Predictable Synchronization Algorithms for Asynchronous Critical Sections

Evandro Coan <https://github.com/evandrocoan>
Eduardo Demeneck Onghero <do.demeneck@gmail.com>

Table of contents

Motivação ←

A abordagem tradicional para a coordenação do acesso à recursos compartilhados em ambientes multi thread envolve a utilização de algoritmos bloqueantes de sincronização. Esses algoritmos forçam threads a bloquearem nos casos de disputas no acesso à recursos compartilhados.

Uma outra abordagem possível envolve o uso de algoritmos baseados em delegação e sincronização não bloqueante para a coordenação de threads. Com esse tipo de algoritmo, threads delegam a execução de operações envolvendo acessos a recursos compartilhados à outras threads. A sincronização não bloqueante é utilizada para garantir certas propriedades às operações dos algoritmos baseados em delegação, de modo a evitar problemas decorrentes do bloqueio de threads, como variações de latência e inversões de prioridade, altamente relevantes no contexto de sistemas embarcados multi core, que, por interagirem diretamente com objetos físicos, possuem necessidades diferenciadas de latência e throughput.

Objetivos ←

O objetivo principal deste trabalho é a implementação de um algoritmo de sincronização com suporte à seções críticas assíncronas.

Objetos específicos ←

- Implementação de um algoritmo de sincronização baseado em delegação para seções críticas assíncronas, utilizando o conceito seções guardadas.
- Implementação de uma nova versão do algoritmo com suporte para seções críticas síncronas.
- Realização de testes para avaliar a corretude dos algoritmos implementados.

Metodologia ←

Será realizada uma pesquisa na literatura com foco nos conceitos básicos que fundamentam algoritmos de sincronização para seções críticas assíncronas, de modo a compreender detalhes de funcionamento desse

tipo de algoritmo e então decidir sobre a melhor maneira de implementá-los no EPOS.

Após entendidos claramente os conceitos, o desenvolvimento seguirá com a metodologia programar e testar (Cowboy Coding), que consiste em escrever um pedaço de código e testar sua execução, verificando se o resultado consiste com o esperado. Os programas de teste serão programas simples, que permitam verificar se o sistema está respondendo corretamente às expectativas.

Cronograma ←

Tarefa	01/10	08/10	22/10	05/11	19/11	26/11	21-28/11
T0	D0						
T1		D1					
T2			D2				
T3				D3			
T4					D4		
T5						D5	
Apresentação							Ap

Tarefas e Entregáveis ←

T0 - Planejamento ←

- Escrever o plano de projeto detalhado, contendo uma fundamentação teórica robusta, apresentando os conceitos chave relacionados à algoritmos de sincronização para seções críticas assíncronas, assim como exemplos de algoritmos desse tipo. Além disso, o plano deve conter o detalhamento do projeto, explicando o que será feito durante o semestre e quais entregáveis serão produzidos.
- Escrever programas de teste que provem a viabilidade do projeto, demonstrando que as operações CAS e FAS, necessárias para a implementação do algoritmo, são suportadas pelo EPOS.

Entregável: D0

1. Plano de projeto
    1. Descrição da metodologia adotada.
    2. Lista de tarefas/entregáveis para cada entrega com seu cronograma.
    3. Fundamentação teórica.

T1 - Revisão do Planejamento ←

- Realizar as correções no plano de projeto com base no feedback obtido no D0.

Entregável: D1

1. Plano de projeto revisado.

T2 - Implementação I ←

- Implementação de um novo componente de sincronização no EPOS. Esse componente será

responsável por prover mecanismos para a execução não bloqueante de seções críticas assíncronas. Uma limitação dessa primeira versão do componente está relacionada aos tipos de seções críticas suportadas, que no caso serão apenas seções críticas assíncronas. Isso porque, o suporte a seções críticas síncronas exige a utilização de mecanismos para a sincronização unilateral de threads, que serão o tema principal da tarefa T3.
- Definição de uma convenção de programação, que poderá ser utilizada por threads do EPOS para solicitar a execução assíncronas de seções críticas, e que juntamente com o componente Guard, compõe um sistema de execução que possibilita a execução de seções críticas assíncronas de maneira não bloqueante.

Entregável: D2

1. Novo componente de sincronização do EPOS, o Guard.
2. Uma convenção de programação para a submissão de requisições de execução de seções críticas.

T3 - Implementação II ←

- Estender o componente de sincronização apresentado em T2 para suportar também seções críticas síncronas. Para isso, serão utilizados *futures*, cuja implementação será baseada em uma descrição apresentada em "Guarded sections: Structuring aid for wait-free synchronisation". G. Drescher and W. Schröder-Preikschat (2015) [2].
- Modificar a convenção de programação para integrar os mecanismos de *futures*, de modo a intermediar a comunicação entre threads e seções críticas bloqueantes.

Entregável: D3

1. Componente Guarda com suporte a seções críticas síncronas.
2. Novo mecanismo de *futures*.

T4 - Implementação III ←

- Aprimorar o modo como as seções críticas são definidas e permitir que sejam definidas seções críticas a partir de funções com um número arbitrário de parâmetros, assim como ocorre com threads.
- Substituir os mecanismos de sincronização das aplicações de teste pela nova implementação com uso de guarda.

Entregável: D4

1. As aplicações *synchronizer_test.cc*, *scheduler_cpu_affinity_test.cc* e *semaphore_test.cc* utilizando o algoritmo de guarda em vez de mutexes e semáforos.

T5 - Validação ←

- Realização de testes em nível acadêmico, com o objetivo de garantir, na medida do possível, a corretude dos componentes implementados até aqui.
- Desenvolvimento de um relatório final, onde será apresentado o que foi feito durante o projeto e quais foram seus resultados.

Entregável: D5

1. Relatório final, que é uma versão do relatório inicial com as correções de planejamento adicionadas ao longo da execução do projeto.
2. O código dos novos testes realizados, junto com o resultado dos testes

Tarefas Opcionais ←

Ver a seção Lista de futuros projetos para o semestre que vem.

Fundamentação Teórica ←

Antigamente quando não existia poder computacional, não existia o conceito atual de threads por que não havia memória suficiente. Assim toda a programação de múltiplas tarefas deveria ser feita pelo programador manualmente, para poder-se extrair o máximo da máquina. Agora, hoje em dia com processadores com milhares de núcleos, volta a ter que se programar a máquina mais diretamente para poder garantir 100% (cem por cento) de throughput de todos os núcleos dos processadores.

O trabalho de referência [1] propõe dois algoritmos de baixo nível, um para arquiteturas UMA {1}, outro para arquiteturas NUMA {2}, que permitem que o programador programe a iteração com a máquina, de forma que ele diga o que a máquina deve fazer, enquanto aguarda que suas seções críticas sejam executadas assincronamente. Regiões críticas assíncronas significam que threads podem requisitar a execução de arbitrárias regiões críticas, sem ter que bloquear a sua execução. Somente a região crítica terá sua execução adiada, devido a necessidade da exclusão mútua. Enquanto a região crítica aguarda sua execução em uma fila, a thread principal pode continuar sua execução. Em contraste, as tradicionais regiões críticas síncronas, exigem que a thread bloqueie a sua execução até que a região crítica seja executada.

Entretanto, note que o programador da aplicação deve tratar de isolar explicitamente em seu código quais são as suas regiões críticas com normalmente faz, e chamar/enviar essas regiões críticas para a fila da exclusão mútua dos algoritmo utilizado. A diferença é que elas não serão mais bloqueantes, assim o programador deve prever o que irá acontecer quando ele chegar em um ponto de execução de seu código, que é dependente dos resultados da execução da região crítica. Assim, no pior dos casos, a execução da thread terá que bloquear a execução, enquanto aguarda a chegada das novas informações provenientes da região crítica.

Sincronização não bloqueante ←

Em ambientes multi thread, técnicas de sincronização precisam ser empregadas para coordenar o acesso à recursos compartilhados e evitar a ocorrência de problemas, como por exemplo, condições de corrida, que podem levar o sistema a se comportar de maneira indefinida ou errônea. No contexto da sincronização de threads, as partes do programa onde recursos compartilhados são acessados por diferentes threads são chamadas de seções críticas.

Técnicas tradicionais de sincronização utilizam mecanismos bloqueantes para coordenar a execução de seções críticas, impedindo que diferentes seções críticas executem simultaneamente e garantindo a propriedade de exclusão mútua. Nesses casos, quando uma thread tenta executar uma seção crítica, e os recursos compartilhados relacionados à essa seção crítica estão ocupados, a thread é bloqueada até que esses recursos sejam liberados. Esse comportamento simplifica a implementação de sistema concorrentes, mas pode levar a problemas de inversão de prioridade, tempos de espera indefinidos, convoying e até deadlocks.

Uma estratégia diferente para lidar com a sincronização de threads envolve a utilização de mecanismos e técnicas de sincronização não bloqueantes. Diferente do que ocorre com as técnicas tradicionais, no caso da sincronização não bloqueante, recursos compartilhados podem ser acessados concorrentemente por múltiplas threads sem a necessidade de bloqueio. Para isso, algoritmos não bloqueantes são cuidadosamente construídos com base em primitivas de sincronização atômicas do tipo *read-write-modify*, geralmente implementadas em hardware, e podem ser utilizados para aumentar o nível de paralelismo do sistema.

Um nicho onde a sincronização não bloqueante se apresenta como uma alternativa viável é no mundo das aplicações para sistemas embarcados multicore. Como essas aplicações geralmente lidam diretamente com entidades físicas, elas se beneficiam de algumas das vantagens desse tipo de algoritmo, como suas garantias de progresso e a preditibilidade de latência de suas operações.

Garantias de Progresso ←

Algoritmos de sincronização não bloqueantes podem ser classificados de acordo com as garantias de progresso de suas operações. Quando todas as operações de um determinado algoritmo oferecem garantias de progresso de um determinado nível, ou de níveis superiores, pode-se dizer esse é o nível de garantias de progresso oferecido pelo algoritmo como um todo. As garantias de progresso de operações de sincronização não bloqueantes são classificadas em três níveis:

- Obstruction-free: É a garantia de progresso de nível mais baixo. Uma operação é considerada obstruction-free se uma thread, executando de maneira isolada, sem sofrer interferências, i.e. competir com outras threads por recursos {3 section 2.1}, consegue completar sua execução em um número finito de passos.

- Lock-free: Um operação é lock-free, se quando invocada por múltiplas threads, garante que pelo menos uma delas threads irá terminar em um número finito de passos. Dessa forma, operações lock-free nunca sofrem com problemas de deadlock e livelock {4}, pois pelo menos uma das threads executando a operação irá progredir. Implementações de soluções lock-free geralmente envolvem a utilização de laços, que geralmente envolvem a primitiva atômica CAS, onde threads repetem certos passos um número arbitrário de vezes, que depende da comportamento da operação, até conseguir prosseguir. Certos autores referem-se a algoritmos lock-free e algoritmos não bloqueantes de maneira análoga, o que pode gerar certa confusão, já que nem todo algoritmo não bloqueante é lock-free {5}, pois podem também ser categorizados como wait-free ou obstruction-free.

- Wait-free: Uma operação é dita wait-free, se, quando invocada por múltiplas threads, garante que todas irão terminar em um número finito de passos. Essa propriedade é especialmente importante para sistemas com grande escala de concorrência, pois independentemente do número de threads executando operações wait-free concorrentemente, nenhuma jamais precisará esperar ou bloquear [13], e como o progresso é garantido para todas as threads, nenhuma sofrerá com problemas de *starvation*.

Primitivas de Sincronização ←

A criação de algoritmos de sincronização não bloqueantes pode envolver a utilização de primitivas de sincronização atômicas do tipo *read-write-modify*. Essas primitivas geralmente leem uma posição de memória e, simultaneamente, escrevem um novo valor em seu lugar. Exemplos de operações atômicas

desse tipo são: compare-and-swap (CAS), fetch-and-operation (FAθ), load-link/store-conditional (LL/SC) e test-and-set (TAS). A seguir serão detalhadas duas dessas operações, CAS e FAS (fetch-and-store), uma especialização da operação FAθ. Ambas são utilizadas na implementação de um algoritmo de sincronização não bloqueante que será apresentado mais adiante.

A operação FAS realiza, de maneira atômica, uma leitura e uma escrita em um endereço de memória específico, e pode ser entendida como uma versão atômica do pseudocódigo, apresentado a seguir:

Listagem 1: Pseudocódigo da implementação da operação FAS

```
int fas(address * location, int replacement) {
        int old = *location;
        *location = replacement;
        return old;
}
```

A operação CAS, cuja implementação lógica em pseudocódigo pode ser vista abaixo, possui um funcionamento similar ao da operação FAS, no entanto, a escrita no endereço de memória sendo acessado ocorre de forma condicional, acontecendo apenas caso o resultado da comparação entre o valor atual desse endereço e o valor de uma determinada variável seja positivo. Na prática, a escrita só acontece caso o valor do endereço sendo acessado já é conhecido por quem está executando o CAS.

Listagem 2: Pseudocódigo da implementação da operação CAS

```
int cas(address location, int compare, int replacement) {
    int old = *location;
    if(*location == compare) {
        *location = replacement;
    }
    return old;
}
```

Um detalhe importante sobre essas operações é o fato de processadores geralmente não suportarem todos os tipos de primitivas *read-write-modify* em hardware. A arquitetura RISC-V, por exemplo, implementa apenas as operações FAθ e LL/SC, enquanto a arquitetura x86 não implementa a operação LL/SC, mas implementa a operação CAS.

Sincronização Baseada em Delegação ←

Com algoritmos de sincronização baseados em delegação, threads podem delegar a execução de suas seções críticas à outras threads. Com isso, as seções críticas são desacopladas das threads que requisitam sua execução, que agora podem decidir se continuam trabalhando, sem bloquear, após delegar a execução de uma seção crítica, ou se preferem bloquear e continuar apenas após a execução da seção crítica terminar. Além disso, deve haver algum protocolo ou sistema de execução cujos mecanismos garantam que as seções críticas eventualmente serão executadas e que a propriedade da exclusão mútua será mantida.

Técnicas de sincronização baseadas em delegação podem definir uma única thread, geralmente presa a um core específico, como responsável pela execução das seções críticas do programa. Um dos benefícios dessa abordagem é o melhor aproveitamento da localidade de dados das caches do sistema, já que agora, todos os acessos aos dados das seções críticas são feitos pela mesma thread. Uma outra possível abordagem, permite que a responsabilidade de execução de seções críticas seja transferida entre threads de acordo com as necessidades do sistema. Nesse caso, diminui-se o aproveitamento da localidade das caches, mas remove-se a necessidade de se manter uma thread, ou até mesmo um core do processador, restrita a execução de seções críticas.

Em situações ideais, onde threads não precisam do resultado de suas seções críticas, ou seja, não existem dependências unilaterais de dados entre a seção crítica e a thread requisitando sua execução, o bloqueio dessas threads se torna opcional. Seções críticas que exibem essas características são chamadas de seções críticas assíncronas. Algoritmos de sincronização baseados em delegação para seções críticas assíncronas permitem que threads deleguem a execução de seções críticas utilizando mecanismos do tipo fire-and-forget, podendo continuar com sua própria execução e esquecer sobre a seção crítica cuja execução foi requisitada.

No entanto, isso nem sempre é possível e, em alguns casos, a sincronização baseada em delegação traz novos desafios. Caso existam dependências de dados entre uma seção crítica e a thread que requisitou sua execução, essa seção crítica é caracterizada como uma seção crítica síncrona. Com esse tipo de seção crítica, surge a necessidade de bloqueio da thread requerente, o que pode ocorrer no momento em que é feita a requisição de execução, ou apenas quando os dados que podem ainda não ter sido calculados, são acessados. No primeiro caso, podem ser utilizados mecanismos de bloqueio simples, como semáforos ou mutexes, enquanto no segundo, exige-se a utilização de mecanismos mais complexos, como *futures* e *observables*.

Guardas ←

Todos os conceitos apresentados até aqui convergem em um algoritmo de sincronização, baseado em delegação, para seções críticas assíncronas, que dá origem a uma nova entidade de sincronização: a Guarda. Guardas possuem duas operações básicas, uma para a entrada e uma para a saída de seções críticas, vouch e clear, respectivamente, além de uma convenção de programação, apresentada na Listagem 1, que deve ser utilizada pelas threads do programa para requisitar a execução assíncrona de seções críticas.

Listagem 3: Protocolo para a requisição de execução de seções críticas

```
Guard guard = ...; // Shared
...
Job * job = ...
Job * cur;
if (NULL != (cur = guard.vouch(job))) do {
    run(cur);
} while (NULL != (cur = guard.clear()));
```

O algoritmo da guarda também introduz o conceito de uma entidade chamada *sequencer*, que será responsável pela execução das seções críticas do programa. Toda thread, após emitir uma requisição de execução de seção crítica, pode ter que assumir o papel de *sequencer*. Ao assumir esse papel, a thread

deve trabalhar na execução de seções críticas até que não hajam mais requisições pendentes a serem executadas. A convenção de programação definida para submissão de requisições, junto com o modo de funcionamento das operações vouch e clear, impede que mais de um *sequencer* esteja ativo ao mesmo tempo, fazendo com que a execução das seções críticas ocorra de forma sequencial, garantindo a propriedade da exclusão mútua.

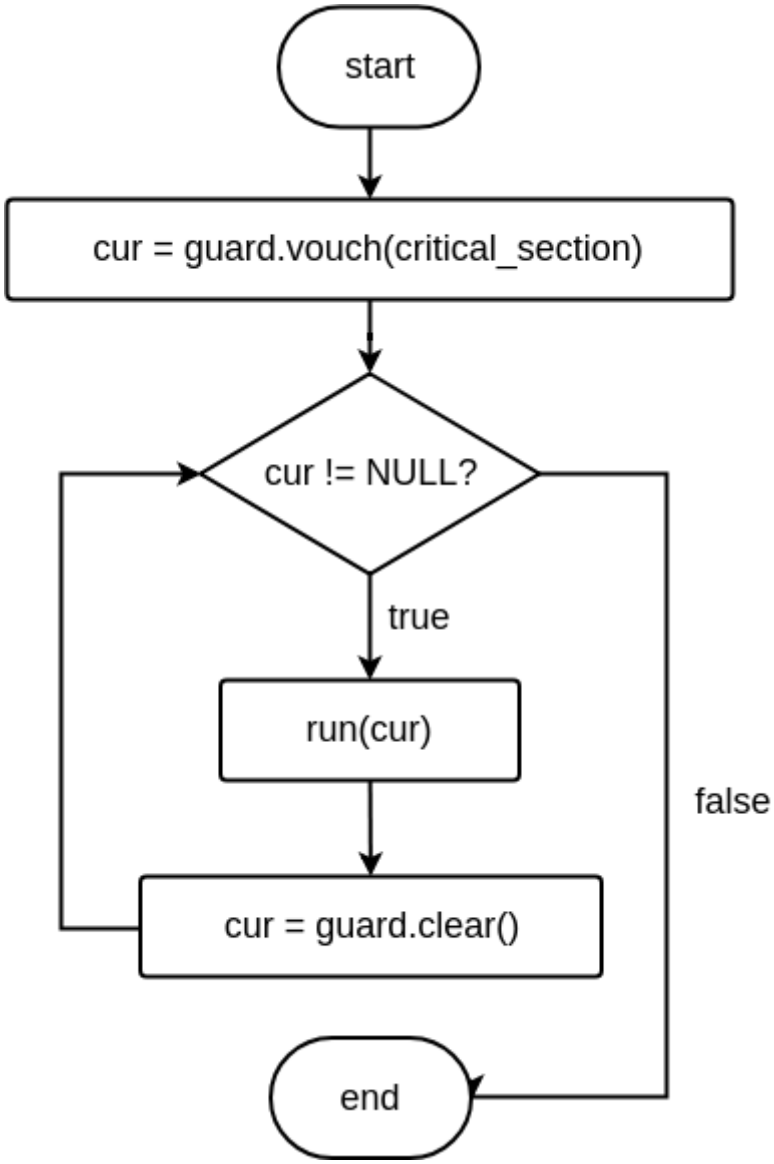

Figura 1: Diagrama demonstrando o funcionamento do protocolo de submissão de seçẽos críticas
Fonte:: Própria

Em termos gerais, o funcionamento do protocolo de submissão de requisições de execução de seções críticas à guarda, que pode ser observado no diagrama da Figura 1, se dá da seguinte maneira: seguindo a convenção de programação estipulada, threads devem realizar um vouch para submeter requisições à guarda. Caso já exista um *sequencer* ativo, vouch retornará *NULL*, sinalizando para a thread requerente que ela pode continuar com sua execução. No entanto, caso nenhuma thread esteja atuando como *sequencer*, vouch retornará um job, sinalizando para a thread que ela deve atuar como *sequencer*, começando pela execução do job retornado. Após terminar a execução desse job, a thread *sequencer* invoca a operação clear repetidamente, e enquanto houverem seções críticas à serem executadas, clear retornará novos jobs, que também serão executados pelo *sequencer*. Quando não houverem mais seções críticas à serem executadas, clear retornará *NULL*, sinalizando à thread que ela deve abandonar o papel de

sequencer e continuar com sua execução normal.

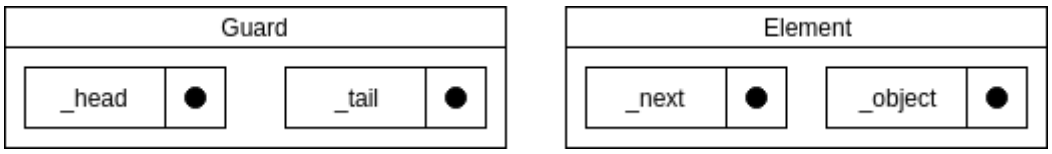

Figura 2: Atributos da guarda e de seus elementos
Fonte:: Própria

A partir da descrição geral do funcionamento do algoritmo, percebe-se que guardas comportam-se como filas, onde jobs representando seções críticas são armazenados. A Figura 2 representa a estrutura geral de uma guarda e de seus elementos, que correspondem a elementos de fila simples, com apenas um nível de encadeamento. Além de ser do tipo FIFO, como apenas o *sequencer* pode remover elementos, a fila representada pela guarda pode ser considerada uma fila multiple-producer-single-consumer (MPSC), pois, enquanto várias threads podem, simultaneamente, inserir elementos na guarda, apenas o *sequencer* pode removê-los. Essa característica diminui muito a complexidade de implementação de operações wait-free para a inserção e remoção de elementos, como poderá ser notado quando o código das operações vouch e clear for apresentado.

Uma implementação para vouch é apresentada na Listagem 2. Essa implementação utiliza as primitivas atômicas CAS e FAS, sinalizadas por V1 e V2 no código, para coordenar acessos a elementos da guarda compartilhados por múltiplas threads, de maneira a evitar a ocorrência de condições de corrida. Mais especificamente, FAS é utilizada, em V1, para coordenar a inserção de novos elementos na guarda por invocações distintas de vouch. Enquanto CAS é utilizada, em V2, para lidar com o caso especial onde a guarda possui apenas um elemento, que pode ser acessado simultaneamente por vouch e clear, e, portanto, deve ser acessado de maneira coordenada.

Listagem 4: Implementação da operação vouch

```
Element * vouch (Element * item) {
    item->next = NULL ;
    Element * last = FAS(_tail, item ); // V1
    if (last && CAS (last->_next, NULL, item )) // V2
        return NULL ;
    _head = item ; // V3
    return item ;
}
```

A operação clear é responsável por remover elementos da guarda. Implementações wait-free de operações de remoção de elementos de estruturas de dados tendem a ser extremamentes complexas, no entanto, como destacado previamente, o comportamento MPSC da guarda acaba simplificando muito o código de clear, apresentado na Listagem 3. Assim como no caso de vouch, as operações FAS e CAS são utilizadas para garantir que invocações simultâneas de clear por múltiplas threads nunca resultem em condições de corrida. No trecho de código sinalizado por C1 na Listagem 3, a operação FAS é utilizada para coordenar a definição de qual elemento deve ser a próxima cabeça da guarda, cujo valor é armazenado na variável next, além disso, FAS também marca, simultaneamente, next->_next com o valor mágico DONE, utilizado

para controlar a interação entre execuções simultâneas de vouch e clear, onde pode haver a necessidade de uma transição de *sequencer*. Caso a variável next tenha recebido o valor *NULL*, tem-se que a lista possui menos de um elemento, e a semântica de clear indica que não apenas a cabeça da lista deve ser modificada mas também a sua cauda, nesse caso, ambas modificações afetam o mesmo elemento e precisam ser coordenadas, por isso são implementadas por operações CAS.

As implementações de vouch e clear apresentadas são wait-free, pois permitem a invocação dessas operações por múltiplas threads com garantias de progresso para cada uma delas. Dessa forma, caso já exista um *sequencer* ativo, o progresso wait-free de todas as threads que submeterem requisições à guarda é garantido, pois esse processo envolve apenas uma invocação à vouch, que também é wait-free. No entanto, as garantias de progresso se tornam mais complexas para o caso de threads que precisam assumir o papel de *sequencer*. Em sistemas onde threads submetem seções críticas à guarda com muita frequência, uma thread que assume o papel de *sequencer* pode ter que executar seções críticas indefinidamente, o que a impedirá de progredir com o resto do seu código. Além disso, seções críticas maliciosas, que executam indefinidamente, também podem impedir o progresso do *sequencer*. No entanto, esse tipo de seção crítica nunca deve existir, e sua presença pode ser considerada um erro de programação.

Listagem 5: Implementação da operação clear

```
Element * clear() {
    Element * item = _head;
    Element * next = FAS(item->_next, DONE); // C1
    if (!next)
        CAS(_tail , item, NULL); // C2
    CAS(_head, item, next);  // C3
    return next ;
}
```

Até agora, o algoritmo tratou apenas de seções críticas assíncronas, onde o código das threads não depende de dados calculados nas seções críticas, mas esse nem sempre é o caso. Dessa forma, o algoritmo da guarda pode ser estendido para lidar também com seções críticas síncronas. Para isso, são adicionados mecanismos como *futures* [14], que permitem a comunicação de dados entre threads e suas seções críticas, de modo a satisfazer dependências unilaterais de dados.

Além dos problemas já citados, várias considerações podem ser feitas em relação à threads de prioridades diferentes compartilhando uma mesma guarda. Nesses casos, inversões de prioridade podem acontecer nos casos onde threads de alta prioridade se tornam *sequencer* e são forçadas a executar seções críticas de threads de baixa prioridade, ou em casos onde seções críticas relacionadas à threads de alta prioridade são executadas por threads de baixa prioridade. Para solucionar esses problemas, podem ser desenvolvidas variações do algoritmo, onde o papel de *sequencer* pode ser renegociado para que as definições de prioridade do sistema sejam respeitadas.

Atores ←

Existe uma variação do algoritmo da guarda, chamado de algoritmo dos atores, onde a execução das seções críticas é realizada por uma thread servidora dedicada. Em aspectos funcionais, os dois algoritmos são muito parecidos. Ambos possuem duas operações principais, uma para a inserção e uma para a

remoção de seções críticas de filas, e uma convenção de programação que deve ser utilizada pelas threads do programa para delegar a execução de suas seções críticas.

Algoritmos de sincronização baseados em delegação onde uma única thread dedicada é responsável pela execução de seções críticas aproveitam ao máximo a localidade no acesso à dados compartilhados, mas pagam um preço por restringirem uma thread, ou até mesmo um core do processador, apenas à execução de seções críticas. Devido a essas característica, o algoritmo dos atores tende a se comportar melhor em sistemas com muitos núcleos de processamento (many-core).

Outro detalhe a ser considerado com esse tipo de abordagem é o overhead decorrente da necessidade da thread dedicada ser posta para dormir quando não existem seções críticas à serem executadas.

Comparação entre o uso do guarda e semáforo ←

Uma versão equivalente ao uso de guardas, pode ser implementada utilizando várias threads e um semáforo inicializado em 1 para controle da entrada da seção crítica, i.e., exclusão mútua. Na implementação apresentada para o algoritmo de guardas, utiliza-se um ponteiro de função para passar para o *sequencer*, qual será a seção crítica a ser executada. Enquanto, na versão com threads, teria-se que criar uma nova thread, além da thread atual, passando um ponteiro de função para a seção crítica. E então essa nova thread irá testar o valor do semáforo antes de executar a seção crítica.

A seguir, vê-se resumidamente o fluxo de execução do algoritmo do guarda, já explicado anteriormente. Nele, a *Thread A* primeiro empilha um ponteiro de função **job** na fila de tarefas da variável global **my_guard** e então verifica se a *Thread A*, deve ou não se tornar o *sequencer*. Como a *Thread A* é a primeira thread a empilhar uma região crítica, **guard_vouch()** retorna um valor não nulo, indicando a próxima tarefa a ser executada. Assim, a *Thread A* torna-se o *sequencer*. E logo em seguida, chama o método **do_things()** que executa a função que contém a região crítica representado por **next_job**. No final da execução, a *Thread A* chamará **guard_clear()**, que neste ambiente de exemplo retornará *NULL*, pois não há outras threads/seções críticas.

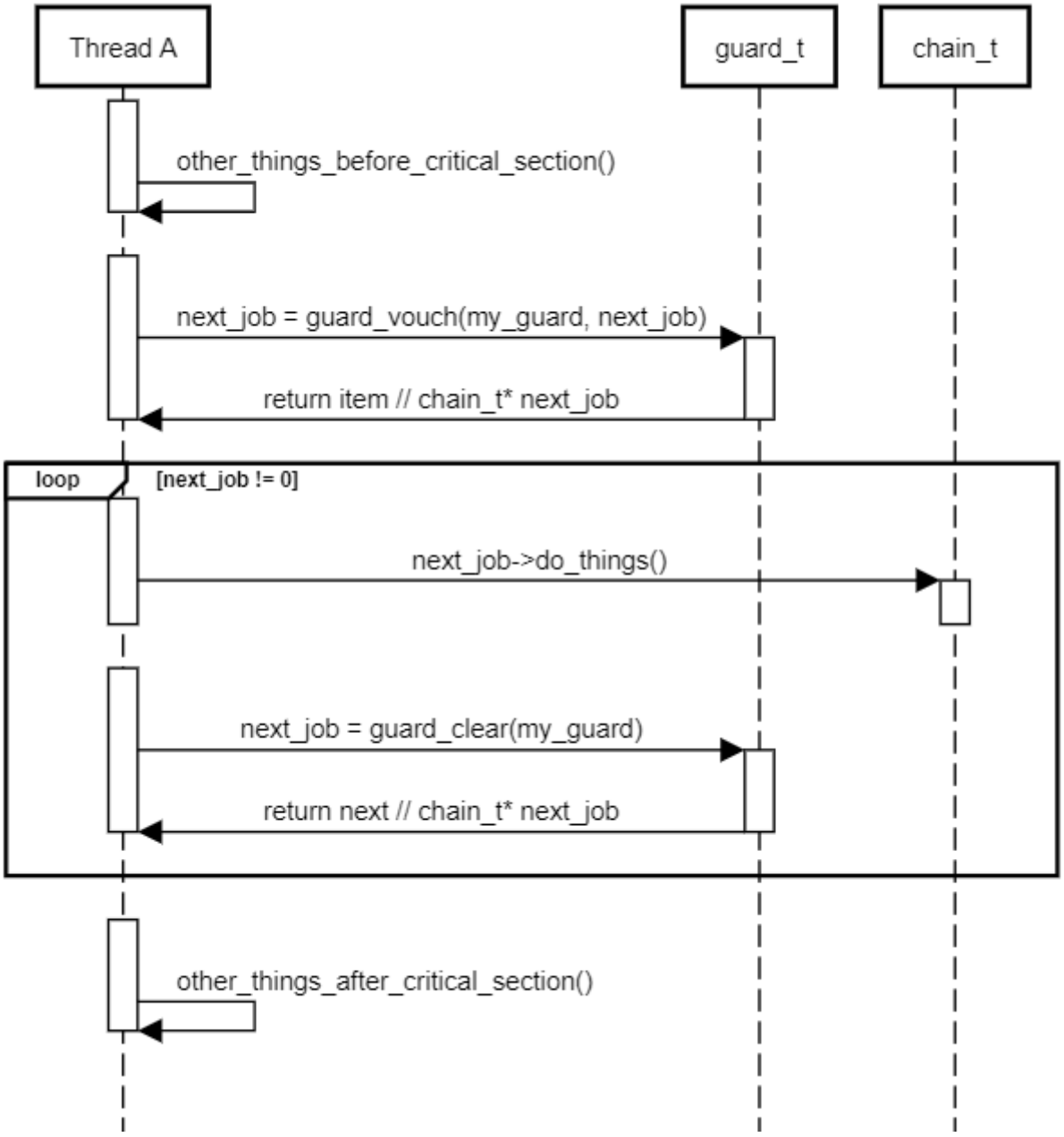

Figura 3: Resumo do algoritmo do Guarda
Fonte: Própria

Agora, vê-se um equivalente do mesmo exemplo do guarda, mas utilizando um semáforo global inicializado em 1 para todas as threads, pois para garantir exclusão mútua somente pode existir uma thread executando a região crítica. Primeiro, cria-se uma thread com new, passando como parâmetro um **semáforo** compartilhado e um ponteiro de função **job**. É importante notar que não se faz **join()** nesta thread, pois se está tratando de uma seção crítica assíncrona, e precisa-se também de uma implementação especial de thread que antes de chamar o ponteiro de função de **job**, dê um *lock()* no semáforo, e depois de completar a função, delete a si própria. Comparando este novo código com o algoritmo do guarda, vê-se que ele ficou muito mais simples, entretanto, perde-se muito na eficiência pois para executar uma seção crítica assincronamente, tem-se que criar exclusivamente uma nova thread para cada uma das threads já existentes, tendo a criação do dobro de threads no sistema em um momento de alta competição pela seção crítica.

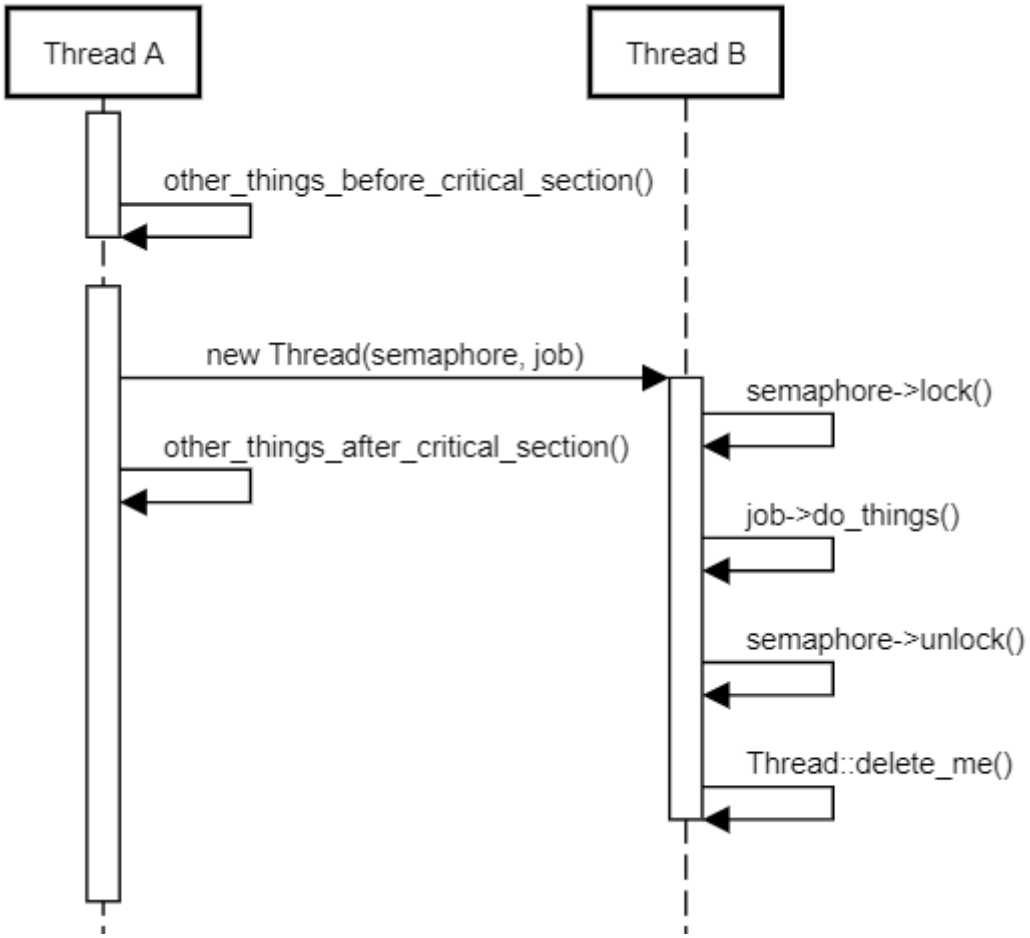

Figura 4: Resumo do funcionamento de um semáforo, equivalente ao algoritmo do Guarda
Fonte: Própria

O leitor mais atento pode perceber que, abordou-se somente uma implementação sem dependência de dados com a seção crítica e, que a versão do algoritmo do guarda apresentado neste exemplo também não executou assincronamente. Somente a segunda versão, que utiliza um semáforo fez uma execução assíncrona da seção crítica. Este é um dos problemas do algoritmo do guarda. A thread que for eleita como *sequencer*, será impedida de executar sua seção crítica assincronamente, pois ela própria tem que executar a sua seção crítica mais as seções críticas das outras threads que empilharem seus **jobs**. Com isso, pode-se ter como já falado anteriormente, o problema de *starvation* do *sequencer*, que pode infinitamente continuar recebendo novos **jobs** para executar. Entretanto, ter-se somente uma thread exclusivamente executando todas as seções críticas, traz-se a vantagem de localidade da cache caso o *sequencer* execute sempre em um mesmo núcleo do processador.

