Arquitecturas que se verán en la cátedra

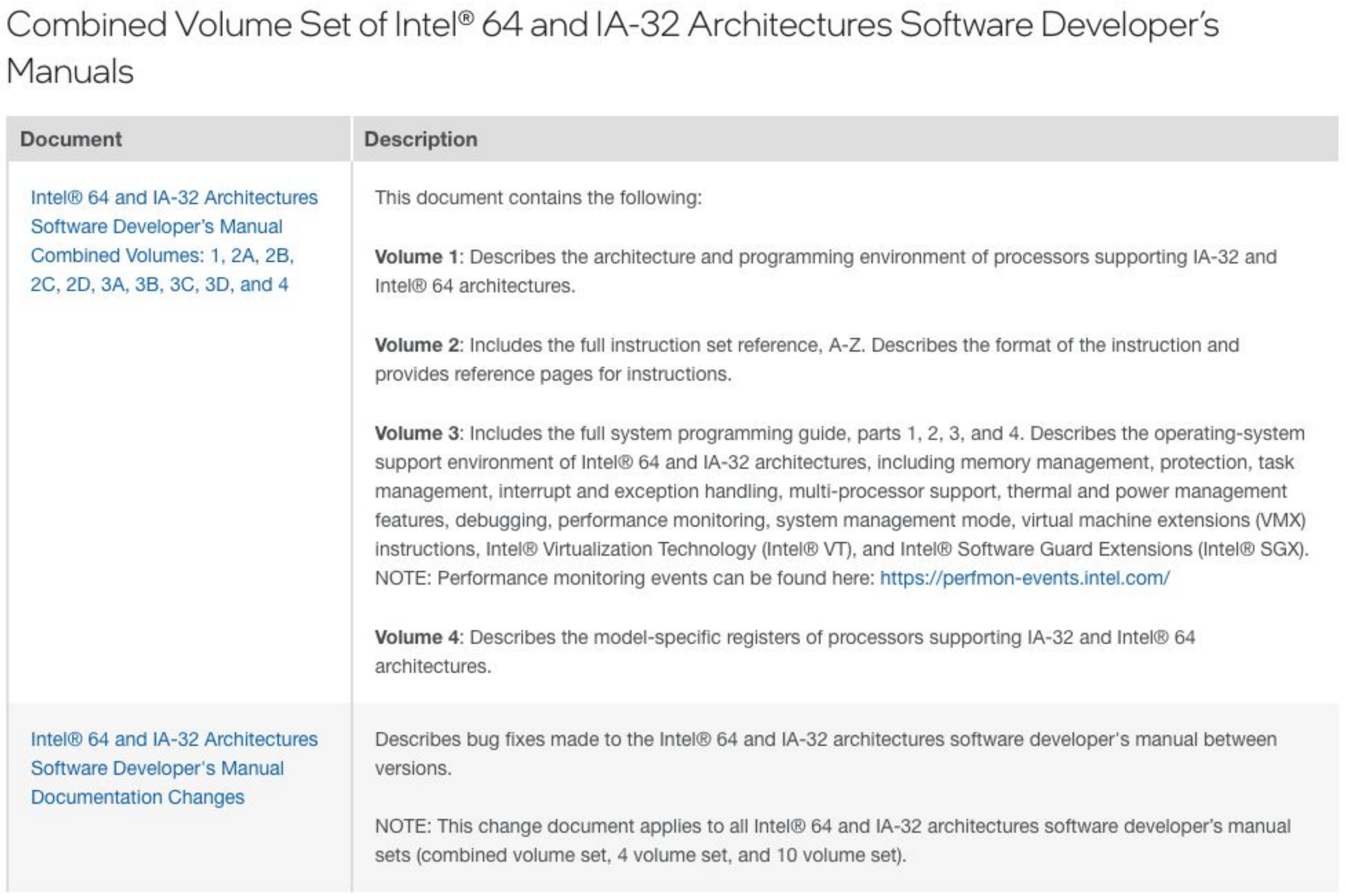
RISC-V

intel®

x86

ISA	Pages	Words	Hours to read	Weeks to read
RISC-V	236	76,702	6	0.2
ARM-32	2736	895,032	79	1.9
x86-32	2198	2,186,259	182	4.5

Figure 1.6: Number of pages and words of ISA manuals [Waterman and Asanović 2017a], [Waterman and Asanović 2017b], [Intel Corporation 2016], [ARM Ltd. 2014]. Hours and weeks to complete assumes reading at 200 words per minute for 40 hours a week. Based in part of Figure 1 of [Baumann 2017].

