Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

An Updated Emulated Architecture to Support the Study of Operating Systems

Relatore:
Prof. Renzo Davoli

Presentata da:
Mattia Biondi

Correlatore:
Prof. Michael Goldweber

Sessione Straordinaria
Anno Accademico 2018/2019

dedica

Abstract

One of the most effective ways to learn something new is by actively practising it, and there is—maybe—no better way to study an Operating Systems course than by building your OS.

However, it is important to emphasize how the realization of an operating system capable of running on a real hardware machine could be an overly complex and unsuitable task for an undergraduate student. Nonetheless, it is possible to use a simplified computer system simulator to achieve the goal of teaching Computer Science foundations in the university environment, thus allowing students to experience a quite realistic representation of an operating system.

μMPS has been created for this purpose, a pedagogically appropriate machine emulator, based around the MIPS R2/3000 microprocessor, which features an accessible architecture that includes a rich set of easily programmable devices. μMPS has an almost two decades old historical development and the outcome of this following thesis is the third version of the software, dubbed μMPS3. This second major revision aims to simplify, even more, the emulator's complexity in order to lighten the load of work required by the students during the OS design and implementation. Two of these simplifications are the removal of the virtual memory bit, which allowed address translation to be turned on and off, and the replacement of the tape device, used as storage devices, with a new flash drive device—certainly something more familiar to the new generation of students.

Thanks to the employment of this software and the feed-backs received over the last decade, it has been possible to realize not just this following thesis, but also to develop some major improvements, which concern everything from the project building tools to the front-end, making μMPS a modern and reliable educational software.

Sommario

Contents

List of Tables

Chapter 1

Introduction

1.1 Background

The study and the consequent implementation of how an operating system works is, by now, a long-established and consolidated practice in every Computer Science's curriculum. It is, actually, one of the crucial components of a computer and it is responsible of ensuring it's basic operations, by coordinating and managing the system's resources like processor, memory, devices and processes, thus allowing hardware and software to interface each other.

This is probably the first real example of "big project" which students must experience, thanks to complex intercommunication system that has to exist between the different machine's components, and the study of it allows to comprehend the most common software engineering practices.

The approach on practical contexts is essential to fully understand how a machine works behind the theoretical notions studied in the early stages of the course of study, and it is usually followed by the debate of which is the best teaching choice concerning processor architectures.

Obviously, there is not only one way of how a central processor unit can be implemented, and while older realizations—although applicable for educational purposes—are now obsolete and incompatible with current platforms,

modern ones are designed to achieve high speed and quality, which makes them overly complex and unsuitable for the pedagogic experience.

Over the years, the MIPS architecture has become one of the landmarks in this teaching choice due to its clean and elegant instruction set, despite being excessively convoluted to student's perception, because of the high level of details obscuring the basic underlying features of it. A potential solution to this problem is the adoption of a simplified computer system simulator, like μMPS, to bring together an adequate level of understanding and a realistic representation of a real operating system.

1.2 History of μMPS

μMPS is based on the machine emulator MPS [1], designed and realized by Professor Renzo Davoli and one of his graduate students Mauro Morsiani, in the late 1990s at the University of Bologna.

The initial purpose was to bring back to life the layout and implementation experience of an operating system through an educational emulator, which could be run on real hardware. This practice was already possible in the past years, when the processor's architecture studied were the same available on real machines, but it has gone lost through the years because of the high-speed development of new more complex technologies used on physical implementations.

MPS was able to emulate the MIPS R3000 processor along with five other different device categories: disks, tapes, network adapters, printers and terminals.

The concerned CPU was genuinely emulated together with its complex virtual memory management system, which was the main subject of the feed-backs received during class testing of MPS as a pedagogical tool at the University of Bologna and Xavier University, in an undergraduate operating systems courses, taught respectively by Renzo Davoli and Michael Goldweber.

It was tested through *Kaya*'s [2] implementation, one of the variety of graduate-level projects that the emulator can support.

The urge of simplification led to the creation of μMPS, virtually identical to MPS but with the addition of a virtual memory management subsystem which had to resemble as much as possible to the conceptual one found in popular introductory OS texts. The only other difference was the new novice-friendly graphical user interface, significantly improved again in 2011 by Tomislav Jonjic during the development of the first major revision of the emulator, μMPS2 [3], which also implemented the support to up to sixteen MIPS R3000-style processors.

1.3 μMPS3

More than ten years have passed since the first release of μMPS and as many have passed from the moment it has been developed into a consolidated educational tool in the two already acknowledged teaching courses.