Outro problema que tanto a versão com semáforo quanto a versão com guardas apresentam, é quando a seção crítica é uma função recursiva. No caso do semáforo, o sistema entrará em deadlock durante a primeira recursão, pois devido a exclusão mútua que o algoritmos possuem, somente existe uma função executando a seção crítica ao mesmo tempo, e o deadlock será imediato ao início da chamada recursiva. A diferença é que o deadlock, não impede o progresso das regiões não críticas do sistema, por que por exemplo, na segunda vez que for chamado **guard_vouch()**, ele irá empilhar um ponteiro de função **job** que irá esperar para sempre, mas o *sequencer* continuará executando o resto do programa que vem depois da seção crítica, pois este é o comportamento do guarda quando já existem **jobs** na seção crítica.

Este foi um exemplo de problema tanto do algoritmo do guarda quando com semáforos, onde uma região crítica pode impedir que seções críticas sejam executadas, mas permitir que as regiões não críticas

continuem executando. Note que isso pode causar falta de memória no sistema pois, toda que uma seção crítica tentar ser executada, mais um ponteiro de função (ou thread no caso do semáforo) será empilhado na fila de execução do *sequencer*, que ficará eternamente como *sequencer* sem executar mais nenhuma linha de código de seções críticas.

Outro problema muito similar do algoritmo do guarda, é que quando a seção crítica é explicitamente bloqueada por algum motivo, seja uma operação de IO ou seja por que ela possuía um semáforo que bloqueou. Bloqueios dentro da seção crítica no algoritmo do guarda, causarão diretamente a degradação/atraso da execução de todas as seções críticas do sistema, pois é somente a thread do *sequencer* que é habilitado a executar as seções críticas. Note também que na versão assíncrona implementada com semáforo, terá-se a possibilidade de travar todo o sistema caso aconteça o mesmo problema, pois mesmo que cada thread execute isoladamente, atrasos na execução de uma seção crítica, atrasam a liberação do semáforo, o que causa o atraso no sistema como um todo.

Futures ←

Seguindo as referências bibliográficas, encontrou-se que *futures* foram descritas pela primeira vez em 1977 no artigo "The incremental garbage collection of processes". *Futures* facilitam a resolução do problema de bloqueio por dependência de dados. No diagrama de sequência a seguir, como implementar o bloqueio no *bloco 4* e sinalizar para o *bloco 4* que o *bloco 2* já terminou, enquanto o *bloco 3* executa?

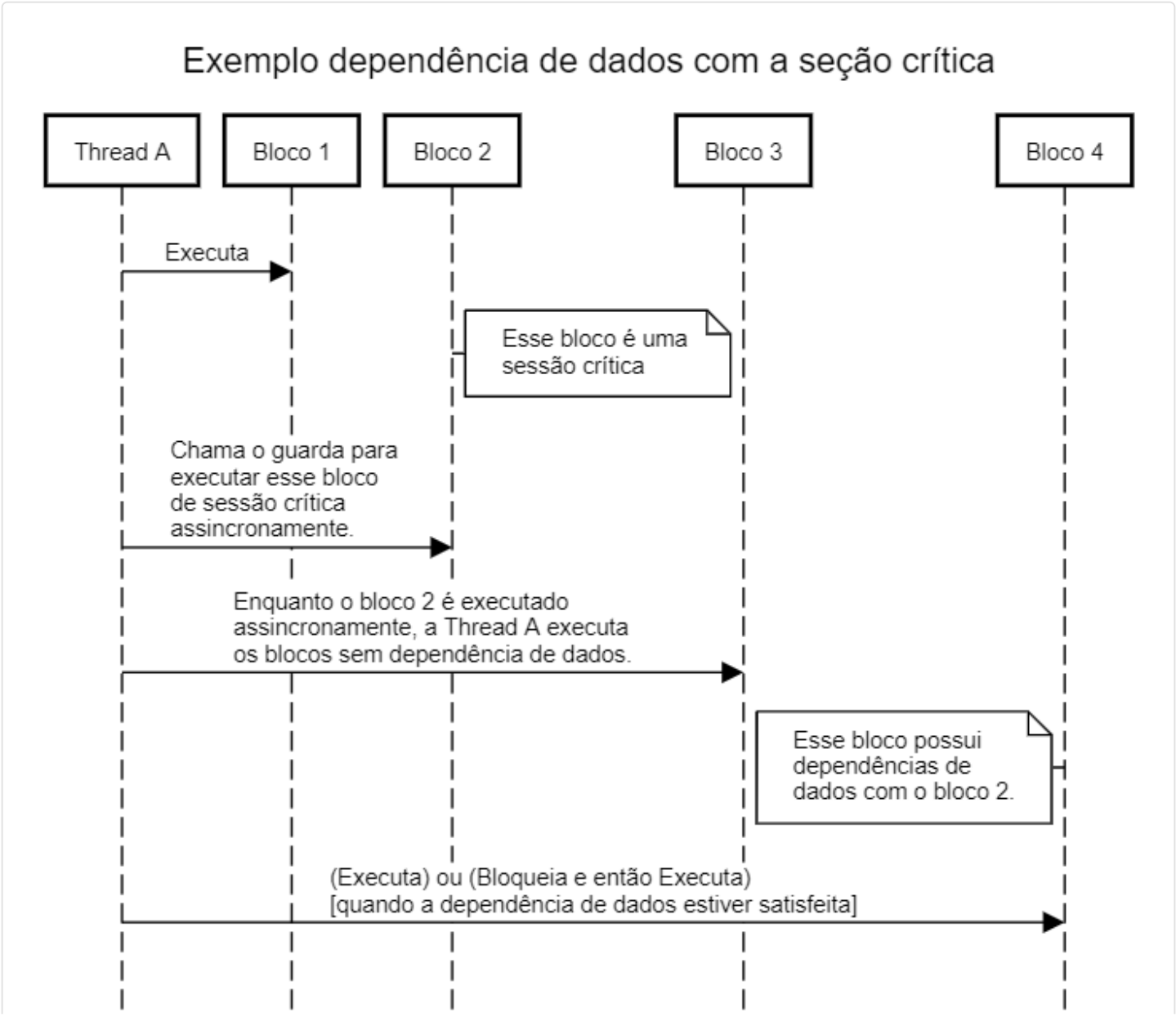

Figura 5: Exemplo de dependência de dados com a seção crítica
Fonte: Própria

O *bloco 4* somente pode ser executado quando a sessão crítica no *bloco 2* já tiver sido executada. Uma vez que o fluxo de execução chega nesse ponto, a *Thread 1* deve bloquear caso o *bloco 2* ainda não tenha sido executado pelo *sequencer*. Isso não seria um problema para o trabalho por que quando existe dependência de dados, o código que é dependente fica encapsulado em uma **closure** [12] que é chamada quando o resultado do **bloco 2** fica pronto. A seguir ve-se o fluxo de execução desse caso com o uso de *futures*:

Figura 6: Exemplo de dependência de dados com a seção críticas utilizando *Future*
Fonte: Própria

Nessa nova versão com *futures* vê-se um novo problema não abordado pelo artigo de referência [1]. Nesse fluxo de execução, quem deve executar o *bloco 4*? A *Thread A* não pode mais executar este bloco porque ele foi condicionado a ser executado depois do *bloco 2*, que é executado assíncronamente pelo *sequencer*. Entretanto, não pode-se deixar que o *sequencer* execute o *bloco 4* por que o *sequencer* somente é encarregado de executar as seções críticas, e permitir que ele execute outros blocos não críticos, irá atrasar toda pipeline de execução de seções críticas.

Isso trás uma alternativa implementação de *futures* que o artigo [1] sugere, onde cada *future* possui um semáforo acoplado, e a execução do *bloco 4* não é delegada ao *sequencer*, mas sim a *Thread A* que irá bloquear automaticamente quando o fluxo de execução chegar ao *bloco 4* e o resultado do *bloco 2* ainda não estiver disponível. Caso o resultado do *bloco 2* já esteja disponível, a *Thread A* não irá bloquear e seguirá executando o *bloco 4*. Infelizmente, pode-se não querer o programa bloqueie quando existe a dependência de dados explicita como essa alternativa sugere.

Por isso, a seguir vê-se um diagrama de sequência sobre a implementação de *future* descrita no artigo [14] inicial de 1977. Este artigo descreve que, além das tradicionais *chamadas-por-valor* e *chamadas-por-referência*, também existem as *chamadas-por-future*, onde cada parâmetro da função é ligado a um processo separado (chamado *future*). Este processo é dedicado a calcular o valor do argumento que a *future* representa, o que completamente permite a execução paralela dos argumentos da função, assim aumentado o poder expressivo da linguagem de programação.

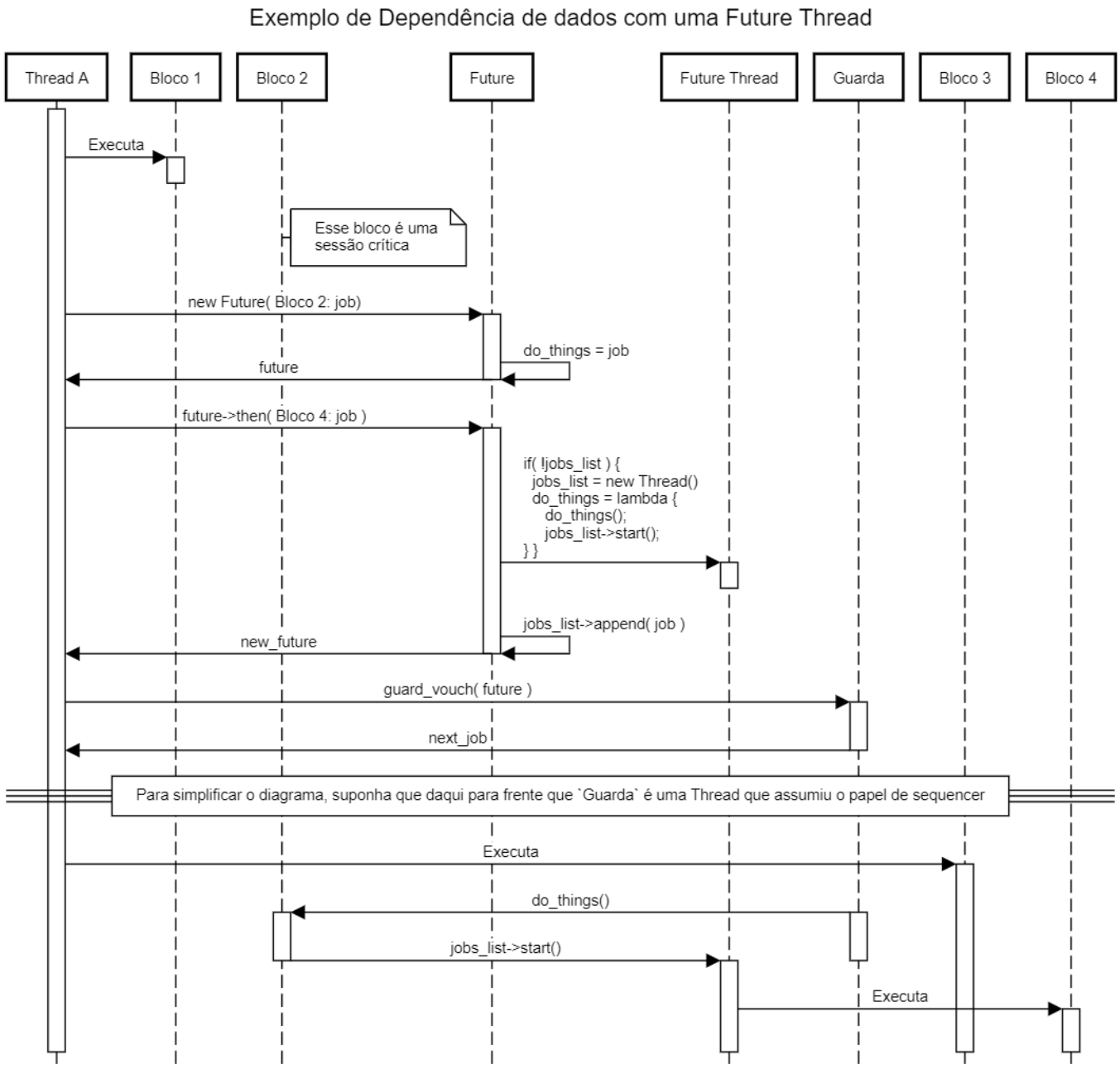

Figura 7: Exemplo de dependência de dados com um *Future Thread*
Fonte: Própria

No exemplo anterior, foi simplificado funcionamento do algoritmo do guarda e abstraiu-se quem é o *sequencer* fazendo com que a entidade que representa tipo **Guarda**, chame o método *do_things()* como se ele fosse o *sequencer*. A versão estritamente correta da implementação seria fazer com que uma thread como *Thread A* fosse o *sequencer*, e então a *Thread A* deveria fazer a chamada de *do_things()*. Além dessa simplificação, assume-se que a implementação de Thread utilizada não inicia a execução imediatamente após sua criação. Ela espera até que o método **start()** seja chamado, e que a thread permita a execução de vários funções, uma após a outra. As funções que esta thread precisa executar são adicionadas através do método *append()*.

Os eventos que acontecem no diagrama de sequência são os seguintes, primeiro a *Thread A* executa um bloco de código não crítico. Depois ela cria uma *Future* com um **job** que é um ponteiro de função para o *bloco 2*. O método *then()* da variável *future* é utilizada para adicionar os blocos de código que são dependentes da seção crítica no *bloco 2*. Os resultados da execução da seção crítica no *bloco 2* são passado em diante para o *bloco 4* como parâmetros que o ponteiro de função de entrada do *bloco 4* aceita.

Depois de criada a *future*, a *Thread A* chama o método *guard_vouch()* passando a *future* como parâmetro. Então o algoritmo do guarda segue o fluxo da sua execução como já explicado na seção Guarda. Por simplicidade, assumi-se que depois que o método *guard_vouch()* retornou, a *Thread A* não assumiu o papel do *sequencer*, mas que a entidade Guarda do diagrama de sequência é o atual *sequencer* em execução, e que já de imediato o *sequencer* chamou o método *do_things()* da *future*. Uma vez que o método *do_things()* completou sua execução, o *sequencer* para de executar o *bloco 2*, e chama o método *jobs_list->start()* que chamada a *Future Thread* para realizar a execução dos blocos dependentes da seção crítica. Uma vez que isso acontece, o *sequencer* inicia a execução de uma outra seção crítica, que foi omitida no diagrama. Assim, permiti-se que o *sequencer* exclusivamente execute seções críticas, enquanto a *Future Thread* executa os blocos dependentes da seção crítica, assincronamente, junto com a execução do *bloco 3* da *Thread A*.

Claramente esta implementação mostrada é simples, e trata-se de uma leve modificação da versão original [14]. Seu defeito é sempre realizar a criação de uma nova thread para cada variável *Future*, pois assim tem-se a criação de muitas threads no sistema, já que sempre cria-se uma thread nova depois que adiciona-se o primeiro bloco com dependência de dados. A implementação inicial [14] dispõe de um serviço de execução que trata de criar novas threads na medida que o sistema necessita. Por exemplo, um sistema embarcado pode não possuir memória suficiente para que muitas thread sejam criadas, então esse serviço de threads limitaria o número máximo de threads que podem ser criadas ao mesmo tempo, e caso esse limite seja ultrapassado, novas requisições aguardam em uma fila, i.e., bloqueiam até que recursos estejam disponíveis para a execução das seções de código dependentes da seção crítica

Um serviço/implementação mais avançado pode automaticamente informar a *Thread A* que, existe um bloco de código dependente da seção crítica ainda não executado, e assim, antes que a *Thread A* termine sua execução, ela verifica se tal condição é verdadeira, e caso, sim, ela aguarda por este bloco está disponível para execução antes de chamar método *exit()* da *Thread A*. Uma desvantagem é que a *Thread A* pode ainda não ter terminado de executar o *bloco 3* quando o bloco dependente da seção crítica estiver terminado. Assim, teria-se o atraso da execução do *bloco 4*. Outra desvantagem desta alternativa é que o bloco de código dependente pode demorar muito tempo antes que ele possa ser executado, e assim, o sistema desperdiçaria a memória ocupada pela *Thread A* que ficou esperando. Assim, a alternativa original de manter um serviço especializado de threads em executar os blocos de código dependentes trás uma melhor vantagem de economia de memória, entretanto ele conta com o overhead de sua manutenção, que conta com criações de novas thread quando a demanda for alta e destruição de threads quando a demanda

por blocos dependentes de código for baixa. Mas talvez em uma implementação mais esperta pode-se utilizar a idle thread [21] do sistema para realizar a manutenção de tal serviço, assim reaproveitando recursos do sistema inutilizados.

Futures versus Promises ←

A linguagem JavaScript recentemente em 2015, incorporou nativamente a classe *Promise* {6}. Conceitualmente elas são muito similares a *futures* [14], mas a implementação de JavaScript é bastante peculiar ao modelo de processamento de JavaScript, que é baseado na existência de uma única thread no sistema [16, 18]. A seguir vê-se uma breve ilustração da única thread que existe no mundo JavaScript. Essa thread chama-se *Event loop*, e tudo o que é feito, passa por ela. No caso da implementação das *promises*, quando uma *promise* é resolvida e obtém o seu valor, elas furam [16] a fila de *callbacks*, para assim serem executadas o mais brevemente possível, preferencialmente mais cedo, ao contrário do modelo usual utilizado para timers, de executar algum tempo depois que o timer expirou, preferencialmente mais tarde pois são adicionados no final da fila.

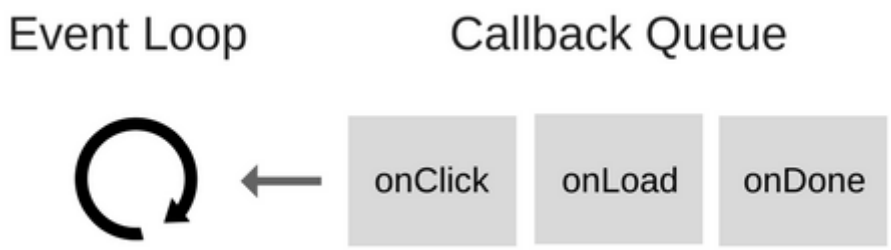

Figura 8: Ilustração do funcionamento programa JavaScript
Fonte: Referência [16]

Listagem 6: Nota Sobre Multithreading em JavaScript

> O ambiente de execução (runtime) JavaScript é do tipo single thread, ou seja, apenas um script é executado de cada vez. Se, por exemplo, o usuário clicar em um botão de uma página e esta ação ("onclick") implicar na execução de um script então nenhum outro script poderá ser executado enquanto aquele estiver sendo executado. Os scripts ficam em uma fila aguardando a vez de serem executados, um de cada vez.
>
> A especificação HTML 5 introduziu o conceito de web worker que permite que um script seja executado em uma thread própria, distinta daquela thread que executa os scripts associados aos eventos produzidos pelo usuário enquanto está interagindo com a página HTML.

Fonte: Referência [22]

Assim, *promises* em JavaScript funcionam não para fazer computações pesadas, mas sim esperar por eventos assíncronos e que podem demorar muito tempo para acontecer, como por exemplo, esperar pela resposta de uma requisição de rede. A seguir vê-se um exemplo do uso de uma *promise* que executa assincronamente. Em JavaScript como somente existe uma thread, então tudo o que se executa é sequencialmente {7}. Na execução de *promises* em JavaScript, refere-se assincronamente para dizer que ao se executar este trecho de código, ele não irá bloquear, e somente algum momento mais tarde (assíncrono, fora de sincronia), a soma será realizada, e novamente sem bloquear a execução da única thread que existe, o *Event loop*.

Listagem 7: Implementação de uma Promise JavaScript

```
function sum(xPromise, yPromise) {
    return Promise.all([xPromise, yPromise])
    .then(function(values) {
        return values[0] + values[1];
    } );
}

sum(fetchX(), fetchY())
.then(function(sum) {
    console.log(sum);
});
```

Fonte: Referência [16]

Assuma de os método *fetchX()* e *fetchY()* returnam alguma *promise* depois de fazerem alguma computação. A função *sum()* mostrada anteriormente, recebe duas *promises* como parâmetro, e chama o método estático *Primise.all()*, que recebe um array de *promises* e retorna uma nova *promise* que irá chamar o seu correspondente *callback* indicado por *then()*, assim que todas as suas "*sub-promises*" forem resolvidas, i.e., efetivamente conterem um valor ao invés de somente serem uma promessa de conter algum valor.

Diferente de JavaScript, em C++ e outras linguagens como Java, existem as implementações dos tipos *Futures* e *Promises*. A biblioteca *std* do C++ defines as classes template *std::promise* {8} e *std::future* {9}. A funcionalidade empregada a esses tipos são a transmissão de resultados da computação de uma thread para outra thread que aguarda os resultados. De maneira tradicional, para receber um valor de uma thread precisa-se compartilhar uma variável de condição e um ponteiro comuns a ambas as threads. Uma vez que a outra thread obtém o valor e coloca o mesmo no ponteiro compartilhado, pede-se para a variável de condição liberar a passagem. Então a outra thread que aguardava o resultado desbloqueia e pode seguir com a execução. A desvantagem dessa abordagem é a necessidade de manter e operar diretamente a variável de condição e o ponteiro, e caso queira-se compartilhar mais resultados entre as diferentes threads, a programação fica ainda mais complicada pois precisa-se manter sincronia com mais variáveis de condição. Já com *std::promise* e *std::promise*, pode-se abstrair essas operações repetitivas e simplificar a programação. A seguir vê-se um exemplo de utilização destas classes para compartilhamento de dados entre duas threads:

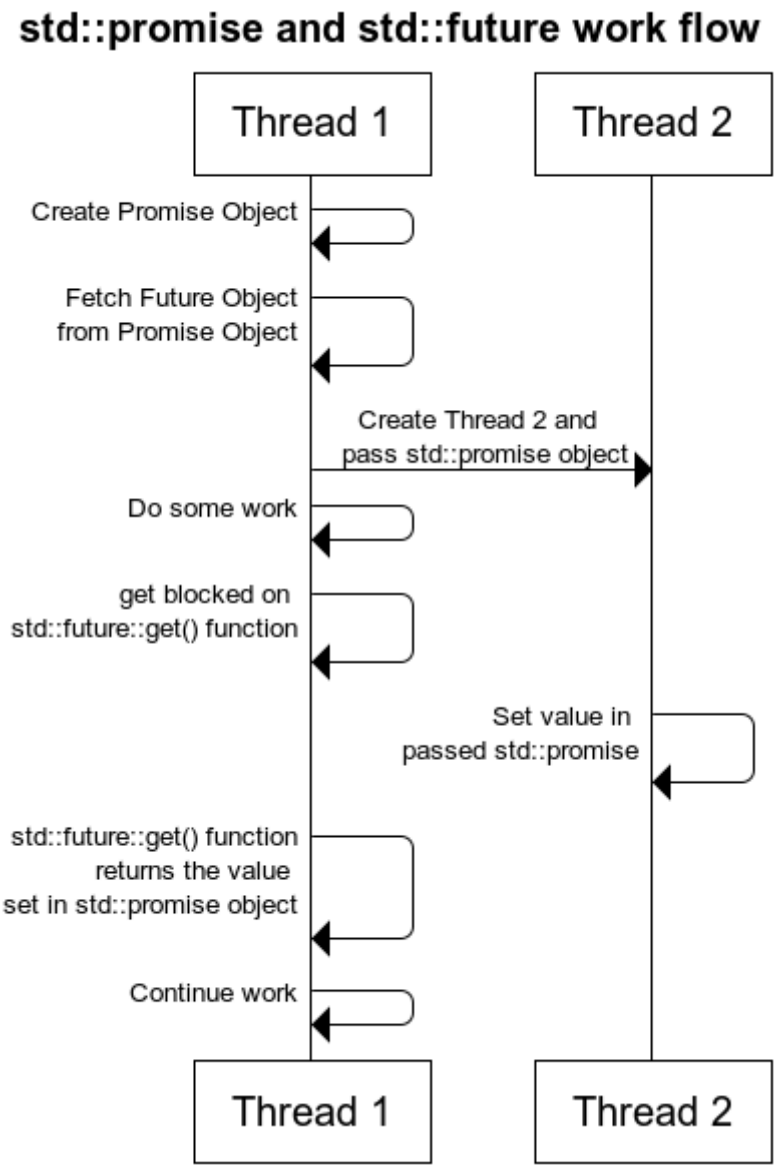

Figura 9: Resumo do funcionamento entre os tipos *Future* e *Promise* em C++
Fonte: Referência [17]

No exemplo anterior, tem-se a *Thread 1* criando um objeto do tipo *std::promise*, então solicitando que o objeto *std::promise* retorne seu objeto do tipo *std::future*. Em seguida, cria-se a *Thread 2* passando o objeto *std::promise*. Uma vez que a *Thread 1* tentar acessar o valor dentro de seu objeto *std::future*, ela irá bloquear automaticamente caso esse resultado ainda não tenha sido colocado dentro do *std::future* através de seu objeto *std::promise* correspondente. A seguir vê-se um exemplo de código em C++ que segue o padrão descrito no diagrama anterior:

Listagem 8: Implementação de uma Promise C++

```
#include <iostream>
#include <thread>
#include <future>

void initiazer(std::promise<int> * promiseObject)
{
    std::cout << "Inside Thread" << std::endl;
```

```
    promiseObject->set_value(35);
}

int main()
{
    std::promise<int> promiseObj;
    std::future<int> futureObject = promiseObj.get_future();
    std::thread thread(initiazer, &promiseObj);
    std::cout << futureObject.get() << std::endl;
    thread.join();
    return 0;
}
```

Fonte: Referência [17]

Futures versus Observables ←

Não há muito o que dizer sobre a diferença entre *futures* [14] e *observables* [20]. Cada um deles servem a propósitos bem específicos, entretanto o domínio de soluções de problemas entre *futures* e *observables* possuí uma intersecção válida quando não utiliza-se *futures* que realizam o bloqueio da thread, mas sim *futures* que registram uma lista de blocos de execução dependentes para serem chamados assim que o resultado da future estiver completo. Neste contexto, a versão equivalente para *observables* seria registrar a lista de blocos dependentes do resultado como observadores.

A desvantagem de utilizar *observables* no lugar de uma *futures* seria o encadeamento manual de *observables* necessário para cada um dos blocos de dados da cadeia. A desvantagem de utilizar *futures* no lugar de *observables* seria a impossibilidade de registrar vários blocos como dependentes do mesmo resultado, pois *observables* permitem que vários ouvintes sejam registrados e que estes ouvinte seja chamados várias vezes, i.e., a cada vez que um novo resultado é produzido. Enquanto *futures* chamam seu ouvinte uma única vez para um único resultado gerado.

Diagrama das Estruturas de Dados Originais ←

Para o funcionamento do algoritmo [1], utiliza-se algumas estruturas de dados em "C". A seguir vê-se as relações entre elas:

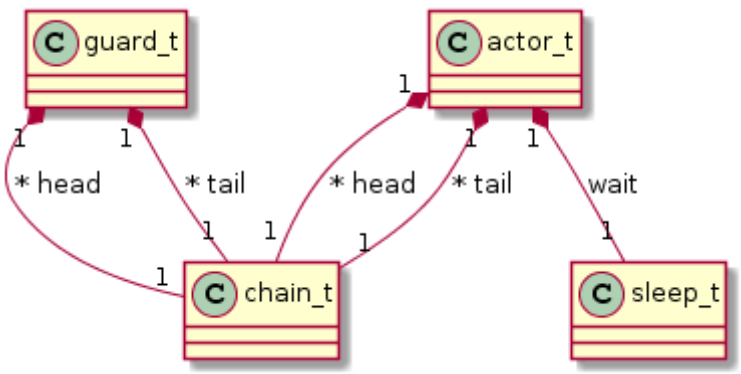

Figura 10: Diagrama de classes do tipos de dados do algoritmo do Guarda
Fonte: Própria

Listagem 9: Estruturas de dados em C

```
typedef struct
{
    chain_t* next;
} chain_t;

typedef struct
{
    chain_t* head;
    chain_t* tail;
} guard_t;

typedef struct
{
    chain_t* head;
    chain_t* tail;
    sleep_t wait;
} actor_t;
```

Diagrama de Sequência - Versão original em C ←

Exemplo do fluxo de execução de uma única Thread que se torna o *sequencer*, e executa sua seção crítica:

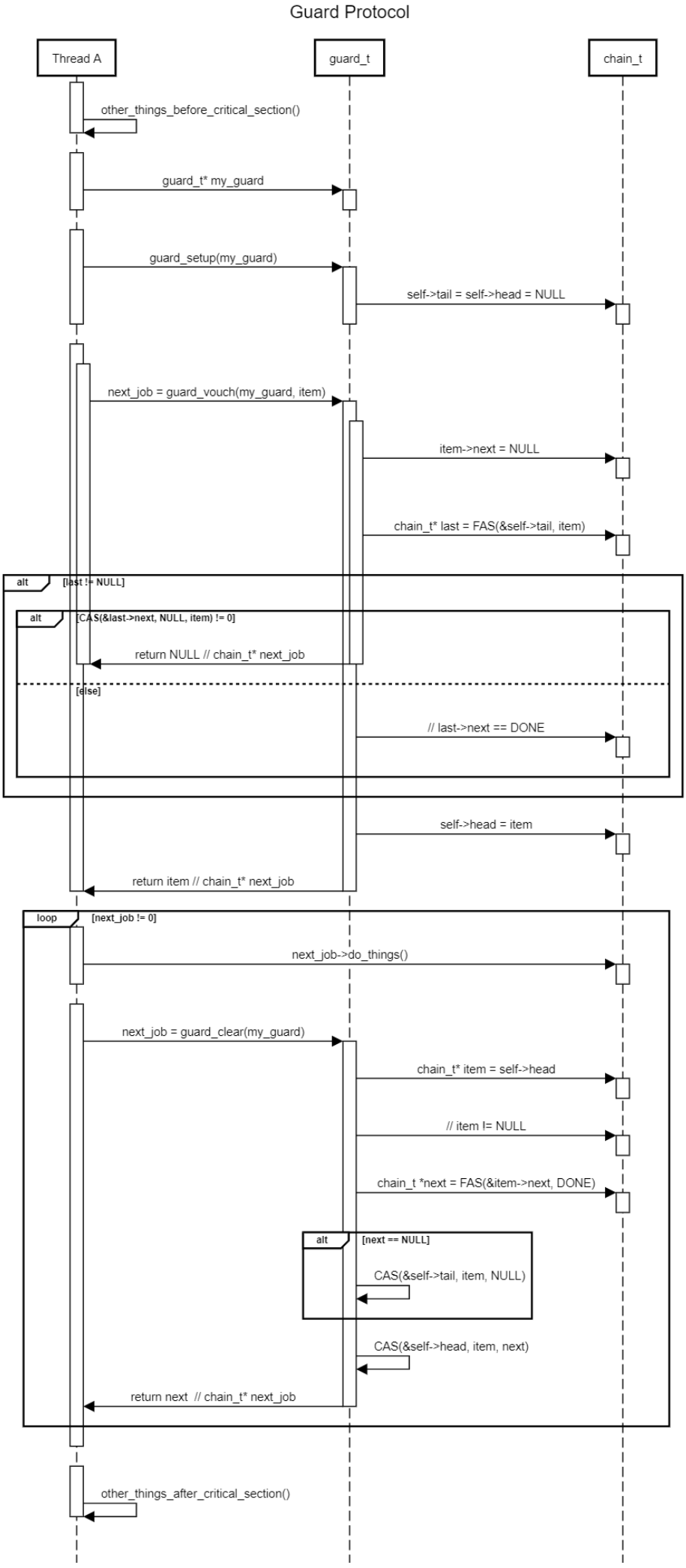
Guard Protocol
Thread A
guard_t
chain_t
other_things_before_critical_section()
guard_t* my_guard
guard_setup(my_guard)
self->tail = self->head = NULL
next_job = guard_vouch(my_guard, item)
item->next = NULL
chain_t* last = FAS(&self->tail, item)
alt
[last != NULL]
alt
[CAS(&last->next, NULL, item) != 0]
return NULL // chain_t* next_job
[else]
// last->next == DONE
self->head = item
return item // chain_t* next_job
loop
[next_job != 0]
next_job->do_things()
next_job = guard_clear(my_guard)
chain_t* item = self->head
// item != NULL
chain_t *next = FAS(&item->next, DONE)
alt
[next == NULL]
CAS(&self->tail, item, NULL)
CAS(&self->head, item, next)
return next // chain_t* next_job
other_things_after_critical_section()

Figura 11: Exemplo detalhado da execução do algoritmo do Guarda em *C*
Fonte: Própria

A seguir vê-se a implementação do algoritmo do guarda, como apresentado no artigo [1], utilizando a linguagem "C".

Listagem 10: Implementação do algoritmo do Guarda em C

```
void guard_setup(guard_t* self)
{
    self->head = self->tail = NULL;
}

chain_t* guard_vouch(guard_t* self, chain_t* item)
{
    item->next = NULL;
    chain_t* last = FAS(&self->tail, item); // V1
    if (last)
    {
        if (CAS(&last->next, NULL, item)) // V2
            return NULL;
        // last->next == DONE
    }
    self->head = item; // V3
    return item;
}

chain_t* guard_clear(guard_t* self)
{
    chain_t* item = self->head; // C1
    // item != NULL
    chain_t* next = FAS(&item->next, DONE); // C2
    if (!next)
        CAS(&self->tail, item, NULL); // C3
    CAS(&self->head, item, next); // C4
    return next;
}
```

Análise de viabilidade ←

No EPOS x86 não hà uma implementação para a operação FAS, uma das duas primitivas atômicas utilizadas no algoritmo das guards. Uma possível implementação para essa operação é apresentada a seguir:

Listagem 11: Implementação da operação de FAS

```
template<typename T>
```

```
static T fas(volatile T & value, volatile T replacement) {
    ASM("lock xchgl %1, %2" : "=a"(replacement) : "a"(replacement), "m"(value) :
"memory");
    return replacement;
}
```

Mesmo que não consiga-se realizar uma implementação correta para *FAS()*, pode-se implementar a FAS utilizando CAS. Por exemplo, uma chamada FAS seria *FAS(entrada, saída)* e a versão com CAS equivalente seria *CAS(entrada, entrada, saída)*. Assim, a diferença seria que utilizar um CAS pode ser menos eficiente ao invés de utilizar somente um *FAS()* implementado em assembly.

Como ARM não suporta CAS, apenas LC/SC, pretende-se limitar a implementação do algoritmo apenas à versão do EPOS para a arquitetura x86.

A seguir vê-se um simples programa que utiliza a operação *CAS()* atualmente implementada no EPOS.

Listagem 12: Arquivo */app/cas_test.cc*

```
// EPOS CAS Component Test Program

#include <utility/ostream.h>
#include <architecture/ia32/cpu.h>

using namespace EPOS;
OStream cout;

int main()
{
    cout << endl << "Welcome to the CPU::cas() instruction test!" << endl;
    int original = 5;
    int compare = 5;
    int replacement = 6;
    int replaced;

    cout << "original=" << original
            << ", compare=" << compare
            << ", replacement=" << replacement
            << ", replaced=" << replaced
            << endl;

    replaced = CPU::cas(original, compare, replacement);

    cout << "original=" << original
            << ", compare=" << compare
            << ", replacement=" << replacement
            << ", replaced=" << replaced
            << endl;

    cout << "The CPU::cas() instruction set ran successfully!" << endl << endl;
}
```

Resultado da execução:

Listagem 13: Exemplo de execução de */app/cas_test.cc*

```
Welcome to the CPU::cas() instruction test!
original=5, compare=5, replacement=6, replaced=0
original=6, compare=5, replacement=6, replaced=5
The CPU::cas() instruction set ran successfully!

The last thread has exited!
Rebooting the machine ...
```

Implementação ←

Código disponível em svn:

1. https://github.com/evandrocoan/predictable_synchronisation_algorithms_for_asynchronous_critical_sections
2. https://epos.lisha.ufsc.br/svn/makers/predictable_synchronisation_algorithms_for_asynchronous_critical_sections

Para a implementação do algoritmo das guardas no EPOS foram definidas três novas classes, que representam as principais abstrações envolvidas na versão básica do algoritmo, seções críticas, elementos de guarda e a guarda em si. Essas classes encapsulam toda a lógica do algoritmo e fornecem interfaces para a delegação da execução de seções críticas por threads arbitrárias.

Neste capítulo será descrito como ocorreu o processo de implementação das novas classes que implementam o algoritmo das guardas, assim como os testes realizados para validação da implementação.

Critical_Section ←

Assim como ocorre com os algoritmos de sincronização tradicionais, uma das bases dos algoritmos de sincronização baseados em delegação é o conceito de seções críticas. Dessa forma, foi definida uma nova classe para representar essas entidades.

A classe que representa seções críticas foi criada com base na classe Function_Handler e recebeu o nome de *Critical_Section*. Essa classe possui apenas dois atributos, _handler e _link. O atributo _handler representa a função relacionada à seção crítica e o atributo _link representa um elemento de lista simples. Além desses dois atributos, a classe *Critical_Section* também implementa o método público run(), que é utilizado pelo sequencer para executar uma seção crítica, que na verdade é o mesmo trabalho realizado pelo operator “()”, que deixava o código pouco intuitivo. A Listagem 5 exibe o código da classe *Critical_Section*.

