G.C.C. 7.2.0 for EPOS IA32

Authors

Evandro Chagas Ribeiro da Rosa

Lucas Cavalcante de Sousa

Otto Menegasso Pires

Motivation

Due to outdated version of current EPOS' compiler, the latest standard available is C++0x, a previous version of C++11. Furthermore many features cannot be used by the epos user, features such as the ones listed bellow.

Some C++ new features unimplemented in C++0x of GCC 4.4.4

Lambda

Lambda expressions were implemented in C++11 and are a mechanism for specifying a function object. The primary use for a lambda is to specify a simple action to be performed by some function.

Lambda expressions bring some advantages, one of them is making the code easier to read. This might sound a little weird because some programmers think that lambda expressions by themselves are ugly, but consider this case:

An example of functional programming in c++

```
std::for_each( begin, end, doer );
```

The problem with this is that the function (object) doer:
- Specifies **what's done** in the loop
- Yet somewhat **hides what's actually done** (you have to look up the function object's operator()'s implementation)
- Must be **defined in a different scope** than the std::for_each call
- Contains a certain amount of **boilerplate code**
- Is often **throw-away code** that's not used for anything but this one loop construct

Lambda considerably improves on the aspects listed above, using as example the std::sort, where the third argument is a binary function that accepts two elements in the range as arguments and returns a value convertible to bool.

On pre C++11 we could sort a vector in reverse order that way:

```
bool comp(int a, int b) { return a > b; } int main() { std::vector<int> vint = {32,71,12,45,26,80,53,33}; std::sort(vint.begin(), vint.end(), comp); }
```

In C++11 we can use a lambda as the function comp:

```
int main() { std::vector<int> vint = {32,71,12,45,26,80,53,33}; //auto = bool (*)(int a, int b) auto comp = [](int a, int b) { return a > b; }; std::sort(vint.begin(), vint.end(), comp); }
```

By bringing the implementation of the function closer to where it is used the Lambda Expression makes the code easier to read . That way you don’t need to break the reading flow of your code because you had to look elsewhere.

An interesting aspect of lambda expressions in C++ is your capacity to capture variables. By “capture” we mean you can use a variable declared outside of the lambda inside the expression. Or, if you are using C++14 or above, your captured variable can have an initializing expression inside the lambda.

Captured variables can have an initializing expression:

```
auto timer = [val = system_clock::now()] { return system_clock::now() - val; }; // ... do stuff ... timer(); // returns time since timer creation
```

In this example, val is assigned the current time which is then returned by the lambda expression. val doesn't need to be an existing variable, so this is in effect a mechanism for adding data members to the lambda. The types of these members are inferred by the compiler.

Also, in C++14 we are allowed to capture move-only variables:

```
auto p = make_unique<int>(10); auto lmb = [p = move(p)] { return *p; }
```

Null pointer constant

The constant NULL can be implemented like:

```
#define NULL 0
```

It may lead to ambiguity:

```
int foo(int); int foo(int*); foo(NULL); // don't know if use foo(int) or foo(int*)
```

To solve this ambiguity nullptr can be user instead of NULL, because nullptr is always consider a pointer type:

```
int foo(int); int foo(int*); foo(nullptr); // always use foo(int*)
```

Forward declarations for enums

Now you can just declare an enum without specifying its elements, but you need to declare its storage type.

It can be useful in headers like:

```
enum number : unsigned; std::ostream& operator<<(std::ostream &os, number &n);
```

So the enum can be specified at another file.

Structured Bindings

Structured Bindings provide an easy way to access all values of an tuple or user-defined type.

Consider the following struct:

```
struct val{ int a; char b; float c; };
```

On Pre C++17 we would access each value individually as:

val aa = { 12,'a',3.123 }; auto i = aa.a; auto f = aa.b; auto g = aa.c;

Or using std::tie to unpack all of them as follows:

val aa = { 12,'a',3.123 }; int i; char f; float h; std::tie(i,f,g) = aa;

On C++17 we could use automatic type deduction combined with structured bindings to access all values in a single line v.g.:

val aa = { 12,'a',3.123 }; auto [i, f, g] = aa;

std::map<int,int> myMap = {{1,2},{3,4}}; for (const auto & [k,v] : myMap) { // now you can using the map iterator }

Init Statement in conditionals

This structure enables an init statement to if and switch as the init statement that while and for already have. As the while and for init statement, there is no leaking into the ambient scope and this structure prevents an explicit scope.

Consider the following leaking example:

auto var = foo(); if (var != SUCCESS) { // ... } else { // ... }

On above example var is leaking into the ambient scope. We could explicit create a new scope, avoiding the leaking:

{ auto var = foo(); if (var != SUCCESS) { // ... } else { // ... } }

At least with the new Init Statement we can avoid the leaking and the explicit scope as:

if (auto var = foo(); var != SUCCESS) { // ... } else { // ... }

In resume, the init statement work as following:

if (init; cond) { // ... }

switch (init; cond) { // ... }

Inline Variables

This feature extends the idea of header defined inline functions to variables and constant. An inline variable means that it is immediately defined and potentially repeated between the translation units, and that the compiler ensures they will be the same between then. So now it is possible to define static variables or global variables at header level, letting the linker to always refer to a same entity even if the definition is seen more time due to multiple inclusion inside different translation units.