As already mentioned, the students' observations during this period were essential to in-depth testing the μMPS emulator as a pedagogical tool, showing as a result that further changes were needed. Some examples of these modifications would be the memory's structure and management system of both physical and virtual ones, a nuance probably still too complex and confusing, which could feel like a regression, since the attempt to simplify it has led to the removal of the VM bit. This was originally introduced in the first version of μMPS, as a distinctive feature from its predecessor MPS. On the other hand, other changes are related to the passage of the years, such as the replacement of memory tapes with flash drive devices, also know as "USB sticks" or "SD cards".

Tapes are probably still studied in their operation nowadays, but they cannot be found in today's practical contexts, and therefore it could be difficult to understand for the new generation of students, who can't find a match in current technological implementations.

Another consequence of μMPS's modernization has been the shift from the historical and well established Autotools to a more modern building tool, CMake, that speeds up the compilation process in addition to simplify the project's structure. It also fits best with the current emulator's graphical user interface, originally built upon Qt4, which also undergoes to the migration to the new version, Qt5.

These and more changes are collected together in a major release of the project, μMPS3, consequently bringing even more reliability in terms of pedagogical tool.

Due to implementation aspects—unlike his predecessor μMPS2—this new version is not backward compatible with older ones.

`here I could talk about the Debian Packaging I will do when everything is finalized`

1.4 Document's Structure

Chapter 2

Memory Management

As for the two prior versions, the memory subsystem of μMPS3, being based on the MIPS 3000 microprocessor, is divided into physical and virtual. Both parts have undergone significant modifications in this major revision, and they will be described in detail in this chapter along with the changes with respect to their old version. The main reasons for the modifications made are the further simplification of the work required by the user for the kernel implementation and the better clarification of the emulator's memory internal view which caused some confusion in the past.

2.1 Physical Memory

The physical address space is divided into two big areas: a kernel reserved space, from address `0x0000.0000` to `0x1FFF.FFFF`, and the installed RAM, from address `0x2000.0000` to `RAMTOP`.

This last value is calculated upon the value retrieved from the configuration file, settable from the machine's configuration dialog in the front-end emulator, which goes from a minimum of 8 to a maximum of 512 memory frames. Being the size of each frame of 4 kilobyte, μMPS3 can have from 32KB up top 2MB of installed RAM. Hence, the value of `RAMTOP` range from `0x2000.8000` to `0x2020.0000`.

The first area was reserved for:

- Execution ROM code, which layed in a read-only segment from address `0x0000.0000` to `ROMTOP`;
- prova.

This address space, corresponding to the first 0.5GB of physical memory, causes the raising of an Address Error when trying to access it while the processor was in user-mode.

2.2 TLB Floor Address

Come già citato nel cap. 2 (credo, memory), una delle conseguenze dell'implementazione del TLB Floor Adress come ultima parola del Bus Register Area è stato lo shifting (in su) di 1 parola (4 byte) delle due aree successive,Installed Devices Bit Map e Interrupting Devices Bit Map. Un'altra differenza rispetto ad uMPS2 è, dunque, il valore delle costanti da utilizzare durante il calcolo del starting address of a device's device register, che dovranno essere aumentate di una parola rispetto alla versione precedente.

2.3 VM Bit Removal

Chapter 3

Exception Handling

3.1 Primary Design

3.2 BIOS Data area

Chapter 4

Device Interfaces

Since its first implementation, besides the MIPS R3000 microprocessor, μMPS has always been able to emulate five different device categories: disks, tapes, network adapters, printers and terminals. Furthermore, it can support up to eight instances of each device type.

This chapter will provide the reasons why, along with this major revision, one of the devices has been replaced with a new one.

As for the functioning and implementation of the remaining devices, the reader may refer to the manual *μMPS2 Principles of Operation* [4].

4.1 Tapes

MPS has been conceived and developed in the late '90s, and many updates and improvements have followed to make it as it is nowadays.

It is simple to imagine how much can technology achieve in over 20 years, and μMPS' devices are not exempt from this process. Therefore, it may not be a surprise talking about how hard disk drives are, nowadays, less and less used, outclassed by new solid-state drives, despite still being common knowledge.

Network adapters are still present and properly taught in every IT class, although changed in their appearance, from physical to wireless, and certainly

much faster, but the speed of the devices is not what concerns μMPS, on the contrary, the goal is to teach their functioning.

Whereas there is not a real reason to think about the replacement of printers and terminals, there is one last device category about which the same cannot be said: tape devices.

In modern ages, tape devices are only studied in History of computer science's classes, and are no longer found in real implementations if not in rare cases. As a result, only older users still know what tapes are, their functioning and their use, therefore new generations are slowly losing consciousness of what they are or admittedly never heard of them.

Given all of this, it would have been a misstep not to use them, as when μMPS2 was released in 2011, technology was going through a transaction, and those who were still students knew of them. However, after almost ten years, the situation has changed, and tapes' final days have come.