O construtor da classe *Critical_Section* recebe um objeto representando uma função, que deve ser compatível com o tipo Function, definido no arquivo *handler.h*, como parâmetro. Além disso, o construtor cria o _link relativo à seção crítica sendo construída, que será utilizado para adição e remoção de seções críticas de estruturas de dados, da mesma forma como acontece com objetos da classe Thread.

Listagem 14: Seção crítica

```
class Critical_Section
{
public:
    friend class Guard;
    typedef Handler::Function Function;
    typedef List_Elements::Singly_Linked<Critical_Section> Element;

public:
    Critical_Section(Function * h): _handler(h), _link(this) {}
    ~Critical_Section() {}

    void operator()() { _handler(); }
    void run()        { _handler(); } // Alias for ()

private:
    Function * _handler;
    Element _link; // Inspired by the thread code
};
```

Exemplos da utilização da classe *Critical_Section* durante a submissão de seções críticas à guarda estão na seção sobre os testes realizados.

Element ←

Como elementos da guarda são elementos de lista simples, com apenas um nível de encadeamento, optou-se por utilizar a classe List_Elements::Singly_Linked, Listagem 6, definida no arquivo *list.h*, para representá-los.

No entanto, foi necessário adicionar uma relação de amizade entre a classe Singly_Linked e a classe Guard, para que as operações CAS e FAS dos métodos vouch e clear da classe guardam possam acessar diretamente o atributo _next do elemento de lista. Isso porque, não é possível efetuar as operações CAS e FAS utilizando getters.

Listagem 15: Link contendo a seção crítica

```
// Simple List Element
template<typename T>
class Singly_Linked
{
public:
    friend class _UTIL::Guard;
    typedef T Object_Type;
    typedef Singly_Linked Element;

public:
    Singly_Linked(const T * o): _object(o), _next(0) {}
```

```
    T * object() const { return const_cast<T *>(_object); }

    Element * next() const { return _next; }
    void next(Element * e) { _next = e; }

private:
    const T * _object;
    Element * _next;
};
```

Guarda ←

O código onde da classe Guard, definida no arquivo *utility/guard.h*, é mostrado na Listagem 7. Essa classe implementa a estrutura da guarda descrita no capítulo de fundamentação teórica.

Listagem 16: Definição da interface do Guarda

```
class Guard
{
public:
   typedef Closure::Element Element;

private:
   static const int NULL = 0;
   static const int DONE = 1;

public:
   Guard();
   ~Guard();

   void submit(Closure * cs);
   Element * vouch(Element * item);
   Element * clear();

private:
   Element * _head;
   Element * _tail;
};
```

Além das operações vouch, clear e a operação de criação da guarda (setup), já apresentadas no capítulo de fundamentação teórica, a classe guarda implementa uma nova operação, chamada de submit. Essa operação encapsula o protocolo de submissão de seções críticas à guarda e pode ser utilizada por threads para a delegação da execução de seções críticas. A Listagem 8 apresenta o construtor e destrutor da guarda, assim como a implementação do método submit().

Listagem 17: Nova convenção para a submissão de requisições de execução de seções críticas

```
// Object Methods
```

```
Guard::Guard(): _head(0), _tail(0) {
    DB( Synchronizer, TRC, "Guard(head=" << _head << ", tail=" << _tail
            << ") => " << this << endl )
}

Guard::~Guard() {
    DB( Synchronizer, TRC, "~Guard(this=" << this << ")" << endl )
}

void Guard::submit(Closure * cs)
{
   Element * cur = vouch(&(cs->_link));
   if (cur != NULL) do {
       cur->object()->run();
       cur = clear();
   } while (cur != NULL);
}
```

A implementação dos métodos vouch(...) e clear são apresentadas na Listagem 8. Essas implementações não diferem muito dos códigos apresentados no capítulo de fundamentação teórica, lembrando que a operação vouch é responsável por adicionar novos elementos à guarda e sinalizar threads quando essas precisarem assumir o papel de sequencer, e a operação clear é responsável pela remoção de elementos da guarda e por sinalizar ao sequencer quando a guarda estiver vazia.

Uma adição importante foi a deleção de seções críticas que já foram executadas. Essa deleção deve geralmente ocorrer durante a execução do método clear. No entanto, existe uma situação excepcional, que pode ocorrer quando a guarda possui apenas um elementos e operações vouch e clear executam simultaneamente. Nesse caso, a passagem do papel de sequencer da thread executando clear para a thread executando vouch. Quando isso acontecer, o sequencer executando vouch precisará também deletar o elemento da guarda previamente removido da lista pela operação clear.

Listagem 18: Implementações de vouch() e clear()

```
Guard::Element * Guard::vouch(Element * item)
{
    DB( Synchronizer, TRC, "Guard::vouch(this=" << this
            << " head=" << _head << " tail=" << _tail
            << " item=" << item << ", size=" << CPU::finc(_size) + 1 )

    item->next(reinterpret_cast<Element *>(NULL));
    Element * last = CPU::fas(_tail, item);
    DB( Synchronizer, TRC, ", last=" << last << ")" << endl )

    if (last) {
        if (CPU::cas(last->_next, reinterpret_cast<Element *>(NULL), item)
                    == reinterpret_cast<Element *>(NULL))

            return reinterpret_cast<Element *>(NULL);
        delete last->object();
    }
```

```
    _head = item;
    return item;
}

Guard::Element * Guard::clear()
{
    DB( Synchronizer, TRC, "Guard::clear(this=" << this
            << " head=" << _head << " tail=" << _tail
            << ", size=" << CPU::fdec(_size) - 1 )

    Element * item = _head;
    Element * next = CPU::fas(item->_next, reinterpret_cast<Element *>(DONE));
    DB( Synchronizer, TRC, ", next=" << next << ")" << endl )

    bool mine = true;
    if (!next)
        mine = CPU::cas(_tail, item, reinterpret_cast<Element *> (NULL)) == item;

    CPU::cas(_head, item, next);
    if (mine)
        delete item->object();

    return next;
}
```

FAS ←

A operação FAS, uma das duas primitivas atômicas utilizados no algoritmo das guardas, ainda não havia sido implementada no EPOS, Dessa forma, foi adicionada ao arquivo */include/archtecture/ia_32/cpu.h*, onde estão implementadas as primitivas atômicas de sincronização no EPOS, uma nova implementação para FAS.

O código de FAS foi inspirado no código das outras operações atômicas e nas descrições das instruções atômicas nos manuais da Intel.

Além disso, assim como ocorre com as outras operações atômicas, foi adicionada uma implementação em software, independente de arquitetura, no arquivo */include/cpu.h*. No entanto, como essa operação não possui suporte de hardware, ela não deve provê garantias de atomicidade.

Listagem 19: Implementação do FAS

```
// /include/archtecture/ia_32/cpu.h
template<typename T>
static T fas(volatile T & value, volatile T replacement)
{
    ASM("lock xchgl %1, %2" : "=a"(replacement) : "a"(replacement), "m"(value) :
"memory");
    return replacement;
}
```

```
// /include/cpu.h
static int fas(volatile int & value, volatile int & replacement) {
     int old = value;
     value = replacement;
     return old;
 }
```

debug_sync.h ←

Para auxiliar e melhorar o entendimento das aplicações multithread, foi criado uma macro especial para realizar o debug das aplicações e novos componentes do EPOS criados, como *closure.h*, *guard.h* e *stringstream.h*.

Sua implementação baseia-se somente em semáforo global uma simples macro que replica seu uso. Ela pode ser desativada completamente, não adicionando a diretiva *#define DEBUG_SYNC* do pré-processador C que realiza sua ativação.

Listagem 20: Arquivo */include/utility/debug_sync.h*

```
// EPOS Debug Utility Declarations

#include <utility/debug.h>
#include <semaphore.h>

#ifndef __debug_sync_h
#define __debug_sync_h

// You can define it anywhere before including this file
// #define DEBUG_SYNC

#ifdef DEBUG_SYNC

__BEGIN_UTIL
    Semaphore _debug_syncronized_semaphore_lock;
__END_UTIL

    // A debug function cannot call this recursively, otherwise a deadlock happens.
    // Unless you replace the `_debug_syncronized_semaphore_lock` by a recursive
    // semaphore or mutex. See:
    //
https://stackoverflow.com/questions/36619715/a-shared-recursive-mutex-in-standard-c
    #define DB(name,level,...) do { \
        _debug_syncronized_semaphore_lock.lock(); \
            db<name>(level) << __VA_ARGS__; \
        _debug_syncronized_semaphore_lock.unlock(); } while(0);

#else
    #define DB(name,level,...) db<name>(level) << __VA_ARGS__;

#endif
```

```
#define LOG(...) DB(Debug, WRN, __VA_ARGS__)

#endif
```

stringstream.h ←

Para auxiliar o uso do algoritmo do *Guarda*, e permite que faça-se a formatação prévia de strings antes de envia-las para a seção crítica, criou-se uma nova class auxiliar chamada *StringStream* criada no arquivo *stringstream.h*. Ela é uma extensão simples da classe *OStream* do EPOS, que já sabe como fazer a formatação de vários tipos. Assim, na implementação do *stringstream.h* somente implementou-se um wrapper sobre a manipulação de todas as strings que já foram formatadas pela super classe *OStream*.

Listagem 21: Modificações do arquivo */include/utility/ostream.h*

```
class StringStream;

...
    // Implemented on `ostream.cc`, because `stringstream.h` includes `ostream.h`,
    // hence, `ostream.h` cannot include `stringstream.h` back (cyclic reference)
    OStream & operator<<(const StringStream * stream);

    // // ostream.cc
    // #include <utility/stringstream.h>
    //
    // OStream & OStream::operator<<(const StringStream * stream)
    // {
    //     return operator<<(stream->buffer());
    // }

private:
    virtual void print(const char * s) { _print(s); }
```

Foi necessário tornar o método *print()* da classe *OStream* virtual e para facilitar o uso, também permitiu-se que classe *OStream* possa imprimir diretamente ponteiros do tipo *StringStream*. Entretanto, depois de declarar o método *print()* com o modificador *virtual*, faz-se com que não se tenha nenhuma saída de texto ao executar a aplicação.

No exemplo a seguir, vê-se a execução de uma aplicação no modo *hysterically_debugged = true* após adicionar-se o modificador *virtual* na assinatura da função *void print(const char * s) { _print(s); }*.

Listagem 22: Saída da execução de uma aplicação com *hysterically_debugged = true*

```
EPOS bootable image tool

  EPOS mode: library
  Machine: pc
  Model: legacy_pc
```

```
  Processor: ia32 (32 bits, little-endian)
  Memory: 262144 KBytes
  Boot Length: 512 - 512 (min - max) KBytes
  Node id: will get from the network
  EPOS Image UUID: 7bb9c483078f0692
  Creating EPOS bootable image in "scheduler_cpu_affinity_test.img":
    Adding boot strap "/Epos2x86/img/pc_boot": done.
    Adding setup "/Epos2x86/img/pc_setup": done.
    Adding application "scheduler_cpu_affinity_test": done.
    Adding system info: done.

  Adding specific boot features of "pc": done.

  Image successfully generated (172868 bytes)!

make[2]: Leaving directory '/Epos2x86/img'
(cd img && make --print-directory run)
make[2]: Entering directory '/Epos2x86/img'
# qemu-system-i386 -smp 2 -m 262144k -nographic -no-reboot -drive
format=raw,index=0,if=floppy,file=scheduler_cpu_affinity_test.img | tee
scheduler_cpu_affinity_test.out
qemu-system-i386 -smp 2 -m 262144k -nographic -no-reboot -drive
format=raw,index=0,if=floppy,file=scheduler_cpu_affinity_test.img
<0>: make[2]: Leaving directory '/Epos2x86/img'
make[1]: Leaving directory '/home/linux/OperatingSystems/Epos2x86'
```

Percebe-se que ao compilar e executar a aplicação *scheduler_cpu_affinity_test.cc*, em modo debug histérico, a única coisa que foi impresso no tela foi a primeira parte *<0>:* da mensagem que indica qual CPU está executando aquela mensagem. Mas basta remover-ser o identificador *virtual* da função *OStream::print*, que tudo volta a funcionar normalmente.

Portando, devido ao bug com o uso da identificador *virtual*, fez-se o uso de outra alternativa mais agressiva do que o polimorfismo permitido pelo identificador *virtual*. Após muita pesquisa, chegou-se a uma implementação que faz uso de muitos *static_cast*'s e utiliza-se métodos estáticos para permitir que o objeto que derivam da super classe possam funcionar corretamente, 'imitando' o funcionamento que obtém-se com o uso do identificador *virtual*.

Agora, tem-se uma nova super classe meta programada chamada *OStream_Base*, que será base de todas os tipos de stream, incluindo *OStream* e o novo tipo *StringStream*. A seguir encontra-se a sua implementação:

Listagem 23: Arquivo */include/utility/ostream.h*

```
template<class StreamType>
class OStream_Base
{
public:
    struct Begl {};
    struct Endl {};
```

```
    struct Hex {};
    struct Dec {};
    struct Oct {};
    struct Bin {};

public:
    OStream_Base(): _base(10) {}

    StreamType& operator<<(const Hex & hex) {
        _set_base(16);
        return *static_cast<StreamType*>(this);
    }
    StreamType& operator<<(const Dec & dec) {
        _set_base(10);
        return *static_cast<StreamType*>(this);
    }
    StreamType& operator<<(const Oct & oct) {
        _set_base(8);
        return *static_cast<StreamType*>(this);
    }
    StreamType& operator<<(const Bin & bin) {
        _set_base(2);
        return *static_cast<StreamType*>(this);
    }

    StreamType& operator<<(char c) {
        char buf[2];
        buf[0] = c;
        buf[1] = '\0';
        print(buf);
        return *static_cast<StreamType*>(this);
    }
    StreamType& operator<<(unsigned char c) {
        return operator<<(static_cast<unsigned int>(c));
    }

    StreamType& operator<<(int i) {
        char buf[64];
        buf[itoa(i, buf)] = '\0';
        print(buf);
        return *static_cast<StreamType*>(this);
    }
    StreamType& operator<<(short s) {
        return operator<<(static_cast<int>(s));
    }
    StreamType& operator<<(long l) {
        return operator<<(static_cast<int>(l));
    }

    StreamType& operator<<(unsigned int u) {
        char buf[64];
        buf[utoa(u, buf)] = '\0';
        print(buf);
        return *static_cast<StreamType*>(this);
```

```
}
StreamType& operator<<(unsigned short s) {
    return operator<<(static_cast<unsigned int>(s));
}
StreamType& operator<<(unsigned long l) {
    return operator<<(static_cast<unsigned int>(l));
}

StreamType& operator<<(long long int u) {
    char buf[64];
    buf[llitoa(u, buf)] = '\0';
    print(buf);
    return *static_cast<StreamType*>(this);
}

StreamType& operator<<(unsigned long long int u) {
    char buf[64];
    buf[llutoa(u, buf)] = '\0';
    print(buf);
    return *static_cast<StreamType*>(this);
}

StreamType& operator<<(const void * p) {
    char buf[64];
    buf[ptoa(p, buf)] = '\0';
    print(buf);
    return *static_cast<StreamType*>(this);
}

StreamType& operator<<(const char * s) {
    print(s);
    return *static_cast<StreamType*>(this);
}

StreamType& operator<<(float f) {
    if(f < 0.0001f && f > -0.0001f)
        (*this) << "0.0000";

    int b = 0;
    int m = 0;

    float x = f;
    if(x >= 0.0001f) {
        while(x >= 1.0000f) {
            x -= 1.0f;
            b++;
        }
        (*this) << b << ".";
        for(int i = 0; i < 3; i++) {
            m = 0;
            x *= 10.0f;
            while(x >= 1.000f) {
                x -= 1.0f;
                m++;
```

```
                }
                (*this) << m;
            }
        } else {
            while(x <= -1.000f) {
                x += 1.0f;
                b++;
            }
            (*this) << "-" << b << ".";
            for(int i = 0; i < 3; i++) {
                m = 0;
                x *= 10.0f;
                while(x <= -1.000f) {
                    x += 1.0f;
                    m++;
                }
                (*this) << m;
            }
        }
        return *static_cast<StreamType*>(this);
    }

protected:
    int itoa(int v, char * s)
    {
        unsigned int i = 0;

        if(v < 0) {
            v = -v;
            s[i++] = '-';
        }

        return utoa(static_cast<unsigned int>(v), s, i);
    }

    int utoa(unsigned int v, char * s, unsigned int i = 0)
    {
        unsigned int j;

        if(!v) {
            s[i++] = '0';
            return i;
        }

        if(v > 256) {
            if(_base == 8 || _base == 16)
                s[i++] = '0';
            if(_base == 16)
                s[i++] = 'x';
        }

        for(j = v; j != 0; i++, j /= _base);
        for(j = 0; v != 0; j++, v /= _base)
            s[i - 1 - j] = _digits[v % _base];
```

```
    return i;
}

int llitoa(long long int v, char * s)
{
    unsigned int i = 0;

    if(v < 0) {
        v = -v;
        s[i++] = '-';
    }

    return llutoa(static_cast<unsigned long long int>(v), s, i);
}

int llutoa(unsigned long long int v, char * s, unsigned int i = 0)
{
    unsigned long long int j;

    if(!v) {
        s[i++] = '0';
        return i;
    }

    if(v > 256) {
        if(_base == 8 || _base == 16)
            s[i++] = '0';
        if(_base == 16)
            s[i++] = 'x';
    }

    for(j = v; j != 0; i++, j /= _base);
    for(j = 0; v != 0; j++, v /= _base)
        s[i - 1 - j] = _digits[v % _base];

    return i;
}

int ptoa(const void * p, char * s)
{
    unsigned int j, v = reinterpret_cast<unsigned int>(p);

    s[0] = '0';
    s[1] = 'x';

    for(j = 0; j < sizeof(void *) * 2; j++, v >>= 4)
        s[2 + sizeof(void *) * 2 - 1 - j]
            = _digits[v & 0xf];

    return j + 2;
}

// https://stackoverflow.com/questions/34222703/how-to-override-static-method
inline void print(const char * s)
```

```
        { StreamType::print(static_cast<StreamType*>(this), s); }

    inline void _set_base(int v) { _base = v; }

    int _base;
    static const char _digits[];
};

// Class Attributes
// https://stackoverflow.com/questions/3531060/how-to-initialize-a-static-const-member
template<class OStream>
const char OStream_Base<OStream>::_digits[] = "0123456789abcdef";
```

Como por padrão o operator de stream *operator<<* returna um objeto o mesmo tipo que a classe atual, e não da classe derivada, tem-se o problema de perder-se o tipo do objeto quando utiliza-se os métodos herdados da super classe. Por isso a classe base, agora é uma meta classe, e realiza a conversão do retorno de todos os seus operadores de stream *operator<<*.

Para contornar o problema de não poder-se utilizar o identificador *virtual* no método *inline void print(const char * s)* logo acima, faz-se uso da meta programação com template *StreamType*, para acessar diretamente o método estático da classe derivada passando como primeiro parâmetro o ponteiro para o objeto do tipo da classe derivada, após realizar sua conversão *StreamType::print(static_cast<StreamType*>(this), s)*.

Assim, após refatorar-se todo o conteúdo da antiga *OStream*, tem-se uma nova *OStream* que deriva da super classe *OStream_Base*:

Listagem 24: Arquivo */include/utility/ostream.h*

```
//
https://stackoverflow.com/questions/11761506/inheritance-function-that-returns-self-typ
e
class OStream : public OStream_Base<OStream>
{
public:
    struct Err {};

public:
    OStream(): _error(false) {}

    // Implemented on `ostream.cc`, because `stringstream.h` includes `ostream.h`,
    // hence, `ostream.h` cannot include `stringstream.h` back (cyclic reference)
    OStream & operator<<(const StringStream * stream);

    OStream & operator<<(const Begl & begl) {
        if(Traits<System>::multicore)
            _print_preamble();
        return *this;
    }
```

```
    OStream & operator<<(const Endl & endl) {
        if(Traits<System>::multicore)
            _print_trailler(_error);
        OStream_Base<OStream>::print("\n");
        _set_base(10);
        return *this;
    }

    OStream & operator<<(const Err & err) {
        _error = true;
        return *this;
    }

    using OStream_Base<OStream>::operator<<;

    // Adding virtual to the print() function caused EPOS to completely break
    static void print(OStream* that, const char * s) { _print(s); }

private:
    volatile bool _error;
};
```

Agora, a implementação do novo tipo *StringStream* resume-se ao seguinte:

Listagem 25: Arquivo */include/utility/stringstream.h*

```
class StringStream : public OStream_Base<StringStream>
{
private:
    char* _buffer;
    unsigned int _last_position;
    const unsigned int _buffer_size;

public:
    StringStream(const unsigned int _buffer_size) :
            _last_position(0), _buffer_size(_buffer_size)
    {
        DB( Synchronizer, TRC, "StringStream::StringStream(_buffer_size="
                << _buffer_size << ") => " << reinterpret_cast<int *>(this) << endl )

        assert(_buffer_size > 0);
        _buffer = new char[_buffer_size];
    }

    ~StringStream() {
        DB( Synchronizer, TRC, "StringStream::~StringStream(this="
                << reinterpret_cast<int *>(this) << ", _buffer="
                << reinterpret_cast<int *>(_buffer) << ")" << endl )

        delete _buffer;
    }
```

```
    const char * const buffer() const {
        return _buffer;
    }

    StringStream & operator<<(const StringStream & stream) {
        OStream_Base<StringStream>::print(stream.buffer());
        return *this;
    }
    StringStream & operator<<(const StringStream * stream) {
        OStream_Base<StringStream>::print(stream->buffer());
        return *this;
    }

    static void print(StringStream* that, const char* string) {
        DB( Synchronizer, TRC, "StringStream::print(this="
                << reinterpret_cast<int *>(that)
                << "), string=" << string << ", " )

        unsigned int string_size = strlen(string);
        unsigned int total_size = string_size + that->_last_position;

        DB( Synchronizer, TRC, "string_size=" << string_size << ", "
                << "total_size=" << total_size )

        // https://linux.die.net/man/3/strncpy
        if( total_size >= that->_buffer_size ) {
            total_size = that->_buffer_size - 1;

            strncpy(&that->_buffer[that->_last_position],
                    string, total_size - that->_last_position);

            that->_buffer[total_size] = '\0';
        }
        else {
            strcpy(&that->_buffer[that->_last_position], string);
        }

        that->_last_position = total_size;
        DB( Synchronizer, TRC, ", _last_position=" << that->_last_position
                << ", _buffer=" << that->_buffer << endl )
    }

public:
    StringStream & operator<<(const Begl & begl) {
        return *this;
    }

    StringStream & operator<<(const Endl & endl) {
        OStream_Base<StringStream>::print("\n");
        _set_base(10);
        return *this;
    }

    using OStream_Base<StringStream>::operator<<;
```

```
};
```

Segue-se alguns testes de extremo de uso da *StringStream*, quando o buffer está quase cheio:

Listagem 26: Arquivo */app/stringstream_test.cc*

```
// EPOS Semaphore Component Test Program
#define DEBUG_SYNC

#include <utility/stringstream.h>
using namespace EPOS;

int main()
{
    LOG( endl )
    LOG( "Welcome to the `stringstream.h` test program" << endl )

    StringStream stream48chars{50};
    StringStream stream49chars{50};
    StringStream stream50chars{50};
    StringStream stream51chars{50};

    stream48chars << "666666666666666666666666666666666666666666660048";
    stream49chars << "666666666666666666666666666666666666666666600049";
    stream50chars << "666666666666666666666666666666666666666666000050";
    stream51chars << "666666666666666666666666666666666666666660000051";

    LOG( &stream48chars << endl )
    LOG( &stream49chars << endl )
    LOG( &stream50chars << endl )
    LOG( &stream51chars << endl << endl )

    StringStream base23chars{50};
    StringStream stream48chars2{50};
    StringStream stream49chars2{50};
    StringStream stream50chars2{50};
    StringStream stream51chars2{50};

    base23chars    << "77777777777777777777";
    stream48chars2 << base23chars << "888888888888888888880048";
    stream49chars2 << base23chars << "888888888888888888800049";
    stream50chars2 << base23chars << "888888888888888888000050";
    stream51chars2 << base23chars << "888888888888888888880000051";

    LOG( &stream48chars2 << endl )
    LOG( &stream49chars2 << endl )
    LOG( &stream50chars2 << endl )
    LOG( &stream51chars2 << endl )

    return 0;
}
```

Resultado da execução:

Listagem 27: Exemplo de execução de */app/stringstream_test.cc*

```
Welcome to the `stringstream.h` test program
66666666666666666666666666666666666666666660048
666666666666666666666666666666666666666666600049
666666666666666666666666666666666666666666600005
666666666666666666666666666666666666666666600000

777777777777777777777888888888888888888880048
7777777777777777777778888888888888888888800049
7777777777777777777778888888888888888888800005
7777777777777777777778888888888888888888800000
```

Parallel Makefile ←

Para auxiliar o desenvolvimento, reduzindo o tempo de espera pela compilação completa do EPOS com um *make veryclean*, foi paralelizado a compilação do EPOS alterando alguns *Makefile*'s:

1. *src/architecture/makefile*
2. *src/component/makefile*
3. *src/machine/makefile*
4. *src/utility/makefile*
5. *tools/makefile*

A alteração feita nesses *Makefile*'s foi a seguinte:

Listagem 28: Novo *makefile* paralelizado

```
# How can I write a makefile to auto-detect and parallelize the build with GNU Make?
# https://stackoverflow.com/questions/2527496/how-can-i-write-a-makefile-to-auto
ifeq ($J,)
    ifeq ($(OS),Windows_NT)
        NPROCS := $(NUMBER_OF_PROCESSORS)
    else
        ifeq ($(UNAME),Darwin)
          NPROCS := $(shell sysctl -n hw.ncpu)
        else
          NPROCS := $(shell grep -c ^processor /proc/cpuinfo)
        endif
    endif
else
  NPROCS := ""
endif
all:
    ${MAKE} all_multithread -j$(NPROCS)
all_multithread: $(SUBDIRS)
```

A alteração não foi feita somente no *Makefile* raíz por que alguns *Makefile*'s como *src/system* precisam ser executados sequencialmente. A seleção de quais *Makefile*'s poderiam ser paralelizados foi feita pela tentativa e erro. Adicionava-se o comando de paralelização, e verifica-se o EPOS compilava corretamente depois de um *make veryclean*.

Para paralelizar o comando *make veryclean*, simplesmente precisou-se editar o *Makefile* raíz *src/makefile*, adicionado o seguinte comando:

Listagem 29: Novo comando *make clean*

```
clean:
    ${MAKE} clean_multithread -j$(NPROCS)
clean_multithread:
```

Para *make veryclean* funcionar paralelamente, foi preciso remover a definição *MAKECLEAN* no arquivo principal *makedefs* 4237faf && e4faae6, por que para que o comando *-j* de paralelização do *Makefile* funcione, as chamadas recursivas de makefile precisam ser feitas utilizando a variável *${MAKE}* ao contrário de chamar o *Makefile* diretamente como *MAKECLEAN := make -i clean* fazia.

Note que, aleatoriamente durante algumas vezes no final do processo de compilação, quando será feita a ligação entre o código do EPOS e aplicação compilada, pode-se ter um ou vários erros de ligação sobre símbolos faltando. Para resolver isso, basta somente pedir para compilar de novo. Algumas outras vezes muito raras, pode ser necessário pedir para compilar até 3 vezes, antes que o linker consiga encontrar todos os símbolos.

Listagem 30: Exemplo de falho do linker para *make veryclean APPLICATION=guard_semaphore_test*

```
(cd app && make --print-directory)
make[2]: Entering directory '/Epos2x86/app'
/Epos2x86/bin/eposcc -Wa,--32 -c -ansi -O2  -o guard_semaphore_test.o
guard_semaphore_test.cc
/Epos2x86/bin/eposcc --library --gc-sections --nmagic -o guard_semaphore_test
guard_semaphore_test.o
/Epos2x86/lib/pc_init_system.o: In function `global constructors keyed to
_ZN4EPOS1S11init_systemE':
init_system.cc:(.text._GLOBAL__I__ZN4EPOS1S11init_systemE+0xe5): undefined reference to
`EPOS::S::System::init()'
makefile:8: recipe for target 'guard_semaphore_test' failed
make[2]: Leaving directory '/Epos2x86/app'
make[2]: *** [guard_semaphore_test] Error 1
makefile:17: recipe for target 'app' failed
make[1]: Leaving directory '/home/linux/OperatingSystems/Epos2x86'
make[1]: *** [app] Error 2
makefile:23: recipe for target 'run' failed
make: *** [run] Error 2
```

Depois de ter um erro desse, é importante não executar o *make veryclean APPLICATION=guard_semaphore_test* novamente, mas somente *make APPLICATION=guard_semaphore_test*'', *sem o* veryclean'' por que a aplicação foi totalmente compilada, mas como foi compilada paralelamente, algumas threads ainda não tinham terminado no momento que o linker começou a executar. Assim, falta descobrir como implementar uma barreira faça com que o linker esperar, antes de iniciar o processo de ligação.

Assert ←

Durante o desenvolvimento do trabalho, foi habilitado as instruções de assert do EPOS, no arquivo */include/system/config.h*. Entretanto, ao fazer isso, foram relevados alguns erros de compilação e de execução do EPOS.

Um dos erros relevados pelos asserts foi construtor da classe *Thread*, que deixa de ser atômico quando compila-se o EPOS para várias CPU's. Ao executar o trecho de código do construtor da classe *Thread* a seguir, as interrupções são ligadas pelo operator new. Isso, é um problema por que o construtor da classe *Thread* inicia com um *lock()* e termina com um *unlock()*, portanto, não pode-se ter alguém no meio do processo reativando as interrupções.

Listagem 31: Trecho do construtor de */src/component/thread.cc*

```
// Somewhere on the new operator, it is being called enable() and disable() CPU
interrupt
if(Traits<MMU>::colorful && color != WHITE)
    _stack = new (color) char[stack_size];
else
    _stack = new (SYSTEM) char[stack_size];
```

Para resolver-se esse problema, tentou-se estudar o código do operador de new, para encontrar onde tal anomalia estava acontecendo. Entretanto, mesmo que descoberto onde isso esteja acontecendo, tentar alterar ou modificar tal código, seria muito problemático. Portanto, ao contrario de lutar contra o problema, abraçou-se ele. Assim, modificou-se arquivo *cpu.h* adicionado-se o suporte a uma pilha de chamadas *lock()* e *unlock()*. Assim, pode-se recursivamente chamar funções que precisa ser atómicas, garantido que a chamada mais externa continuará sempre atômica.

Listagem 32: Mudanças no arquivo */include/architecture/ia32/cpu.h*

```
static void int_enable()
{
    if( int_disabled() )
    {
        if( _not_reenable == 0 ) {
            ASM("sti");
        }
        else {
            _not_reenable -= 1;
        }
    }
```

```
}

static void int_disable()
{
    if( int_enabled() ) {
        ASM("cli");
    }
    else {
        _not_reenable += 1;
    }
}
```

A seguir, vê-se um resumo das commits com errors relacionadas a ativação do assert.

Listagem 33: Lista de commits criadas pela ativação do *assert()*

```
Date:   Tue Nov 13 16:55:14 2018 -0200
    Enabled asserts on config.h

diff --git a/include/system/config.h b/include/system/config.h
index 23e2476..83b045b 100644
--- a/include/system/config.h
+++ b/include/system/config.h
@@ -71,8 +71,8 @@ namespace EPOS {
 //==========================================================================
 // ASSERT (for pre and post conditions)
 //==========================================================================
-//#define assert(expr)    ((expr) ? static_cast<void>(0) : Assert::fail (#expr,
__FILE__, __LINE__, __PRETTY_FUNCTION__))
-#define assert(expr)    (static_cast<void>(0))
+#define assert(expr)    ((expr) ? static_cast<void>(0) : Assert::fail (#expr,
__FILE__, __LINE__, __PRETTY_FUNCTION__))
+// #define assert(expr)    (static_cast<void>(0))

 //==========================================================================
 // CONFIGURATION


Date:   Tue Nov 13 16:55:03 2018 -0200
    Make the debug.h assert messages more visible

diff --git a/include/utility/debug.h b/include/utility/debug.h
index 6508d48..ade97f7 100644
--- a/include/utility/debug.h
+++ b/include/utility/debug.h
@@ -116,7 +116,7 @@ class Assert
 {
 public:
     static void fail(const char * __assertion, const char * __file, unsigned int
__line, const char * __function) {
-        db<void>(ERR) << "Assertion fail: " << __assertion << ", function=" <<
```

```
__function << ", file=" << __file << ", line=" << __line << endl;
+        db<void>(ERR) << endl << endl << "Assertion fail: " << __assertion << ",
function=" << __function << ", file=" << __file << ", line=" << __line << endl << endl;
     }
 };


Date:   Tue Nov 13 16:51:29 2018 -0200
    Fixed tstp.h:1255: error: incomplete type 'EPOS::S::Cipher' used in
    nested name specifier.

diff --git a/include/tstp.h b/include/tstp.h
index cd593d1..8cbfbc3 100644
--- a/include/tstp.h
+++ b/include/tstp.h
@@ -1251,7 +1251,8 @@ public:

             db<TSTP>(INF) << "Node ID: " << _id << endl;

-            assert(Cipher::KEY_SIZE == sizeof(Node_ID));
+            // There is not cipher.h header
+            // assert(Cipher::KEY_SIZE == sizeof(Node_ID));
             _cipher.encrypt(_id, _id, _auth);
         }
         ~Security();


Date:   Tue Nov 13 16:20:05 2018 -0200
    Fixed ic.h:503: error: array subscript is above array bounds

diff --git a/include/machine/pc/ic.h b/include/machine/pc/ic.h
index 49c8f00..4aabea4 100644
--- a/include/machine/pc/ic.h
+++ b/include/machine/pc/ic.h
@@ -499,8 +499,11 @@ public:
     static void int_vector(const Interrupt_Id & i, const Interrupt_Handler & h) {
         db<IC>(TRC) << "IC::int_vector(int=" << i << ",h=" << reinterpret_cast<void
*>(h) <<")" << endl;
         assert(i < INTS);
+        // When enabling the asserts expressions, the compiler keeps complaining about
         // WARNING: in static member function 'static void EPOS::S::FPGA::init()':
error: array subscript is above array bounds
-        _int_vector[i] = h;
+        // Then, do some hack to overrule it  and let the above assert() do its thing
on runtime.
+        unsigned int index = i >= INTS ? INTS - 1 : i;
+        _int_vector[index] = h;
     }

     static void enable() {


Date:   Tue Nov 13 16:14:31 2018 -0200
    Fixed pmu.h:602: assert error: comparison between signed and unsigned
```

```
    integer expressions

diff --git a/include/architecture/ia32/pmu.h b/include/architecture/ia32/pmu.h
index e8f7cc7..7184c17 100644
--- a/include/architecture/ia32/pmu.h
+++ b/include/architecture/ia32/pmu.h
@@ -599,7 +599,7 @@ public:
     Intel_Sandy_Bridge_PMU() {}

     static bool config(const Channel & channel, const Event & event, const Flags &
flags = NONE) {
-        assert((channel < CHANNELS) && (event < EVENTS));
+        assert((channel < CHANNELS) && (static_cast<unsigned int>(event) < EVENTS));
         db<PMU>(TRC) << "PMU::config(c=" << channel << ",e=" << event << ",f=" <<
flags << ")" << endl;

         if(((channel == 0) && (event != INSTRUCTION)) || ((channel == 1) && (event !=
DVS_CLOCK)) || ((channel == 2) && (event != CLOCK))) {


Date:   Tue Nov 13 16:11:18 2018 -0200
    Fixed assert error: 'i' was not declared in this scope on ic.h

diff --git a/include/machine/pc/ic.h b/include/machine/pc/ic.h
index a85fbcb..49c8f00 100644
--- a/include/machine/pc/ic.h
+++ b/include/machine/pc/ic.h
@@ -505,7 +505,7 @@ public:

     static void enable() {
         db<IC>(TRC) << "IC::enable()" << endl;
-        assert(i < INTS);
+        // assert(i < INTS);
         Engine::enable();
     }

@@ -517,7 +517,7 @@ public:

     static void disable() {
         db<IC>(TRC) << "IC::disable()" << endl;
-        assert(i < INTS);
+        // assert(i < INTS);
         Engine::disable();
     }


Date:   Thu Nov 15 16:01:15 2018 -0200
    Fixed the thread.cc constructor not being atomic while building
    EPOS with multiple CPUs.
diff --git a/include/architecture/ia32/cpu.h b/include/architecture/ia32/cpu.h
index d6d51d2..1ced73e 100644
--- a/include/architecture/ia32/cpu.h
+++ b/include/architecture/ia32/cpu.h
@@ -313,8 +313,29 @@ public:
```

```
     static Hertz clock() { return _cpu_clock; }
     static Hertz bus_clock() { return _bus_clock; }

-    static void int_enable() { ASM("sti"); }
-    static void int_disable() { ASM("cli"); }
+    static void int_enable()
+    {
+        if( int_disabled() )
+        {
+            if( _not_reenable == 0 ) {
+                ASM("sti");
+            }
+            else {
+                _not_reenable -= 1;
+            }
+        }
+    }
+
+    static void int_disable()
+    {
+        if( int_enabled() ) {
+            ASM("cli");
+        }
+        else {
+            _not_reenable += 1;
+        }
+    }
+
     static bool int_enabled() { return (flags() & FLAG_IF); }
     static bool int_disabled() { return !int_enabled(); }

@@ -624,6 +645,7 @@ private:
     static void init();

 private:
+    static int _not_reenable;
     static unsigned int _cpu_clock;
     static unsigned int _bus_clock;
 };
diff --git a/src/architecture/ia32/cpu.cc b/src/architecture/ia32/cpu.cc
index 8357fa0..e712aa2 100644
--- a/src/architecture/ia32/cpu.cc
+++ b/src/architecture/ia32/cpu.cc
@@ -8,6 +8,7 @@ extern "C" { void _exec(void *); }
 __BEGIN_SYS

 // Class attributes
+int CPU::_not_reenable = 0;
 unsigned int CPU::_cpu_clock;
 unsigned int CPU::_bus_clock;

diff --git a/src/component/thread.cc b/src/component/thread.cc
index 9495641..27dd942 100644
--- a/src/component/thread.cc
```

```
+++ b/src/component/thread.cc
@@ -26,6 +26,7 @@ void Thread::constructor_prologue(const Color & color, unsigned int
stack_size)
     _thread_count++;
     _scheduler.insert(this);

+    // Somewhere on the new operator, it is being called enable() and disable() CPU
interrupt
     if(Traits<MMU>::colorful && color != WHITE)
         _stack = new (color) char[stack_size];
     else
@@ -50,10 +51,14 @@ void Thread::constructor_epilogue(const Log_Addr & entry, unsigned
int stack_siz
     if((_state != READY) && (_state != RUNNING))
         _scheduler.suspend(this);

-    if(preemptive && (_state == READY) && (_link.rank() != IDLE))
+    if(preemptive && (_state == READY) && (_link.rank() != IDLE)) {
+        db<Scheduler<Thread> >(TRC) << "Thread::constructor_epilogue(locked=" <<
locked() << ")" << endl;
+        assert(locked());
         reschedule(_link.rank().queue());
-    else
+    }
+    else {
         unlock();
+    }
 }


@@ -124,6 +129,9 @@ void Thread::priority(const Priority & c)
     }

     if(preemptive) {
+        db<Scheduler<Thread> >(TRC) << "Thread::priority(locked=" << locked() << ")"
<< endl;
+        assert(locked());
+
         reschedule(old_cpu);
         if(smp) {
             lock();
@@ -312,6 +320,7 @@ void Thread::wakeup_all(Queue * q)
 void Thread::reschedule()
 {
     db<Scheduler<Thread> >(TRC) << "Thread::reschedule()" << endl;
+    db<Scheduler<Thread> >(TRC) << "Thread::reschedule(locked=" << locked() << ")" <<
endl;

     // lock() must be called before entering this method
     assert(locked());
@@ -325,6 +334,10 @@ void Thread::reschedule()

 void Thread::reschedule(unsigned int cpu)
 {
```

```
+    db<Scheduler<Thread> >(TRC) << "Thread::reschedule(cpu=" << cpu << ")" << endl;
+    db<Scheduler<Thread> >(TRC) << "Thread::reschedule(locked=" << locked() << ")" <<
endl;
+    assert(locked());
+
     if(!smp || (cpu == Machine::cpu_id()))
         reschedule();
     else {
@@ -338,6 +351,8 @@ void Thread::reschedule(unsigned int cpu)
 void Thread::rescheduler(const IC::Interrupt_Id & interrupt)
 {
     lock();
+    // if( Traits<IC>::dispatch_debugged )
+    db<Synchronizer>(TRC) << "Thread::rescheduler(interrupt=" << interrupt << ")" <<
endl;

     reschedule();
 }
```

Futures ←

Das 2 implementações distintas de futures apresentadas, realiza-se a implementação do modelo de future bloqueante, apresentado no artigo referência [2]. Como a implementação atual do algoritmo do guarda não suporta passagens de parâmetros variádicos, fez-se a implementação dos métodos para que aceitem vários parâmetros com meta programação. Ver a seção Closure Metaprogramada.