So it is possible to do:

```
struct MyClass { static inline const int value = 123; };
```

Instead of:

```
struct MyClass { static const int value; };
```

With a separate source file to redefine it as following:

```
const int MyClass::value = 123;
```

Constexpr if

This feature provide a static-if, allowing branches of an if statement to be discard at compile-time based on a constant conditional expression.

```
if constexpr(cond) statement1; // Discarded if cond is false else statement2; // Discarded if cond is true
```

It can be well used on metaprogramming/template code.

e.g.: a metaprogramming fibonacci:

```
template<int N> constexpr int fibonacci() {return fibonacci<N-1>() + fibonacci<N-2>(); } template<> constexpr int fibonacci<1>() { return 1; } template<> constexpr int fibonacci<0>() { return 0; }
```

Now it could be implemented with constexpr if as following:

```
template<int N> constexpr int fibonacci() { if constexpr (N>=2) return fibonacci<N-1>() + fibonacci<N-2>(); else return N; }
```

Folding expressions

This features make possible to performe an logic/arithmetic operation over a parameter pack. This could be done on post C++11 by doing a recursion.

e.g. A sum:

```
auto sum(){ return 0; } template<typename T1, typename ...T> auto sum(T1 s, T... ts) { return s + sum(ts...); }
```

Now it could be done simpler as:

```
template<typename ...Args> sum(Args ...args){ return (args + ...); }
```

This feature suportes the following operations:

```
+ - * / % ^ & | ~ = < > << >> += -= *= /= %= ^= &= |= <<= >>= == != <= >= && || , .* ->*
```

Moreover, the fold expressions has four forms and they differ by having or not an initial value and by the direction that the operation is reduced.

```
```

(... op pack)
Where pack is an unexpanded parameter pack, and op is one of the operations listed above. Here the expression is reduced from left to right with op.

```
```

(pack op ...)
Here the expression is reduced from right to left with op.

```
```

(init op ... op pack)
Where init is an initial value. Here the expression is reduced from left to right with op and initial value init.

```
```

(pack op ... op init)
Here the expression is reduced from right to left with op and initial value init.

The initial value is crucial when the binary operation op has no default value, that is, when an operation does not have an default value for empty packs.

So...

Our main motivation is to make new features, as the ones listed above, available for the epos user and, if possible, to modernize EPOS making it even more readable.

Goals

In order to make available features from the latest C++ standard (as the ones listed above) , we will upgrade the EPOS G.C.C. compiler to 7.2.0, the newest version available.

Methodology

- Upgrade: Update to subsequent C++ standard version.
- Refactoring: Correct errors due to previous upgrade.
- Coding: Implement and test an EPOS application with the new standard.
- Restart: Repeat the process.

Tasks

1. Development of the project plan and literature review.
2. Compile G.C.C 7.2.0 for IA32 and make it work with EPOS.
3. Upgrade EPOS to C++14 standard.
4. Upgrade EPOS to C++17 standard.
5. Upgrade some EPOS' feature to C++17 standard.

Deliverables

- **D1** - Development of the project plan and literature review
- **D2** - Toolchain used to make a G.C.C. 7.2.0 compatible EPOS and any required changes on EPOS to work properly with the new G.C.C. version.
- **D3** - EPOS compatible with C++14 and an EPOS application with a feature from the C++14.

- **D4** - EPOS compatible with C++17 and an EPOS application with a feature from the C++17.
- **D5** - Some EPOS' feature with C++17 .

Schedule

Task	W1	02/10	W3	W4	W5	30/10	06/11	13/11	W9/11
Task1	x	D1							
Task2		x	x	x	x	D2			
Task3						x	D3		
Task4							x	D4	
Task5									D5

Toolchain

Our objective is to use `C++17` on EPOS. To active this out first step is to update EPOS's toolchain. The EPOS' toolchain is script that create an GCC cross-compiler for IA32 architecture compatible with EPOS. The new toolchain was build with Binutils 2.28.1, GCC 7.2.0 and Newlib 2.5.0. We used the script `toolchain.sh` to build it.

To make sure that the toolchain was not based GLibC, we used `prove.sh` to compile and link a simple program and list its symbols, that way we could see if there is any link with GLibC. We get the following output that guarantees we are not using GLibC.

```
00400014 B __bss_start 00400004 d __CTOR_END__ 00400000 d __CTOR_LIST__ 0040000c d __DTOR_END__ 00400008 d __DTOR_LIST__ 00400014 D _edata 004001b8 B _end 00000050 T __epos_app_entry 00000b80 T _exit 00000b60 T __exit 00002eb0 T _fini 0000246a t .GO 00000000 T _init 00400020 b initialized.877 00000040 T main 00000b90 T _panic 00000960 T _print 00000048 T _start
```

We have some problem running the toolchain script. In some computers its gives an error related to libstdc++-v3, that we couldn't figure out.

The specs from the computer used are listed below and the installed packages are listed in `packages.txt` and with more information in `packages_infos.txt`.