Tapes, in μMPS2, were implemented as read-only DMA (Direct Memory Access) devices, capable of transferring 4KB blocks of data per time, concurrently for each installed device of this category. Internally, they were divided into equal sized frames of 4KB each, through the use of four marked codes: `EOT` (end-of-tape), `EOF` (end-of-file), `EOB` (end-of-block), `TS` (tape-start), to scan the reading process and simulating the movement of a head on a tape.

In order to let the students work with a storage device slower than disks, but still familiar to them, μMPS3 had to completely remove tapes to implement a new category of devices: flash drive devices.

4.2 Flash Devices

A flash drive device, also know as "USB sticks" or "SD cards", is a storage device using flash memory, which is a solid-state memory, therefore not implemented through physical disks or tapes, and can be electrically erased and reprogrammed. Unlike disks or tapes, a "seek" operation—that moves a head assembly over a physical surface—is not needed before reading from

or writing to a specific block or sector.

In order to reproduce the same experience in μMPS3, the easiest way is to take a similar implemented device, the disk, remove the "seek" operation from it and simplify the coordinates system (cylinder, head, sector) to a single contiguous block addressable space [`0...MAXBLOCKS-1`].

Hence, a flash drive device is a readable/writable DMA device, divided into equal-sized frames of the same dimension of μMPS3's framesize, 4KB, each one accessible via specific block number.

μMPS3 uses a 3 byte (24 bit) address space for flash device, consequently, each device in this category can have up to 2^{24} (`0xFFFFFF`) blocks of memory, which is equivalent to a maximum size of 64GB.

As already said, a flash drive device is usually slower than a disk device. However, the `umps3-mkdev` utility, which comes with μMPS3's installation and allows the user to create a disk or a flash device image, accepts a speed argument for both of them: "seek time" for disks and "write time" for flash devices. Therefore, this does not prevent anyone to voluntarily create a flash device significantly faster than a disk device.

The speed related argument is optional, and, if not given, `umps3-mkdev` will use default values:

```
#define DFLSEEKTIME 100
#define DFLWTIME    DFLSEEKTIME * 10
```

Like intuitively understable from parameters' name, the first one is for disks' default seek time, while the latter is for flash devices' default write time. This means that, if not voluntarily indicated by the user who creates a device, the flash devices' write time will always be ten times slower the disks' seek time.

Flash devices' read time can't be directly defined, but it will be calculated *ad-hoc* when needed by multiplying the device's set write time by:

```
#define READRATIO   3/4
```

This way, the read speed will always be 3/4 of the write speed, because → `WAIT FOR PROFS' ANSWER`

4.2.1 Creation

As `umps2-mkdev` allowed to create tape devices' image files, `umps3-mkdev` permits to create flash device ones via the following syntax (also visible by running the utility with no arguments):

```
$ umps3-mkdev -f <flashfile>.umps <file> [blocks [wt]]
```

where:

- `-f` : specify that we want to create a flash device image instead o a disk's one (`-d`);
- `<flashfile>` : the flash device image file's name that will be created;
- `<file>` : existing file to be written into the new flash device image;
- `blocks` : number of blocks [`1...0xFFFFFF`] (default = 512);
- `wt` : average write time (in microseconds) [1...100000] (default = 1000);

Going more in detail, after correctly running the previous command, the utility will decode command-line's argument, if existing.

Then, it will attempt to open, via a common `fopen`, the flash device's image file in write-mode. If the operation is successfull, a `FLASHFILEID` (`0x0153504D`) will be written into the first word of memory of the file by using an `fwrite` function. This file recognition tag is used to distinguish this type of file from all the other types of files generated or recognized by μMPS3.

Number of blocks and write speed are also subsequently written, which, along with the `FLASHFILEID`, compose the *header* of the flash device. These three parameters will be needed by μMPS3, when interacting with the device, to correctly simulate it.

Lastly, after opening it in read-mode by using an `fread` function, the existing file will be written inside the image file block-by-block.

Once the *end-of-file* indicator is reached, in case no errors have occurred, the flash device image file will be successfully created.

4.2.2 Usage

The use of a flash device, in μMPS3, is not significantly different from the use of any other device. As reported on the manual *μMPS2 Principles of Operation* [4]:

> Each single device is operated by a controller. Controllers exchange information with the processor via device registers; special memory locations. A device register is a consecutive 4-word block of memory. By writing and reading specific fields in a given device register, the processor may both issue commands and test device status and responses. μMPS3 implements the full-handshake interrupt-driven protocol. Specifically:
>
> 1. Communication with device i is initiated by the writing of a command code into device i's device register.
> 2. Device i's controller responds by both starting the indicated operation and setting a status field in i's device register.
> 3. When the indicated operation completes, device i's controller will again set some fields in i's device register; including the status field. Furthermore, device i's controller will generate an interrupt exception by asserting the appropriate interrupt line. The generated interrupt exception informs the processor that the requested operation has concluded and that the device requires its attention.
> 4. The interrupt is acknowledged by writing the acknowledge command code in device i's device register.
> 5. Device i's controller will de-assert the interrupt line and the protocol can restart. For performance purposes, writing a new command after the interrupt is generated will both acknowledge the interrupt and start a new operation immediately.