Uma vez que a future é resolvida, i.e., seu método *resolve()* é chamado e seu valor é definido, ele não pode ser mais alterado, e qualquer um que tentar obter o valor da future com *get_value()* não precisará mais bloquear, pois o valor não é indefinido. Futures são implementadas totalmente no arquivo header *.h* por que são meta classes que são instanciadas pelo compilador e seu código é gerado somente caso alguém crie uma especialização com algum tipo de dado.

Listagem 34: Implementação de future em */include/utility/future.h*

```
// EPOS Guard Component Declarations

#ifndef __future_h
#define __future_h

#include <semaphore.h>
#include <condition.h>
#include <utility/debug_sync.h>

__BEGIN_UTIL

template<typename FutureType>
class Future
{
public:
    Future(): _is_resolved(false), _condition()  {
```

```
        DB( Synchronizer, TRC, "Future(_is_resolved=" << _is_resolved
                << ", _condition=" << _condition.size()
                << ") => " << this << endl )
    }

    ~Future() {
        DB( Synchronizer, TRC, "~Future(this=" << this << ")" << endl );
    }

    FutureType get_value() {
        DB( Synchronizer, TRC, "Future::get_value(this=" << this
                << " _is_resolved=" << _is_resolved
                << " _condition=" << _condition.size()
                <<  ")" << endl )
        // Optimization once _is_resolved is set true, we do not need lock anymore
        if( _is_resolved ) {
            return _value;
        }

        // If _is_resolved is not set to true, lock and double check _is_resolved
        Thread::lock();
        if( !_is_resolved ) {
            _condition.wait(); // implicit Thread::unlock()
        }
        else {
            Thread::unlock();
        }
        return _value;
    }

    void resolve(FutureType value) {
        DB( Synchronizer, TRC, "Future::resolve(this=" << this
                << " _is_resolved=" << _is_resolved
                << " _condition=" << _condition.size()
                <<  ")" << endl )
        Thread::lock();
        assert( !_is_resolved );
        // If the instruction pointer got until here, and the thread is unscheduled,
        // and another thread call `resolve()`, then, the `assert` will not work,
        // if the whole resolve() call is not atomic.
        _value = value;
        _is_resolved = true;
        _condition.broadcast(); // implicit Thread::unlock()
    }

private:
    volatile bool _is_resolved;

    Condition _condition;
    FutureType _value;
};

__END_UTIL
```

```
#endif
```

Agora ve-sê um simples programa que demonstra o uso de futures:

Listagem 35: Arquivo de Teste */app/future_simple_test.cc*

```
// EPOS Synchronizer Component Test Program
#define DEBUG_SYNC

#include <thread.h>
#include <utility/future.h>
#include <alarm.h>

using namespace EPOS;

int producerFunction(Future<int>* future) {
    LOG( "producerFunction ()" << endl )
    Delay thinking(1000000);
    future->resolve(10);

    LOG( "producerFunction (resolving future=" << future << " to 10)" << endl )
    return 0;
}

int consumerFunction(Future<int>* future) {
    LOG( "consumerFunction ()" << endl )

    auto value = future->get_value();
    LOG( "consumerFunction (result=" << value << ")" << endl )

    value = future->get_value();
    LOG( "consumerFunction (result=" << value << ")" << endl )
    return 0;
}

int main()
{
    LOG( endl )
    LOG( "Starting main application..." << endl )
    Future<int>* future = new Future<int>();

    Thread* consumer = new Thread(&consumerFunction, future);
    Thread* producer = new Thread(&producerFunction, future);

    consumer->join();
    producer->join();

    LOG( "Exiting main application..." << endl )
    return 0;
}
```

Resultado da execução:

Listagem 36: Exemplo de execução de */app/future_simple_test.cc*

```
Starting main application...
Future(_is_resolved=0, _condition=0) => 0x00097f5c
consumerFunction ()
Future::get_value(this=0x00097f5c _is_resolved=0 _condition=0)
producerFunction ()
Future::resolve(this=0x00097f5c _is_resolved=0 _condition=1)
consumerFunction (result=10)
Future::get_value(this=0x00097f5c _is_resolved=1 _condition=0)
consumerFunction (result=10)
producerFunction (resolving future=0x00097f5c to 10)
Exiting main application...
```

A seguir vê-se um simples exemplo de uso de futures. Nesse exemplo, evolve-se varias classes to EPOS, fazendo seu uso em conjunto com o algoritmo do guarda.

Listagem 37: Arquivo de Teste */app/future_guard_test.cc*

```
// EPOS Synchronizer Component Test Program
#define DEBUG_SYNC

#include <thread.h>
#include <utility/guard.h>
#include <utility/future.h>
#include <alarm.h>

using namespace EPOS;
static volatile int counter = 0;

Guard guard;

void increment_counter(Future<int>* future) {
    Delay thinking(1000000);
    counter = counter + 1;

    LOG( "increment_counter (counter=" << counter << ")" << endl )
    future->resolve(counter);
}

int functionA() {
    LOG( "functionA ()" << endl )
    Future<int>* future = new Future<int>();

    guard.submit(&increment_counter, future);
    auto value = future->get_value();
```

```
    LOG( "functionA (result=" << value << ")" << endl )
    return 0;
}

int functionB() {
    LOG( "functionB ()" << endl )
    Future<int>* future = new Future<int>();

    guard.submit(&increment_counter, future);
    auto value = future->get_value();

    LOG( "functionB (result=" << value << ")" << endl )
    return 0;
}

int main()
{
    LOG( endl )
    LOG( "Starting main application..." << endl )

    Thread* producer = new Thread(&functionA);
    Thread* consumer = new Thread(&functionB);

    consumer->join();
    producer->join();

    LOG( "Exiting main application..." << endl )
    return 0;
}
```

Resultado da execução:

Listagem 38: Exemplo de execução de */app/future_guard_test.cc*

```
Starting main application...
functionA ()
Future(_is_resolved=0, _condition=0) => 0x0008fef4
functionB ()
Future(_is_resolved=0, _condition=0) => 0x0008feb4
Future::get_value(this=0x0008feb4 _is_resolved=0 _condition=0)
increment_counter (counter=1)
Future::resolve(this=0x0008fef4 _is_resolved=0 _condition=0)
increment_counter (counter=2)
Future::resolve(this=0x0008feb4 _is_resolved=0 _condition=1)
functionB (result=2)
Future::get_value(this=0x0008fef4 _is_resolved=1 _condition=0)
functionA (result=1)
Exiting main application...
```

Closure Metaprogramada ←

A *Critical_Section* inicialmente descrita não apresenta a possibilidade de definir-se/criar-se funções com um número variado de argumentos. Por isso, essa implementação foi substituída por uma *Critical_Section* meta programada, que faz uso de templates variádicos do C++ 11.

A seguir ve-sê a implementação da classe base de todas as *Critical_Section*. Uma classe base é necessária por que o algoritmo do Guarda implementa uma lista encadeada de seções críticas, que em nosso caso são presentados por *Closures*. Assim, como a implementação de lista, que o algoritmo do Guarda faz uso, não permite que tipos diferentes sejam adicionados na lista, faz-se uso do polimorfismo para criar um lista com diferentes closures meta programadas.

Listagem 39: Base de todas *Closures* meta programadas, Arquivo: */include/utility/critical_section.h*

```
class Critical_Section_Base
{
    /// The Guard class requires access to our the _next private attribute
    friend class Guard;

public:
    Critical_Section_Base(): _next(this) {
        DB( Synchronizer, TRC, "Critical_Section_Base(_next=" << &_next
                << ") => " << this << endl )
    }

    /// This must to be virtual otherwise the derived classes objects destructor
    /// method would not be called when accessed by a base class pointer.
    virtual ~Critical_Section_Base() {
        DB( Synchronizer, TRC, "~Critical_Section_Base(this=" << this
                << " _next=" << &_next << ")" << endl )
    }

    /// Returns void because the base class Critical_Section_Base() cannot be a
    /// template class, as it is general interface for all Closures used on the
    /// Guard algorithm. Also, it does not make sense to return something from
    /// the closure when in its code is ran by the sequencer on another thread,
    /// detached from the original code.
    ///
    /// This must to be virtual otherwise the derived classes objects run()
    /// method would not be called when accessed by a base class pointer.
    virtual void start() = 0;

private:
    // Inspired by the thread code
    Critical_Section_Base* _next;
};
```

A implementação da *Critical_Section* meta programada é simplesmente a herança privada da implementação de *Closure*, mais a herança da classe base de todas as seções críticas, a *Critical_Section_Base*.

Listagem 40: Definição da *Critical_Section* meta programada baseada na implementação de *Closures*

```
template<typename... Tn>
class Critical_Section: public Critical_Section_Base, private Closure<void( Tn... )>
{
    using Closure<void( Tn... )>::run;

public:
    Critical_Section(void(*_entry)(Tn ...), Tn ... an) :
            Closure<void( Tn... )>::Closure( _entry, an ... )
    {
        DB( Synchronizer, TRC, "Critical_Section(_entry="
                << reinterpret_cast<void *>(_entry) << ") => " << this << endl )
    }

    inline void start() {
        run();
    }
};
```

Argumentos Variádicos ←

Para poder-se criar Closures com um número variado do parâmetros, fez-se uso de um meta programa que calcula, recursivamente as posições dos parâmetros da closures.

Listagem 41: Cálculo do index dos parâmetros da seção crítica dentro da closure

```
template<int ...>
struct MetaSequenceOfIntegers { };

template<int AccumulatedSize, typename Tn, int... GeneratedSequence>
struct GeneratorOfIntegerSequence;

template<
            int AccumulatedSize,
            typename Grouper,
            typename Head,
            typename... Tail,
            int... GeneratedSequence
        >
struct GeneratorOfIntegerSequence<
        AccumulatedSize, Grouper( Head, Tail... ), GeneratedSequence... >
{
    typedef typename GeneratorOfIntegerSequence
            <
                AccumulatedSize + sizeof(Head),
                Grouper( Tail... ),
                GeneratedSequence...,
```

```
                AccumulatedSize
            >::type type;
};

template<int AccumulatedSize, typename Grouper, int... GeneratedSequence>
struct GeneratorOfIntegerSequence<AccumulatedSize, Grouper(), GeneratedSequence...>
{
  typedef MetaSequenceOfIntegers<GeneratedSequence...> type;
};
```

Este programa é executado em tempo de compilação, e para uma dada entrada, como por exemplo *GeneratorOfIntegerSequence< 0, int(char, int, char) >::type()*, gera o seguinte código, que calcula o pacote de parâmetros *<0, 1, 5>*, que significa que o primeiro parâmetro da função, que é o tipo *char*, está na posição 0, enquanto o segundo parâmetro do tipo *int* está na posição 1, que e o último parâmetro, *char* está na posição 5. Tais valores devem-se aos respectivos tamanhos de *char* e *int* serem 1 e 4:

Listagem 42: Trecho de código gerado pelo meta programa, obtido com compilador *LLVM Clang*

```
$ clang++ -Xclang -ast-print -fsyntax-only \
  parameter_pack_sequencer_size_generator.cpp > expanded.cpp

template<> struct GeneratorOfIntegerSequence<
            0, int (char, int, char), <>>
{
    typedef typename GeneratorOfIntegerSequence<
            0 + sizeof(char), int (int, char), 0>::type type;
};

template<> struct GeneratorOfIntegerSequence<1, int (int, char), <0>>
{
    typedef typename GeneratorOfIntegerSequence<
            1 + sizeof(int), int (char), 0, 1>::type type;
};

template<> struct GeneratorOfIntegerSequence<5, int (char), <0, 1>>
{
    typedef typename GeneratorOfIntegerSequence<
            5 + sizeof(char), int (), 0, 1, 5>::type type;
};

template<> struct GeneratorOfIntegerSequence<6, int (), <0, 1, 5>>
{
    typedef MetaSequenceOfIntegers<0, 1, 5> type;
};
```

Segue o meta programa que implementa closures ou functors de parâmetros variádicos:

Listagem 43: Implementação da Closure meta programa, arquivo */include/utility/closure.h*

```
// https://stackoverflow.com/questions/34957810/variadic-templates-parameter-pack
template<typename Tn>
class Closure;

template<typename ReturnType, typename... Tn>
class Closure<ReturnType( Tn... )>
{
public:
    typedef ReturnType(*Function)(Tn ...);
    static const unsigned int PARAMETERS_COUNT = sizeof...( Tn );
    static const unsigned int PARAMETERS_LENGTH = SIZEOF<Tn ...>::Result;

private:
    Function _entry;
    char* _parameters;

public:
    Closure(Function _entry, Tn ... an): _entry(_entry)
    {
        DB( Synchronizer, TRC, "Closure(_entry=" << reinterpret_cast<void *>(_entry)
                << ", PARAMETERS_COUNT=" << PARAMETERS_COUNT
                << ", PARAMETERS_LENGTH=" << PARAMETERS_LENGTH
                << ", sizeof=" << sizeof(*this) << ") => " << this << endl )

        if(PARAMETERS_LENGTH)
            _parameters = new char[PARAMETERS_LENGTH];

        pack_helper( _parameters, an ... );
    }

    ~Closure() {
        DB( Synchronizer, TRC, "~Closure(this=" << this
                << ", _entry=" << reinterpret_cast<void *>(_entry)
                << ", PARAMETERS_COUNT=" << PARAMETERS_COUNT
                << ", PARAMETERS_LENGTH=" << PARAMETERS_LENGTH
                << ", sizeof=" << sizeof(*this) << ")" << endl )

        if(PARAMETERS_LENGTH)
            delete _parameters;
    }

    inline ReturnType run() {
        return operator()();
    }

    inline ReturnType operator()() {
        return _unpack_and_run(
                typename GeneratorOfIntegerSequence< 0, int(Tn...) >::type() );
    }

private:
    template<int ...Sequence>
    inline ReturnType _unpack_and_run(MetaSequenceOfIntegers<Sequence...>)
    {
```

```
        DB( Synchronizer, TRC, "Closure::_unpack_and_run(this=" << this << ")" << endl
)
        return _entry( unpack_helper<Sequence, Tn>()... );
    }

    template<const int position, typename T>
    inline T unpack_helper()
    {
        DB( Synchronizer, TRC, "Closure::unpack_helper(Head=" << sizeof( T )
                << ", address=" << reinterpret_cast<int *>(_parameters + position)
                << "(" << reinterpret_cast<int *>(_parameters) << ")"
                << ", position=" << position << ")" << endl )

        return *reinterpret_cast<T *>( _parameters + position );
    }

public:
    template<typename Head, typename ... Tail>
    static void pack_helper(char* pointer_address, Head head, Tail ... tail)
    {
        DB( Synchronizer, TRC, "Closure::pack_helper(Head=" << sizeof( Head )
                << ", address=" << reinterpret_cast<int *>(pointer_address)
                << ")" << endl )

        *reinterpret_cast<Head *>(pointer_address) = head;
        pack_helper(pointer_address + sizeof( Head ), tail ...);
    }

    static void pack_helper(char* pointer_address) {}
};
```

Em tempo de compilação, ao detectar a criação de um Objeto closure, o compilador irá criar uma definição específica para a classe Closure de acordo com os argumentos *(function, arg1, ..., argN)*. Algo similar ao exemplo a seguir:

Listagem 44: Ilustração de um meta programa depois de compilado

```
class Closure<funcReturntype(char, …, int), char, …, int>{
  ...
}
```

O tipo *funcReturnType* é inferido a partir do tipo de retorno de function. Os tipos *(char, ... , int)* são inferidos a partir dos tipos dos argumentos *('a', ..., 1)*.

Em tempo de execução, o construtor da classe Closure irá:

1. Alocar memória para o atributo *_parameters*, que é do tipo array de char, de acordo com o tamanho dos tipos dos argumentos *(char, ... , 1)*
2. Utilizar a função *pack_helper()* para colocar *('a', ... , 1)* no array *_parameters*, tratando o fato de possuírem de tipos diferentes.

Exemplo de Uso ←

Veja um exemplo completo de teste/uso das closures meta programadas:

Listagem 45: Arquivo */app/closure_test.cc*

```
// EPOS Synchronizer Component Test Program
#define DEBUG_SYNC

#include <utility/closure.h>
using namespace EPOS;

template<typename ReturnType, typename ... Tn>
Closure< ReturnType(Tn ...) > create_closure( ReturnType(*_entry)( Tn ... ), Tn ... an
)
{
    auto closure = new Closure< ReturnType(Tn ...) >( _entry, an ... );
    LOG( "create_closure " << closure << endl )
    return *closure;
}

template<typename ... Tn>
void test_closure(Tn ... an) {
    auto closure = create_closure(an ...);
    closure();
    LOG( "    void" << endl << endl )
}

template<typename ... Tn>
void test_closure_with_return(Tn ... an) {
    auto closure = create_closure(an ...);
    auto return_value = closure();
    LOG( "    " << return_value << endl << endl )
}

char test_function1(char arg1, int arg2, bool arg3) {
    LOG( "test_function1 " << arg1 << ", " <<  arg2 << ", " << arg3 << endl )
    return 'A';
}

char test_function2(const char* arg1, const char* arg2, char arg3) {
    LOG( "test_function2 " << arg1 << ", " <<  arg2 << ", " << arg3 << endl )
    return 'B';
}

char test_function3() {
    LOG( "test_function3 " << endl )
    return 'C';
}

void test_function4() {
    LOG( "test_function4 " << endl )
}
```

```
void test_function5(const char* arg1) {
    LOG( "test_function5 " << arg1 << endl )
}

int main()
{
    LOG( endl )
    LOG( "main: begin()" << endl )

    test_closure_with_return( &test_function1, 'a', 10, false );
    test_closure_with_return( &test_function2, "test1", "test2", 'b' );
    test_closure_with_return( &test_function3 );

    test_closure( &test_function4 );
    test_closure( &test_function5, "Testa 3" );
    test_closure( &test_function5, "Testa 4" );

    LOG( "main: exiting()" << endl )
    return 0;
}
```

Resultado da execução:

Listagem 46: Exemplo de execução de */app/closure_test.cc*

```
main: begin()
Closure::Closure(_entry=0x0009ff78,
        PARAMETERS_COUNT=3, PARAMETERS_LENGTH=6, sizeof=8) => 0x00097f64
Closure::pack_helper(Head=1, address=0x00097f54)
Closure::pack_helper(Head=4, address=0x00097f55)
Closure::pack_helper(Head=1, address=0x00097f59)
create_closure 0x00097f64
Closure::_unpack_and_run(this=0x0009fef4)
Closure::unpack_helper(Head=1, address=0x00097f59(0x00097f54), position=5)
Closure::unpack_helper(Head=4, address=0x00097f55(0x00097f54), position=1)
Closure::unpack_helper(Head=1, address=0x00097f54(0x00097f54), position=0)
test_function1 a, 10, 0
    A

Closure::~Closure(this=0x0009fef4,
        _entry=0x0009fef4, PARAMETERS_COUNT=3, PARAMETERS_LENGTH=6, sizeof=8)
Closure::Closure(_entry=0x0009ff74,
        PARAMETERS_COUNT=3, PARAMETERS_LENGTH=9, sizeof=8) => 0x00097f54
Closure::pack_helper(Head=4, address=0x00097f44)
Closure::pack_helper(Head=4, address=0x00097f48)
Closure::pack_helper(Head=1, address=0x00097f4c)
create_closure 0x00097f54
Closure::_unpack_and_run(this=0x0009ff34)
Closure::unpack_helper(Head=1, address=0x00097f4c(0x00097f44), position=8)
Closure::unpack_helper(Head=4, address=0x00097f48(0x00097f44), position=4)
Closure::unpack_helper(Head=4, address=0x00097f44(0x00097f44), position=0)
```

```
test_function2 test1, test2, b
    B

Closure::~Closure(this=0x0009ff34,
        _entry=0x0009ff34, PARAMETERS_COUNT=3, PARAMETERS_LENGTH=9, sizeof=8)
Closure::Closure(_entry=0x0009fe8c,
        PARAMETERS_COUNT=0, PARAMETERS_LENGTH=0, sizeof=8) => 0x00097f44
create_closure 0x00097f44
Closure::_unpack_and_run(this=0x0009fe84)
test_function3
    C

Closure::~Closure(this=0x0009fe84,
        _entry=0x0009fe84, PARAMETERS_COUNT=0, PARAMETERS_LENGTH=0, sizeof=8)
Closure::Closure(_entry=0x0009fe8c,
        PARAMETERS_COUNT=0, PARAMETERS_LENGTH=0, sizeof=8) => 0x00097f34
create_closure 0x00097f34
Closure::_unpack_and_run(this=0x0009fe84)
test_function4
    void

Closure::~Closure(this=0x0009fe84,
        _entry=0x0009fe84, PARAMETERS_COUNT=0, PARAMETERS_LENGTH=0, sizeof=8)
Closure::Closure(_entry=0x0009fe8c,
        PARAMETERS_COUNT=1, PARAMETERS_LENGTH=4, sizeof=8) => 0x00097f24
Closure::pack_helper(Head=4, address=0x00097f14)
create_closure 0x00097f24
Closure::_unpack_and_run(this=0x0009fe84)
Closure::unpack_helper(Head=4, address=0x00097f14(0x00097f14), position=0)
test_function5 Testa 3
    void

Closure::~Closure(this=0x0009fe84,
        _entry=0x0009fe84, PARAMETERS_COUNT=1, PARAMETERS_LENGTH=4, sizeof=8)
Closure::Closure(_entry=0x0009fe8c,
        PARAMETERS_COUNT=1, PARAMETERS_LENGTH=4, sizeof=8) => 0x00097f14
Closure::pack_helper(Head=4, address=0x00097f04)
create_closure 0x00097f14
Closure::_unpack_and_run(this=0x0009fe84)
Closure::unpack_helper(Head=4, address=0x00097f04(0x00097f04), position=0)
test_function5 Testa 4
    void

Closure::~Closure(this=0x0009fe84,
        _entry=0x0009fe84, PARAMETERS_COUNT=1, PARAMETERS_LENGTH=4, sizeof=8)
main: exiting()
```

Dada a função *test_function2* do exemplo anterior como tendo a seguinte assinatura, *char test_function2(const char *, const char *, char)*, tem-se o seguinte código meta programado, gerado durante a compilação, para criar a *Closure* da *test_function2*:

Listagem 47: Trecho de código gerado pelo meta programa, obtido com compilador *LLVM Clang*

```
$ clang++ -Xclang -ast-print -fsyntax-only closure_test.cpp > expanded.cpp

template<> class Closure<char (const char *, const char *, char)>
{
public:
    typedef char (*Function)(const char *, const char *, char);

    static const unsigned int PARAMETERS_COUNT = sizeof...(Tn);
    static const unsigned int PARAMETERS_LENGTH = SIZEOF<
            const char *, const char *, char>::Result;

    Closure<char (const char *, const char *, char)>::Function _entry;
    char *_parameters;

    Closure(Closure<char (const char *, const char *, char)>::Function _entry,
            const char *an, const char *an, char an) : _entry(_entry)
    {
        this->_parameters = new char [PARAMETERS_LENGTH];
        pack_helper(this->_parameters, an, an, an);
    }

    ~Closure<char (const char *, const char *, char)>()
    {
        delete this->_parameters;
    }

    char operator()()
    {
        return this->_run(typename GeneratorOfIntegerSequence<
                0, int (const char *, const char *, char)>::type());
    }

private:
    template<> char _run<<0, 8, 16>>(MetaSequenceOfIntegers<0, 8, 16>)
    {
        return this->_entry(
            this->unpack_helper<0, const char *>(),
            this->unpack_helper<8, const char *>(),
            this->unpack_helper<16, char>()
        );
    }

    template<> const char *unpack_helper<0, const char *>()
    {
        return *reinterpret_cast<const char **>(this->_parameters + 0);
    }

    template<> const char *unpack_helper<8, const char *>()
    {
        return *reinterpret_cast<const char **>(this->_parameters + 8);
    }

    template<> char unpack_helper<16, char>()
    {
```

```
        return *reinterpret_cast<char *>(this->_parameters + 16);
    }

public:
    template<> static void pack_helper<const char *, <const char *, char>>(
                char *pointer_address, const char *head, const char *tail, char tail)
    {
        *reinterpret_cast<const char **>(pointer_address) = head;
        pack_helper(pointer_address + sizeof(const char *), tail, tail);
    }

    template<> static void pack_helper<const char *, <char>>(char *pointer_address,
                const char *head, char tail)
    {
        *reinterpret_cast<const char **>(pointer_address) = head;
        pack_helper(pointer_address + sizeof(const char *), tail);
    }

    template<> static void pack_helper<char, <>>(char *pointer_address, char head)
    {
        *reinterpret_cast<char *>(pointer_address) = head;
        pack_helper(pointer_address + sizeof(char));
    }

    static void pack_helper(char *pointer_address)
    {
    }
}
```

Integração com Guarda ←

Para integrar a nova interface de *Critical_Section* no algoritmo do Guarda, foi necessário somente modificar o método *Guard::submit*, que agora passa a ser um método meta programado:

Listagem 48: Trecho de código de */include/utility/guard.h*

```
class Guard
{
    int _size;
    Thread* _sequencer;

    Critical_Section_Base * _head;
    Critical_Section_Base * _tail;

    static const int NULL = 0;
    static const int DONE = 1;

public:
    Guard();
    ~Guard();
```

```
    template<typename ... Tn>
    void submit( void(*entry)( Tn ... ), Tn ... an )
    {
        // Creates a closure with the critical section parameters
        Critical_Section<Tn ...>* cs = new Critical_Section<Tn ...>(entry, an ...);

        Critical_Section_Base * cur = vouch(cs);
        if (cur != reinterpret_cast<Critical_Section_Base *>(NULL)) do {
            _sequencer = Thread::self();
            cur->start();
            cur = clear();
        } while (cur != reinterpret_cast<Critical_Section_Base *>(NULL));
    }

    Critical_Section_Base * vouch(Critical_Section_Base * item);
    Critical_Section_Base * clear();
};

Guard::Guard(): _size(0), _sequencer(0), _head(0), _tail(0) {
    DB( Synchronizer, TRC, "Guard(head=" << _head << ", tail=" << _tail
            << ") => " << this << endl )
}

Guard::~Guard() {
    DB( Synchronizer, TRC, "~Guard(this=" << this << ")" << endl )
}

Critical_Section_Base* Guard::vouch(Critical_Section_Base * item)
{
    DB( Synchronizer, TRC, "Guard::vouch(this=" << this
            << " head=" << _head
            << " tail=" << _tail
            << " thread=" << Thread::self()
            << " sequencer=" << _sequencer
            << " size=" << _size + 1
            << " item=" << item )
    CPU::finc(_size);

    item->_next = reinterpret_cast<Critical_Section_Base *>(NULL);
    Critical_Section_Base * last = CPU::fas(_tail, item);
    DB( Synchronizer, TRC, " last=" << last << ")" << endl )

    if( last ) {
        if( CPU::cas( last->_next, reinterpret_cast<Critical_Section_Base *>(NULL),
item )
                  == reinterpret_cast<Critical_Section_Base *>(NULL) )

            return reinterpret_cast<Critical_Section_Base *>(NULL);
        delete last;
    }
    _head = item;
    return item;
}
```

```
Critical_Section_Base* Guard::clear()
{
    DB( Synchronizer, TRC, "Guard::clear(this=" << this
            << " head=" << _head
            << " tail=" << _tail
            << " thread=" << Thread::self()
            << " sequencer=" << _sequencer
            << " size=" << _size - 1 )
    CPU::fdec(_size);

    Critical_Section_Base * item = _head;
    Critical_Section_Base * next = CPU::fas( item->_next,
            reinterpret_cast<Critical_Section_Base *>(DONE) );

    DB( Synchronizer, TRC, " next=" << next << ")" << endl )
    bool mine = true;

    if( !next ) {
        mine = CPU::cas( _tail, item, reinterpret_cast<
                Critical_Section_Base *> (NULL) ) == item;
    }

    CPU::cas( _head, item, next );
    if( mine ) {
        delete item;
    }
    return next;
}
```

Obrigatoriamente a nossa closure que representa a seção crítica, tem que retornar *void*, pois no contexto de execução do algoritmo do guarda, uma vez que a *Closure* começa a executar, a única forma que ela tem para se comunicar com a thread original é através de uma memória compartilhada, que em nosso caso é utilizado o mecanismo de *Futures*, como já explicado.

Agora mostra-se um exemplo de uso de closures e do algoritmo do Guarda.

Listagem 49: Arquivo */app/closure_guard_test.cc*

```
// EPOS Synchronizer Component Test Program
#define DEBUG_SYNC

#include <thread.h>
#include <machine.h>
#include <utility/guard.h>
#include <utility/debug_sync.h>
#include <alarm.h>

using namespace EPOS;
Guard counter_guard;

void show(char char1, const char* text1, const char* text2,
```

```
            bool bool1, bool bool2, const char* text3, int int1) {
    LOG( "    char1=" << char1 << ", text1=" << text1
        << ", text2=" << text2 << ", bool1=" << bool1 << ", bool2=" << bool2
        << ", text3=" << text3 << ", int1="  << int1  << endl )
}

int main()
{
    LOG( endl )
    LOG( "main: begin()" << endl )
    counter_guard.submit(&show, 'A', "Test 1", "Test 2", true, false, "Test 3", 10);
    counter_guard.submit(&show, 'B', "Test 4", "Test 5", false, false, "Test 6", 20);

    LOG( "main: exiting()" << endl )
    return 0;
}
```

Resultado da execução:

Listagem 50: Exemplo de execução de */app/closure_guard_test.cc*

```
main: begin()
Closure::Closure(_entry=0x0009ff5c,
        PARAMETERS_COUNT=7, PARAMETERS_LENGTH=19, sizeof=8) => 0x00097f68
Closure::pack_helper(Head=1, address=0x00097f45)
Closure::pack_helper(Head=4, address=0x00097f46)
Closure::pack_helper(Head=4, address=0x00097f4a)
Closure::pack_helper(Head=1, address=0x00097f4e)
Closure::pack_helper(Head=1, address=0x00097f4f)
Closure::pack_helper(Head=4, address=0x00097f50)
Closure::pack_helper(Head=4, address=0x00097f54)
Closure::_unpack_and_run(this=0x00097f68)
Closure::unpack_helper(Head=4, address=0x00097f54(0x00097f45), position=15)
Closure::unpack_helper(Head=4, address=0x00097f50(0x00097f45), position=11)
Closure::unpack_helper(Head=1, address=0x00097f4f(0x00097f45), position=10)
Closure::unpack_helper(Head=1, address=0x00097f4e(0x00097f45), position=9)
Closure::unpack_helper(Head=4, address=0x00097f4a(0x00097f45), position=5)
Closure::unpack_helper(Head=4, address=0x00097f46(0x00097f45), position=1)
Closure::unpack_helper(Head=1, address=0x00097f45(0x00097f45), position=0)
    char1=A, text1=Test 1, text2=Test 2, bool1=1, bool2=0, text3=Test 3, int1=10

Closure::~Closure(this=0x00097f68,
        _entry=0x00097f68, PARAMETERS_COUNT=7, PARAMETERS_LENGTH=19, sizeof=8)
Closure::Closure(_entry=0x0009ff5c,
        PARAMETERS_COUNT=7, PARAMETERS_LENGTH=19, sizeof=8) => 0x00097f68
Closure::pack_helper(Head=1, address=0x00097f45)
Closure::pack_helper(Head=4, address=0x00097f46)
Closure::pack_helper(Head=4, address=0x00097f4a)
Closure::pack_helper(Head=1, address=0x00097f4e)
Closure::pack_helper(Head=1, address=0x00097f4f)
Closure::pack_helper(Head=4, address=0x00097f50)
```

```
Closure::pack_helper(Head=4, address=0x00097f54)
Closure::_unpack_and_run(this=0x00097f68)
Closure::unpack_helper(Head=4, address=0x00097f54(0x00097f45), position=15)
Closure::unpack_helper(Head=4, address=0x00097f50(0x00097f45), position=11)
Closure::unpack_helper(Head=1, address=0x00097f4f(0x00097f45), position=10)
Closure::unpack_helper(Head=1, address=0x00097f4e(0x00097f45), position=9)
Closure::unpack_helper(Head=4, address=0x00097f4a(0x00097f45), position=5)
Closure::unpack_helper(Head=4, address=0x00097f46(0x00097f45), position=1)
Closure::unpack_helper(Head=1, address=0x00097f45(0x00097f45), position=0)
    char1=B, text1=Test 4, text2=Test 5, bool1=0, bool2=0, text3=Test 6, int1=20

Closure::~Closure(this=0x00097f68,
        _entry=0x00097f68, PARAMETERS_COUNT=7, PARAMETERS_LENGTH=19, sizeof=8)
main: exiting()
```

Guarda para Linux ←

A seguir, encontra-se uma reimplementação de todos os algoritmos já apresentados, mas com a diferença que tratam-se de uma versão dos mesmos componentes para uma máquina nativa Linux. Tal reimplementação foi feita por que uma das aplicações de teste não conseguia funcionar no EPOS devido algum problema desconhecido. Para mais informações, veja a seção guard_scheduler_cpu_affinity_test.cc.