OS: Arch Linux x86_64
Host: 80E6 Lenovo Z40-70
Kernel: 4.13.8-1-ARCH
CPU: Intel i7-4500U (4) @ 3.000GHz
GPU: Intel Integrated Graphics
GPU: NVIDIA GeForce 840M

All files mentioned are available at https://gitlab.com/evandro-crr/epos-ia32-gcc-7.2.0.git and you can download the GCC 7.2.0 for IA32 archtecture compatible with EPOS at https://gitlab.com/evandro-crr/epos-ia32-gcc-7.2.0/tree/master/ia32-gcc-7.2.0

Refactoring EPOS

To use `std=c++17` to make EPOS we had to refactor the source code to the new standard. It was a gradual work, first we changed to `std=c++11`, then to `std=c++14` and finally to `std=c++17`. All process can be seen at this Git repository. At the time we successfully compiled the EPOS under a new standard we marked a commit with a tag.

To build the EPOS' trunk with the new GCC using `make` we need to specify that we want to use our new compiler. There are two options:

- add the `epos-gcc-7.2.0/bin` to `$PATH`, and change EPOS' makedefs with `ia32_COMP_PREFIX := epos-ia32-`
- change EPOS' makedefs with `ia32_COMP_PREFIX := <path-to-epos-gcc-7.2.0/bin>/epos-ia32-`

Errors while compiling EPOS to C++11

Here we describe all the erros we have found while compiling EPOS with `-std=c++11` and executing `make`. We also described their meaning and how we solved them. All sections are related to Git commits in our repository, thus they can be replicated at will.

sector .ctors and .dtors changed names to .init_array and .fini_array

Around six years ago, GCC adopted the .init_array and .fini_array as a mean to initialize global variables. This new standard has already been supported by glibc since 1999 and its main features are:

- It guarantees the priority in the initialization of variables.
- It addresses an old quirk of the .ctors that is the fact it is iterated backwards, making the accesses to the disk really slow.

During the global variables' initialization the method `__do_global_ctors_aux`, called by `_init`, iterating throw the global constructors that was stored at sector `.ctors`.

Due to the change of sectors name, the application falls to an error during the initialization. You can see that **`CTOR_END`**, where the iteration begin, point to an empty sector.

sector .ctors

```
□□□□□□□□
```

00400000 <__CTOR_LIST__>: 400000: ff (bad) 400001: ff (bad) 400002: ff (bad) 400003: ff 00 incl (%eax) 00400004 <__CTOR_END__>: 400004: 00 00 add %al,(%eax)

It should be some thing like this:

sector .ctors

```
□□□□□□□□
```

00400000 <__CTOR_LIST__>: 400000: ff (bad) 400001: ff (bad) 400002: ff (bad) 400003: ff 60 00 jmp *0x0(%eax) 400006: 00 00 add %al,(%eax) 400008: 10 02 adc %al,(%edx) 40000a: 00 00 add %al,(%eax) 40000c: a0 05 00 00 10 mov 0x10000005,%al 400011: 06 push %es ... 40002c: 30 51 00 xor %dl,0x0(%ecx) 40002f: 00 00 add %al,(%eax) 400031: 52 push %edx 00400034 <__CTOR_END__>: 400034: 00 00 add %al,(%eax)

But there was a sector called `.init_array`. After some failed attempts to use the sector .init_array we decide to just disable the `init/fini_array` with flag `--disable-initfini-array` while compiling GCC. That makes the compiler does not use any of the new features. But we still had to change the sections

names from “.ctors” and “.dtors” to “.init_array” and “.fini_array” even to access the constructors and destructors the old way.

src/architecture/ia32/ia32_crtend.c

```
void _init() __attribute__ ((section(".init"))); typedef void (*fptr)(void); static fptr __CTOR_END__[1] __attribute__((section(".init_array"))) = { (fptr)0 }; static fptr __DTOR_END__[1] __attribute__((used, section(".fine_array"))) = { (fptr)0 }; static void __do_global_ctors_aux() { fptr * p; for(p = __CTOR_END__ - 1; *p != (fptr) -1; p--) (*p)(); } void _init() { __do_global_ctors_aux(); } void __epos_app_entry() __attribute__ ((section(".init"), weak, alias ("_init")));
```

src/architecture/ia32/ia32_crtbegin.c

```
void _fini() __attribute__ ((section(".fini"))); typedef void (*fptr) (void); static fptr __CTOR_LIST__[1] __attribute__ ((used, section(".init_array"), aligned(sizeof(fptr)))) = { (fptr)(-1) }; static fptr __DTOR_LIST__[1] __attribute__ ((section("..fini_array"), aligned(sizeof(fptr)))) = { (fptr)(-1) }; static void __do_global_dtors_aux() { fptr * p; for(p = __DTOR_LIST__ + 1; *p; p++) (*p)(); } void _fini() { static int initialized = 0; if(!initialized) { initialized = 1; __do_global_dtors_aux(); } }
```

If you want to know why this new standard was adopted we recommend reading this bugzilla discussion thread .

union doesn't support flexible array member

Error log

```
union doesn't support flexible array member: In file included from bignum.cc:3:0: include/utility/bignum.h:29:20: error: flexible array member in union Digit data[]; ^ include/utility/bignum.h:33:20: error: flexible array member in union Digit data[]; ^
```

C specifies that, as a special case, the last element of a structure with more than one named member may have an incomplete array type; this is called a flexible array member. These flexible array members may be defined only in structures, not in unions.