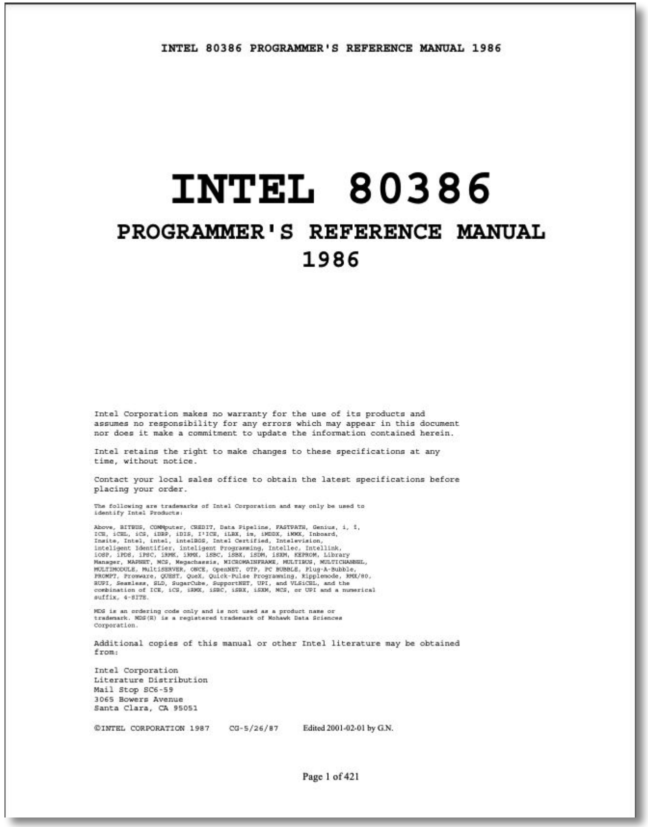


Combined Volume Set of Intel® 64 and IA-32 Architectures Software Developer's Manuals

Document	Description
Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4	This document contains the following: **Volume 1**: Describes the architecture and programming environment of processors supporting IA-32 and Intel® 64 architectures. **Volume 2**: Includes the full instruction set reference, A-Z. Describes the format of the instruction and provides reference pages for instructions. **Volume 3**: Includes the full system programming guide, parts 1, 2, 3, and 4. Describes the operating-system support environment of Intel® 64 and IA-32 architectures, including memory management, protection, task management, interrupt and exception handling, multi-processor support, thermal and power management features, debugging, performance monitoring, system management mode, virtual machine extensions (VMX) instructions, Intel® Virtualization Technology (Intel® VT), and Intel® Software Guard Extensions (Intel® SGX). NOTE: Performance monitoring events can be found here: https://perfmon-events.intel.com/ **Volume 4**: Describes the model-specific registers of processors supporting IA-32 and Intel® 64 architectures.
Intel® 64 and IA-32 Architectures Software Developer's Manual Documentation Changes	Describes bug fixes made to the Intel® 64 and IA-32 architectures software developer's manual between versions. NOTE: This change document applies to all Intel® 64 and IA-32 architectures software developer's manual sets (combined volume set, 4 volume set, and 10 volume set).

https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

Versión acotada del manual x86



INTEL 80386 PROGRAMMER'S REFERENCE MANUAL 1986

INTEL 80386

PROGRAMMER'S REFERENCE MANUAL
1986

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Distribution
Mail Stop SC6-59
3065 Bowers Avenue
Santa Clara, CA 95051

Page 1 of 421

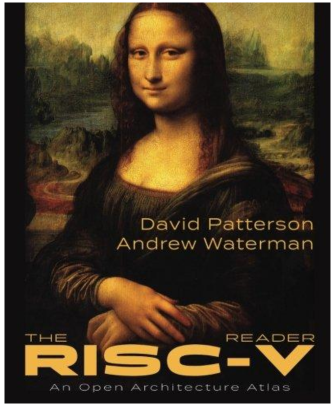
https://css.csail.mit.edu/6.858/2014/readings/i386.pdf