Para compilar um programa que que incluí essa implementação, você utilizar um linha de comando parecida com a seguinte: *g++ -ggdb -O0 -o test application.cpp -m32 --std=c++11 -lpthread && ./test*

Listagem 51: Arquivo */non-epos-guard/count_sync_guard/guard.h*

```
#include <mutex>
#include <atomic>
#include <cassert>

#include <iomanip>
#include <iostream>
#include <condition_variable>

#define ASM __asm__ __volatile__


// You can define it anywhere before including this file
// #define DEBUG

#ifdef DEBUG
    std::recursive_mutex _debug_syncronized_semaphore_lock;

    #define DB(...) do { \
        _debug_syncronized_semaphore_lock.lock(); \
            std::cout << __VA_ARGS__ << std::flush; \
        _debug_syncronized_semaphore_lock.unlock(); } while(0);

    #define LOG(...) DB(__VA_ARGS__)
```

```
#else
    #define DB(...)
    #define LOG(...) std::cout << __VA_ARGS__ << std::flush;

#endif


template<typename T>
static T fas(T volatile & value, T replacement)
{
    ASM("lock xchgl %1, %2"
            : "=r"(replacement)
            : "r"(replacement), "m"(value)
            : "memory");
    return replacement;
}

template<typename T>
static T cas(T volatile & value, T compare, T replacement)
{
    ASM("lock cmpxchgl %2, %3\n"
            : "=a"(compare)
            : "a"(compare), "r"(replacement), "m"(value)
            : "memory");
    return compare;
}


class Display
{
private:
    static const int LINES = 25;
    static const int COLUMNS = 8000;
    static const int TAB_SIZE = 8;

    // Special characters
    enum {
        ESC  = 0x1b,
        CR   = 0x0d,
        LF   = 0x0a,
        TAB  = 0x09,
    };

public:
    Display() {}

    static void clear() {
        _line = 0;
        _column = 0;
        escape();
        put('2');
        put('J');
    };
```

```
    static void putc(char c) {
        switch(c) {
        case '\n':
            scroll();
            _line++;
            break;
        case '\t':
            put(TAB);
            _column += TAB_SIZE;
            break;
        default:
            _column++;
            put(c);
            if(_column >= COLUMNS) scroll();
        }
    };

    static void puts(const char * s) {
        while(*s != '\0')
            putc(*s++);
    }

    static void geometry(int * lines, int * columns) {
        *lines = LINES;
        *columns = COLUMNS;
    }

    static void position(int * line, int * column) {
        *line = _line;
        *column = _column;
    }
    static void position(int line, int column) {
        _line = line;
        _column = column;
        escape();
        puti(_line);
        put(';');
        puti(_column);
        put('H');
    }

private:
    static void put(char c) {
        std::cout << c;
    }

    static void escape() {
        put(ESC);
        put('[');
    }

    static void puti(unsigned char value) {
        unsigned char digit = '0';
        while(value >= 100) {
```

```
            digit++;
            value -= 100;
        }
        put(digit);

        digit = '0';
        while(value >= 10) {
            digit++;
            value -= 10;
        }
        put(digit);

        put('0' + value);
    }

    static void scroll() {
        put(CR);
        put(LF);
        _column = 0;
    }

    static void init() {
        _line = 0;
        _column = 0;
    }

private:
    static int _line;
    static int _column;
};

int Display::_line;
int Display::_column;


template<typename FutureType>
class Future
{
public:
    Future(): _size(0), _is_resolved(false) {
        DB( "Future(_is_resolved=" << _is_resolved
                << ", _condition=" << _size
                << ") => " << this << std::endl )
    }

    ~Future() {
        DB( "~Future(this=" << this << ")" << std::endl );
    }

    FutureType get_value() {
        DB( "Future::get_value(this=" << this
                << " _is_resolved=" << _is_resolved
                << " _condition=" << _size
                <<  ")" << std::endl )
```

```
        // Optimization once _is_resolved is set true, we do not need lock anymore
        if( _is_resolved ) {
            return _value;
        }

        // If _is_resolved is not set to true, lock and double check _is_resolved
        std::unique_lock<std::mutex> lock(_lock);
        if(!_is_resolved) {
            ++_size;

            // We cannot call _lock.unlock(); before _condition.wait(lock); because
            // 1. It would allow this thread to be preempted
            // 2. Then, some other thread could call resolve()
            // Once this other thread completes the resolve() call, and this
            // thread is rescheduled, we would finally call _condition.wait(lock);
            // but doing so, would cause this thread to be locked indefinitely
            // because we will never call resolve() anymore
            _condition.wait(lock);
        }
        return _value;
    }

    void resolve(FutureType value) {
        DB( "Future::resolve(this=" << this
                << " _is_resolved=" << _is_resolved
                << " _condition=" << _size
                <<  ")" << std::endl )
        _lock.lock();
        assert(!_is_resolved);
        // If the instruction pointer got until here, and the thread is unscheduled,
        // and another thread call `resolve()`, then, the `assert` will not work,
        // if the whole resolve() call is not atomic.
        _value = value;
        _is_resolved = true;
        _lock.unlock();
        _condition.notify_all();
    }

private:
    FutureType _value;
    std::atomic<int> _size;
    volatile std::atomic<bool> _is_resolved;

    std::mutex _lock;
    std::condition_variable _condition;
};



// SIZEOF Type Package
template<typename ... Tn>
struct SIZEOF
{ static const unsigned int Result = 0; };

template<typename T1, typename ... Tn>
```

```
struct SIZEOF<T1, Tn ...>
{ static const unsigned int Result = sizeof(T1) + SIZEOF<Tn ...>::Result ; };

//
https://stackoverflow.com/questions/7858817/unpacking-a-tuple-to-call-a-matching-functi
on
template<int ...>
struct MetaSequenceOfIntegers { };

template<int AccumulatedSize, typename Tn, int... GeneratedSequence>
struct GeneratorOfIntegerSequence;

template<
            int AccumulatedSize,
            typename Grouper,
            typename Head,
            typename... Tail,
            int... GeneratedSequence
        >
struct GeneratorOfIntegerSequence<
        AccumulatedSize, Grouper( Head, Tail... ), GeneratedSequence... >
{
    typedef typename GeneratorOfIntegerSequence
            <
                AccumulatedSize + sizeof(Head),
                Grouper( Tail... ),
                GeneratedSequence...,
                AccumulatedSize
            >::type type;
};

template<int AccumulatedSize, typename Grouper, int... GeneratedSequence>
struct GeneratorOfIntegerSequence<AccumulatedSize, Grouper(), GeneratedSequence...>
{
  typedef MetaSequenceOfIntegers<GeneratedSequence...> type;
};


//
https://stackoverflow.com/questions/34957810/variadic-templates-parameter-pack-and-its
template<typename Tn>
class Closure;

template<typename ReturnType, typename... Tn>
class Closure<ReturnType( Tn... )>
{
public:
    typedef ReturnType(*Function)(Tn ...);
    static const unsigned int PARAMETERS_COUNT = sizeof...( Tn );
    static const unsigned int PARAMETERS_LENGTH = SIZEOF<Tn ...>::Result;

private:
    Function _entry;
    char* _parameters;
```

```
public:
    Closure(Function _entry, Tn ... an): _entry(_entry)
    {
        DB( "Closure(_entry=" << reinterpret_cast<void *>(_entry)
                << ", PARAMETERS_COUNT=" << PARAMETERS_COUNT
                << ", PARAMETERS_LENGTH=" << PARAMETERS_LENGTH
                << ", sizeof=" << sizeof(*this) << ") => " << this << std::endl )

        if(PARAMETERS_LENGTH)
            _parameters = new char[PARAMETERS_LENGTH];

        pack_helper( _parameters, an ... );
    }

    ~Closure() {
        DB( "~Closure(this=" << this
                << ", _entry=" << reinterpret_cast<void *>(_entry)
                << ", PARAMETERS_COUNT=" << PARAMETERS_COUNT
                << ", PARAMETERS_LENGTH=" << PARAMETERS_LENGTH
                << ", sizeof=" << sizeof(*this) << ")" << std::endl )

        if(PARAMETERS_LENGTH)
            delete _parameters;
    }

    inline ReturnType run() {
        return operator()();
    }

    inline ReturnType operator()() {
        return _unpack_and_run(
                typename GeneratorOfIntegerSequence< 0, int(Tn...) >::type() );
    }

private:
    template<int ...Sequence>
    inline ReturnType _unpack_and_run(MetaSequenceOfIntegers<Sequence...>)
    {
        DB( "Closure::_unpack_and_run(this=" << this << ")" << std::endl )
        return _entry( unpack_helper<Sequence, Tn>()... );
    }

    template<const int position, typename T>
    inline T unpack_helper()
    {
        DB( "Closure::unpack_helper(Head=" << sizeof( T )
                << ", address=" << reinterpret_cast<int *>(_parameters + position)
                << "(" << reinterpret_cast<int *>(_parameters) << ")"
                << ", position=" << position << ")" << std::endl )

        return *reinterpret_cast<T *>( _parameters + position );
    }

public:
```

```
    template<typename Head, typename ... Tail>
    static void pack_helper(char* pointer_address, Head head, Tail ... tail)
    {
        DB( "Closure::pack_helper(Head=" << sizeof( Head )
                << ", address=" << reinterpret_cast<int *>(pointer_address)
                << ")" << std::endl )

        *reinterpret_cast<Head *>(pointer_address) = head;
        pack_helper(pointer_address + sizeof( Head ), tail ...);
    }

    static void pack_helper(char* pointer_address) {}
};


class Critical_Section_Base
{
    friend class Guard;
    Critical_Section_Base* _next;

public:
    Critical_Section_Base(): _next(this) {
        DB( "Critical_Section_Base(_next=" << _next
                << ") => " << this << std::endl )
    }

    virtual ~Critical_Section_Base() {
        DB( "~Critical_Section_Base(this=" << this
                << " _next=" << _next << ")" << std::endl )
    }

    virtual void start() = 0;
};

template<typename... Tn>
class Critical_Section: public Critical_Section_Base, private Closure<void( Tn... )>
{
    using Closure<void( Tn... )>::run;

public:
    Critical_Section(void(*_entry)(Tn ...), Tn ... an) :
            Closure<void( Tn... )>::Closure( _entry, an ... )
    {
        DB( "Critical_Section(_entry="
                << reinterpret_cast<void *>(_entry) << ") => " << this << std::endl )
    }

    inline void start() {
        run();
    }
};


class Guard
```

```
{
    std::atomic<int> _size;
    void* _sequencer;

    Critical_Section_Base* _head;
    Critical_Section_Base* _tail;

    // static const unsigned int NULL = 0;
    static const unsigned int DONE = 1;

public:
    Guard() : _size(0), _sequencer(0), _head(0), _tail(0)
    {
        DB( "Guard(head=" << _head << ", tail=" << _tail
                << ") => " << this << std::endl )
    }

    ~Guard() {
        DB( "~Guard(this=" << this << ")" << std::endl )
    }

    template<typename ... Tn>
    void submit( void(*entry)( Tn ... ), Tn ... an )
    {
        // Creates a closure with the critical section parameters
        Critical_Section<Tn ...>* cs = new Critical_Section<Tn ...>(entry, an ...);

        Critical_Section_Base * cur = vouch(cs);
        if (cur != reinterpret_cast<Critical_Section_Base *>(NULL)) do {
            _sequencer = get_thread_id();
            cur->start();
            cur = clear();
        } while (cur != reinterpret_cast<Critical_Section_Base *>(NULL));
    }

    Critical_Section_Base* vouch(Critical_Section_Base * item)
    {
        DB( "Guard::vouch(this=" << this
                << " head=" << _head
                << " tail=" << _tail
                << " thread=" << get_thread_id()
                << " sequencer=" << _sequencer
                << " size=" << _size + 1
                << " item=" << item )
        ++_size;

        item->_next = reinterpret_cast<Critical_Section_Base *>(NULL);
        Critical_Section_Base * last = fas( _tail, item );
        DB( " last=" << last << ")" << std::endl )

        if( last ) {
            if( cas( last->_next, reinterpret_cast<Critical_Section_Base *>(NULL), item
)
                        == reinterpret_cast<Critical_Section_Base *>(NULL) )
```

```
                return reinterpret_cast<Critical_Section_Base *>(NULL);
            delete last;
        }
        _head = item;
        return item;
    }

    Critical_Section_Base * clear()
    {
        DB( "Guard::clear(this=" << this
                << " head=" << _head
                << " tail=" << _tail
                << " thread=" << get_thread_id()
                << " sequencer=" << _sequencer
                << " size=" << _size - 1 )
        --_size;

        Critical_Section_Base * item = _head;
        Critical_Section_Base * next = fas( item->_next,
                reinterpret_cast<Critical_Section_Base *>(DONE) );
        DB( " next=" << next << ")" << std::endl )

        bool mine = true;
        if( !next ) {
            mine = cas( _tail, item, reinterpret_cast<
                    Critical_Section_Base *> (NULL) ) == item;
        }

        cas( _head, item, next) ;
        if( mine ) {
            delete item;
        }
        return next;
    }

    static void* get_thread_id() {
        std::stringstream ss;
        ss << std::this_thread::get_id();
        return reinterpret_cast<void *>( std::stoull( ss.str() ) );
    }
};
```

Testes ←

Cria-se um arquivo de testes baseado no exemplo de http://pages.cs.wisc.edu/~remzi/OSTEP/threads-intro.pdf.

1. Os exemplos com o EPOS multicore, com duas CPUS, variavelmente gerava exceções, que não ocorrem na versão single core. Não pode-se identificar exatamente qual a causa dessas exceções, que são do tipo GPF e PF. Uma constatação importante é que essas exceções ocorrem mesmo em versões minimamente modificadas das aplicações de testes padrões do EPOS.
2. Por duas vezes consegue-se fazer a contagem falhar, ambas vezes utilizando o critério de

escalonamento RR. No entanto, nas últimas muitas execuções a contagem voltou a ocorrer corretamente, o que é indesejado, mesmo com a utilização do escalonamento RR. Com duas CPUS.
3. Foram definidos três programas de teste para a aplicação contadora simples: uma sem sincronização, uma utilizando semáforos e outra com guardas.

Todos os testes a seguir foram realizados utilizando *static const unsigned int CPUS = 4*.

fas_test.cc ←

Aqui testa-se a implementação nova de FAS feita, repetidamente fazendo a troca de variáveis em duas thread executando em CPUs diferentes.

Listagem 52: Arquivo de Teste */app/fas_test.cc*

```
// EPOS Synchronizer Component Test Program
#define DEBUG_SYNC

#include <utility/ostream.h>
#include <utility/debug_sync.h>
#include <thread.h>
#include <cpu.h>

using namespace EPOS;
static const int iterations = 1e5;

int old = 0;
int current = 10;
int next = 11;

#define check(thread, name) \
    LOG( thread << name \
            << ", old=" << old \
            << ", current=" << current \
            << ", next=" << next \
            << endl )

int myThread1() {
    check("Thread 1", ", begin")

    for (int i = 0; i < iterations; i++)
    {
        old = CPU::fas(current, next);
        current = CPU::fas(next, old);
        next = CPU::fas(old, current);
        // check("Thread 1", ", now")
    }

    check("Thread 1", ", result")
    return 0;
}

int myThread2() {
```

```
    check("Thread 2", ", begin")

    for (int i = 0; i < iterations; i++) {
        old = CPU::fas(current, next);
        current = CPU::fas(next, old);
        next = CPU::fas(old, current);
        // check("Thread 2", ", now")
    }

    check("Thread 2", ", result")
    return 0;
}

int main()
{
    LOG( endl )
    LOG( "iterations=" << iterations << endl )
    Thread p1(&myThread1);
    Thread p2(&myThread2);

    check("Thread 0", ", main")
    p1.join();
    p2.join();

    check("Thread 0", ", end")
    return 0;
}
```

Resultado da execução:

Listagem 53: Exemplo de execução de */app/fas_test.cc*

```
<1>: MMU is disabled! :<1>
<2>: MMU is disabled! :<2>
<3>: MMU is disabled! :<3>
<0>: iterations=100000 :<0>
<0>: Thread 0, main, old=0, current=10, next=11 :<0>
<2>: Thread 1, begin, old=0, current=10, next=11 :<2>
<1>: Thread 2, begin, old=10, current=10, next=11 :<1>
<2>: Thread 1, result, old=10, current=10, next=11 :<2>
<1>: Thread 2, result, old=10, current=10, next=11 :<1>
<1>: Thread 0, end, old=10, current=10, next=11 :<1>
<0>: The last thread has exited! :<0>
<0>: Rebooting the machine ... :<0>
```

count_sync_guard.cc ←

Neste teste, utiliza-se a implementação do guarda para controlar o acesso a seção crítica *increment_counter()*, que simplesmente soma um contador. No final da execução, espera-se que resultado da conta seja precisamente o número de iterações feita.

Listagem 54: Arquivo de teste */app/count_sync_guard.cc*

```
// EPOS Synchronizer Component Test Program
#define DEBUG_SYNC

#include <thread.h>
#include <machine.h>
#include <utility/guard.h>
#include <alarm.h>

using namespace EPOS;

static volatile int counter = 0;
static const int iterations = 1e3;

Guard counter_guard;
Guard display_guard;
Thread * pool[5];

void show(char arg, const char * type) {
    LOG( arg << ": " << type << " (counter=" << counter << ")" << endl )
}

void increment_counter() {
    counter = counter + 1;
    LOG( "increment_counter (counter=" << counter << ")" << endl )
}

int mythread(char arg) {
    display_guard.submit(&show, arg, "begin");

    for (int i = iterations; i > 0 ; i--) {
        counter_guard.submit(&increment_counter);
        // Delay label(1000);
    }

    display_guard.submit(&show, arg, "end");
    return 0;
}

int main()
{
    LOG( endl )
    LOG( "main: begin (counter=" << counter << ")" << endl )

    pool[0] = new Thread(&mythread, 'A');
    pool[1] = new Thread(&mythread, 'B');
    pool[2] = new Thread(&mythread, 'C');
    pool[3] = new Thread(&mythread, 'D');
    pool[4] = new Thread(&mythread, 'E');

    LOG( "main: start joining the threads (counter=" << counter << ")" << endl )
```

```
    pool[0]->join();
    pool[1]->join();
    pool[2]->join();
    pool[3]->join();
    pool[4]->join();

    LOG( "main: done with both (counter=" << counter << ")" << endl )
    delete pool[0];
    delete pool[1];
    delete pool[2];
    delete pool[3];
    delete pool[4];

    LOG( "main: exiting (counter=" << counter << ")" << endl )
    return 0;
}
```

Resultado da execução:

Listagem 55: Exemplo de execução de */app/count_sync_guard.cc*

```
<1>: MMU is disabled! :<1>
<2>: MMU is disabled! :<2>
<3>: MMU is disabled! :<3>
<0>:  :<0>
main: begin (counter=0)
<0>: main: start joining the threads (counter=0) :<0>
<0>: A: begin (counter=0) :<0>
<1>: B: begin (counter=1476) :<1>
<1>: C: begin (counter=7900) :<1>
<3>: D: begin (counter=11002) :<3>
<2>: C: end (counter=12407) :<2>
<2>: E: begin (counter=18387) :<2>
<2>: A: end (counter=25320) :<2>
<3>: D: end (counter=27383) :<3>
<2>: E: end (counter=33451) :<2>
<2>: B: end (counter=50000) :<2>
<2>: main: done with both (counter=50000) :<2>
<2>: main: exiting (counter=50000) :<2>
<0>:  :<0>
The last thread has exited!
<0>: Rebooting the machine ... :<0>
```

count_sync_uncoordinated.cc ←

O que se espera deste teste, ao contrário do teste anterior, é que a contagem final de errada, pois nenhuma primitiva de sincronização é utilizada, assim tem-se que a operação de incremento, que não é atômica, seja interrompida pelo metade, e no fim o resultado fique errado.

Listagem 56: Arquivo de teste */app/count_sync_uncoordinated.cc*

```
// EPOS Synchronizer Component Test Program
#define DEBUG_SYNC

#include <semaphore.h>
#include <utility/debug_sync.h>
#include <thread.h>
#include <machine.h>
#include <alarm.h>

using namespace EPOS;
int counter = 0;

// mythread()
// Simply adds 1 to counter repeatedly, in a loop
// No, this is not how you would add 10,000,000 to
// a counter, but it shows the problem nicely.
int mythread(char arg) {
    LOG( arg << ": begin : " << endl)

    for (int i = 0; i < 1e6; i++) {
        if (counter%100000 == 0){
            LOG( arg << " : " << counter << endl )
        }
        counter = counter + 1;
        // LOG( "Counting " << counter << endl )
    }

    LOG( arg << ": done" << endl )
    return 0;
}

// main()
// Just launches two threads (pthread_create)
// and then waits for them (pthread_join)
int main()
{
    LOG( endl )
    LOG( "main: begin (counter = " << counter << ")" << endl )

    Thread * p1 = new Thread(&mythread, 'A');
    Thread * p2 = new Thread(&mythread, 'B');
    Thread * p3 = new Thread(&mythread, 'C');
    Thread * p4 = new Thread(&mythread, 'D');
    Thread * p5 = new Thread(&mythread, 'E');
    Thread * p6 = new Thread(&mythread, 'F');

    // join waits for the threads to finish
    p1->join();
    p2->join();
    p3->join();
```

```
    p4->join();
    p5->join();
    p6->join();

    LOG( "main: done with both (counter = " << counter << ")"<< endl )
    return 0;
}
```

Resultado da execução:

Listagem 57: Exemplo de execução de */app/count_sync_uncoordinated.cc*

```
<1>: MMU is disabled! :<1>
<2>: MMU is disabled! :<2>
<3>: MMU is disabled! :<3>
<0>: main: begin (counter = 0) :<0>
<2>: A: begin :  :<2>
<2>: A : 0 :<2>
<1>: B: begin :  :<1>
<3>: C: begin :  :<3>
<0>: D: begin :  :<0>
<2>: E: begin :  :<2>
<1>: F: begin :  :<1>
<1>: F : 100000 :<1>
<2>: E : 100000 :<2>
<3>: A : 100000 :<3>
<0>: B : 100000 :<0>
<2>: C : 100000 :<2>
<1>: D : 100000 :<1>
<0>: F : 200000 :<0>
<3>: C : 200000 :<3>
<2>: E : 200000 :<2>
<3>: A : 200000 :<3>
<0>: B : 200000 :<0>
<1>: D : 200000 :<1>
<3>: E : 300000 :<3>
<0>: F : 300000 :<0>
<2>: A : 300000 :<2>
<2>: A : 300000 :<2>
<3>: C : 300000 :<3>
<2>: E : 400000 :<2>
<1>: F : 400000 :<1>
<1>: D : 345974 :<1>
<1>: B : 400000 :<1>
<3>: D : 346916 :<3>
<0>: F : 385967 :<0>
<0>: E : 393666 :<0>
<2>: A : 400000 :<2>
<1>: C : 477901 :<1>
<0>: F : 498945 :<0>
<3>: D : 446608 :<3>
```

```
<1>: E : 476901 :<1>
<0>: B : 484604 :<0>
<3>: C : 500000 :<3>
<1>: E : 528187 :<1>
<0>: D : 525792 :<0>
<2>: A : 571028 :<2>
<2>: F : 600000 :<2>
<1>: E : 567826 :<1>
<3>: A : 654100 :<3>
<1>: B : 588204 :<1>
<1>: E : 600000 :<1>
<3>: D : 689233 :<3>
<0>: B : 700000 :<0>
<0>: A : 600000 :<0>
<0>: D : 646539 :<0>
<3>: F : 658486 :<3>
<1>: C : 691064 :<1>
<0>: D : 700000 :<0>
<0>: D : 700000 :<0>
<1>: C : 700000 :<1>
<3>: F : 700000 :<3>
<2>: A : 700000 :<2>
<3>: B : 700000 :<3>
<0>: E : 700000 :<0>
<0>: C : 800000 :<0>
<3>: A : 800000 :<3>
<0>: D : 800000 :<0>
<1>: F : 817259 :<1>
<2>: B : 800000 :<2>
<2>: D : 900000 :<2>
<2>: D : 800000 :<2>
<3>: E : 800000 :<3>
<0>: C : 900000 :<0>
<2>: A : 900000 :<2>
<3>: F : 900000 :<3>
<2>: B : 900000 :<2>
<0>: D : 900000 :<0>
<3>: C : 1000000 :<3>
<2>: F : 1000000 :<2>
<3>: E : 900000 :<3>
<1>: A : 1000000 :<1>
<1>: A : 1000000 :<1>
<0>: B : 1000000 :<0>
<3>: F : 1100000 :<3>
<3>: C : 1100000 :<3>
<0>: D : 1000000 :<0>
<1>: E : 1000000 :<1>
<2>: B : 1100000 :<2>
<3>: C : 1100000 :<3>
<0>: A: done :<0>
<1>: F: done :<1>
<3>: B: done :<3>
<0>: C: done :<0>
<1>: E: done :<1>
```

```
<2>: D : 1100000 :<2>
<2>: D: done :<2>
<2>: main: done with both (counter = 1106138) :<2>
<0>: The last thread has exited! :<0>
<0>: Rebooting the machine ... :<0>
```

count_sync_semaphore_simple.cc ←

Neste teste, utiliza-se os mecanismo de sincronização do semáforo, e ao contrário do teste anterior *count_sync_uncoordinated.cc*, deste teste espera-se que a contagem seja correta, pois utiliza-se o mecanismo de sincronização do semáforo para garantir a atomicidade da operação de adição.

Listagem 58: Arquivo de teste */app/count_sync_semaphore_simple.cc*

```
// EPOS Synchronizer Component Test Program
#define DEBUG_SYNC

#include <utility/debug_sync.h>
#include <semaphore.h>
#include <thread.h>
#include <machine.h>

using namespace EPOS;
static volatile int counter = 0;

Semaphore counter_lock;

// mythread()
// Simply adds 1 to counter repeatedly, in a loop
// No, this is not how you would add 10,000,000 to
// a counter, but it shows the problem nicely.
int mythread(char arg) {
    LOG( arg << ": begin" << endl )
    for (int i = 0; i < 1e4; i++) {
        counter_lock.p();
        counter = counter + 1;
        counter_lock.v();
    }

    LOG( arg << ": done" << endl )
    return 0;
}

// main()
// Just launches two threads (pthread_create)
// and then waits for them (pthread_join)
int main()
{
    LOG( endl )
    LOG( "main: begin (counter = " << counter << ")" << endl )
```

```
    Thread p1(&mythread, 'A');
    Thread p2(&mythread, 'B');

    // join waits for the threads to finish
    p1.join();
    p2.join();

    LOG( "main: done with both (counter = " << counter << ")"<< endl )
    return 0;
}
```

Resultado da execução:

Listagem 59: Exemplo de execução de */app/count_sync_semaphore_simple.cc*

```
<1>: MMU is disabled! :<1>
<2>: MMU is disabled! :<2>
<3>: MMU is disabled! :<3>
<0>: main: begin (counter = 0) :<0>
<0>: A: begin :<0>
<0>: B: begin :<0>
<3>: B: done :<3>
<3>: A: done :<3>
<3>: main: done with both (counter = 20000) :<3>
<0>: The last thread has exited! :<0>
<0>: Rebooting the machine ... :<0>
```

guard_semaphore_test.cc ←

Trata-se do mesmo programa original do EPOS chamada *semaphore_test.cc*, também conhecido como *Philosophers Dinner*. A diferença, é que esta versão utiliza um *Guarda* e várias variáveis *Future*, ao contrário de utilizar alguns semáforos para sincronização.

Sua implementação com o algoritmo do *Guarda* foi muito mais complexa do que a implementação original com semáforos. Para efeitos de desenvolvimento, foram utilizadas macros do *preprocessador C* para simultaneamente criar-se dois programas, dependendo das diretivas passadas ao *preprocessador C*. Ao definir-se macro *#define CONSOLE_MODE*, tem-se uma versão do programa que não utiliza de artefatos gráficos, i.e., desenhar uma mesa simulando uma janta de filósofos visualmente. No modo *CONSOLE_MODE*, mostra-se em detalhes várias mensagem de log, que mostram o funcionamento interno do programa.

Listagem 60: Arquivo de teste */app/guard_semaphore_test.cc*

```
// EPOS Semaphore Component Test Program
#define DEBUG_SYNC

#include <utility/ostream.h>
```

```
#include <utility/stringstream.h>
#include <utility/guard.h>
#include <utility/future.h>
#include <thread.h>
#include <alarm.h>
#include <display.h>

#if !defined(nullptr)
    #define nullptr 0
#endif

using namespace EPOS;

// #define CONSOLE_MODE
const int iterations = 10;

Guard table;

OStream cout;
Thread * phil[5];

bool is_chopstick_free[5];
Future<int>* locked_futures[5];

#ifdef CONSOLE_MODE
    #define LVL WRN
#else
    #define LVL FFF
#endif

void show_message(const char * message, int line, int column) {
#ifdef CONSOLE_MODE
    DB( Synchronizer, LVL, message << "(" << line << ")" << endl )
#else
    Display::position( line, column );
    cout << message;
#endif
}

void show_message(StringStream * message, int line, int column) {
#ifdef CONSOLE_MODE
    DB( Synchronizer, LVL, message )
#else
    Display::position( line, column );
    cout << message;
#endif
    delete message;
}

void release_chopstick(int philosopher_index, int chopstick_index,
        Future<int>* future_chopstick, const char* which_chopstick)
{
    DB( Synchronizer, LVL, "Philosopher=" << philosopher_index
            << ", release chopstick=" << chopstick_index
```

```
            << ", is_chopstick_free=" << is_chopstick_free[chopstick_index]
            << ", future_chopstick=" << future_chopstick
            << ", locked_futures=" << locked_futures[chopstick_index]
            << ", " << which_chopstick )

    is_chopstick_free[chopstick_index] = true;
    delete future_chopstick;

    if( locked_futures[chopstick_index] ) {
        DB( Synchronizer, LVL, endl )

        auto old = locked_futures[chopstick_index];
        locked_futures[chopstick_index] = nullptr;
        old->resolve(1);
    }
    else {
        assert( locked_futures[chopstick_index] == nullptr );
        DB( Synchronizer, LVL, endl )
    }
}

void get_chopstick(int philosopher_index, int chopstick_index,
        Future<int>* future_chopstick, const char* which_chopstick)
{
    DB( Synchronizer, LVL, "Philosopher=" << philosopher_index
            << ", getting chopstick=" << chopstick_index
            << ", is_chopstick_free=" << is_chopstick_free[chopstick_index]
            << ", " << which_chopstick )

    if( is_chopstick_free[chopstick_index] ) {
        is_chopstick_free[chopstick_index] = false;

        DB( Synchronizer, LVL, ", future_chopstick=" << future_chopstick << endl )
        future_chopstick->resolve(1);
    }
    else {
        assert( locked_futures[chopstick_index] == nullptr );
        locked_futures[chopstick_index] = future_chopstick;

        DB( Synchronizer, LVL, ", locked_futures="
                << locked_futures[chopstick_index] << endl )
    }
}

int philosopher(int philosopher_index, int line, int column)
{
    int first = (philosopher_index < 4)? philosopher_index : 0;
    int second = (philosopher_index < 4)? philosopher_index + 1 : 4;

#ifdef CONSOLE_MODE
    line = philosopher_index;
#endif

    for(int i = iterations; i > 0; i--)
```

```
    {
        table.submit( &show_message, "    thinking", line, column );
        Delay thinking(2000000);

        // Get the first chopstick
        Future<int>* chopstick1 = new Future<int>();
        table.submit( &get_chopstick, philosopher_index, first, chopstick1, "FIRST " );
        chopstick1->get_value();

        // Get the second chopstick
        Future<int>* chopstick2 = new Future<int>();
        table.submit( &get_chopstick, philosopher_index, second, chopstick2, "SECOND"
);
        chopstick2->get_value();

    #ifdef CONSOLE_MODE
        StringStream* stream1 = new StringStream{100};
        *stream1 << "Philosopher=" << philosopher_index
                << ", got     the first=" << first
                << " and second=" << second << " chopstick\n";
        table.submit( &show_message, stream1, line, column );
    #endif

        table.submit( &show_message, "    eating  ", line, column );
        Delay eating(1000000);

        // Release the chopsticks
        table.submit( &release_chopstick, philosopher_index, second, chopstick2,
"SECOND" );
        table.submit( &release_chopstick, philosopher_index, first, chopstick1, "FIRST
" );

    #ifdef CONSOLE_MODE
        StringStream* stream2 = new StringStream{100};
        *stream2 << "Philosopher=" << philosopher_index
                << ", release the first=" << first
                << " and second=" << second << " chopstick\n";
        table.submit( &show_message, stream2, line, column );
    #endif
    }

    table.submit( &show_message, "    done    ", line, column );
    return iterations;
}

void setup_program()
{
    Display::clear();
    Display::position(0, 0);
    cout << "The Philosopher's Dinner:" << endl;

    for(int i = 0; i < 5; i++)
    {
        is_chopstick_free[i] = true;
```

```
        locked_futures[i] = nullptr;
    }

    phil[0] = new Thread(&philosopher, 0,  5, 32);
    phil[1] = new Thread(&philosopher, 1, 10, 44);
    phil[2] = new Thread(&philosopher, 2, 16, 39);
    phil[3] = new Thread(&philosopher, 3, 16, 24);
    phil[4] = new Thread(&philosopher, 4, 10, 20);

    cout << "Philosophers are alive and hungry!" << endl;

    Display::position(7, 44);
    cout << '/';
    Display::position(13, 44);
    cout << '\\';
    Display::position(16, 35);
    cout << '|';
    Display::position(13, 27);
    cout << '/';
    Display::position(7, 27);
    cout << '\\';
    Display::position(19, 0);

    cout << "The dinner is served ..." << endl;
}

int main()
{
    table.submit( &setup_program );

    for(int i = 0; i < 5; i++) {
        int ret = phil[i]->join();
        StringStream* stream = new StringStream(100);

        *stream << "Philosopher " << i << " ate " << ret << " times\n";
        table.submit( &show_message, stream, 20 + i, 0 );
    }

    for(int i = 0; i < 5; i++)
        delete phil[i];

    cout << "The end!" << endl;
    return 0;
}
```

Resultado da execução sem o modo *CONSOLE_MODE*:

Listagem 61: Exemplo de execução de */app/guard_semaphore_test.cc*

```
The Philosopher's Dinner:
Philosophers are alive and hungry!
```

```
                                  done

                          \                /


                      done                    done

                          /                \


                           done           done


The dinner is served ...
Philosopher 0 ate 10 times
Philosopher 1 ate 10 times
Philosopher 2 ate 10 times
Philosopher 3 ate 10 times
Philosopher 4 ate 10 times
The end!
```

Resultado da execução com o modo *CONSOLE_MODE*:

Listagem 62: Exemplo de execução de */app/guard_semaphore_test.cc*

```
The Philosopher's Dinner:
Philosophers are alive and hungry!




                          \                /




                          /                \

                                    |


The dinner is served ...
    thinking(0)
    thinking(1)
    thinking(2)
    thinking(3)
    thinking(4)
Philosopher=0, getting chopstick=0, is_chopstick_free=1, FIRST ,
```

```
future_chopstick=0x00083e30
Philosopher=0, getting chopstick=1, is_chopstick_free=1, SECOND,
future_chopstick=0x00083e08
Philosopher=0, got     the first=0 and second=1 chopstick
    eating  (0)
Philosopher=1, getting chopstick=1, is_chopstick_free=0, FIRST ,
locked_futures=0x00083de0
Philosopher=2, getting chopstick=2, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083db8
Philosopher=2, getting chopstick=3, is_chopstick_free=1, SECOND,
future_chopstick=0x00083d90
Philosopher=2, got     the first=2 and second=3 chopstick
    eating  (2)
Philosopher=3, getting chopstick=3, is_chopstick_free=0, FIRST ,
locked_futures=0x00083d68
Philosopher=4, getting chopstick=0, is_chopstick_free=0, FIRST ,
locked_futures=0x00083d40
Philosopher=0, release chopstick=1, is_chopstick_free=0, future_chopstick=0x00083e08,
locked_futures=0x00083de0, SECOND
Philosopher=1, getting chopstick=2, is_chopstick_free=0, SECOND,
locked_futures=0x00083cec
Philosopher=0, release chopstick=0, is_chopstick_free=0, future_chopstick=0x00083e30,
locked_futures=0x00083d40, FIRST
Philosopher=4, getting chopstick=4, is_chopstick_free=1, SECOND,
future_chopstick=0x00083c98
Philosopher=4, got     the first=0 and second=4 chopstick
    eating  (4)
Philosopher=0, release the first=0 and second=1 chopstick
    thinking(0)
Philosopher=2, release chopstick=3, is_chopstick_free=0, future_chopstick=0x00083d90,
locked_futures=0x00083d68, SECOND
Philosopher=3, getting chopstick=4, is_chopstick_free=0, SECOND,
locked_futures=0x00083c44
Philosopher=2, release chopstick=2, is_chopstick_free=0, future_chopstick=0x00083db8,
locked_futures=0x00083cec, FIRST
Philosopher=1, got     the first=1 and second=2 chopstick
    eating  (1)
Philosopher=2, release the first=2 and second=3 chopstick
    thinking(2)
Philosopher=4, release chopstick=4, is_chopstick_free=0, future_chopstick=0x00083c98,
locked_futures=0x00083c44, SECOND
Philosopher=3, got     the first=3 and second=4 chopstick
    eating  (3)
Philosopher=4, release chopstick=0, is_chopstick_free=1, future_chopstick=0x00083d40,
locked_futures=0x00000000, FIRST
Philosopher=4, release the first=0 and second=4 chopstick
    thinking(4)
Philosopher=1, release chopstick=2, is_chopstick_free=1, future_chopstick=0x00083cec,
locked_futures=0x00000000, SECOND
Philosopher=1, release chopstick=1, is_chopstick_free=1, future_chopstick=0x00083de0,
locked_futures=0x00000000, FIRST
Philosopher=1, release the first=1 and second=2 chopstick
    thinking(1)
Philosopher=0, getting chopstick=0, is_chopstick_free=1, FIRST ,
```

```
future_chopstick=0x00083c1c
Philosopher=0, getting chopstick=1, is_chopstick_free=1, SECOND,
future_chopstick=0x00083bf4
Philosopher=0, got     the first=0 and second=1 chopstick
    eating  (0)
Philosopher=3, release chopstick=4, is_chopstick_free=1, future_chopstick=0x00083c44,
locked_futures=0x00000000, SECOND
Philosopher=3, release chopstick=3, is_chopstick_free=1, future_chopstick=0x00083d68,
locked_futures=0x00000000, FIRST
Philosopher=3, release the first=3 and second=4 chopstick
    thinking(3)
Philosopher=2, getting chopstick=2, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083bcc
Philosopher=2, getting chopstick=3, is_chopstick_free=1, SECOND,
future_chopstick=0x00083ba4
Philosopher=2, got     the first=2 and second=3 chopstick
    eating  (2)
Philosopher=4, getting chopstick=0, is_chopstick_free=0, FIRST ,
locked_futures=0x00083b7c
Philosopher=0, release chopstick=1, is_chopstick_free=0, future_chopstick=0x00083bf4,
locked_futures=0x00000000, SECOND
Philosopher=0, release chopstick=0, is_chopstick_free=0, future_chopstick=0x00083c1c,
locked_futures=0x00083b7c, FIRST
Philosopher=4, getting chopstick=4, is_chopstick_free=1, SECOND,
future_chopstick=0x00083b28
Philosopher=4, got     the first=0 and second=4 chopstick
    eating  (4)
Philosopher=0, release the first=0 and second=1 chopstick
    thinking(0)
Philosopher=1, getting chopstick=1, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083b00
Philosopher=1, getting chopstick=2, is_chopstick_free=0, SECOND,
locked_futures=0x00083ad8
Philosopher=2, release chopstick=3, is_chopstick_free=0, future_chopstick=0x00083ba4,
locked_futures=0x00000000, SECOND
Philosopher=2, release chopstick=2, is_chopstick_free=0, future_chopstick=0x00083bcc,
locked_futures=0x00083ad8, FIRST
Philosopher=1, got     the first=1 and second=2 chopstick
    eating  (1)
Philosopher=2, release the first=2 and second=3 chopstick
    thinking(2)
Philosopher=3, getting chopstick=3, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083ab0
Philosopher=3, getting chopstick=4, is_chopstick_free=0, SECOND,
locked_futures=0x00083a88
Philosopher=4, release chopstick=4, is_chopstick_free=0, future_chopstick=0x00083b28,
locked_futures=0x00083a88, SECOND
Philosopher=3, got     the first=3 and second=4 chopstick
    eating  (3)
Philosopher=4, release chopstick=0, is_chopstick_free=1, future_chopstick=0x00083b7c,
locked_futures=0x00000000, FIRST
Philosopher=4, release the first=0 and second=4 chopstick
    thinking(4)
Philosopher=1, release chopstick=2, is_chopstick_free=1, future_chopstick=0x00083ad8,
```

```
locked_futures=0x00000000, SECOND
Philosopher=1, release chopstick=1, is_chopstick_free=0, future_chopstick=0x00083b00,
locked_futures=0x00000000, FIRST
Philosopher=1, release the first=1 and second=2 chopstick
    thinking(1)
Philosopher=0, getting chopstick=0, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083a60
Philosopher=0, getting chopstick=1, is_chopstick_free=1, SECOND,
future_chopstick=0x00083a38
Philosopher=0, got     the first=0 and second=1 chopstick
    eating  (0)
Philosopher=3, release chopstick=4, is_chopstick_free=1, future_chopstick=0x00083a88,
locked_futures=0x00000000, SECOND
Philosopher=3, release chopstick=3, is_chopstick_free=0, future_chopstick=0x00083ab0,
locked_futures=0x00000000, FIRST
Philosopher=3, release the first=3 and second=4 chopstick
    thinking(3)
Philosopher=2, getting chopstick=2, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083a10
Philosopher=2, getting chopstick=3, is_chopstick_free=1, SECOND,
future_chopstick=0x000839e8
Philosopher=2, got     the first=2 and second=3 chopstick
    eating  (2)
Philosopher=0, release chopstick=1, is_chopstick_free=0, future_chopstick=0x00083a38,
locked_futures=0x00000000, SECOND
Philosopher=0, release chopstick=0, is_chopstick_free=0, future_chopstick=0x00083a60,
locked_futures=0x00000000, FIRST
Philosopher=0, release the first=0 and second=1 chopstick
    thinking(0)
Philosopher=4, getting chopstick=0, is_chopstick_free=1, FIRST ,
future_chopstick=0x000839c0
Philosopher=4, getting chopstick=4, is_chopstick_free=1, SECOND,
future_chopstick=0x00083998
Philosopher=4, got     the first=0 and second=4 chopstick
    eating  (4)
Philosopher=1, getting chopstick=1, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083970
Philosopher=1, getting chopstick=2, is_chopstick_free=0, SECOND,
locked_futures=0x00083948
Philosopher=2, release chopstick=3, is_chopstick_free=0, future_chopstick=0x000839e8,
locked_futures=0x00000000, SECOND
Philosopher=2, release chopstick=2, is_chopstick_free=0, future_chopstick=0x00083a10,
locked_futures=0x00083948, FIRST
Philosopher=1, got     the first=1 and second=2 chopstick
    eating  (1)
Philosopher=2, release the first=2 and second=3 chopstick
    thinking(2)
Philosopher=3, getting chopstick=3, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083920
Philosopher=3, getting chopstick=4, is_chopstick_free=0, SECOND,
locked_futures=0x000838f8
Philosopher=4, release chopstick=4, is_chopstick_free=0, future_chopstick=0x00083998,
locked_futures=0x000838f8, SECOND
Philosopher=3, got     the first=3 and second=4 chopstick
```

```
    eating  (3)
Philosopher=4, release chopstick=0, is_chopstick_free=0, future_chopstick=0x000839c0,
locked_futures=0x00000000, FIRST
Philosopher=4, release the first=0 and second=4 chopstick
    thinking(4)
Philosopher=1, release chopstick=2, is_chopstick_free=1, future_chopstick=0x00083948,
locked_futures=0x00000000, SECOND
Philosopher=1, release chopstick=1, is_chopstick_free=0, future_chopstick=0x00083970,
locked_futures=0x00000000, FIRST
Philosopher=1, release the first=1 and second=2 chopstick
    thinking(1)
Philosopher=0, getting chopstick=0, is_chopstick_free=1, FIRST ,
future_chopstick=0x000838d0
Philosopher=0, getting chopstick=1, is_chopstick_free=1, SECOND,
future_chopstick=0x000838a8
Philosopher=0, got     the first=0 and second=1 chopstick
    eating  (0)
Philosopher=3, release chopstick=4, is_chopstick_free=1, future_chopstick=0x000838f8,
locked_futures=0x00000000, SECOND
Philosopher=3, release chopstick=3, is_chopstick_free=0, future_chopstick=0x00083920,
locked_futures=0x00000000, FIRST
Philosopher=3, release the first=3 and second=4 chopstick
    thinking(3)
Philosopher=2, getting chopstick=2, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083880
Philosopher=2, getting chopstick=3, is_chopstick_free=1, SECOND,
future_chopstick=0x00083858
Philosopher=2, got     the first=2 and second=3 chopstick
    eating  (2)
Philosopher=0, release chopstick=1, is_chopstick_free=0, future_chopstick=0x000838a8,
locked_futures=0x00000000, SECOND
Philosopher=0, release chopstick=0, is_chopstick_free=0, future_chopstick=0x000838d0,
locked_futures=0x00000000, FIRST
Philosopher=0, release the first=0 and second=1 chopstick
    thinking(0)
Philosopher=4, getting chopstick=0, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083830
Philosopher=4, getting chopstick=4, is_chopstick_free=1, SECOND,
future_chopstick=0x00083808
Philosopher=4, got     the first=0 and second=4 chopstick
    eating  (4)
Philosopher=1, getting chopstick=1, is_chopstick_free=1, FIRST ,
future_chopstick=0x000837e0
Philosopher=1, getting chopstick=2, is_chopstick_free=0, SECOND,
locked_futures=0x000837b8
Philosopher=2, release chopstick=3, is_chopstick_free=0, future_chopstick=0x00083858,
locked_futures=0x00000000, SECOND
Philosopher=2, release chopstick=2, is_chopstick_free=0, future_chopstick=0x00083880,
locked_futures=0x000837b8, FIRST
Philosopher=1, got     the first=1 and second=2 chopstick
    eating  (1)
Philosopher=2, release the first=2 and second=3 chopstick
    thinking(2)
Philosopher=3, getting chopstick=3, is_chopstick_free=1, FIRST ,
```

```
future_chopstick=0x00083790
Philosopher=3, getting chopstick=4, is_chopstick_free=0, SECOND,
locked_futures=0x00083768
Philosopher=4, release chopstick=4, is_chopstick_free=0, future_chopstick=0x00083808,
locked_futures=0x00083768, SECOND
Philosopher=3, got     the first=3 and second=4 chopstick
    eating  (3)
Philosopher=4, release chopstick=0, is_chopstick_free=0, future_chopstick=0x00083830,
locked_futures=0x00000000, FIRST
Philosopher=4, release the first=0 and second=4 chopstick
    thinking(4)
Philosopher=1, release chopstick=2, is_chopstick_free=1, future_chopstick=0x000837b8,
locked_futures=0x00000000, SECOND
Philosopher=1, release chopstick=1, is_chopstick_free=0, future_chopstick=0x000837e0,
locked_futures=0x00000000, FIRST
Philosopher=1, release the first=1 and second=2 chopstick
    thinking(1)
Philosopher=0, getting chopstick=0, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083740
Philosopher=0, getting chopstick=1, is_chopstick_free=1, SECOND,
future_chopstick=0x00083718
Philosopher=0, got     the first=0 and second=1 chopstick
    eating  (0)
Philosopher=3, release chopstick=4, is_chopstick_free=1, future_chopstick=0x00083768,
locked_futures=0x00000000, SECOND
Philosopher=3, release chopstick=3, is_chopstick_free=0, future_chopstick=0x00083790,
locked_futures=0x00000000, FIRST
Philosopher=3, release the first=3 and second=4 chopstick
    thinking(3)
Philosopher=2, getting chopstick=2, is_chopstick_free=1, FIRST ,
future_chopstick=0x000836f0
Philosopher=2, getting chopstick=3, is_chopstick_free=1, SECOND,
future_chopstick=0x000836c8
Philosopher=2, got     the first=2 and second=3 chopstick
    eating  (2)
Philosopher=0, release chopstick=1, is_chopstick_free=0, future_chopstick=0x00083718,
locked_futures=0x00000000, SECOND
Philosopher=0, release chopstick=0, is_chopstick_free=0, future_chopstick=0x00083740,
locked_futures=0x00000000, FIRST
Philosopher=0, release the first=0 and second=1 chopstick
    thinking(0)
Philosopher=4, getting chopstick=0, is_chopstick_free=1, FIRST ,
future_chopstick=0x000836a0
Philosopher=4, getting chopstick=4, is_chopstick_free=1, SECOND,
future_chopstick=0x00083678
Philosopher=4, got     the first=0 and second=4 chopstick
    eating  (4)
Philosopher=1, getting chopstick=1, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083650
Philosopher=1, getting chopstick=2, is_chopstick_free=0, SECOND,
locked_futures=0x00083628
Philosopher=2, release chopstick=3, is_chopstick_free=0, future_chopstick=0x000836c8,
locked_futures=0x00000000, SECOND
Philosopher=2, release chopstick=2, is_chopstick_free=0, future_chopstick=0x000836f0,
```

```
locked_futures=0x00083628, FIRST
Philosopher=1, got     the first=1 and second=2 chopstick
    eating  (1)
Philosopher=2, release the first=2 and second=3 chopstick
    thinking(2)
Philosopher=3, getting chopstick=3, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083600
Philosopher=3, getting chopstick=4, is_chopstick_free=0, SECOND,
locked_futures=0x000835d8
Philosopher=4, release chopstick=4, is_chopstick_free=0, future_chopstick=0x00083678,
locked_futures=0x000835d8, SECOND
Philosopher=3, got     the first=3 and second=4 chopstick
    eating  (3)
Philosopher=4, release chopstick=0, is_chopstick_free=0, future_chopstick=0x000836a0,
locked_futures=0x00000000, FIRST
Philosopher=4, release the first=0 and second=4 chopstick
    thinking(4)
Philosopher=1, release chopstick=2, is_chopstick_free=1, future_chopstick=0x00083628,
locked_futures=0x00000000, SECOND
Philosopher=1, release chopstick=1, is_chopstick_free=0, future_chopstick=0x00083650,
locked_futures=0x00000000, FIRST
Philosopher=1, release the first=1 and second=2 chopstick
    thinking(1)
Philosopher=0, getting chopstick=0, is_chopstick_free=1, FIRST ,
future_chopstick=0x000835b0
Philosopher=0, getting chopstick=1, is_chopstick_free=1, SECOND,
future_chopstick=0x00083588
Philosopher=0, got     the first=0 and second=1 chopstick
    eating  (0)
Philosopher=3, release chopstick=4, is_chopstick_free=1, future_chopstick=0x000835d8,
locked_futures=0x00000000, SECOND
Philosopher=3, release chopstick=3, is_chopstick_free=0, future_chopstick=0x00083600,
locked_futures=0x00000000, FIRST
Philosopher=3, release the first=3 and second=4 chopstick
    thinking(3)
Philosopher=2, getting chopstick=2, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083560
Philosopher=2, getting chopstick=3, is_chopstick_free=1, SECOND,
future_chopstick=0x00083538
Philosopher=2, got     the first=2 and second=3 chopstick
    eating  (2)
Philosopher=0, release chopstick=1, is_chopstick_free=0, future_chopstick=0x00083588,
locked_futures=0x00000000, SECOND
Philosopher=0, release chopstick=0, is_chopstick_free=0, future_chopstick=0x000835b0,
locked_futures=0x00000000, FIRST
Philosopher=0, release the first=0 and second=1 chopstick
    thinking(0)
Philosopher=4, getting chopstick=0, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083510
Philosopher=4, getting chopstick=4, is_chopstick_free=1, SECOND,
future_chopstick=0x000834e8
Philosopher=4, got     the first=0 and second=4 chopstick
    eating  (4)
Philosopher=1, getting chopstick=1, is_chopstick_free=1, FIRST ,
```

```
future_chopstick=0x000834c0
Philosopher=1, getting chopstick=2, is_chopstick_free=0, SECOND,
locked_futures=0x00083498
Philosopher=2, release chopstick=3, is_chopstick_free=0, future_chopstick=0x00083538,
locked_futures=0x00000000, SECOND
Philosopher=2, release chopstick=2, is_chopstick_free=0, future_chopstick=0x00083560,
locked_futures=0x00083498, FIRST
Philosopher=1, got     the first=1 and second=2 chopstick
    eating  (1)
Philosopher=2, release the first=2 and second=3 chopstick
    thinking(2)
Philosopher=3, getting chopstick=3, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083470
Philosopher=3, getting chopstick=4, is_chopstick_free=0, SECOND,
locked_futures=0x00083448
Philosopher=4, release chopstick=4, is_chopstick_free=0, future_chopstick=0x000834e8,
locked_futures=0x00083448, SECOND
Philosopher=3, got     the first=3 and second=4 chopstick
    eating  (3)
Philosopher=4, release chopstick=0, is_chopstick_free=0, future_chopstick=0x00083510,
locked_futures=0x00000000, FIRST
Philosopher=4, release the first=0 and second=4 chopstick
    thinking(4)
Philosopher=1, release chopstick=2, is_chopstick_free=1, future_chopstick=0x00083498,
locked_futures=0x00000000, SECOND
Philosopher=1, release chopstick=1, is_chopstick_free=0, future_chopstick=0x000834c0,
locked_futures=0x00000000, FIRST
Philosopher=1, release the first=1 and second=2 chopstick
    thinking(1)
Philosopher=0, getting chopstick=0, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083420
Philosopher=0, getting chopstick=1, is_chopstick_free=1, SECOND,
future_chopstick=0x000833f8
Philosopher=0, got     the first=0 and second=1 chopstick
    eating  (0)
Philosopher=3, release chopstick=4, is_chopstick_free=1, future_chopstick=0x00083448,
locked_futures=0x00000000, SECOND
Philosopher=3, release chopstick=3, is_chopstick_free=0, future_chopstick=0x00083470,
locked_futures=0x00000000, FIRST
Philosopher=3, release the first=3 and second=4 chopstick
    thinking(3)
Philosopher=2, getting chopstick=2, is_chopstick_free=1, FIRST ,
future_chopstick=0x000833d0
Philosopher=2, getting chopstick=3, is_chopstick_free=1, SECOND,
future_chopstick=0x000833a8
Philosopher=2, got     the first=2 and second=3 chopstick
    eating  (2)
Philosopher=0, release chopstick=1, is_chopstick_free=0, future_chopstick=0x000833f8,
locked_futures=0x00000000, SECOND
Philosopher=0, release chopstick=0, is_chopstick_free=0, future_chopstick=0x00083420,
locked_futures=0x00000000, FIRST
Philosopher=0, release the first=0 and second=1 chopstick
    thinking(0)
Philosopher=4, getting chopstick=0, is_chopstick_free=1, FIRST ,
```

```
future_chopstick=0x00083380
Philosopher=4, getting chopstick=4, is_chopstick_free=1, SECOND,
future_chopstick=0x00083358
Philosopher=4, got     the first=0 and second=4 chopstick
    eating  (4)
Philosopher=1, getting chopstick=1, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083330
Philosopher=1, getting chopstick=2, is_chopstick_free=0, SECOND,
locked_futures=0x00083308
Philosopher=2, release chopstick=3, is_chopstick_free=0, future_chopstick=0x000833a8,
locked_futures=0x00000000, SECOND
Philosopher=2, release chopstick=2, is_chopstick_free=0, future_chopstick=0x000833d0,
locked_futures=0x00083308, FIRST
Philosopher=1, got     the first=1 and second=2 chopstick
    eating  (1)
Philosopher=2, release the first=2 and second=3 chopstick
    thinking(2)
Philosopher=3, getting chopstick=3, is_chopstick_free=1, FIRST ,
future_chopstick=0x000832e0
Philosopher=3, getting chopstick=4, is_chopstick_free=0, SECOND,
locked_futures=0x000832b8
Philosopher=4, release chopstick=4, is_chopstick_free=0, future_chopstick=0x00083358,
locked_futures=0x000832b8, SECOND
Philosopher=3, got     the first=3 and second=4 chopstick
    eating  (3)
Philosopher=4, release chopstick=0, is_chopstick_free=0, future_chopstick=0x00083380,
locked_futures=0x00000000, FIRST
Philosopher=4, release the first=0 and second=4 chopstick
    thinking(4)
Philosopher=1, release chopstick=2, is_chopstick_free=1, future_chopstick=0x00083308,
locked_futures=0x00000000, SECOND
Philosopher=1, release chopstick=1, is_chopstick_free=0, future_chopstick=0x00083330,
locked_futures=0x00000000, FIRST
Philosopher=1, release the first=1 and second=2 chopstick
    thinking(1)
Philosopher=0, getting chopstick=0, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083290
Philosopher=0, getting chopstick=1, is_chopstick_free=1, SECOND,
future_chopstick=0x00083268
Philosopher=0, got     the first=0 and second=1 chopstick
    eating  (0)
Philosopher=3, release chopstick=4, is_chopstick_free=1, future_chopstick=0x000832b8,
locked_futures=0x00000000, SECOND
Philosopher=3, release chopstick=3, is_chopstick_free=0, future_chopstick=0x000832e0,
locked_futures=0x00000000, FIRST
Philosopher=3, release the first=3 and second=4 chopstick
    thinking(3)
Philosopher=2, getting chopstick=2, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083240
Philosopher=2, getting chopstick=3, is_chopstick_free=1, SECOND,
future_chopstick=0x00083218
Philosopher=2, got     the first=2 and second=3 chopstick
    eating  (2)
Philosopher=0, release chopstick=1, is_chopstick_free=0, future_chopstick=0x00083268,
```

```
locked_futures=0x00000000, SECOND
Philosopher=0, release chopstick=0, is_chopstick_free=0, future_chopstick=0x00083290,
locked_futures=0x00000000, FIRST
Philosopher=0, release the first=0 and second=1 chopstick
    thinking(0)
Philosopher=4, getting chopstick=0, is_chopstick_free=1, FIRST ,
future_chopstick=0x000831f0
Philosopher=4, getting chopstick=4, is_chopstick_free=1, SECOND,
future_chopstick=0x000831c8
Philosopher=4, got     the first=0 and second=4 chopstick
    eating  (4)
Philosopher=1, getting chopstick=1, is_chopstick_free=1, FIRST ,
future_chopstick=0x000831a0
Philosopher=1, getting chopstick=2, is_chopstick_free=0, SECOND,
locked_futures=0x00083178
Philosopher=2, release chopstick=3, is_chopstick_free=0, future_chopstick=0x00083218,
locked_futures=0x00000000, SECOND
Philosopher=2, release chopstick=2, is_chopstick_free=0, future_chopstick=0x00083240,
locked_futures=0x00083178, FIRST
Philosopher=1, got     the first=1 and second=2 chopstick
    eating  (1)
Philosopher=2, release the first=2 and second=3 chopstick
    thinking(2)
Philosopher=3, getting chopstick=3, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083150
Philosopher=3, getting chopstick=4, is_chopstick_free=0, SECOND,
locked_futures=0x00083128
Philosopher=4, release chopstick=4, is_chopstick_free=0, future_chopstick=0x000831c8,
locked_futures=0x00083128, SECOND
Philosopher=3, got     the first=3 and second=4 chopstick
    eating  (3)
Philosopher=4, release chopstick=0, is_chopstick_free=0, future_chopstick=0x000831f0,
locked_futures=0x00000000, FIRST
Philosopher=4, release the first=0 and second=4 chopstick
    thinking(4)
Philosopher=1, release chopstick=2, is_chopstick_free=1, future_chopstick=0x00083178,
locked_futures=0x00000000, SECOND
Philosopher=1, release chopstick=1, is_chopstick_free=0, future_chopstick=0x000831a0,
locked_futures=0x00000000, FIRST
Philosopher=1, release the first=1 and second=2 chopstick
    thinking(1)
Philosopher=0, getting chopstick=0, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083100
Philosopher=0, getting chopstick=1, is_chopstick_free=1, SECOND,
future_chopstick=0x000830d8
Philosopher=0, got     the first=0 and second=1 chopstick
    eating  (0)
Philosopher=3, release chopstick=4, is_chopstick_free=1, future_chopstick=0x00083128,
locked_futures=0x00000000, SECOND
Philosopher=3, release chopstick=3, is_chopstick_free=0, future_chopstick=0x00083150,
locked_futures=0x00000000, FIRST
Philosopher=3, release the first=3 and second=4 chopstick
    thinking(3)
Philosopher=2, getting chopstick=2, is_chopstick_free=1, FIRST ,
```

```
future_chopstick=0x000830b0
Philosopher=2, getting chopstick=3, is_chopstick_free=1, SECOND,
future_chopstick=0x00083088
Philosopher=2, got     the first=2 and second=3 chopstick
    eating  (2)
Philosopher=0, release chopstick=1, is_chopstick_free=0, future_chopstick=0x000830d8,
locked_futures=0x00000000, SECOND
Philosopher=0, release chopstick=0, is_chopstick_free=0, future_chopstick=0x00083100,
locked_futures=0x00000000, FIRST
Philosopher=0, release the first=0 and second=1 chopstick
    thinking(0)
Philosopher=4, getting chopstick=0, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083060
Philosopher=4, getting chopstick=4, is_chopstick_free=1, SECOND,
future_chopstick=0x00083038
Philosopher=4, got     the first=0 and second=4 chopstick
    eating  (4)
Philosopher=1, getting chopstick=1, is_chopstick_free=1, FIRST ,
future_chopstick=0x00083010
Philosopher=1, getting chopstick=2, is_chopstick_free=0, SECOND,
locked_futures=0x00082fe8
Philosopher=2, release chopstick=3, is_chopstick_free=0, future_chopstick=0x00083088,
locked_futures=0x00000000, SECOND
Philosopher=2, release chopstick=2, is_chopstick_free=0, future_chopstick=0x000830b0,
locked_futures=0x00082fe8, FIRST
Philosopher=1, got     the first=1 and second=2 chopstick
    eating  (1)
Philosopher=2, release the first=2 and second=3 chopstick
    thinking(2)
Philosopher=3, getting chopstick=3, is_chopstick_free=1, FIRST ,
future_chopstick=0x00082fc0
Philosopher=3, getting chopstick=4, is_chopstick_free=0, SECOND,
locked_futures=0x00082f98
Philosopher=4, release chopstick=4, is_chopstick_free=0, future_chopstick=0x00083038,
locked_futures=0x00082f98, SECOND
Philosopher=3, got     the first=3 and second=4 chopstick
    eating  (3)
Philosopher=4, release chopstick=0, is_chopstick_free=0, future_chopstick=0x00083060,
locked_futures=0x00000000, FIRST
Philosopher=4, release the first=0 and second=4 chopstick
    thinking(4)
Philosopher=1, release chopstick=2, is_chopstick_free=1, future_chopstick=0x00082fe8,
locked_futures=0x00000000, SECOND
Philosopher=1, release chopstick=1, is_chopstick_free=0, future_chopstick=0x00083010,
locked_futures=0x00000000, FIRST
Philosopher=1, release the first=1 and second=2 chopstick
    thinking(1)
Philosopher=0, getting chopstick=0, is_chopstick_free=1, FIRST ,
future_chopstick=0x00082f70
Philosopher=0, getting chopstick=1, is_chopstick_free=1, SECOND,
future_chopstick=0x00082f48
Philosopher=0, got     the first=0 and second=1 chopstick
    eating  (0)
Philosopher=3, release chopstick=4, is_chopstick_free=1, future_chopstick=0x00082f98,
```

```
locked_futures=0x00000000, SECOND
Philosopher=3, release chopstick=3, is_chopstick_free=0, future_chopstick=0x00082fc0,
locked_futures=0x00000000, FIRST
Philosopher=3, release the first=3 and second=4 chopstick
    thinking(3)
Philosopher=2, getting chopstick=2, is_chopstick_free=1, FIRST ,
future_chopstick=0x00082f20
Philosopher=2, getting chopstick=3, is_chopstick_free=1, SECOND,
future_chopstick=0x00082ef8
Philosopher=2, got     the first=2 and second=3 chopstick
    eating  (2)
Philosopher=0, release chopstick=1, is_chopstick_free=0, future_chopstick=0x00082f48,
locked_futures=0x00000000, SECOND
Philosopher=0, release chopstick=0, is_chopstick_free=0, future_chopstick=0x00082f70,
locked_futures=0x00000000, FIRST
Philosopher=0, release the first=0 and second=1 chopstick
    done    (0)
Philosopher 0 ate 10 times
Philosopher=4, getting chopstick=0, is_chopstick_free=1, FIRST ,
future_chopstick=0x00082ed0
Philosopher=4, getting chopstick=4, is_chopstick_free=1, SECOND,
future_chopstick=0x00082ea8
Philosopher=4, got     the first=0 and second=4 chopstick
    eating  (4)
Philosopher=1, getting chopstick=1, is_chopstick_free=1, FIRST ,
future_chopstick=0x00082e80
Philosopher=1, getting chopstick=2, is_chopstick_free=0, SECOND,
locked_futures=0x00082e58
Philosopher=2, release chopstick=3, is_chopstick_free=0, future_chopstick=0x00082ef8,
locked_futures=0x00000000, SECOND
Philosopher=2, release chopstick=2, is_chopstick_free=0, future_chopstick=0x00082f20,
locked_futures=0x00082e58, FIRST
Philosopher=1, got     the first=1 and second=2 chopstick
    eating  (1)
Philosopher=2, release the first=2 and second=3 chopstick
    done    (2)
Philosopher=3, getting chopstick=3, is_chopstick_free=1, FIRST ,
future_chopstick=0x00082e30
Philosopher=3, getting chopstick=4, is_chopstick_free=0, SECOND,
locked_futures=0x00082e08
Philosopher=4, release chopstick=4, is_chopstick_free=0, future_chopstick=0x00082ea8,
locked_futures=0x00082e08, SECOND
Philosopher=3, got     the first=3 and second=4 chopstick
    eating  (3)
Philosopher=4, release chopstick=0, is_chopstick_free=0, future_chopstick=0x00082ed0,
locked_futures=0x00000000, FIRST
Philosopher=4, release the first=0 and second=4 chopstick
    done    (4)
Philosopher=1, release chopstick=2, is_chopstick_free=1, future_chopstick=0x00082e58,
locked_futures=0x00000000, SECOND
Philosopher=1, release chopstick=1, is_chopstick_free=0, future_chopstick=0x00082e80,
locked_futures=0x00000000, FIRST
Philosopher=1, release the first=1 and second=2 chopstick
    done    (1)
```

```
Philosopher 1 ate 10 times
Philosopher 2 ate 10 times
Philosopher=3, release chopstick=4, is_chopstick_free=1, future_chopstick=0x00082e08,
locked_futures=0x00000000, SECOND
Philosopher=3, release chopstick=3, is_chopstick_free=0, future_chopstick=0x00082e30,
locked_futures=0x00000000, FIRST
Philosopher=3, release the first=3 and second=4 chopstick
    done    (3)
Philosopher 3 ate 10 times
Philosopher 4 ate 10 times
The end!
```

guard_scheduler_cpu_affinity_test.cc ←

Trata-se do mesmo programa original do EPOS chamada *scheduler_cpu_affinity_test.cc*, também conhecido como *Concurrent Philosophers Dinner*. Nesta nova versão utiliza um *Guarda* e várias variáveis *Future*, ao contrário de utilizar alguns semáforos para sincronização.

Listagem 63: Arquivo */app/guard_scheduler_cpu_affinity_test.cc*

```
// EPOS Scheduler Test Program
// #define DEBUG_SYNC

#include <utility/ostream.h>
#include <utility/stringstream.h>
#include <utility/guard.h>
#include <utility/future.h>
#include <machine.h>
#include <display.h>
#include <thread.h>

using namespace EPOS;

#if !defined(nullptr)
    #define nullptr 0
#endif

// #define CONSOLE_MODE
const int iterations = 10;
OStream cout;

Guard table;
Thread * phil[5];

bool is_chopstick_free[5];
Future<int>* locked_futures[5];

int philosopher(int n, int l, int c);
void think(unsigned long long n);
void eat(unsigned long long n);
unsigned long long busy_wait(unsigned long long n);
```

```
// Fix `error: undefined reference to endl` when compiling with hysterically_debugged = true
EPOS::S::U::OStream::Endl EPOS::S::U::endl;

#ifdef CONSOLE_MODE
    #define LVL WRN
    #define EXTRA(arg) arg
#else
    #define LVL FFF
    #define EXTRA(arg)
#endif

void show_message(const char * message, int line, int column) {
#ifdef CONSOLE_MODE
    DB( Synchronizer, LVL, message << "(" << line << ")" << endl )
#else
    Display::position( line, column );
    cout << message;
#endif
}

void show_message(StringStream * message, int line, int column) {
#ifdef CONSOLE_MODE
    DB( Synchronizer, LVL, message )
#else
    Display::position( line, column );
    cout << message;
#endif
    delete message;
}

void setup_program()
{
    Display::clear();
    Display::position(0, 0);
    cout << "The Philosopher's Dinner:" << endl;

    for(int i = 0; i < 5; i++)
    {
        is_chopstick_free[i] = true;
        locked_futures[i] = nullptr;
    }

    phil[0] = new Thread(&philosopher, 0,  5, 30);
    phil[1] = new Thread(&philosopher, 1, 10, 44);
    phil[2] = new Thread(&philosopher, 2, 16, 39);
    phil[3] = new Thread(&philosopher, 3, 16, 21);
    phil[4] = new Thread(&philosopher, 4, 10, 17);

    cout << "Philosophers are alive and hungry!" << endl;

    Display::position(7, 44);
    cout << '/';
    Display::position(13, 44);
```

```
    cout << '\\';
    Display::position(16, 35);
    cout << '|';
    Display::position(13, 27);
    cout << '/';
    Display::position(7, 27);
    cout << '\\';
    Display::position(19, 0);

    cout << "The dinner is served ..." << endl;
}

// Regular expression for plugin: https://github.com/evandroforks/HighlightWords
// /\) => (\dx[a-f\d]{8,8})/ /Philosopher=\d/ \bGuard::vouch\b \bGuard::clear\b
int main()
{
    table.submit( &setup_program );

    for(int i = 0; i < 5; i++) {
        int ret = phil[i]->join();
        StringStream* stream = new StringStream{100};

        *stream << "Philosopher " << i << " ate " << ret << " times\n";
        table.submit( &show_message, stream, 20 + i, 0 );
    }

    for(int i = 0; i < 5; i++)
        delete phil[i];

    cout << "The end!" << endl;
    return 0;
}

void release_chopstick(int philosopher_index, int chopstick_index,
        Future<int>* future_chopstick, const char* which_chopstick)
{
    DB( Synchronizer, LVL, "Philosopher=" << philosopher_index
            << ", release chopstick=" << chopstick_index
            << ", is_chopstick_free=" << is_chopstick_free[chopstick_index]
            << ", future_chopstick=" << future_chopstick
            << ", locked_futures=" << locked_futures[chopstick_index]
            << ", " << which_chopstick )

    is_chopstick_free[chopstick_index] = true;
    delete future_chopstick;

    if( locked_futures[chopstick_index] ) {
        DB( Synchronizer, LVL, endl )

        auto old = locked_futures[chopstick_index];
        locked_futures[chopstick_index] = nullptr;
        old->resolve(1);
    }
    else {
```

```
        assert( locked_futures[chopstick_index] == nullptr );
        DB( Synchronizer, LVL, endl )
    }
}

void get_chopstick(int philosopher_index, int chopstick_index,
        Future<int>* future_chopstick, const char* which_chopstick)
{
    DB( Synchronizer, LVL, "Philosopher=" << philosopher_index
            << ", getting chopstick=" << chopstick_index
            << ", is_chopstick_free=" << is_chopstick_free[chopstick_index]
            << ", " << which_chopstick )

    if( is_chopstick_free[chopstick_index] ) {
        is_chopstick_free[chopstick_index] = false;

        DB( Synchronizer, LVL, ", future_chopstick=" << future_chopstick << endl )
        future_chopstick->resolve(1);
    }
    else {
        assert( locked_futures[chopstick_index] == nullptr );
        locked_futures[chopstick_index] = future_chopstick;

        DB( Synchronizer, LVL, ", locked_futures="
                << locked_futures[chopstick_index] << endl )
    }
}

int philosopher(int philosopher_index, int line, int column)
{
    int first = (philosopher_index < 4)? philosopher_index : 0;
    int second = (philosopher_index < 4)? philosopher_index + 1 : 4;

#ifdef CONSOLE_MODE
    line = philosopher_index;
#endif

    for(int i = iterations; i > 0; i--) {
        StringStream* stream1 = new StringStream{101};
        *stream1 << "thinking[" << Machine::cpu_id() << "]"
                EXTRA( << " (" << line << ")" << "\n" );
        table.submit( &show_message, stream1, line, column );
        think(1000000);

        StringStream* stream2 = new StringStream{102};
        *stream2 << "  hungry[" << Machine::cpu_id() << "]"
                EXTRA( << " (" << line << ")" << "\n" );
        table.submit( &show_message, stream2, line, column );

        // Get the first chopstick
        Future<int>* chopstick1 = new Future<int>();
        table.submit( &get_chopstick, philosopher_index, first, chopstick1, "FIRST " );
        chopstick1->get_value();
```

```
        // Get the second chopstick
        Future<int>* chopstick2 = new Future<int>();
        table.submit( &get_chopstick, philosopher_index, second, chopstick2, "SECOND"
);
        chopstick2->get_value();

    #ifdef CONSOLE_MODE
        StringStream* stream6 = new StringStream{103};
        *stream6 << "Philosopher=" << philosopher_index
                << ", got     the first=" << first
                << " and second=" << second << " chopstick" << "\n";
        table.submit( &show_message, stream6, line, column );
    #endif

        StringStream* stream3 = new StringStream{104};
        *stream3 << " eating[" << Machine::cpu_id() << "] " EXTRA( << "\n" );
        table.submit( &show_message, stream3, line, column );

        eat(500000);

        StringStream* stream4 = new StringStream{105};
        *stream4 << "    ate[" << Machine::cpu_id() << "]"
                EXTRA( << " (" << line << ")" << "\n" );
        table.submit( &show_message, stream4, line, column );

        // Release the chopsticks
        table.submit( &release_chopstick, philosopher_index, first, chopstick1, "FIRST
" );
        table.submit( &release_chopstick, philosopher_index, second, chopstick2,
"SECOND" );

    #ifdef CONSOLE_MODE
        StringStream* stream7 = new StringStream{106};
        *stream7 << "Philosopher=" << philosopher_index
                << ", release the first=" << first
                << " and second=" << second << " chopstick" << "\n";
        table.submit( &show_message, stream7, line, column );
    #endif
    }

    StringStream* stream5 = new StringStream{107};
    *stream5 << "  done[" << Machine::cpu_id() << "]  "
            EXTRA( << " (" << line << ")" << "\n" );
    table.submit( &show_message, stream5, line, column );

    return iterations;
}

void eat(unsigned long long n) {
    static unsigned long long v;
    v = busy_wait(n);
}

void think(unsigned long long n) {
```

```
    static unsigned long long v;
    v = busy_wait(n);
}

unsigned long long busy_wait(unsigned long long n)
{
    volatile unsigned long long v;
    for(long long int j = 0; j < 20 * n; j++)
        v &= 2 ^ j;
    return v;
}
```

Esta versão com o algoritmo do *Guarda* apresenta inúmeros problemas e não consegue executar até o fim. Apesar de sua implementação ser basicamente idêntica a implementação de *guard_semaphore_test.cc*, a maioria das vezes, o programa entra em trava, no momento que o método *lock()* da classe *Thread* tenta obter o spin lock.