As a solution we used a C struct hack, we initialized the array with size 0. Actually this is a GCC non-standard extension, while compiling with GNU Standard (using `-pedantic-errors`) it does implies an error, but even in this case, it is possible to bypass by initializing the array with size 1 without getting compiling errors about this matter. As this is not an issue in this project, the array was initialized with the C struct hack.

calloc signature has changed

Error log

```
In file included from malloc.cc:4:0: include/utility/malloc.h: In function 'void* calloc(size_t, unsigned int)': include/utility/malloc.h:21:19: error: declaration of 'void* calloc(size_t, unsigned int)' conflicts with built-in declaration 'void* calloc(long unsigned int, long unsigned int)' [-Werror=builtin-declaration-mismatch] inline void * calloc(size_t n, unsigned int bytes) { ^
```

The calloc signature in C++ is `void* calloc(size_t, size_t)`, as size_t is always big enough to store the biggest theoretically possible non-pointer object. On the newer version it is equivalent to `long unsigned`. To avoid further issues with maximum size of variables we changed the signature to `void*`

`calloc(size_t, size_t)` (on our C++11 Git Tag though, it's still with `long unsigned`) ; since the size_t type changes his size accordingly.

insert_first, insert_tail and insert_head was not declared in the scope

Error log

```
include/utility/list.h:486:25: error: 'insert_first' was not declared in this scope, and no declarations were found by argument-dependent lookup at the point of instantiation [-fpermissive] insert_first(e); ~~~~~~~~~~~~~^~~ include/utility/list.h:486:25: note: declarations in dependent base 'EPOS::S::U::Simple_List<EPOS::S::U::Data_Observer<EPOS::S::U::Buffer<EPOS::S::NIC, EPOS::S::Ethernet::Frame, void, EPOS::S::U::Dummy>, short unsigned int>, EPOS::S::U::List_Elements::Singly_Linked_Ordered<EPOS::S::U::Data_Observer<EPOS::S::U::Buffer< EPOS::S::NIC, EPOS::S::Ethernet::Frame, void, EPOS::S::U::Dummy>, short unsigned int>, short unsigned int> >' are not found by unqualified lookup include/utility/list.h:486:25: note: use 'this->insert_first' instead include/utility/list.h:497:28: error: 'insert_tail' was not declared in this scope, and no declarations were found by argument-dependent lookup at the point of instantiation [-fpermissive] insert_tail(e); ~~~~~~~~~~~~~^~~ include/utility/list.h:497:28: note: declarations in dependent base 'EPOS::S::U::Simple_List<EPOS::S::U::Data_Observer<EPOS::S::U::Buffer<EPOS::S::NIC, EPOS::S::Ethernet::Frame, void, EPOS::S::U::Dummy>, short unsigned int>, EPOS::S::U::List_Elements::Singly_Linked_Ordered<EPOS::S::U::Data_Observer<EPOS::S::U::Buffer< EPOS::S::NIC, EPOS::S::Ethernet::Frame, void, EPOS::S::U::Dummy>, short unsigned int>, short unsigned int> >' are not found by unqualified lookup include/utility/list.h:497:28: note: use 'this->insert_tail' instead include/utility/list.h:501:28: error: 'insert_head' was not declared in this scope, and no declarations were found by argument-dependent lookup at the point of instantiation [-fpermissive] insert_head(e); ~~~~~~~~~~~~~^~~ include/utility/list.h:501:28: note: declarations in dependent base 'EPOS::S::U::Simple_List<EPOS::S::U::Data_Observer<EPOS::S::U::Buffer<EPOS::S::NIC, EPOS::S::Ethernet::Frame, void, EPOS::S::U::Dummy>, short unsigned int>, EPOS::S::U::List_Elements::Singly_Linked_Ordered<EPOS::S::U::Data_Observer<EPOS::S::U::Buffer< EPOS::S::NIC, EPOS::S::Ethernet::Frame, void, EPOS::S::U::Dummy>, short unsigned int>, short unsigned int> >' are not found by unqualified lookup include/utility/list.h:501:28: note: use 'this->insert_head' instead
```

As it is, we could not understand why this methods are out of scope since class Simple_Ordered_List is an specialization of Simple_List and those methods are implemented there. But explicitly using them was enough to solve it.

size of _pmc_handler is CHANNELS again:

Error log

```
include/architecture/ia32/pmu.h: In static member function 'static void EPOS::S::PMU_handler<VERSION>::perf_int_init() [with int VERSION = 6]': include/architecture/ia32/pmu.h:1155:17 error: iteration 2 invokes undefined behavior [-Werror=aggressive-loop-optimizations] _pmc_handler[i] = 0; ^~~~~~~~~~~~~ include/architecture/ia32/pmu.h:1154:27 note: within this loop for (int i=0;i<8; i++) ~^~ include/architecture/ia32/pmu.h: In static member function 'static void EPOS::S::PMU_handler<VERSION>::PMU_int_handler(const unsigned int&) [with int VERSION = 6]': include/architecture/ia32/pmu.h:1201:34 error: iteration 2 invokes undefined behavior [-Werror=aggressive-loop-optimizations] if ((_pmc_handler[i] != 0) && ((perf_ovf_msr&(1ULL << i)) !=
```