Especificaciones RISC-V

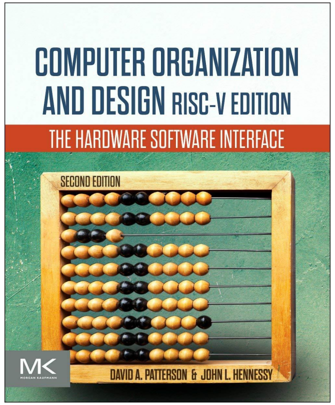
https://riscv.org/technical/specifications/

The RISC-V Instruction Set Manual
Volume I: Unprivileged ISA
Document Version 20191213
Editors: Andrew Waterman, Krste Asanovi´c, SiFive Inc.,

The RISC-V Instruction Set Manual
Volume II: Privileged Architecture
Document Version 20211203
Editors: Andrew Waterman, Krste Asanovi´c, John
Hauser, SiFive Inc.,



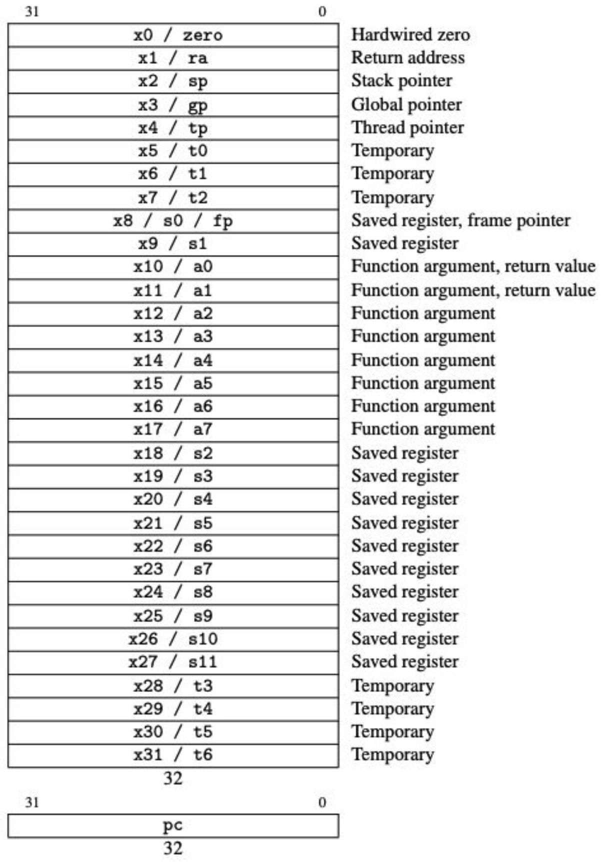
The RISC-V Reader: An Open Architecture Atlas
Paperback – November 7, 2017
by David Patterson, Andrew Waterman


COMPUTER ORGANIZATION AND DESIGN RISC-V EDITION
THE HARDWARE SOFTWARE INTERFACE
SECOND EDITION
MK
DAVID A. PATTERSON & JOHN L. HENNESSY

Computer Organization and Design RISC-V Edition: The Hardware Software Interface - 2nd Edition
by David A. Patterson, John L. Hennessy

31 25	24 20	19 15	14 12	11 7	6 0	
imm[31:12]				rd	0110111	U lui
imm[31:12]				rd	0010111	U auipc
imm[20|10:1|11|19:12]				rd	1101111	J jal
imm[11:0]		rs1	000	rd	1100111	I jalr
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	B beq
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	B bne
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	B blt
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	B bge
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	B bltu
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	B bgeu
imm[11:0]		rs1	000	rd	0000011	I lb
imm[11:0]		rs1	001	rd	0000011	I lh
imm[11:0]		rs1	010	rd	0000011	I lw
imm[11:0]		rs1	100	rd	0000011	I lbu
imm[11:0]		rs1	101	rd	0000011	I lhu
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	S sb
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	S sh
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	S sw
imm[11:0]		rs1	000	rd	0010011	I addi
imm[11:0]		rs1	010	rd	0010011	I slti
imm[11:0]		rs1	011	rd	0010011	I sltiu
imm[11:0]		rs1	100	rd	0010011	I xori
imm[11:0]		rs1	110	rd	0010011	I ori
imm[11:0]		rs1	111	rd	0010011	I andi
0000000	shamt	rs1	001	rd	0010011	I slli
0000000	shamt	rs1	101	rd	0010011	I srli
0100000	shamt	rs1	101	rd	0010011	I srai
0000000	rs2	rs1	000	rd	0110011	R add
0100000	rs2	rs1	000	rd	0110011	R sub
0000000	rs2	rs1	001	rd	0110011	R sll
0000000	rs2	rs1	010	rd	0110011	R slt
0000000	rs2	rs1	011	rd	0110011	R sltu
0000000	rs2	rs1	100	rd	0110011	R xor
0000000	rs2	rs1	101	rd	0110011	R srl
0100000	rs2	rs1	101	rd	0110011	R sra
0000000	rs2	rs1	110	rd	0110011	R or
0000000	rs2	rs1	111	rd	0110011	R and
0000 pred	succ	00000	000	00000	0001111	I fence
0000 0000	0000	00000	001	00000	0001111	I fence.i
000000000000		00000	000	00000	1110011	I ecall
000000000001		00000	000	00000	1110011	I ebreak
csr		rs1	001	rd	1110011	I csrrw
csr		rs1	010	rd	1110011	I csrrs
csr		rs1	011	rd	1110011	I csrrc
csr		zimm	101	rd	1110011	I csrrwi
csr		zimm	110	rd	1110011	I csrrsi
csr		zimm	111	rd	1110011	I csrrci