Anteriormente, EPOS já apresentou problemas na execução da versão original do programa de teste *scheduler_cpu_affinity_test.cc* quando as asserções foram ativadas no arquivo *config.h*. Veja a seção Assert. Depois de se ter problemas com o *scheduler_cpu_affinity_test.cc*, as mudanças lá propostas foram retiras, restaurando o código do EPOS para sua versão original, sem asserções ou outras modificações, entretanto, esse novo programa de teste que faz uso do Guarda, continua apresentado os mesmos erros.

A seguir vê-sê um exemplo de problema de alocação de memória que pode acontece durante a execução da aplicação. Neste exemplo de problema, ocorre uma sobre-escrita de uma memória de um objeto na heap por um objeto na stack de outro objeto na heap.

1. Na linha 1088 ocorre a destruição de um objeto
2. Na linha 1090 vê-se que foi liberado o endereço de memória 0x0007fd65
3. Na linha 1101 vê-se que o endereço liberado anteriormente é alocado para um novo objeto do tipo *StringStream*
4. Entretanto na linha 1144 vê-se que o endereço do objeto 0x0007fd65 foi colocado como endereço de um objeto _link do tipo *List_Elements::Singly_Linked<Critical_Section_Base>*
5. Assim, mais tarde na linha 1248, durante a execução do objeto *StringStream*, vemos que foi mostrado lixo na tela: *9ý*. Mas o correto deveria ter sido imprimir a mensagem *thinking1 (4)*.
6. Também nota-se que durante a chamada do destrutor do objeto *StringStream*, linha 1248, foram mostrados valores de lixo: *~StringStream(this=0x0007fd65, _buffer=0x0007fd3d, _buffer_size=523601), _last_position=24624)*. O correto deveria ser *~StringStream(this=0x0007fd65, _buffer=não conhecido, _buffer_size=102), _last_position=16)*.
7. Log com a execução completa: https://gist.github.com/evandrocoan/52869eb0dae7ec3b40525628ff2be276

Listagem 64: Trecho do log de execução de */app/guard_scheduler_cpu_affinity_test.cc*

```
1088: <1>: ~Closure(this=0x0007fd85, _entry=0x00006030, PARAMETERS_COUNT=3,
PARAMETERS_LENGTH=12, sizeof=8) :<1>
1089: <1>: Heap::free(this=0x004000b4,ptr=0x0007fd65,bytes=16) :<1>
1090: <1>: Grouping_List::insert_merging(e=0x0007fd65) :<1>
```

```
1091: <1>: ~Critical_Section_Base(this=0x0007fd79 _link=0x0007fd7d) :<1>
1092: <1>: Heap::free(this=0x004000b4,ptr=0x0007fd75,bytes=24) :<1>
1093: <1>: Grouping_List::insert_merging(e=0x0007fd75) :<1>
1094: <1>: List::remove(e=0x0007fd8d) => {p=0x00093f24,o=0x0007fd8d,n=0x00000000} :<1>
1095: <1>: List[0x004000b4]::head=0x0001d000 => {p=0x00000000,o=0x0001d<0>:
Spin::acquire[SPIN=0x004000c8,ID=0x0008febc]() => {owner=589500,level=1} :<0>
1096: <0>: Heap::alloc(this=0x004000b4,bytes=16Grouping_List::search_decrementing(s=20)
:<0>
1097: <0>: List[0x004000b4]::head=0x0001d000 =>
{p=0x00000000,o=0x0001d000,n=0x00100000} :<0>
1098: <0>: List[0x004000b4]::tail=0x0007fd8d =>
{p=0x00093f24,o=0x0007fd8d,n=0x00000000} :<0>
1099: <0>: ) => 0x0007fd65 :<0>
1100: <0>: Spin::release[SPIN=0x004000c8,ID=589500]() => {owner=0,level=0} :<0>
1101: <0>: StringStream(_buffer_size=102) => 0x0007fd65 :<0>
1102: <0>: Spin::acquire[SPIN=0x004000c8,ID=0x0008febc]() => {owner=589500,level=1}
:<0>
1103: <0>:
Heap::alloc(this=0x004000b4,bytes=103Grouping_List::search_decrementing(s=107) :<0>
1104: <0>: List[0x004000b4]::head=0x0001d000 =>
{p=0x00000000,o=0x0001d000,n=0x00100000} :<0>
1105: <0>: List[0x004000b4]::tail=0x0007fd8d =>
{p=0x00093f24,o=0x0007fd8d,n=0x00000000} :<0>
1106: <0>: ) => 0x0007fcfa :<0>
1107: <0>: Spin::release[SPIN=0x004000c8,ID=589500]() => {owner=0,level=0} :<0>
1108: <0>: StringStream::print(this=0x0007fd65), string=  hungry[, string_size=9,
total_size=9, _last_position=9, _buffer=  hungry[ :<0>
1109: <0>: StringStream::print(this=0x0007fd65), string=0, string_size=1,
total_size=10, _last_position=10, _buffer=  hungry[0 :<0>
1110: <0>: StringStream::print(this=0x0007fd65), 000,n=0x00100000} :<1>
1111: <1>: List[0x004000b4]::tail=0x0007fd8d =>
{p=0x00093f24,o=0x0007fd8d,n=0x00000000} :<1>
1112: <1>: List::remove_tail() :<1>
1113: <1>: List[0x004000b4]::head=0x0001d000 =>
{p=0x00000000,o=0x0001d000,n=0x00100000} :<1>
1114: <1>: List[0x004000b4]::tail=0x0007fd8d =>
{p=0x00093f24,o=0x0007fd8d,n=0x00000000} :<1>
1115: <1>: List[0x004000b4]::head=0x0001d000 =>
{p=0x00000000,o=0x0001d000,n=0x00100000} :<1>
1116: <1>: List[0x004000b4]::tail=0x00093f24 =>
{p=0x00401000,o=0x00093f24,n=0x00000000} :<1>
1117: <1>: List[0x004000b4]::head=0x0001d000 =>
{p=0x00000000,o=0x0001d000,n=0x00100000} :<1>
1118: <1>: List[0x004000b4]::tail=0x00093f24 =>
{p=0x00401000,o=0x00093f24,n=0x00000000} :<1>
1119: <1>: Spin::acquire[SPIN=0x004000c8,ID=0x0008be84]() => {owner=573060,level=1}
:<1>
1120: <1>: Heap::alloc(this=0x004000b4,bytes=20Grouping_List::search_decrementing(s=24)
:<1>
1121: <1>: List[0x004000b4]::head=0x0001d000 =>
{p=0x00000000,o=0x0001d000,n=0x00100000} :<1>
1122: <1>: List[0x004000b4]::tail=0x00093f24 =>
{p=0x00401000,o=0x00093f24,n=0x00000000} :<1>
1123: <1>: ) => 0x0007fd79 :<1>
```

```
1124: <1>: Spin::release[SPIN=0x004000c8,ID=573060]() => {owner=0,level=0} :<1>
1125: <1>: Condition(<0>: Alarm::lock(size=0) :<0>
1126: <0>: Thread::lock(this=0x0008febc, level=0, owner=0) :<0>
1127: <0>: Spin::acquire[SPIN=0x00400120,ID=0x0008febc]() => {owner=589500,level=1}
:<0>
1128: <0>: Alarm::unlock(size=0) :<0>
1129: <0>: Thread::unlock(this=0x0008febc, level=1, owner=589500) :<0>
1130: <0>: Spin::release[SPIN=0x00400120,ID=589500]() => {owner=0,level=0} :<0>
1131: string=], <0>: string_size=1, total_size=11, _last_position=11, _buffer=
hungry[0] :<0>
1132: <0>: StringStream::print(this=0x0007fd65), string= (, string_size=2,
total_size=13, _last_position=13, _buffer=  hungry[0] ( :<0>
1133: <0>: StringStream::print(this=0x0007fd65), string=1, string_size=1,
total_size=14, _last_position=14, _buffer=  hungry[0] (1 :<0>
1134: <0>: StringStream::print(this=0x0007fd65), string=), string_size=1,
total_size=15, _last_position=15, _buffer=  hungry[0] (1) :<0>
1135: <0>: StringStream::print(this=0x0007fd65), string=
1136: , string_size=1, total_size=16, _last_position=16, _buffer=  hungry[0] (1)
1137:  :<0>
1138: <0>: Spin::acquire[SPIN=0x004000c8,ID=0x0008febc]() => {owner=589500,level=1}
:<0>
1139: <0>: Heap::alloc(this=0x004000b4,bytes=20Grouping_List::search_decrementing(s=24)
:<0>
1140: <0>: List[0x004000b4]::head=0x0001d000 =>
{p=0x00000000,o=0x0001d000,n=0x00100000} :<0>
1141: <0>: List[0x004000b4]::tail=0x00093f24 =>
{p=0x00401000,o=0x00093f24,n=0x00000000} :<0>
1142: <0>: ) => 0x0007fd61 :<0>
1143: <0>: Spin::release[SPIN=0x004000c8,ID=589500]() => {owner=0,level=0} :<0>
1144: <0>: Critical_Section_Base(_link=0x0007fd65) => 0x0007fd61 :<0>
1145: <0>: Closure(_entry=0x00006030, PARAMETERS_COUNT=3, PARAMETERS_LENGTH=12,
sizeof=8) => 0x0007fd6d :<0>
1146: <0>: Spin::acquire[SPIN=0x004000c8,ID=0x0008febc]() => {owner=589500,level=1}
:<0>
1147: <0>: Heap::alloc(this=0x004000b4,bytes=12Grouping_List::search_decrementing(s=16)
:<0>
1148: <0>: List[0x004000b4]::head=0x0001d000 =>
{p=0x00000000,o=0x0001d000,n=0x00100000} :<0>
1149: <0>: List[0x004000b4]::tail=0x00093f24 =>
{p=0x00401000,o=0x00093f24,n=0x00000000} :<0>
1150: <0>: ) => 0x0007fd51 :<0>
1151: <0>: Spin::release[SPIN=0x004000c8,ID=589500]() => {owner=0,level=0} :<0>
1152: <0>: Closure::pack_helper(Head=4, address=0x0007fd51) :<0>
1153: <0>: Closure::pack_helper(Head=4, address=0x0007fd55) :<0>
1154: <0>: Closure::pack_helper(Head=4, address=0x0007fd59) :<0>
1155: <0>: Critical_Section(_entry=0x00006030) => 0x0007fd61 :<0>
1156: <0>: Guard::vouch(this=0x00400050 head=0x00000000 tail=0x00000000
item=0x0007fd65, size=1) => 0x0007fd7d :<1>
1157: <1>: Future(_is_resolved=0, _condition=0) => 0x0007fd79 :<1>
```

Este programa foi compilado com o mode *hysterically_debugged* = *true*. Mas para que ele pudesse funcionar foi adicionado em seu trais a redução da frequência do alarme de usuário para *static const int FREQUENCY = 10; // Hz* por que foram adicionados inúmeros novas mensagens de log por todo código do

EPOS, incluindo no tratador de interrupções do alarme, o que aumentou simplificativamente o tempo em que ele demora para executar, assim causando com que novas interrupções do timer acontecessem antes que o timer tivesse terminado de imprimir a mensagem atual, causando com que o EPOS morresse após várias interrupções encadeadas.

A seguir, vê-se outro problema encontrado durante outra execução de *guard_scheduler_cpu_affinity_test.cc*. Na linha 8010 da imagem seguir, encontra-se a liberação de um spin lock que tem *level* resultante igual a 0 (zero) e que a thread que era sua dona possui o endereço *0x0008be84*. Entretanto, magicamente na linha 8148, vê-se que ele algora possui um valor *level* de 1 (um) e que sua thread dona continua sendo a thread *0x0008be84* que acabou de ter liberado ele. Analisando mais atentamente, percebe-se que tratam-se de objetos de spin lock diferentes, ambos operados pela thread *0x0008be84*. Mais tarde, o sistema trava durante a aquisição de outro spin lock no método da classe *Thread::lock()*.

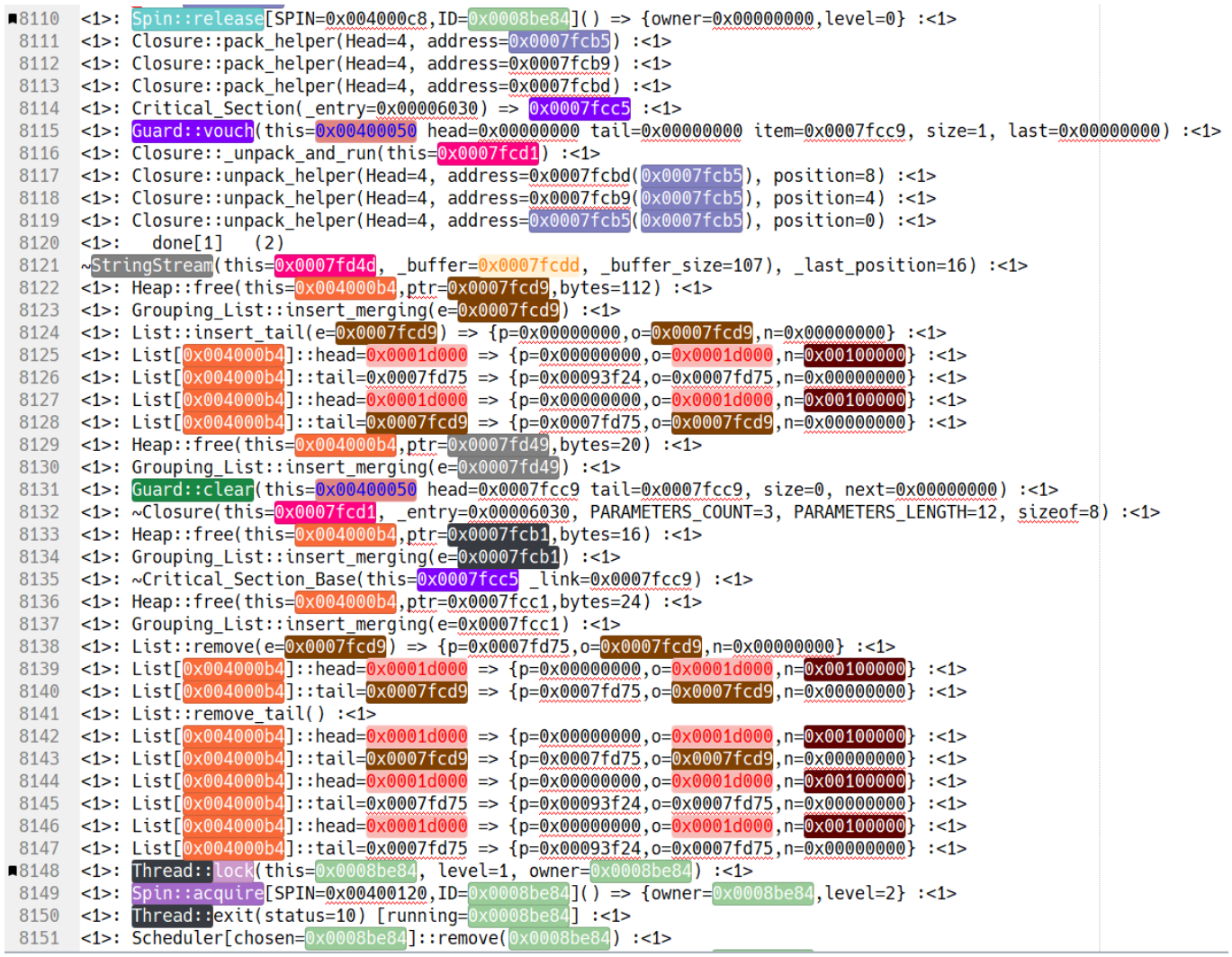

Figura 12: Aquisição de spin locks

Fonte:: Própria

guard_scheduler_cpu_affinity.cc (Linux) ←

Devido a impossibilidade de conseguir depurar e fazer a aplicação *guard_scheduler_cpu_affinity_test.cc* funcionar no EPOS com multicores, foi então realizada toda a implementação dos algoritmos do Guarda e seus mecanismos auxiliares em um Linux nativo. Para encontrar a implementação completa do algoritmo do Guarda e seus mecanismos auxiliares, veja a seção Guarda para Linux.

A seguir encontrar-e o mesmo programa original implementado na seção anterior (guard_scheduler_cpu_affinity_test.cc) chamado de *guard_scheduler_cpu_affinity_test.cc*. Nesta nova versão utiliza a implementação da *Guarda* e seus auxiliares para Linux, junto com as versões equivalentes das demais bibliotecas como *std::thread*, *std::condition_variable* e *std::cout*.

Para compilar esse programa, você precisa do include *guard.h* (seção Guarda para Linux) e utilizar a seguinte linha de comando: *g++ -ggdb -O0 -o test application.cpp -m32 --std=c++11 -lpthread && ./test*

Listagem 65: Arquivo */non-epos-guard/count_sync_guard/guard_scheduler_cpu_affinity.cpp*

```
// Scheduler Test Program
// #define DEBUG

#include <thread>
#include <chrono>
#include <sstream>

#include <sched.h>
#include "guard.h"

// #define CONSOLE_MODE
const int iterations = 10;

Guard table;
std::thread* philosophers[5];
Future<int>* philosophers_result[5];

bool is_chopstick_free[5];
Future<int>* locked_futures[5];

void philosopher(int n, int l, int c, Future<int>* r);
void think(unsigned long long n);
void eat(unsigned long long n);
unsigned long long busy_wait(unsigned long long n);

#ifdef CONSOLE_MODE
    #define EXTRA(arg) arg
#else
    #define EXTRA(arg)
#endif

void show_message(const char * message, int line, int column) {
#ifdef CONSOLE_MODE
    LOG( message << "(" << line << ")" << std::endl )
#else
    Display::position( line, column );
    LOG( message )
#endif
}

void show_message(std::stringstream * message, int line, int column) {
#ifndef CONSOLE_MODE
```

```
    Display::position( line, column );
#endif
    LOG( message->str() )
    delete message;
}

void setup_program()
{
    Display::clear();
    Display::position(0, 0);
    LOG( "The Philosopher's Dinner:" << std::endl )

    for(int i = 0; i < 5; i++)
    {
        is_chopstick_free[i] = true;
        philosophers_result[i] = new Future<int>();
        locked_futures[i] = nullptr;
    }

    philosophers[0] = new std::thread(&philosopher, 0,  5, 30, philosophers_result[0]);
    philosophers[1] = new std::thread(&philosopher, 1, 10, 44, philosophers_result[1]);
    philosophers[2] = new std::thread(&philosopher, 2, 16, 39, philosophers_result[2]);
    philosophers[3] = new std::thread(&philosopher, 3, 16, 21, philosophers_result[3]);
    philosophers[4] = new std::thread(&philosopher, 4, 10, 17, philosophers_result[4]);

    LOG( "Philosophers are alive and hungry!" << std::endl )

    Display::position(7, 44);
    LOG( '/' )
    Display::position(13, 44);
    LOG( '\\' )
    Display::position(16, 35);
    LOG( '|' )
    Display::position(13, 27);
    LOG( '/' )
    Display::position(7, 27);
    LOG( '\\' )
    Display::position(19, 0);

    LOG( "The dinner is served ..." << std::endl )
}

// g++ -ggdb -O0 -o test application.cpp -m32 --std=c++11 -lpthread && ./test;
// Regular expression for plugin: https://github.com/evandroforks/HighlightWords
// /\) => (\dx[a-f\d]{8,8})/ /Philosopher=\d/ \bGuard::vouch\b \bGuard::clear\b
int main()
{
    DB( "main() => " << Guard::get_thread_id() << std::endl )
    table.submit( &setup_program );

    for(int i = 0; i < 5; i++) {
        philosophers[i]->join();
        int ret = philosophers_result[i]->get_value();
        std::stringstream* stream = new std::stringstream();
```

```
        *stream << "Philosopher " << i << " ate " << ret << " times\n";
        table.submit( &show_message, stream, 20 + i, 0 );
    }

    for(int i = 0; i < 5; i++) {
        delete philosophers[i];
        delete philosophers_result[i];
    }

    LOG( "The end!" << std::endl )
    return 0;
}

void release_chopstick(int philosopher_index, int chopstick_index,
        Future<int>* future_chopstick, const char* which_chopstick)
{
    DB( "Philosopher=" << philosopher_index
            << ", release chopstick=" << chopstick_index
            << ", is_chopstick_free=" << is_chopstick_free[chopstick_index]
            << ", future_chopstick=" << future_chopstick
            << ", locked_futures=" << locked_futures[chopstick_index]
            << ", " << which_chopstick )

    is_chopstick_free[chopstick_index] = true;
    delete future_chopstick;

    if( locked_futures[chopstick_index] ) {
        DB( std::endl )

        auto old = locked_futures[chopstick_index];
        locked_futures[chopstick_index] = nullptr;
        old->resolve(1);
    }
    else {
        assert( locked_futures[chopstick_index] == nullptr );
        DB( std::endl )
    }
}

void get_chopstick(int philosopher_index, int chopstick_index,
        Future<int>* future_chopstick, const char* which_chopstick)
{
    DB( "Philosopher=" << philosopher_index
            << ", getting chopstick=" << chopstick_index
            << ", is_chopstick_free=" << is_chopstick_free[chopstick_index]
            << ", " << which_chopstick )

    if( is_chopstick_free[chopstick_index] ) {
        is_chopstick_free[chopstick_index] = false;

        DB( ", future_chopstick=" << future_chopstick << std::endl )
        future_chopstick->resolve(1);
    }
    else {
```

```
        assert( locked_futures[chopstick_index] == nullptr );
        locked_futures[chopstick_index] = future_chopstick;

        DB( ", locked_futures="
                << locked_futures[chopstick_index] << std::endl )
    }
}

void philosopher(int philosopher_index, int line, int column, Future<int>* result)
{
    DB( "philosopher(index=" << philosopher_index << ") => "
            << Guard::get_thread_id() << std::endl )

    int first = (philosopher_index < 4)? philosopher_index : 0;
    int second = (philosopher_index < 4)? philosopher_index + 1 : 4;

#ifdef CONSOLE_MODE
    line = philosopher_index;
#endif

    for(int i = iterations; i > 0; i--) {
        std::stringstream* stream1 = new std::stringstream();
        *stream1 << "thinking[" << sched_getcpu() << "]"
                EXTRA( << " (" << line << ")" << "\n" );
        table.submit( &show_message, stream1, line, column );
        think(1000000);

        std::stringstream* stream2 = new std::stringstream();
        *stream2 << "  hungry[" << sched_getcpu() << "]"
                EXTRA( << " (" << line << ")" << "\n" );
        table.submit( &show_message, stream2, line, column );

        // Get the first chopstick
        Future<int>* chopstick1 = new Future<int>();
        table.submit( &get_chopstick, philosopher_index, first, chopstick1, "FIRST " );
        chopstick1->get_value();

        // Get the second chopstick
        Future<int>* chopstick2 = new Future<int>();
        table.submit( &get_chopstick, philosopher_index, second, chopstick2, "SECOND"
);
        chopstick2->get_value();

    #ifdef CONSOLE_MODE
        std::stringstream* stream6 = new std::stringstream();
        *stream6 << "Philosopher=" << philosopher_index
                << ", got     the first=" << first
                << " and second=" << second << " chopstick" << "\n";
        table.submit( &show_message, stream6, line, column );
    #endif

        std::stringstream* stream3 = new std::stringstream();
        *stream3 << " eating[" << sched_getcpu() << "] " EXTRA( << "\n" );
        table.submit( &show_message, stream3, line, column );
```

```
        eat(500000);

        std::stringstream* stream4 = new std::stringstream();
        *stream4 << "    ate[" << sched_getcpu() << "]"
                EXTRA( << " (" << line << ")" << "\n" );
        table.submit( &show_message, stream4, line, column );

        // Release the chopsticks
        table.submit( &release_chopstick, philosopher_index, first, chopstick1, "FIRST
" );
        table.submit( &release_chopstick, philosopher_index, second, chopstick2,
"SECOND" );

    #ifdef CONSOLE_MODE
        std::stringstream* stream7 = new std::stringstream();
        *stream7 << "Philosopher=" << philosopher_index
                << ", release the first=" << first
                << " and second=" << second << " chopstick" << "\n";
        table.submit( &show_message, stream7, line, column );
    #endif
    }

    std::stringstream* stream5 = new std::stringstream();
    *stream5 << "  done[" << sched_getcpu() << "]  "
            EXTRA( << " (" << line << ")" << "\n" );
    table.submit( &show_message, stream5, line, column );

    result->resolve(iterations);
}

void eat(unsigned long long n) {
    static unsigned long long v;
    v = busy_wait(n);
}

void think(unsigned long long n) {
    static unsigned long long v;
    v = busy_wait(n);
}

unsigned long long busy_wait(unsigned long long n)
{
    volatile unsigned long long v;
    // for(long long int j = 0; j < 120 * n; j++) v &= 2 ^ j;
    std::this_thread::sleep_for(std::chrono::milliseconds(n/400));
    return v;
}
```

Para este exemplo, foi comentado a penúltima linha no qual fazia um busy wait e substituída por um thread sleep em milisegundos, por questões de conveniência, pois agora este programa de teste roda diretamente em uma máquina nativa e esse busy wait deixa o computador lendo. Mas você pode tranquilamente reativar o busy wait, se quiser ver com os próprios olhos. Mas note que, dependendo do poder de

processamento de sua máquina poderá ser necessário alterar o tamanho máximo do loop busy wait na linha *j < 120 * n*. Originalmente esse valor era 50 ao contrário de 120, entretanto o valor de 50 era insuficiente para causar algum delay significativo, causando o programa terminar quase que imediatamente. Assim, pode ser necessário aumentar ou diminui esse valor dependendo da velocidade de seu processador.

Listagem 66: Exemplo de execução de */non-epos-guard/count_sync_guard/guard_scheduler_cpu_affinity.cpp* em um processador com 4 núcleos

```
The Philosopher's Dinner:
Philosophers are alive and hungry!


                              done[2]

                         \               /


                  done[0]                   done[3]


                         /               \


                     done[3]     |     done[2]


The dinner is served ...
Philosopher 0 ate 10 times
Philosopher 1 ate 10 times
Philosopher 2 ate 10 times
Philosopher 3 ate 10 times
Philosopher 4 ate 10 times
The end!
```

Agora, veja um execução utilizando o modo *#define CONSOLE_MODE*, como apresentado na seção guard_semaphore_test.cc. Mas por simplicidade, sem utilizar o modo *#define DEBUG*, definido no header *guard.h* e exibido pela macro *DB()*. Assim, resultando em um log mais limpo e simples do que o log anteriormente apresentado na seção guard_semaphore_test.cc.

Listagem 67: Exemplo de execução de */non-epos-guard/count_sync_guard/guard_scheduler_cpu_affinity.cpp* em um processador com 4 núcleos

```
Running the command: g++ -ggdb -O0 -o test \
        count_sync_guard/guard_scheduler_cpu_affinity.cpp \
        -m32 --std=c++11 -lpthread && ./test;
The Philosopher's Dinner:
Philosophers are alive and hungry!
```

```
                        \               /




                        /               \

                                |


The dinner is served ...
thinking[1] (1)
thinking[0] (0)
thinking[1] (2)
thinking[2] (3)
thinking[2] (4)
  hungry[1] (1)
  hungry[0] (0)
  hungry[2] (3)
  hungry[3] (4)
  hungry[1] (2)
Philosopher=3, got     the first=3 and second=4 chopstick
 eating[1]
Philosopher=0, got     the first=0 and second=1 chopstick
 eating[0]
    ate[1] (3)
    ate[0] (0)
Philosopher=0, release the first=0 and second=1 chopstick
thinking[0] (0)
Philosopher=3, release the first=3 and second=4 chopstick
thinking[1] (3)
Philosopher=2, got     the first=2 and second=3 chopstick
 eating[0]
Philosopher=4, got     the first=0 and second=4 chopstick
 eating[0]
    ate[0] (2)
Philosopher=2, release the first=2 and second=3 chopstick
thinking[0] (2)
Philosopher=1, got     the first=1 and second=2 chopstick
 eating[0]
    ate[1] (4)
Philosopher=4, release the first=0 and second=4 chopstick
thinking[1] (4)
  hungry[0] (0)
  hungry[1] (3)
Philosopher=0, got     the first=0 and second=1 chopstick
 eating[0]
Philosopher=3, got     the first=3 and second=4 chopstick
    ate[0] (1)
Philosopher=1, release the first=1 and second=2 chopstick
thinking[0] (1)
 eating[1]
```

```
  hungry[1] (4)
    ate[0] (0)
Philosopher=0, release the first=0 and second=1 chopstick
thinking[0] (0)
    ate[0] (3)
Philosopher=3, release the first=3 and second=4 chopstick
thinking[0] (3)
  hungry[0] (2)
Philosopher=4, got     the first=0 and second=4 chopstick
 eating[1]
Philosopher=2, got     the first=2 and second=3 chopstick
 eating[0]
  hungry[0] (1)
    ate[1] (4)
    ate[0] (2)
Philosopher=2, release the first=2 and second=3 chopstick
thinking[0] (2)
Philosopher=1, got     the first=1 and second=2 chopstick
 eating[0]
Philosopher=4, release the first=0 and second=4 chopstick
thinking[1] (4)
  hungry[0] (0)
  hungry[1] (3)
Philosopher=3, got     the first=3 and second=4 chopstick
 eating[1]
    ate[0] (1)
Philosopher=1, release the first=1 and second=2 chopstick
thinking[0] (1)
Philosopher=0, got     the first=0 and second=1 chopstick
 eating[1]
    ate[1] (3)
  hungry[0] (2)
Philosopher=3, release the first=3 and second=4 chopstick
Philosopher=2, got     the first=2 and second=3 chopstick
 eating[0]
    ate[0] (0)
thinking[1] (3)
Philosopher=0, release the first=0 and second=1 chopstick
thinking[0] (0)
  hungry[0] (4)
Philosopher=4, got     the first=0 and second=4 chopstick
 eating[0]
  hungry[0] (1)
    ate[0] (2)
    ate[1] (4)
Philosopher=2, release the first=2 and second=3 chopstick
thinking[0] (2)
Philosopher=4, release the first=0 and second=4 chopstick
thinking[1] (4)
Philosopher=1, got     the first=1 and second=2 chopstick
 eating[1]
  hungry[1] (3)
Philosopher=3, got     the first=3 and second=4 chopstick
 eating[1]
```

```
  hungry[0] (0)
    ate[1] (1)
Philosopher=1, release the first=1 and second=2 chopstick
thinking[1] (1)
Philosopher=0, got     the first=0 and second=1 chopstick
 eating[1]
    ate[1] (3)
  hungry[0] (2)
Philosopher=3, release the first=3 and second=4 chopstick
thinking[1] (3)
  hungry[1] (4)
Philosopher=2, got     the first=2 and second=3 chopstick
 eating[0]
    ate[1] (0)
Philosopher=0, release the first=0 and second=1 chopstick
thinking[1] (0)
Philosopher=4, got     the first=0 and second=4 chopstick
 eating[1]
  hungry[1] (1)
    ate[0] (2)
Philosopher=2, release the first=2 and second=3 chopstick
thinking[0] (2)
Philosopher=1, got     the first=1 and second=2 chopstick
 eating[1]
    ate[1] (4)
Philosopher=4, release the first=0 and second=4 chopstick
thinking[1] (4)
  hungry[1] (3)
Philosopher=3, got     the first=3 and second=4 chopstick
 eating[1]
    ate[1] (1)
Philosopher=1, release the first=1 and second=2 chopstick
thinking[1] (1)
  hungry[1] (0)
Philosopher=0, got     the first=0 and second=1 chopstick
 eating[1]
  hungry[0] (2)
    ate[0] (3)
Philosopher=3, release the first=3 and second=4 chopstick
thinking[0] (3)
Philosopher=2, got     the first=2 and second=3 chopstick
 eating[0]
  hungry[0] (4)
    ate[0] (0)
Philosopher=0, release the first=0 and second=1 chopstick
thinking[0] (0)
Philosopher=4, got     the first=0 and second=4 chopstick
 eating[0]
  hungry[1] (1)
    ate[1] (2)
Philosopher=2, release the first=2 and second=3 chopstick
thinking[1] (2)
Philosopher=1, got     the first=1 and second=2 chopstick
 eating[1]
```

```
    ate[1] (4)
Philosopher=4, release the first=0 and second=4 chopstick
thinking[1] (4)
  hungry[0] (3)
Philosopher=3, got     the first=3 and second=4 chopstick
 eating[0]
  hungry[0] (0)
    ate[1] (1)
Philosopher=0, got     the first=0 and second=1 chopstick
 eating[1]
Philosopher=1, release the first=1 and second=2 chopstick
thinking[1] (1)
    ate[0] (3)
  hungry[1] (2)
Philosopher=3, release the first=3 and second=4 chopstick
thinking[0] (3)
Philosopher=2, got     the first=2 and second=3 chopstick
 eating[0]
  hungry[0] (4)
    ate[0] (0)
Philosopher=0, release the first=0 and second=1 chopstick
thinking[0] (0)
Philosopher=4, got     the first=0 and second=4 chopstick
 eating[0]
    ate[0] (2)
Philosopher=2, release the first=2 and second=3 chopstick
thinking[0] (2)
  hungry[1] (1)
Philosopher=1, got     the first=1 and second=2 chopstick
 eating[1]
    ate[1] (4)
Philosopher=4, release the first=0 and second=4 chopstick
thinking[1] (4)
  hungry[1] (0)
  hungry[0] (3)
Philosopher=3, got     the first=3 and second=4 chopstick
 eating[1]
    ate[1] (1)
Philosopher=0, got     the first=0 and second=1 chopstick
 eating[1]
Philosopher=1, release the first=1 and second=2 chopstick
thinking[1] (1)
  hungry[0] (2)
    ate[1] (0)
Philosopher=0, release the first=0 and second=1 chopstick
  hungry[0] (4)
    ate[0] (3)
Philosopher=3, release the first=3 and second=4 chopstick
thinking[0] (3)
Philosopher=2, got     the first=2 and second=3 chopstick
 eating[0]
thinking[1] (0)
Philosopher=4, got     the first=0 and second=4 chopstick
 eating[0]
```

```
  hungry[1] (1)
    ate[0] (2)
Philosopher=2, release the first=2 and second=3 chopstick
thinking[0] (2)
Philosopher=1, got     the first=1 and second=2 chopstick
    ate[0] (4)
Philosopher=4, release the first=0 and second=4 chopstick
thinking[0] (4)
 eating[1]
  hungry[0] (3)
  hungry[1] (0)
Philosopher=3, got     the first=3 and second=4 chopstick
 eating[0]
    ate[0] (1)
Philosopher=0, got     the first=0 and second=1 chopstick
 eating[0]
Philosopher=1, release the first=1 and second=2 chopstick
thinking[0] (1)
    ate[0] (3)
Philosopher=3, release the first=3 and second=4 chopstick
thinking[0] (3)
  hungry[0] (4)
  hungry[1] (2)
Philosopher=2, got     the first=2 and second=3 chopstick
 eating[1]
    ate[1] (0)
Philosopher=0, release the first=0 and second=1 chopstick
thinking[1] (0)
Philosopher=4, got     the first=0 and second=4 chopstick
 eating[1]
    ate[1] (2)
  hungry[0] (1)
Philosopher=2, release the first=2 and second=3 chopstick
thinking[1] (2)
Philosopher=1, got     the first=1 and second=2 chopstick
 eating[1]
    ate[1] (4)
Philosopher=4, release the first=0 and second=4 chopstick
thinking[1] (4)
  hungry[0] (3)
Philosopher=3, got     the first=3 and second=4 chopstick
 eating[0]
  hungry[1] (0)
    ate[0] (1)
Philosopher=1, release the first=1 and second=2 chopstick
thinking[0] (1)
Philosopher=0, got     the first=0 and second=1 chopstick
 eating[1]
    ate[0] (3)
Philosopher=3, release the first=3 and second=4 chopstick
thinking[0] (3)
    ate[1] (0)
Philosopher=0, release the first=0 and second=1 chopstick
thinking[1] (0)
```

```
  hungry[1] (2)
  hungry[1] (4)
Philosopher=4, got     the first=0 and second=4 chopstick
 eating[0]
Philosopher=2, got     the first=2 and second=3 chopstick
 eating[1]
  hungry[0] (1)
    ate[0] (4)
Philosopher=4, release the first=0 and second=4 chopstick
thinking[0] (4)
    ate[0] (2)
Philosopher=2, release the first=2 and second=3 chopstick
thinking[0] (2)
Philosopher=1, got     the first=1 and second=2 chopstick
 eating[0]
  hungry[1] (0)
  hungry[1] (3)
Philosopher=3, got     the first=3 and second=4 chopstick
 eating[1]
    ate[1] (1)
Philosopher=0, got     the first=0 and second=1 chopstick
 eating[1]
Philosopher=1, release the first=1 and second=2 chopstick
thinking[1] (1)
    ate[1] (0)
  hungry[0] (2)
  hungry[0] (4)
    ate[1] (3)
Philosopher=3, release the first=3 and second=4 chopstick
  done[1]   (3)
Philosopher=0, release the first=0 and second=1 chopstick
  done[1]   (0)
Philosopher 0 ate 10 times
Philosopher=2, got     the first=2 and second=3 chopstick
 eating[1]
Philosopher=4, got     the first=0 and second=4 chopstick
 eating[3]
  hungry[1] (1)
    ate[1] (2)
    ate[3] (4)
Philosopher=4, release the first=0 and second=4 chopstick
  done[3]   (4)
Philosopher=2, release the first=2 and second=3 chopstick
  done[1]   (2)
Philosopher=1, got     the first=1 and second=2 chopstick
 eating[3]
    ate[3] (1)
Philosopher=1, release the first=1 and second=2 chopstick
  done[3]   (1)
Philosopher 1 ate 10 times
Philosopher 2 ate 10 times
Philosopher 3 ate 10 times
Philosopher 4 ate 10 times
The end!
```

guard_synchronizer_test.cc ←

Trata-se do mesmo programa original do EPOS chamado *synchronizer_test.cc*, também conhecido como *Producer Consumer*. Até o presente momento não foi conseguido completar a lógica de execução correta para a aplicação *guard_synchronizer_test.cc*. Igualmente as aplicações anteriores, faz-se uso da mesma estratégia para realizar a sincronização entre as diferentes threads.