0)) ~~~~~~~~~~~~~~~~~~^~~~~ include/architecture/ia32/pmu.h:1198:29 note: within this loop for (Reg32 i = 0; i < CHANNELS-1; i++) ~~^~~~~~~~~~~~

As we could understand `-O2` optimization brings `-faggressive-loop-optimizations` optimization, this flag seems to disable the for condition, so GCC infer that inside the for boundary there are no signed integer overflows or out-of-bound array accesses. And at some point it can causes an infinite loop. As it seems when using pointer arithmetics GCC can not use `-faggressive-loop-optimizations` at those for's, "solving" the error.

array subscript is above array bounds, GCC maybe the wrong one

Error log

e100_init.cc: In static member function 'static void EPOS::S::E100::init(unsigned int ': e100_init.cc:62:18: error: array subscript is above array bounds [-Werror=array-bound] _devices[unit].device = dev; ~~~~~~~~~~~~~~^ e100_init.cc:63:18: error: array subscript is above array bounds [-Werror=array-bound] _devices[unit].interrupt = IC::irq2int(irq); ~~~~~~~~~~~~~~^

It is not the first time that GCC pointed false-positives "array subscript is above array bounds" error, but we could not be sure. Another anomaly is that any new access to this array causes this error. Changing from normal array access to pointer arithmetics access seems to solve it. When disabling all optimizations and using normal array access, there is no problem, so it make us conclude that it is caused by an optimization.

Define Connection destructor as virtual

Error log

In file included from include/channel.h:7:0, from include/communicator.h:6, from include/dhcp.h:8, from dhcp.cc:6: include/tcp.h: In static member function 'static EPOS::S::TCP::Connection* EPOS::S::TCP::attach(EPOS::S::TCP::Observer*, const Port&, const Address&)': include/tcp.h:276:20: error: deleting object of polymorphic class type 'EPOS::S::TCP::Connection' which has non-virtual destructor might cause undefined behavior [-Werror=delete-non-virtual-dtor] delete conn; ^~~~ include/tcp.h: In static member function 'static void EPOS::S::TCP::detach(EPOS::S::TCP::Observer*, EPOS::S::TCP::Connection*)': include/tcp.h:286:16: error: deleting object of polymorphic class type 'EPOS::S::TCP::Connection' which has non-virtual destructor might cause undefined behavior [-Werror=delete-non-virtual-dtor] delete conn; ^~~~

Connection is derived from multiple classes and because one of them has virtual functions, the connection destructor must be virtual, this happens because it is unsafe to delete an instance of a derived class through a pointer to a base class if the base class does not have a virtual destructor, because there will be leak if the derived class (i.e. Connection) has any dynamically allocated objects.

suggest parentheses around operand of '!'

Error log

tcp.cc: In member function 'void EPOS::S::TCP::Connection::close_wait()': tcp.cc:704:12: error: suggest parentheses around operand of '!' or change '&' to '&&' or '!' to '~' [-Werror=parentheses] if(!_current->header()->flags() & RST) ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~

This is a warning brought by `-Wall` and it is a way that GCC tells the user to take more attention to some operations that, in general, people get wrong. Explaining with parenthesis which operator is in the and-operation solves this.

Reg8 was undefined

Error log

include/tstp.h:210:9: error: 'typedef EPOS::S::CPU_Common::Reg8 EPOS::S::IEEE802_15_4::Reg8' is private within this context Reg8 length() const { return MTU; } // Fixme: placeholder ^~~~ In file included from include/tstp.h:3: , from tstp.cc:6: include/ieee802_15_4.h:18:23: note: declared private here typedef CPU::Reg8 Reg8; ^~~~

The code here seemed to be unfinished, we think the author just forgot to add the complete path so we completed it.

gcc sees System: :_preheap as char[[16], so reinterpret cast was necessary GCC now checks allocation size on replacement new and throw a warning if you try to construct some thing bigger than the allocated space.

Error log

init_system.cc: In constructor 'EPOS::S::Init_System::Init_System()': init_system.cc:45:42: error: placement new constructing an object of type 'EPOS::S::Segment' and size '20' in a region of type 'char [16]' and size '16' [-Werror=placement-new=] System::_heap_segment = new (&System::_preheap[0]) Segment(HEAP_SIZE, WHITE, Segment::Flags::SYS);

`_preheap` is declared as `static char _preheap(Traits<System>::multiheap ? sizeof(Segment) : 0) + sizeof(Heap);` and the placement new is made inside the if bellow.

src/init/init_system.cc

if(Traits<System>::multiheap) { System::_heap_segment = new (&System::_preheap[0]) Segment(HEAP_SIZE, WHITE, Segment::Flags::SYS); System::_heap = new (&System::_preheap[sizeof(Segment)]) Heap(Address_Space(MMU::current()).attach(System::_heap_segment, Memory_Map::SYS_HEAP), System::_heap_segment->size()); }

So inside the if we are sure that `_preheap` has enough space for a `Segment`, but GCC doesn't see that way. The alternative was to use a `reinterpret_cast<>` to bypass GCC verification.

