EPOS for Raspberry Pi

Software/Hardware Integration Lab at UFSC
# Table of Contents

**EPOS for Raspberry Pi**  
Table of contents ....................................................................................................................... 1  
1. Running EPOS on Raspberry Pi ............................................................................................... 2  
   1.1. Compiling ................................................................. 2  
   1.2. Running and Debugging  ........................................... 2  
   1.3. Running Raspberry Pi3b in a real Hardware ................................................................. 2  
      1.3.1. Setting up the SD Card ........................................ 2  
      1.3.2. Connecting the UART to your PC .................... 3  
2. ARMv7 and AArch32 ................................................................................................................ 3  
3. Cortex-A53 ............................................................................................................................... 3  
4. Task Memory Model .................................................................................................................. 3  
5. MMU for Paging ........................................................................................................................... 3  
6. Task Context Switching ............................................................................................................. 3  
7. System Calls .............................................................................................................................. 3  
8. System Object Proxies and Agents ............................................................................................ 4  
9. System Call Security .................................................................................................................. 4  
10. Resource Management ............................................................................................................ 4  
11. Inter-Process Communication .................................................................................................. 4  
12. I/O ........................................................................................................................................... 4
EPOS for Raspberry Pi

Table of contents

- EPOS for Raspberry Pi
- 1. Running EPOS on Raspberry Pi
  - 1.1. Compiling
  - 1.2. Running and Debugging
  - 1.3. Running Raspberry Pi3b in a real Hardware
    - 1.3.1. Setting up the SD Card
      - 1.3.1.1. Firmware Files
      - 1.3.1.2. Application Image
    - 1.3.2. Connecting the UART to your PC
- 2. ARMv7 and AArch32
- 3. Cortex-A53
- 4. Task Memory Model
- 5. MMU for Paging
- 6. Task Context Switching
- 7. System Calls
- 8. System Object Proxies and Agents
- 9. System Call Security
- 10. Resource Management
- 11. Inter-Process Communication
- 12. I/O
1. Running EPOS on Raspberry Pi

1.1. Compiling

To compile an APP for Raspberry Pi3b, first configure the application Traits<Build> as follows:

```cpp
template<> struct Traits<Build>: public Traits<void>
{
    static const unsigned int MODE = LIBRARY;
    static const unsigned int ARCHITECTURE = ARMv7; // You can use ARMv8 or ARMv7 on QEMU.
    static const unsigned int MACHINE = Cortex;
    static const unsigned int MODEL = Raspberry_Pi3;
    static const unsigned int CPUS = 1; // or 4
    static const unsigned int NODES = 1; // (> 1 => NETWORKING)
    static const unsigned int EXPECTED_SIMULATION_TIME = 60; // s (0 => not simulated, using real hardware)
};
```

At the directory where you installed EPOS' source code, just type:

```
$ make APPLICATION=<appname>
```

1.2. Running and Debugging

To run and debug applications, follow the steps described in EPOS documentation.

1.3. Running Raspberry Pi3b in a real Hardware

First, to run an application in real Raspberry Pi3 hardware, use ARMv8 as the ARCHITECTURE in Traits<Build>. In EPOS, ARMv8 is very similar to ARMv7, it just replaces the cores() function in cpu.h, as Raspberry Pi3b hardware does not support the ARMv7 implementation of cores() function.

1.3.1. Setting up the SD Card

To boot a Raspberry Pi3b in a real hardware, you first need to configure an SD Card with the EPOS application image and some additional firmware files required by the Raspberry Pi3b hardware.

1.3.1.1. Firmware Files

The additional Firmware files required are available at the Raspberry Pi3b Official Github. From the firmware folder, you only need bootcode.bin and start.elf files. Setup the SD card with a single partition, fat32, and copy the bootcode.bin and start.elf files to the SD card.

1.3.1.2. Application Image

Raspberry Pi3b CPU boot is started by the GPU. The GPU reads the SD card and copies the kernel image to the specific initial address, where the first piece of code in this image is expected to be the Vector Table. The default image names are related to the compatibility to Pi models. You can specifically select your own name in config.txt. Considering no config.txt override, the search order for a Pi3 is:
if kernel8.img is found: boot in 64 bits mode
else if any of kernel8-32.img, kernel7.img, or kernel.img are found: boot in 32 bits mode

The address of the Vector Table changes from 64 and 32 bits modes. For 32 bits, the vector table is initially located at the address 0x00008000, and 0x00080000 for 64 bits.

Currently, EPOS supports only 32 bits Raspberry Pi3b. Thus, after compiling, copy the final application binary file to the SD card, renaming it to kernel8-32.img or kernel7.img or kernel.img.

1.3.2. Connecting the UART to your PC

Warning: Attain to the Raspberry Pi3b energy supply requirements, and to the correct pin connection when connecting the RaspberryPi3b UART / FTDI / PC.

EPOS uses Raspberry MiniUART as the default Serial Display. To connect the Raspberrypi 3b UART to your PC, an FTDI is needed to intermediate the UART pins and connect the EPOS app output to the PC USB over serial protocol. The following configuration is then needed:

<table>
<thead>
<tr>
<th>FTDI</th>
<th>Pi3b</th>
</tr>
</thead>
<tbody>
<tr>
<td>TX</td>
<td>RX (pin 10)</td>
</tr>
<tr>
<td>RX</td>
<td>TX (pin 8)</td>
</tr>
<tr>
<td>GND</td>
<td>GNC (pin 6)</td>
</tr>
</tbody>
</table>

After connecting the FTDI to PC using a USB cable, the Raspberry Pi3b output can be seen by reading the USB content (e.g., using minicom or cutecom). The UART configuration is the following:

<table>
<thead>
<tr>
<th>Baudrate</th>
<th>115200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data bits</td>
<td>8</td>
</tr>
<tr>
<td>Stop bits</td>
<td>1</td>
</tr>
<tr>
<td>Parity</td>
<td>None</td>
</tr>
<tr>
<td>Flow Control</td>
<td>None</td>
</tr>
</tbody>
</table>

2. ARMv7 and AArch32
3. Cortex-A53
4. Task Memory Model
5. MMU for Paging
6. Task Context Switching
7. System Calls
8. System Object Proxies and Agents
9. System Call Security
10. Resource Management
11. Inter-Process Communication
12. I/O