Agora tem-se duas listas encadeada de variáveis *Future*, representando quantas requisições de consumo e produção estão aguardando a disponibilidade do buffer. Os anteriores semáforos agora são somente dois números inteiros que servem como contadores, para representar quantas vagas estão disponíveis nos respectivos produtor e consumidor.

Listagem 68: Arquivo */app/guard_synchronizer_test.cc*

```
// EPOS Synchronizer Component Test Program

#include <utility/ostream.h>
#include <utility/guard.h>
#include <utility/future.h>
#include <utility/debug_sync.h>
#include <utility/list.h>
#include <thread.h>
#include <alarm.h>

#if !defined(nullptr)
    #define nullptr 0
#endif

using namespace EPOS;

#define CONSOLE_MODE
const int iterations = 10;

Guard table;
OStream cout;

const int BUF_SIZE = 16;
char buffer[BUF_SIZE];

int full = 0;
int empty = BUF_SIZE;

Simple_List<Future<bool>*> pending_consume_futures;
Simple_List<Future<bool>*> pending_produce_futures;

#ifdef CONSOLE_MODE
    #define EXTRA(arg)
#else
    #define EXTRA
#endif
```

```
void consume() {
    EXTRA( LOG( "consume(empty=" << empty ) )

    Simple_List<Future<bool>*>::Element* element
            = pending_consume_futures.remove_tail();

    EXTRA( LOG( " element=" << element ) )
    EXTRA( LOG( ")" << endl ) )

    if( element )
    {
        Future<bool>* future = *element->object();
        future->resolve(true);
    }

    ++empty;
}

void can_consume(Future<bool>* can_consume) {
    EXTRA( LOG( "can_consume(full=" << full << ")" << endl ) )

    if(full > 0) {
        --full;
        can_consume->resolve(true);
    }
    else {
        Simple_List<Future<bool>*>::Element* element =
                new Simple_List<Future<bool>*>::Element(&can_consume);
        pending_consume_futures.insert(element);
    }
}

int consumer() {
    EXTRA( LOG( "consumer(this=" << Thread::self() << ")" << endl ) )

    int out = 0;
    for(int i = 0; i < iterations; i++)
    {
        // full.p();
        Future<bool>* future = new Future<bool>();
        table.submit(can_consume, future);
        future->get_value();

        cout << "C<-" << buffer[out] << "\t";
        out = (out + 1) % BUF_SIZE;
        Alarm::delay(5000);

        // empty.v();
        table.submit(consume);
    }

    return 0;
}
```

```
void can_produce(Future<bool>* can_consume) {
    EXTRA( LOG( "can_produce(empty=" << empty << ")" << endl ) )

    if(empty > 0) {
        --empty;
        can_consume->resolve(true);
    }
    else {
        Simple_List<Future<bool>*>::Element* element =
                new Simple_List<Future<bool>*>::Element(&can_consume);
        pending_produce_futures.insert(element);
    }
}

void produce() {
    EXTRA( LOG( "produce(full=" << full ) )

    Simple_List<Future<bool>*>::Element* element
            = pending_produce_futures.remove_tail();

    EXTRA( LOG( " element=" << element ) )
    EXTRA( LOG( ")" << endl ) )

    if( element )
    {
        Future<bool>* future = *element->object();
        future->resolve(true);
    }

    ++full;
}

int main() {
    Thread * cons = new Thread(&consumer);

    // producer
    int in = 0;
    for(int i = 0; i < iterations; i++) {
        // empty.p();
        Future<bool>* future = new Future<bool>();
        table.submit(can_produce, future);
        future->get_value();

        Alarm::delay(5000);
        buffer[in] = 'a' + in;
        cout << "P->" << buffer[in] << "\t";
        in = (in + 1) % BUF_SIZE;

        // full.v();
        table.submit(produce);
    }

    cons->join();
    cout << "The end!" << endl;
```

```
    delete cons;
    return 0;
}
```

Ao executar essa aplicação em modo *CONSOLE_MODE* com 10 iterações, percebe-se que logo após completar-se essas 10 interações, não há nenhum consumo por parte do consumidor. Portanto, ainda precisa-se verificar as interações entre os algoritmos do Guarda e Future no momento em que a thread do consumidor começa a executar, mas não faz nenhum progresso.

Listagem 69: Exemplo de execução de */app/guard_synchronizer_test.cc*

```
can_produce(empty=16)
consumer(this=0x00097f44)
can_consume(full=0)
P->a    produce(full=0 element=0x00000000)
can_produce(empty=15)
P->b    produce(full=1 element=0x00000000)
can_produce(empty=14)
P->c    produce(full=2 element=0x00000000)
can_produce(empty=13)
P->d    produce(full=3 element=0x00000000)
can_produce(empty=12)
P->e    produce(full=4 element=0x00000000)
can_produce(empty=11)
P->f    produce(full=5 element=0x00000000)
can_produce(empty=10)
P->g    produce(full=6 element=0x00000000)
can_produce(empty=9)
P->h    produce(full=7 element=0x00000000)
can_produce(empty=8)
P->i    produce(full=8 element=0x00000000)
can_produce(empty=7)
P->j    produce(full=9 element=0x00000000)
```

producer_consumer (guard_synchronizer_test.cc) ←

Trata-se do mesmo programa original do EPOS chamado *synchronizer_test.cc*, também conhecido como *Producer Consumer*. Nesta versão, faz-se a substituição dos dois semáforos originais por uma guarda, um contador e duas variáveis de condição. O Guarda serve para garantir a exclusão mútua das seções críticas, onde ocorrem acessos a um buffer compartilhado entre as threads produras e consumidoras.

Listagem 70: Arquivo */app/producer_consumer.cc*

```
// EPOS Synchronizer Component Test Program

#include <utility/ostream.h>
#include <thread.h>
```

```
#include <alarm.h>
#include <utility/guard.h>
#include <utility/future.h>
#include <condition.h>
#include <clock.h>
#include <utility/random.h>

using namespace EPOS;

const int iterations = 100;

OStream cout;
Clock clock;

Guard buffer_guard;
Condition full;
Condition empty;
const int BUF_SIZE = 16;
char buffer[BUF_SIZE];
int buffer_count = 0;
int total_count = 0;
int out = 0;
int in = 0;

void check_load(Future<bool> * wait){
    if(buffer_count > 0){ // checking whether the buffer IS empty
        wait->resolve(false);
    } else {
        wait->resolve(true);
    }
}

void load(Future<char> * item)
{
    item->resolve(buffer[out]);
    cout << "C<-" << buffer[out] << "\t";
    total_count++;
    if (total_count > 10){
        cout << "\n";
        total_count = 0;
    }
    out = (out + 1) % BUF_SIZE;
    buffer_count--;
    if (buffer_count == BUF_SIZE-1){
        //cout << "Some Producer Wakeup"  << "\n";
        full.signal(); // wakeup waiting producer
    }
}

void check_store(Future<bool> * wait){
    if(buffer_count >= BUF_SIZE) // checking whether the buffer IS full
        wait->resolve(true);
    else
        wait->resolve(false);
```

```
}

void store(char item)
{
    buffer[in] = item;
    cout << "P->" << item << "\t";
    total_count++;
    if (total_count > 10){
        cout << "\n";
        total_count = 0;
    }
    in = (in + 1) % BUF_SIZE;
    buffer_count++;
    if (buffer_count == 1){
        //cout << "Some Consumer Wakeup"  << "\n";
        empty.signal(); // wakeup waiting consumer
    }
}

int consumer()
{
    for(int i = 0; i < iterations; i++) {
        Future<char> item{};
        Future<bool> wait{};
        buffer_guard.submit(check_load, &wait); // checking whether wait is necessary
        if (wait.get_value()){
            empty.wait(); // wait somehow
        }
        buffer_guard.submit(load, &item); // load item from buffer
        item.get_value();  // consume - part1
        unsigned long int delay_rand = Random::random();
        unsigned long int true_rand = (delay_rand > 9999) ? delay_rand / 10000 :
delay_rand;
        Alarm::delay(true_rand/5); // consume - part3
    }
    return 0;
}

int producer()
{
    for(int i = 0; i < iterations; i++) {
        unsigned long int delay_rand = Random::random();
        unsigned long int true_rand = (delay_rand > 9999) ? delay_rand / 1000 :
delay_rand;
        Alarm::delay(true_rand/20); // consume - part3
        char item = 'a' + in;  // produce - part2
        Future<bool> wait{};
        buffer_guard.submit(check_store, &wait); // checking whether wait is necessary
        if(wait.get_value()){
            //cout << "Producer Waiting"  << "\n";
            full.wait();
            //cout << "Producer Awaking"  << "\n";
        }
        buffer_guard.submit(store, item); // store item on buffer
```

```
    }
    return 0;
}

int main()
{
    cout << "Main started!" << endl;

    unsigned long int ini = clock.now();
    Random::seed(ini);

    Thread * prod = new Thread(&producer);
    Thread * cons = new Thread(&consumer);

    cons->join();
    prod->join();

    cout << "The end!" << endl;

    delete cons;
    delete prod;

    return 0;
}
```

Resultado da execução:

Listagem 71: Exemplo de execução de */app/producer_consumer.cc*

```
Main started!
P->a    C<-a    P->b    C<-b    P->c    C<-c    P->d    C<-d    P->e    C<-e    P->f
C<-f    P->g    C<-g    P->h    C<-h    P->i    C<-i    P->j    C<-j    P->k    C<-k
P->l    C<-l    P->m    C<-m    P->n    C<-n    P->o    C<-o    P->p    C<-p    P->a
C<-a    P->b    C<-b    P->c    C<-c    P->d    P->e    C<-d    C<-e    P->f    C<-f
P->g    C<-g    P->h    C<-h    P->i    C<-i    P->j    C<-j    P->k    C<-k    P->l
C<-l    P->m    C<-m    P->n    C<-n    P->o    C<-o    P->p    C<-p    P->a    C<-a
P->b    C<-b    P->c    C<-c    P->d    C<-d    P->e    C<-e    P->f    C<-f    P->g
C<-g    P->h    C<-h    P->i    C<-i    P->j    C<-j    P->k    C<-k    P->l    C<-l
P->m    C<-m    P->n    C<-n    P->o    C<-o    P->p    C<-p    P->a    C<-a    P->b
C<-b    P->c    C<-c    P->d    C<-d    P->e    C<-e    P->f    C<-f    P->g    C<-g
P->h    C<-h    P->i    C<-i    P->j    C<-j    P->k    C<-k    P->l    C<-l    P->m
C<-m    P->n    C<-n    P->o    C<-o    P->p    C<-p    P->a    C<-a    P->b    C<-b
P->c    C<-c    P->d    P->e    C<-d    P->f    C<-e    P->g    C<-f    C<-g    P->h
C<-h    P->i    C<-i    P->j    C<-j    P->k    C<-k    P->l    C<-l    P->m    C<-m
P->n    C<-n    P->o    C<-o    P->p    C<-p    P->a    C<-a    P->b    C<-b    P->c
C<-c    P->d    C<-d    P->e    C<-e    P->f    C<-f    P->g    C<-g    P->h    C<-h
P->i    C<-i    P->j    C<-j    P->k    C<-k    P->l    C<-l    P->m    C<-m    P->n
C<-n    P->o    C<-o    P->p    C<-p    P->a    C<-a    P->b    C<-b    P->c    C<-c
P->d    C<-d    The end!
```

O que aprendemos com os testes ←

1. Executar aplicações com mais de 1 CPU ativa variavelmente gera exceções (GPF). Isso acontece mesmo com a aplicação do jantar dos filósofos, se os delays forem removidos.
2. Quando requisições são enviadas ao sequencer em uma taxa maior do que ele consegue executar, a memória enche e exceções (PF) ocorrem.
3. Com o KVM não funciona o relógio e o escalonador não era chamado.
4. Sem o RR fica difícil gerar erros de sincronização.
5. Após a primeira execução, o qemu fica muito comportado e para de gerar erros de contagem durante a execução do arquivo corrente mais simples: *count_sync_uncoordinated.cc*
6. Compilando o EPOS utilizado/entregado no trabalho anterior de 2017.2 com GCC 7.2.0, não executa a aplicação principal. Baixamos o código deles, e compilamos. A seguir tem um log completo no modo histérico da execução, mostrando o problema:
   https://gist.github.com/evandrocoan/3159d9bf1c53620e2e01169d7a4b8d92

Lista de futuros projetos para o semestre que vem ←

1. Substituir a nossa implementação de closure pelas funções lambdas do C++ 11. Note que lambdas estão somente disponíveis para C++ 11, então você ter que fazer EPOS executar em uma versão do GCC tal como 7.2.0, que atualmente não funciona. Veja a seção O que aprendemos com os testes.
2. Uma segunda versão da implementação de Futures, com suporte completo requisições bloqueantes e não bloqueantes junto com um serviço de execução que trata de criar novas threads na medida que o sistema necessita executar códigos com dependência de dados. Ver a seção Futures. Tal serviço será reconfigurável, limitando o número máximo de threads que podem existir simultaneamente. Por exemplo, um sistema embarcado pode não possuir memória suficiente para que muitas thread sejam criadas, então esse serviço de threads limitaria o número máximo de threads que podem ser criadas ao mesmo tempo, e caso esse limite seja ultrapassado, novas requisições aguardam em uma fila, i.e., bloqueiam até que recursos estejam disponíveis.
   Tal serviço também, pode ser configurado para iniciar com um número definido de threads que ficam aguardando um bloco de dependência ser requerido, e na medida que surge o diminui a demanda, vai criando e destruindo threads para economizar memória. Por preferência de performance e de não remover recursos do sistema, somente fazer tal balanceamento de carga quando uma idle thread entrar em execução. Assim, não se tira recursos de nenhuma outra thread em execução.
3. A implementação da variação do algoritmo do **Guarda** para arquiteturas NUMA {2}, o algoritmo do **Ator**. Ver a seção Atores. Cuidado por que mesmo este semestre, não conseguimos fazer o algoritmo do Guarda funcionar corretamente utilizando multicores no EPOS. Veja a seção guard_scheduler_cpu_affinity_test.cc. Isso talvez possa ser algum problema com a emulação que o QEMU faz do EPOS, ou um problema com alguma coisa no EPOS, ou algum problema com código que implementamos esse semestre.

Referências de Implementação ←

1. https://stackoverflow.com/questions/18847424/c-create-custom-function-dispatcher-from-variadic-template
2. https://stackoverflow.com/questions/42047795/order-of-parameter-pack-expansion
3. https://stackoverflow.com/questions/29194858/order-of-function-calls-in-variadic-template-expansion
4. https://stackoverflow.com/questions/43553585/how-to-transform-a-variadic-template-argument-to-another-types-for-calling-anoth

5. https://stackoverflow.com/questions/12030538/calling-a-function-for-each-variadic-template-argument-and-an-array
6. https://stackoverflow.com/questions/687490/how-do-i-expand-a-tuple-into-variadic-template-functions-arguments/12650100#12650100
7. https://stackoverflow.com/questions/7858817/unpacking-a-tuple-to-call-a-matching-function-pointer
8. https://en.cppreference.com/w/cpp/language/list_initialization#Notes
9. https://stackoverflow.com/questions/47207621/build-function-parameters-with-variadic-templates
10. https://stackoverflow.com/questions/16868129/how-to-store-variadic-template-arguments
11. https://stackoverflow.com/questions/43026550/how-to-define-a-class-that-can-save-variadic-template-arguments
12. https://stackoverflow.com/questions/17996003/how-to-save-variable-number-of-arguments-using-variadic-template-arguments
13. https://stackoverflow.com/questions/29220248/is-it-possible-to-store-variadic-arguments-into-a-member-variable
14. https://stackoverflow.com/questions/4691657/is-it-possible-to-store-a-template-parameter-pack-without-expanding-it
15. https://arne-mertz.de/2016/11/modern-c-features-variadic-templates/
16. https://gitlab.com/evandro-crr/epos2/tree/master
17. https://stackoverflow.com/questions/13108663/storing-functions-call-and-list-of-parameters-to-invoke-later
18. https://stackoverflow.com/questions/4926433/saving-function-pointerarguments-for-later-use
19. https://appuals.com/fix-cannot-execute-binary-file-exec-format-error-ubuntu/
20. https://stackoverflow.com/questions/30926764/extracting-function-argument-types-as-a-parameter-pack
21. https://stackoverflow.com/questions/24948277/unpacking-arguments-of-a-functional-parameter-to-a-c-template-class
22. https://stackoverflow.com/questions/14783876/c-is-it-possible-to-extract-class-and-argument-types-from-a-member-function
23. https://stackoverflow.com/questions/34836104/how-to-extract-a-selected-set-of-arguments-of-a-variadic-function-and-use-them-t
24. https://stackoverflow.com/questions/28033251/can-you-extract-types-from-template-parameter-function-signature
25. https://stackoverflow.com/questions/11056714/c-type-traits-to-extract-template-parameter-class
26. https://stackoverflow.com/questions/32674839/variadic-member-function-of-template-class
27. https://stackoverflow.com/questions/49217891/class-member-function-in-variadic-template
28. https://stackoverflow.com/questions/50316284/how-to-achieve-variadic-virtual-member-function
29. https://stackoverflow.com/questions/15599679/class-member-function-pointer
30. https://stackoverflow.com/questions/40855835/how-do-you-typedef-a-function-pointer-type-with-parameter-pack-arguments
31. https://stackoverflow.com/questions/8915797/calling-a-function-through-its-address-in-memory-in-c-c
32. https://stackoverflow.com/questions/47005664/call-function-by-known-address-c
33. https://en.cppreference.com/w/cpp/language/parameter_pack
34. https://stackoverflow.com/questions/47037395/forward-types-in-variadic-template-as-values-references-according-to-function-si
35. https://eli.thegreenplace.net/2014/perfect-forwarding-and-universal-references-in-c
36. https://stackoverflow.com/questions/3836648/structure-or-class-with-variable-number-of-members
37. https://stackoverflow.com/questions/43069213/c-class-template-for-automatic-getter-setter-methods-

good-bad-practice
38. https://stackoverflow.com/questions/13980157/c-class-with-template-member-variable
39. https://stackoverflow.com/questions/6261375/how-to-declare-data-members-that-are-objects-of-any-type-in-a-class
40. https://stackoverflow.com/questions/1872220/is-it-possible-to-iterate-over-arguments-in-variadic-macros
41. https://stackoverflow.com/questions/11761703/overloading-macro-on-number-of-arguments
42. https://groups.google.com/forum/#!topic/comp.std.c/d-6Mj5Lko_s
43. https://stackoverflow.com/questions/27941661/generating-one-class-member-per-variadic-template-argument
44. https://stackoverflow.com/questions/14919990/c-how-to-template-a-class-attribute-and-not-the-functions-of-the-class
45. https://stackoverflow.com/questions/40204338/template-parameter-pack-attribute
46. https://stackoverflow.com/questions/5723619/is-it-possible-to-create-function-local-closures-pre-c11
47. https://blog.feabhas.com/2014/03/demystifying-c-lambdas/
48. https://stackoverflow.com/questions/1447199/c-closures-and-templates
49. http://matt.might.net/articles/c++-template-meta-programming-with-lambda-calculus/
50. https://www.gnu.org/software/gcc/gcc-4.4/cxx0x_status.html
51. https://stackoverflow.com/questions/25091436/expand-a-parameter-pack-with-a-counter
52. https://stackoverflow.com/questions/41623422/c-expand-variadic-template-arguments-into-a-statement
53. https://stackoverflow.com/questions/8526598/how-does-stdforward-work
54. https://stackoverflow.com/questions/31204084/is-it-possible-to-call-static-method-form-variadic-template-type-parameter
55. https://stackoverflow.com/questions/25680461/variadic-template-pack-expansion
56. https://stackoverflow.com/questions/21180346/variadic-template-unpacking-arguments-to-typename
57. https://stackoverflow.com/questions/2934904/order-of-evaluation-in-c-function-parameters
58. https://stackoverflow.com/questions/9566187/function-parameter-evaluation-order
59. https://stackoverflow.com/questions/7728478/c-template-class-function-with-arbitrary-container-type-how-to-define-it
60. https://stackoverflow.com/questions/12048221/c11-variadic-template-function-parameter-pack-expansion-execution-order
61. https://stackoverflow.com/questions/34957810/variadic-templates-parameter-pack-and-its-discussed-ambiguity-in-a-parameter-li
62. https://stackoverflow.com/questions/20588191/error-with-variadiac-template-parameter-pack-must-be-expanded
63. https://stackoverflow.com/questions/37200391/multiple-variadic-parameter-pack-for-template-class
64. https://stackoverflow.com/questions/34940875/parameter-pack-must-be-at-the-end-of-the-parameter-list-when-and-why
65. https://stackoverflow.com/questions/15904288/how-to-reverse-the-order-of-arguments-of-a-variadic-template-function
66. https://stackoverflow.com/questions/12906523/how-can-i-interpret-a-stackdump-file
67. https://stackoverflow.com/questions/320001/using-a-stackdump-from-cygwin-executable
68. https://stackoverflow.com/questions/37628530/how-to-debug-using-stackdump-file-in-cygwin
69. https://stackoverflow.com/questions/8305866/how-to-analyze-a-programs-core-dump-file-with-gdb
70. https://stackoverflow.com/questions/5115613/core-dump-file-analysis
71. https://www.linuxquestions.org/questions/programming-9/cygwin-gcc-compiled-code-generates-a-stac

kdump-how-to-analyze-with-gdb-874630
72. http://cygwin.1069669.n5.nabble.com/exe-stackdump-and-core-dump-files-questions-td17219.html
73. https://stackoverflow.com/questions/10208233/what-comes-first-template-instantiation-vs-macro-expansion/10208278#10208278
74. https://stackoverflow.com/questions/44050323/class-member-variables-based-on-variadic-template
75. https://stackoverflow.com/questions/40356688/c-define-a-class-member-type-according-to-its-template
76. https://stackoverflow.com/questions/11394832/how-to-define-a-template-member-function-of-a-template-class
77. https://stackoverflow.com/questions/29668447/c-independent-static-variable-for-each-object-inside-a-method
78. http://www.cplusplus.com/forum/general/142258/
79. https://stackoverflow.com/questions/3778450/is-local-static-variable-per-instance-or-per-class
80. https://stackoverflow.com/questions/347358/inheriting-constructors
81. https://stackoverflow.com/questions/8887864/template-base-constructor-call-in-member-initialization-list-error
82. https://stackoverflow.com/questions/25064515/how-to-call-constructor-of-a-template-base-class-in-a-template-derived-class
83. https://stackoverflow.com/questions/10282787/calling-the-base-class-constructor-from-the-derived-class-constructor
84. https://stackoverflow.com/questions/120876/what-are-the-rules-for-calling-the-superclass-constructor
85. https://stackoverflow.com/questions/12464633/is-it-safe-to-privately-inherit-from-a-class-with-a-non-virtual-destructor
86. https://stackoverflow.com/questions/2004820/inherit-interfaces-which-share-a-method-name
87. https://stackoverflow.com/questions/10535667/does-it-make-any-sense-to-use-inline-keyword-with-templates
88. https://stackoverflow.com/questions/3531060/how-to-initialize-a-static-const-member-in-c
89. https://stackoverflow.com/questions/34222703/how-to-override-static-method-of-template-class-in-derived-class
90. https://stackoverflow.com/questions/11761506/inheritance-function-that-returns-self-type
91. https://stackoverflow.com/questions/115703/storing-c-template-function-definitions-in-a-cpp-file
92. https://stackoverflow.com/questions/19884494/too-few-template-parameter-lists-error
93. https://stackoverflow.com/questions/5901300/how-do-i-view-previous-diff-commits-using-git
94. https://stackoverflow.com/questions/2721846/alternative-to-c-static-virtual-methods
95. https://stackoverflow.com/questions/137038/how-do-you-get-assembler-output-from-c-c-source-in-gcc
96. https://stackoverflow.com/questions/34842152/compare-and-exchange-on-x86-cpu
97. https://stackoverflow.com/questions/47981/how-do-you-set-clear-and-toggle-a-single-bit
98. https://stackoverflow.com/questions/2064692/how-to-print-function-pointers-with-cout
99. https://stackoverflow.com/questions/8747005/backporting-nullptr-to-c-pre-c0x-programs
100. https://stackoverflow.com/questions/10496824/how-to-define-nullptr-for-supporting-both-c03-and-c11
101. https://stackoverflow.com/questions/2741708/makefile-contains-string
102. https://stackoverflow.com/questions/4673720/private-inheritance-using-directive-overloads
103. https://stackoverflow.com/questions/656224/when-should-i-use-c-private-inheritance
104. https://stackoverflow.com/questions/24609872/delete-virtual-function-from-a-derived-class
105. https://stackoverflow.com/questions/18469818/class-inheritance-error-private-member
106. https://stackoverflow.com/questions/9055778/initializing-a-reference-to-member-to-null-in-c
107. http://www.cplusplus.com/reference/cassert/assert/

108. https://en.cppreference.com/w/cpp/atomic/atomic
109. https://stackoverflow.com/questions/31978324/what-exactly-is-stdatomic
110. https://stackoverflow.com/questions/41206861/atomic-increment-and-return-counter
111. https://en.cppreference.com/w/cpp/thread/recursive_mutex
112. https://en.cppreference.com/w/cpp/thread/recursive_mutex/lock
113. https://riptutorial.com/cplusplus/example/30442/std--recursive-mutex
114. https://baptiste-wicht.com/posts/2012/04/c11-concurrency-tutorial-advanced-locking-and-condition-variables.html
115. https://stackoverflow.com/questions/9935190/why-is-a-point-to-volatile-pointer-like-volatile-int-p-useful
116. https://stackoverflow.com/questions/4792449/c0x-has-no-semaphores-how-to-synchronize-threads
117. https://stackoverflow.com/questions/14171955/incorrect-register-rbx-used-withl-suffix
118. https://stackoverflow.com/questions/353180/how-do-i-find-the-name-of-the-calling-function
119. https://stackoverflow.com/questions/3899870/print-call-stack-in-c-or-c
120. https://stackoverflow.com/questions/46425109/access-the-owners-counter-used-by-stdrecursive-mutex
121. https://stackoverflow.com/questions/14581837/gdb-how-to-print-the-current-line-or-find-the-current-line-number
122. https://stackoverflow.com/questions/37616367/have-gdb-show-the-type-of-a-template-argument
123. https://stackoverflow.com/questions/33661179/debug-c-template-with-gdb
124. https://en.cppreference.com/w/cpp/thread/condition_variable
125. https://stackoverflow.com/questions/9212219/converting-int-to-pointer
126. https://stackoverflow.com/questions/5100718/integer-to-hex-string-in-c
127. https://stackoverflow.com/questions/38770996/linux-moving-the-console-cursor-visual
128. https://stackoverflow.com/questions/40046591/how-do-i-move-the-console-cursor-to-x-y-on-unix
129. https://stackoverflow.com/questions/7686939/c-simple-return-value-from-stdthread
130. https://stackoverflow.com/questions/21531096/can-i-use-stdasync-without-waiting-for-the-future-limitation
131. http://man7.org/linux/man-pages/man3/sched_getcpu.3.html
132. https://eli.thegreenplace.net/2016/c11-threads-affinity-and-hyperthreading/
133. https://stackoverflow.com/questions/16034448/obtaining-thread-core-affinity-in-c-11-through-pthreads
134. https://en.cppreference.com/w/cpp/thread/thread/native_handle
135. https://stackoverflow.com/questions/9358681/how-can-i-get-the-processor-id-of-the-current-process-in-c-in-linux
136. http://man7.org/linux/man-pages/man2/getcpu.2.html
137. https://stackoverflow.com/questions/8042822/how-can-i-find-which-processor-each-thread-is-running-on
138. https://docs.microsoft.com/en-us/windows/desktop/api/processthreadsapi/nf-processthreadsapi-getcurrentprocessornumber
139. https://en.cppreference.com/w/cpp/header/thread
140. https://stackoverflow.com/questions/150355/programmatically-find-the-number-of-cores-on-a-machine
141. https://en.cppreference.com/w/cpp/thread/thread/hardware_concurrency
142. https://solarianprogrammer.com/2012/10/17/cpp-11-async-tutorial/
143. https://stackoverflow.com/questions/42128670/c-stdasync-does-not-spawn-a-new-thread
144. https://thispointer.com/c11-multithreading-part-7-condition-variables-explained/

145. http://www.cplusplus.com/reference/condition_variable/condition_variable/
146. https://en.cppreference.com/w/cpp/thread/unique_lock
147. https://en.cppreference.com/w/cpp/thread/condition_variable/wait
148. https://solarianprogrammer.com/2011/12/16/cpp-11-thread-tutorial/
149. https://stackoverflow.com/questions/19217093/terminate-called-after-throwing-an-instance-of-stdout-of-range
150. https://code.i-harness.com/en/q/7167a4
151. https://stackoverflow.com/questions/5101918/thread-id-vs-thread-handle
152. https://stackoverflow.com/questions/7432100/how-to-get-integer-thread-id-in-c11
153. https://stackoverflow.com/questions/19255137/how-to-convert-stdthreadid-to-string-in-c
154. https://en.cppreference.com/w/cpp/thread/thread/id
155. https://stackoverflow.com/questions/16259257/c11-native-handle-is-not-a-member-of-stdthis-thread
156. https://en.cppreference.com/w/cpp/language/integer_literal
157. https://stackoverflow.com/questions/5760252/how-can-i-print-0x0a-instead-of-0xa-using-cout
158. http://www.cplusplus.com/reference/ios/hex/
159. https://stackoverflow.com/questions/42262750/how-to-get-stdthread-of-current-thread
160. http://www.modernescpp.com/index.php/condition-variables
161. https://stackoverflow.com/questions/43759609/stdcondition-variablenotify-all-i-need-an-example
162. https://stackoverflow.com/questions/4184468/sleep-for-milliseconds
163. https://en.cppreference.com/w/cpp/thread/sleep_for
164. https://stackoverflow.com/questions/3292795/how-to-declare-a-templated-struct-class-as-a-friend
165. https://stackoverflow.com/questions/8967521/class-template-with-template-class-friend-whats-really-going-on-here
166. https://stackoverflow.com/questions/25878765/stdpromise-and-stdfuture-in-c
167. https://en.cppreference.com/w/cpp/thread/shared_future
168. https://www.codeproject.com/Tips/712107/Simple-introduction-to-std-future-and-std-async
169. https://en.cppreference.com/w/cpp/thread/future
170. https://www.justsoftwaresolutions.co.uk/threading/locks-mutexes-semaphores.html
171. https://stackoverflow.com/questions/18878377/implementing-a-semaphore-with-stdmutex
172. https://stackoverflow.com/questions/6804044/use-a-mutex-as-a-semaphore
173. https://stackoverflow.com/questions/36248616/how-mutex-semaphore-works
174. https://en.cppreference.com/w/cpp/thread/lock
175. https://stackoverflow.com/questions/38340378/difference-between-stdmutex-lock-function-and-stdlock-guardstdmutex
176. https://stackoverflow.com/questions/7007834/what-is-the-use-case-for-the-atomic-exchange-read-write-operation
177. https://www.quora.com/What-is-the-difference-between-a-mutex-and-a-semaphore
178. https://en.cppreference.com/w/cpp/thread/mutex/lock
179. https://en.cppreference.com/w/cpp/thread/mutex
180. https://stackoverflow.com/questions/46767423/how-can-i-lock-twice-with-the-same-mutex-on-the-same-thread
181. https://stackoverflow.com/questions/13483767/locking-multiple-mutexes
182. https://stackoverflow.com/questions/34915926/is-assignment-equivalent-to-load-store-for-stdatomicbool
183. https://stackoverflow.com/questions/30761225/how-to-use-stdatomic
184. https://stackoverflow.com/questions/45859414/atomic-read-then-write-with-stdatomic
185. https://en.cppreference.com/w/cpp/atomic/atomic/exchange

186. http://www.cplusplus.com/reference/atomic/memory_order/
187. http://www.cplusplus.com/reference/atomic/atomic/operator=/
188. https://en.cppreference.com/w/cpp/language/memory_model
189. https://stackoverflow.com/questions/22424209/tsl-instruction-reference
190. https://en.wikipedia.org/wiki/Double-checked_locking
191. https://en.wikipedia.org/wiki/Test_and_test-and-set
192. https://en.wikipedia.org/wiki/Spinlock
193. https://stackoverflow.com/questions/14498892/stdmutex-vs-stdrecursive-mutex-as-class-member
194. https://stackoverflow.com/questions/36619715/a-shared-recursive-mutex-in-standard-c
195. https://stackoverflow.com/questions/23893835/what-special-purpose-does-unique-lock-have-over-using-a-mutex
196. https://tiki.org/WikiSyntax
197. https://doc.tiki.org/PluginDiv#Controlling_Wrapping_
198. https://doc.tiki.org/Wiki-Syntax-Text

Referências ←

1. Stefan Reif and Wolfgang Schröder-Preikschat, "Predictable Synchronisation Algorithms for Asynchronous Critical Sections," Friedrich-Alexander-Universität Erlangen-Nürnberg, Dept. of Computer Science, Technical Reports, CS-2018-03, February 2018.
2. G. Drescher and W. Schröder-Preikschat, "Guarded sections: Structuring aid for wait-free synchronisation," in Proceedings of the 18th International Symposium On Real-Time Computing (ISORC 2015). IEEE Computer Society Press, 2015, pp. 280–283.
3. S. Reif, T. Hönig, and W. Schröder-Preikschat. 2017. In the Heat of Conflict: On the Synchronisation of Critical Sections. In IEEE ISORC '17. 42–51.
4. D. Klaftenegger, K. Sagonas, and K. Winblad, "Brief announcement: Queue delegation locking," in Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2014). ACM Press, 2014, pp. 70–72.
5. Herlihy, Maurice. "The art of multiprocessor programming." PODC (2006).
6. Håkan Sundell, "Efficient and Practical Non-Blocking Data Structures", Chalmers University of Technology and Göteborg University, Department of Computing Science, 2004
7. Jean-Pierre Lozi , Florian David , Gaël Thomas , Julia Lawall , Gilles Muller, Remote core locking: migrating critical-section execution to improve the performance of multithreaded applications, Proceedings of the 2012 USENIX conference on Annual Technical Conference, p.6-6, June 13-15, 2012, Boston, MA
8. Numa And Uma And Shared Memory Multiprocessors Computer Science Essay https://www.ukessays.com/essays/computer-science/numa-and-uma-and-shared-memory-multiprocessors-computer-science-essay.php.
9. Lock-free Programming Talk at Cppcon 2014 by Herb Sutter https://www.youtube.com/watch?v=c1gO9aB9nbs&list=PLLx8RvOpJ0wlFCGxWAVBTo3CoR9PYkpz2
10. An Introduction to Lock-Free Programming http://preshing.com/20120612/an-introduction-to-lock-free-programming/
11. Lock-Free Programming - Geoff Langdale https://www.cs.cmu.edu/~410-s05/lectures/L31_LockFree.pdf
12. P. J. Landin, "The mechanical evaluation of expressions," The Computer Journal, vol. 6, no. 4, pp. 308–320, 1964 https://www.cs.cmu.edu/~crary/819-f09/Landin64.pdf
13. An Experiment in Wait-Free Synchronisation of Priority-Controlled Simultaneous Processes: Guarded

Sections, 2015 https://opus4.kobv.de/opus4-fau/frontdoor/index/index/year/2015/docId/6061
14. The incremental garbage collection of processes, 1977 https://dl.acm.org/citation.cfm?id=806932
15. Técnicas de Processamento Assíncrono https://ine5646.gitbook.io/livro/javascript/tecnicas-de-processamento-assincrono
16. How JavaScript works: *Event loop* and the rise of Async programming + 5 ways to better coding with async/await https://blog.sessionstack.com/how-javascript-works-event-loop-and-the-rise-of-async-programming-5-ways-to-better-coding-with-2f077c4438b5
17. C++11 Multithreading – Part 8: *std::promise* , *std::promise* and Returning values from Thread https://thispointer.com/c11-multithreading-part-8-stdfuture-stdpromise-and-returning-values-from-thread/
18. CPU intensive javascript computations without blocking the single thread https://benjaminhorn.io/code/cpu-intensive-javascript-computations-without-blocking-the-single-thread/
19. Promise vs Observable https://stackoverflow.com/questions/37364973/promise-vs-observable
20. Promises vs Observables https://medium.com/@mpodlasin/promises-vs-observables-4c123c51fe13
21. When does a thread become idle? https://stackoverflow.com/questions/19784293/when-does-a-thread-become-idle
22. Aplicações para Web Progressivas (PWA), Service Workers https://ine5646.gitbook.io/livro/parte-ii-modelagem-de-aplicacoes-para-web/finalidade/orientadas-a-pagina/progressiva-pwa