Figure 2.3: RV32I opcode map has instruction layout, opcodes, format type, and names. (Table 19.2 of [Waterman and Asanović 2017] is the basis of this figure.)

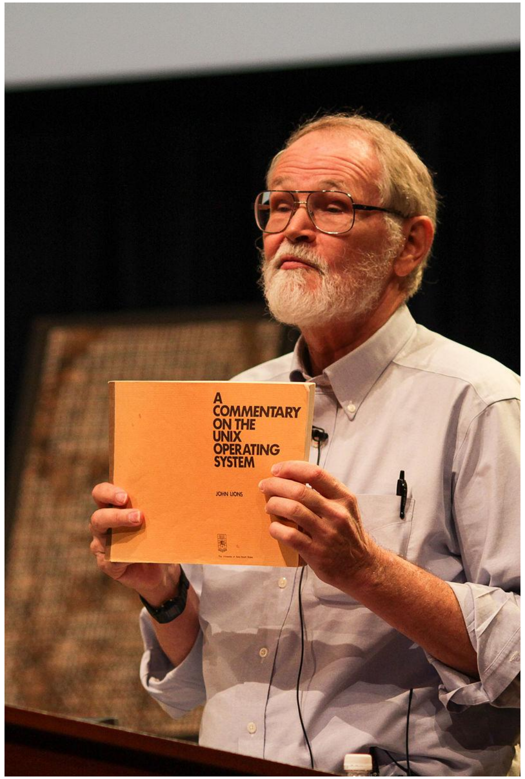


x0 / zero	Hardwired zero
x1 / ra	Return address
x2 / sp	Stack pointer
x3 / gp	Global pointer
x4 / tp	Thread pointer
x5 / t0	Temporary
x6 / t1	Temporary
x7 / t2	Temporary
x8 / s0 / fp	Saved register, frame pointer
x9 / s1	Saved register
x10 / a0	Function argument, return value
x11 / a1	Function argument, return value
x12 / a2	Function argument
x13 / a3	Function argument
x14 / a4	Function argument
x15 / a5	Function argument
x16 / a6	Function argument
x17 / a7	Function argument
x18 / s2	Saved register
x19 / s3	Saved register
x20 / s4	Saved register
x21 / s5	Saved register
x22 / s6	Saved register
x23 / s7	Saved register
x24 / s8	Saved register
x25 / s9	Saved register
x26 / s10	Saved register
x27 / s11	Saved register
x28 / t3	Temporary
x29 / t4	Temporary
x30 / t5	Temporary
x31 / t6	Temporary
pc	

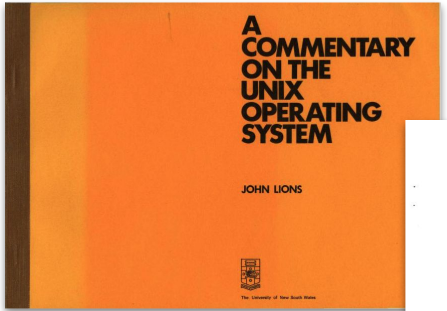
Figure 2.4: The registers of RV32I. Chapter 3 explains the RISC-V calling convention, the rationale behind the various pointers (sp, gp, tp, fp), Saved registers (s0-s11), and Temporaries (t0-t6). (Figure 2.1 and Table 20.1 of [Waterman and Asanović 2017] is the basis of this figure.)