This is the code now with the reinterpret_cast

if(Traits<System>::multiheap) { Segment *heap = reinterpret_cast<Segment*> (&System::_preheap[0]); new (heap) Segment(HEAP_SIZE, WHITE, Segment::Flags::SYS); System::_heap_segment = heap; System::_heap = new (&System::_preheap[sizeof(Segment)]) Heap(Address_Space(MMU::current()).attach(System::_heap_segment, Memory_Map::SYS_HEAP), System::_heap_segment->size()); }

declare Adapter destructor as virtual

Error log

In file included from kernel_binding.cc:4:0: include/framework/agent.h: In member function 'void EPOS::S::Agent::handle_ipc()': include/framework/agent.h:409:16: error: deleting object of polymorphic class type 'EPOS::S::Adapter<EPOS::S::Port<EPOS::S::IPC> >' which has non-virtual destructor might

cause undefined behavior [-Werror=delete-non-virtual-dtor] delete comm; ^~~~
This error is the same as in "Define Connection destructor as virtual", and the solution is the same too, making Adapter destructor virtual.

Perfect forwarding

C++11 introduce the concept of Rvalue reference with `&&`, you can learn more about it in this link.

That's what the following error is about.

Error log

include/framework/message.h:119:18: error: cannot bind rvalue reference of type 'const unsigned int&&' to lvalue of type 'const unsigned int' SERIALIZE(_parms, index, an ...); ~~~~~~~~~~^~~~~~~~~~~~~~~~~~~~~~~~ include/system/meta.h:196:6: note: initializing argument 3 of 'void EPOS::S::SERIALIZE(char*, int, const T&&) [with T = unsigned int]' void SERIALIZE(char * buf, int index, const T && a) { ^~~~~~~~~

SERIALIZE has as third argument a `const T &&` so it necessarily needs to be a Rvalue, and in order to pass a Lvalue we use `static_cast<T&&>` to make it a Rvalue. Another problem is that function `void Message::out(const Tn& ...)` calls SERIALIZE forwarding a parameter pack. Because it is a pack we can not deduce the type of the parameters inside it, so it was necessary to implement the function move bellow, that can infer its type and then cast it to Rvalue.

template <class T> inline T&& move(T& a) { return static_cast<T&&>(a); }

The function move was used to infer the type of `an` for the cast.

Change necessary in SERIALIZE call

- SERIALIZE(_parms, index, an ...); + SERIALIZE(_parms, index, move(an)...);

C++11 Example

Just to use some features from C++11, the application producer_consumer was changed transferring the function consumer to a lambda function.

producer_consumer_with_lambda.cc

#include <utility/ostream.h> #include <thread.h> #include <semaphore.h> #include <alarm.h> using namespace EPOS; const int iterations = 100; OStream cout; const int BUF_SIZE = 16; char buffer[BUF_SIZE]; Semaphore empty(BUF_SIZE); Semaphore full(0); int main() { auto consumer = []() { int out = 0; for(int i = 0; i < iterations; i++) { full.p(); cout << "C<-" << buffer[out] << "\t"; out = (out + 1) % BUF_SIZE; Alarm::delay(5000); empty.v(); } return 0; }; Thread * cons = new Thread(consumer); // producer int in = 0; for(int i = 0; i < iterations; i++) { empty.p(); Alarm::delay(5000); buffer[in] = 'a' + in; cout << "P->" << buffer[in] << "\t"; in = (in + 1) % BUF_SIZE; full.v(); } cons->join(); cout << "The end!" << endl; delete cons; return 0; }

The `Thread` constructor needs a `template <class... Ts> int (*)(Ts...)` as first parameter, but by using `auto` the type of consumer is defined as `main()::<lambda()>`.

Instead of use `auto consumer` we could use `int (*consumer)()`, but it is way less readable. So an alternative was to change the `Thread` constructor as follow.

old definitinon

```
template<typename ... Tn> Thread(int (* entry)(Tn ...), Tn ... an); template<typename ... Tn> Thread(const Configuration & conf, int (* entry)(Tn ...), Tn ... an);
```

new definition

```
template<class T, typename ... Tn> Thread(T entry, Tn ... an); template<class T, typename ... Tn> Thread(const Configuration & conf, T entry, Tn ... an);
```

By just being flexible with the parameter `entry`, and assigning it to a `template <class... Ts> int (*)(Ts...)` as `int (*_entry) (Tn...) = entry;` in the constructor's body, you pass any lambda that can be converted to a `template <class... Ts> int (*)(Ts...)`.

Errors while compiling EPOS to C++14

In this section we describe the only erro we have found while compiling EPOS with `-std=c++14` and executing `make`. As well its meaning and how we solved it.