The flash device registers are located in low-memory, starting at `0x100000D4` for `flash0` up to `0x10000144` for `flash7`. Each register consists of 4 fields:

Field #	Address	Field Name
0	(base) + 0x0	STATUS
1	(base) + 0x4	COMMAND
2	(base) + 0x8	DATA0
3	(base) + 0xc	DATA1

Table 4.1: Device Register Layout

A flash device's register `STATUS` field is read-only and will contain one of the following status codes:

Code	Status	Possible Reason for Code
0	Device Not Installed	Device not installed
1	Device Ready	Device waiting for a command
2	Illegal Operation	Device presented unknown command
3	Device Busy	Device executing a command
4	Read Error	Illegal parameter/hardware failure
5	Write Error	Illegal parameter/hardware failure
6	DMA Transfer Error	Illegal physical address/hardware failure

Table 4.2: Flash Device Status Codes

From [4]:

> Status codes 1, 2, and 4-6 are completion codes. An illegal parameter may be an out of bounds value (e.g. a block number outside of [`0..MAXBLOCK-1`]), or a non-existent physical address for DMA transfers.

A flash device's register `COMMAND` field is read/writable and is used to issue commands to the device:

Code	Command	Operation
0	`RESET`	Reset the device interface
1	`ACK`	Acknowledge a pending interrupt
2	`READBLK`	Read the block located at `BLOCKNUMBER` and copy it into RAM starting at the address in `DATA0`
3	`WRITEBLK`	Copy the 4KB of RAM starting at the address in `DATA0` into the block located at `BLOCKNUMBER`

Table 4.3: Flash Device Command Codes

The format of the `COMMAND` field is illustrated in Table 4.4:

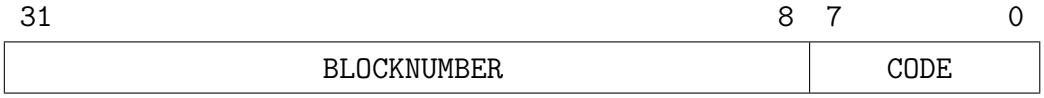

Table 4.4: Flash Device `COMMAND` Field

An operation on a flash device is started by loading the appropriate value into the `COMMAND` field. For the duration of the operation the device's status is "`Device Busy`". Upon completion of the operation an interrupt is raised and an appropriate status code is set; "`Device Ready`" for successful completion or one of the error codes. The interrupt is then acknowledged by issuing an `ACK` or `RESET` command.

A flash device's register `DATA0` field is read/writable and is used to specify the starting physical address for a read or write DMA operation. Since memory is addressed from low addresses to high ones, this address is the lowest word-aligned physical address of the 4KB block about to be transferred.

Each device's register DATA1 field is read-only and describes the physical characteristics of the device's geometry:

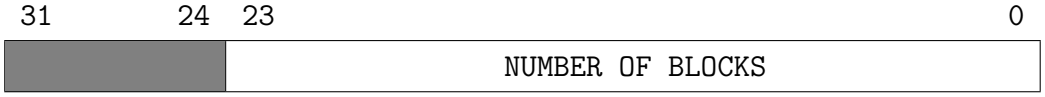

Table 4.5: Flash Device DATA1 Field

Chapter 5

Project Modernization

5.1 CMake Migration

5.2 Qt5 Transition

5.3 Logo and Icon Theme

Chapter 6

Linux Packaging

6.1 Debian Package

6.2 Arch Linux Package

Chapter 7

Conclusions

Knowing how an operating system works should be common knowledge and not something restricted only to the ones who studied in the IT field. If you are reading this document there are high chances you are doing it on a device of your property, which is running an operating system, and you should know how all of this really works in other to really feel like you own this particular system.

Bibliography

[1] M. Morsiani, R. Davoli, *Learning Operating Systems Structure and Implementation through the MPS Computer System Simulator*, in *The Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '99, (New York, NY, USA), pp. 63-67, ACM, March 1999.

[2] M. Goldweber, R. Davoli, and M. Morsiani, *The Kaya OS Project and the μMPS Hardware Emulator*, in *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '05, (New York, NY, USA), pp. 49–53, ACM, June 2005.

[3] M. Goldweber, R. Davoli, and T. Jonjic, *Supporting Operating Systems Projects Using the μMPS2 Hardware Simulator*, in *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '12, (New York, NY, USA), pp. 63–68, ACM, July 2012.

[4] M. Goldweber, R. Davoli, *μMPS2 Principles of Operation*, Lulu Books, August 2011.