Caso de estudio: xv6

A Commentary on the UNIX Operating System



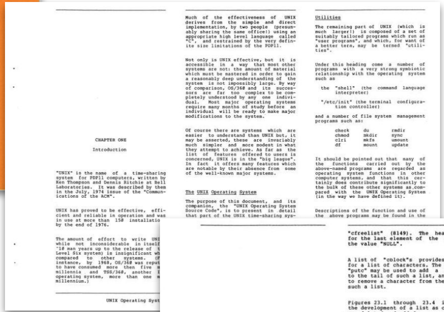
Brian Kernighan



A COMMENTARY ON THE UNIX OPERATING SYSTEM

JOHN LIONS

The University of New South Wales



CHAPTER ONE

Introduction

"UNIX" is the name of a time-sharing system for PDP11 computers, written by Ken Thompson and Dennis Ritchie at Bell Laboratories. It was described by them in the July, 1974 issue of the "Communications of the ACM".

UNIX has proved to be effective, efficient and reliable in operation and was in use at more than 150 installatio
by the end of 1976.

The amount of effort to write UNI
while not inconsiderable in itself
~10 man years up to the release of t
Level Six system) is insignificant wh
compared to other systems. (F
instance, by 1968, OS/360 was reput
to have consumed more then five m
millennia and TSS/360, another I
operating system, more than one m
millennium.)

UNIX Operating Syst

Much of the effectiveness of UNIX derives from the simple and direct implementation, by two people (presumably sharing the same office!) using an appropriate high level language called "C", and restrained by the very definite size limitations of the PDP11.

Not only is UNIX effective, but it is accessible in a way that most other systems are not: the amount of material which must be mastered in order to gain a reasonably deep understanding of the system is not impossibly large. By way of comparison, OS/360 and its successors are far too complex to be completely understood by any one individual. Most major operating systems require many months of study before an individual will be ready to make major modifications to the system.

Of course there are systems which are easier to understand than UNIX but, it may be asserted, these are invariably much simpler and more modest in what they attempt to achieve. As far as the list of features offered to users is concerned, UNIX is in the "big league". In fact it offers many features which are notable by their absence from some of the well-known major systems.

The UNIX Operating System

The purpose of this document, and its companion, the "UNIX Operating System Source Code", is to present in detail that part of the UNIX time-sharing sys-

Utilities

The remaining part of UNIX (which is much larger!) is composed of a set of suitably tailored programs which run as "user programs", and which, for want of a better term, may be termed "utilities".

Under this heading come a number of programs with a very strong symbiotic relationship with the operating system such as

the "shell" (the command language interpreter)

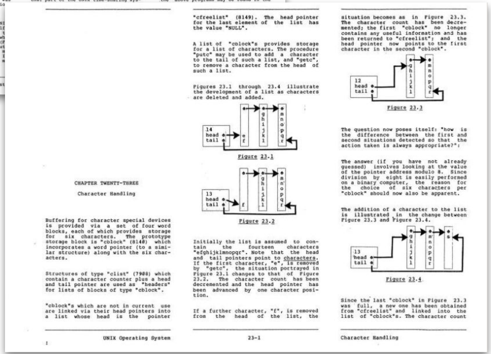
"/etc/init" (the terminal configuration controller)

and a number of file system management programs such as:

```
check    du       rmdir
chmod    mkdir    sync
clri     mkfs     umount
df       mount    update
```

It should be pointed out that many of the functions carried out by the above-named programs are regarded as operating system functions in other computer systems, and that this certainly does contribute significantly to the bulk of these other systems as compared with the UNIX Operating System (in the way we have defined it).

Descriptions of the function and use of the above programs may be found in the



CHAPTER TWENTY-THREE

Character Handling

Buffering for character special devices is provided via a set of four word blocks, each of which provides storage for six characters. The prototype storage block is "cblock" (8140) which incorporates a word pointer (to a similar structure) along with the six characters.

Structures of type "clist" (7908) which contain a character counter plus a head and tail pointer are used as "headers" for lists of blocks of type "cblock".