By pass operator delete (void *ptr, long unsigned size)

Error log

```
lib/libsys_ia32.a(thread.o): In function `EPOS::S::Thread::~Thread()':
thread.cc:(.text._ZN4EPOS1S6ThreadD2Ev+0x178): undefined reference to `operator delete(void*, unsigned long)'
```

The C++14 add two sized delete signature: `void operator delete(void* ptr, std::size_t size)` and `void operator delete[](void* ptr, std::size_t size)`. The idea is to minimize the overhead at deleting objects by passing the control of how many bytes will be deleted to the programmer. When using deletes operations without size, the object structure keeps its size in bytes to be used when the operation is called. With the new ones, one could change how the object is structured to do not store this information, saving space and time (because there will not be necessary to search for its size) by specifying the size in bytes of the structured being deleted.
But to use it at EPOS it would be necessary to adapt the Heap implementation so it would not store the objects sizes, as well as add the support to this new delete operators.
Because our main goal is to compile EPOS with C++14 we just added some simple functions to bypass the necessity of the sized deletes, as follow:

include/utility/malloc.h

```
void operator delete(void * ptr, size_t); void operator delete[](void * ptr, size_t);
```

include/utility/malloc.h

```
void operator delete(void * object, size_t) { return free(object); } void operator delete[](void * object, size_t) { return free(object); }
```

Note that the error says that the undefined function is `delete(void*, unsigned long)` but we actually implemented `delete(void*, std::size_t)` (not in our c++14 tag) this happens because the delete signature uses `std::size_t` and its current implementation is `unsigned long`(as already discussed).

Errors while compiling EPOS to C++17

In this section we describe the only erro we have found while compiling EPOS with `-std=c++17` and executing `make`. As well its meaning and how we solved it.

register keyword was deprecated in C++11

Error log

> In file included from include/cpu.h:119:0, from include/utility/spin.h:6, from include/utility/heap.h:8, from heap.cc:3: include/architecture/ia32/cpu.h: In static member function 'static T EPOS::S::CPU::tsl(volatile T&)': include/architecture/ia32/cpu.h:343:20: error: ISO C++1z does not allow 'register' storage class specifier [-Werror=register] register T old = 1; ^~~ include/architecture/ia32/cpu.h: In static member function 'static T EPOS::S::CPU::finc(volatile T&)': include/architecture/ia32/cpu.h:350:20: error: ISO C++1z does not allow 'register' storage class specifier [-Werror=register] register T old = 1; ^~~ include/architecture/ia32/cpu.h: In static member function 'static T EPOS::S::CPU::fdec(volatile T&)': include/architecture/ia32/cpu.h:357:20: error: ISO C++1z does not allow 'register' storage class specifier [-Werror=register] register T old = -1; ^~~ include/architecture/ia32/cpu.h: In static member function 'static int EPOS::S::CPU::bsf(EPOS::S::CPU_Common::Log_Addr)': include/architecture/ia32/cpu.h:539:31: error: ISO C++1z does not allow 'register' storage class specifier [-Werror=register] register unsigned int pos; ^~~ include/architecture/ia32/cpu.h: In static member function 'static int EPOS::S::CPU::bsr(EPOS::S::CPU_Common::Log_Addr)': include/architecture/ia32/cpu.h:544:22: error: ISO C++1z does not allow 'register' storage class specifier [-Werror=register] register int pos = -1; ^~~ include/architecture/ia32/cpu.h: In instantiation of 'static T EPOS::S::CPU::tsl(volatile T&) [with T = bool]': include/utility/spin.h:55:31: required from here include/architecture/ia32/cpu.h:343:20: error: ISO C++1z does not allow 'register' storage class specifier [-Werror=register] register T old = 1; ^~~

The error addressed here is about the deprecated `register` keyword. It is a storage class specifier, a part of the *decl-specifier-seq* of a name's declaration syntax. Together with the scope of the name, storage class specifiers control two independent properties of the name: Its storage duration and its linkage.

The `register` specifier indicates automatic storage duration. And the most important thing to understand the `register` keyword: its presence could be used as a hint for the optimizer that the declared variable will be heavily used; it is a hint to store its value in a CPU register.

In C, the address of a `register` variable cannot be taken, but until C++17, a variable declared `register` was semantically indistinguishable from a variable declared without any storage class specifiers. And the hint given by `register` could be ignored, and in most implementations it would be ignored if the address of the variable was taken.

So, in C++17, unlike C and previous C++ standards, variables cannot be declared `register`. We then removed the `register` keyword from the declarations above. This change does not cause much impact because it was just a hint to the optimizer as well because of its specific use cases.

C++17 Example

Just to use some features from C++17, we created a *fibonacci* application that uses `constexpr if`.

fib_con_if.cc

```
#include <utility/ostream.h> using namespace EPOS; OStream cout; template<int N> int fibonacci() { if constexpr (N>=2) return fibonacci<N-1>() + fibonacci<N-2>(); else return N; } int main() { cout << fibonacci<44>(); return 0; }
```

To use a *constexpr if statement* we need that the if condition to be a *constant expressions*, to achieve this we used templates.

Conclusion

At the end of our project we reached some unexpected conclusions. We hoped that the new standard could bring some improvements to the actual design of EPOS, but we couldn't find anything that could make a big impact in embedded systems specifically. Although it was a little disappointing we don't think that all this work were for nothing. This could be the first step to upgrade EPOS to a future standard that really brings the wanted improvements.

Future Works

There are some aspects of our project that we couldn't fix while working on it. Our toolchain, as already metioned, only worked on a specific PC configuration and we couldn't replicated it on other machines.

One missing part of our work was a comparison between the assembly code generated by GCC 4.4.4 and GCC 7.2.0. This results would show if the new compiler is worth using.

We also hope that anyone who wants to upgrade EPOS to a new standard in the future can use our work as a base and have less trouble making the transition.

Bibliography

1. Danny Kalev, **The Biggest Changes in C++11 (and Why You Should Care)**, Available on: https://blog.smartbear.com/development/the-biggest-changes-in-c11-and-why-you-should-care/ , Accessed at: 09-24-2017;
2. Marius Bancila, **Ten C++11 Features Every C++ Developer Should Use**, Available on: https://www.codeproject.com/Articles/570638/Ten-Cplusplus-Features-Every-Cplusplus-Developer , Accessed at: 09-24-2017;
3. Bjarne Stroustrup, **C++11 - the new ISO C++ standard**, Available on: http://www.stroustrup.com/C++11FAQ.html , Accessed at: 09-24-2017;
4. **An overview of C++14 language features**, Available on: http://cpprocks.com/an-overview-of-c14-language-features/ , Accessed at: 09-24-2017;
5. Thomas Köppe, **Changes between C++14 and C++17 DIS**, Available on: https://isocpp.org/files/papers/p0636r0.html , Accessed at: 09-24-2017;
6. fenbf, **7 Features of C++17 that will simplify your code**, Available on: https://tech.io/playgrounds/2205/7-features-of-c17-that-will-simplify-your-code/introduction , Accessed at: 09-24-2017;
7. **Status of Experimental C++0x Support in GCC 4.4**, Available on: https://gcc.gnu.org/gcc-4.4/cxx0x_status.html, Accessed at: 09-29-2017;
8. **How do inline variables work?**, Available on: https://stackoverflow.com/questions/38043442/how-do-inline-variables-work , Accessed at: 09-30-2017;
9. **What are the use cases of inline variables introduced in C++17?**, Available on: https://www.quora.com/What-are-the-use-cases-of-inline-variables-introduced-in-C++17 , Accessed at: 09-30-2017;
10. **Inline specifier**, Available on: http://en.cppreference.com/w/cpp/language/inline , Accessed at: 09-30-2017;
11. **C++17: let's have a look at the constexpr if**, Available on:

http://filipjaniszewski.com/2016/07/29/c17-lets-have-a-look-at-the-constexpr-if/ , Accessed at: 30-09-2017
12. **Fold Expressions**, Available on: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4295.html , Accessed at: 01-10-2017;
13. **Folding expressions**, Available on: http://www.modernescpp.com/index.php/fold-expressions , Accessed at: 01-10-2017;
14. **Replace .ctors/.dtors with .init_array/.fini_array on targets supporting them**, Available on: https://gcc.gnu.org/bugzilla/show_bug.cgi?id=46770 , Accessed at: 10-26-2017;
15. **Does union support flexible array members?**, Available on: https://stackoverflow.com/questions/46233400/does-union-support-flexible-array-members/46233509#46233509 , Accessed at: 10-25-2017;
16. **Structure and union specifiers**, Available on: http://c0x.coding-guidelines.com/6.7.2.1.html , Accessed at: 10-25-2017;
17. **Difference between size_t and unsigned int?**, Available on: https://stackoverflow.com/questions/19732319/difference-between-size-t-and-unsigned-int , Accessed at: 10-26-2017;
18. **GCC pre-4.8 Breaks Broken SPEC 2006 Benchmarks**, Available on: https://blog.regehr.org/archives/918 , Accessed at: 10-28-2017;
19. **Options That Control Optimization**, Avalible on: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html , Accessed at: 10-28-2017;
20. **Wrong warnings "array subscript is above array bounds"**, Available on: https://gcc.gnu.org/bugzilla/show_bug.cgi?id=59124#c5 , Accessed at: 10-28-2017;
21. **How to get around GCC '*((void*)& b +4)' may be used uninitialized in this function warning while using boost::optional**, Available on: https://stackoverflow.com/questions/21755206/how-to-get-around-gcc-void-b-4-may-be-used-uninitialized-in-this-funct , Accessed at: 10-28-2017;
22. **C++ Rvalue References Explained**, Available on: http://thbecker.net/articles/rvalue_references/section_01.html, Accessed at 10-30-2017
23. **operator delete, operator delete[]**, Available on: http://en.cppreference.com/w/cpp/memory/new/operator_delete , Accessed at:11-04-2017
24. **Storage class specifiers** , Available on: http://en.cppreference.com/w/cpp/language/storage_duration , Accessed at: 11-12-2017;
25. **Remove Deprecated Use of the *register* Keyword**, Available on: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4340 , Accessed at: 11-12-2017;
26. **Deprecation of the *register* keyword**, Available on: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4193.html#809 , Accessed at: 11-12-2017;