"cblock"s which are not in current use are linked via their head pointers into a list whose head is the pointer "cfreelist" (8149). The head pointer for the last element of the list has the value "NULL".

A list of "cblock"s provides storage for a list of characters. The procedure "putc" may be used to add a character to the tail of such a list, and "getc", to remove a character from the head of such a list.

Figures 23.1 through 23.4 illustrate the development of a list as characters are deleted and added.

Figure 23.1

Figure 23.2

Initially the list is assumed to contain the fourteen characters "efghijklmnopqr". Note that the head and tail pointers point to characters. If the first character, "e", is removed by "getc", the situation portrayed in Figure 23.1 changes to that of Figure 23.2. The character count has been decremented and the head pointer has been advanced by one character position.

If a further character, "f", is removed from the head of the list, the situation becomes as in Figure 23.3. The character count has been decremented; the first "cblock" no longer contains any useful information and has been returned to "cfreelist"; and the head pointer now points to the first character in the second "cblock".

Figure 23.3

The question now poses itself: "how is the difference between the first and second situations detected so that the action taken is always appropriate?":

The answer (if you have not already guessed) involves looking at the value of the pointer address modulo 8. Since division by eight is easily performed on a binary computer, the reason for the choice of six characters per "cblock" should now also be apparent.

The addition of a character to the list is illustrated in the change between Figure 23.3 and Figure 23.4.

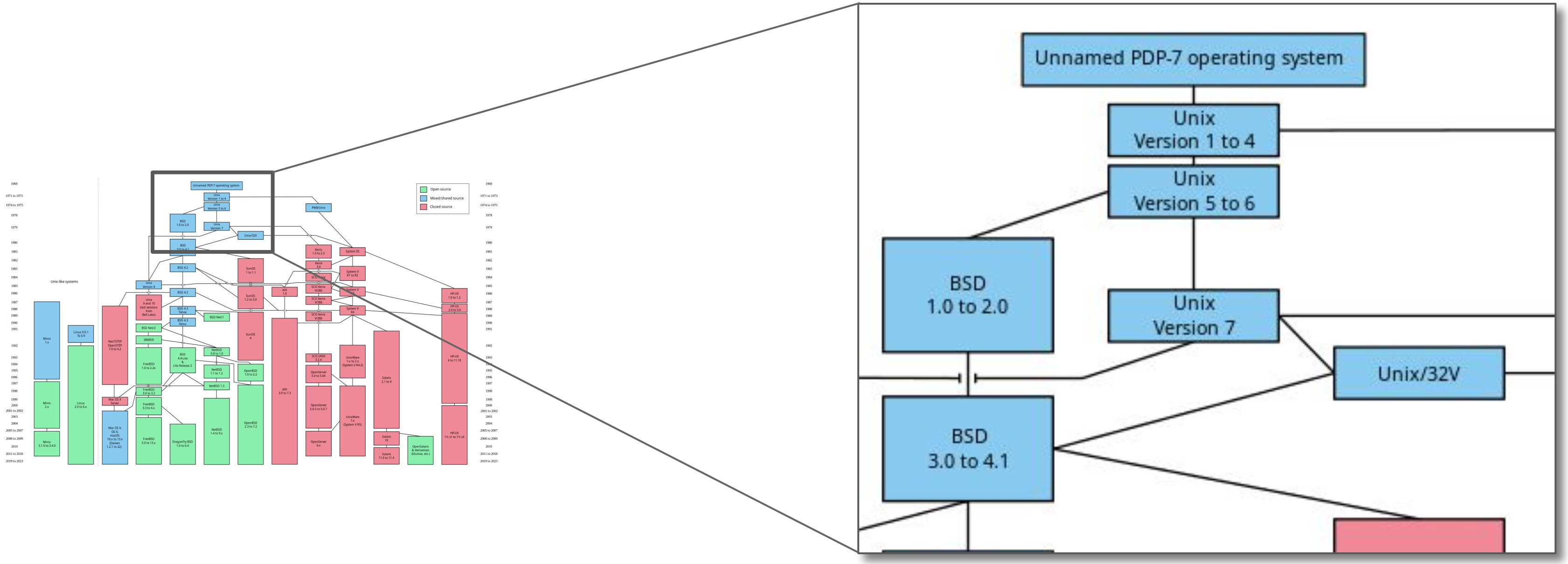
Figure 23.4

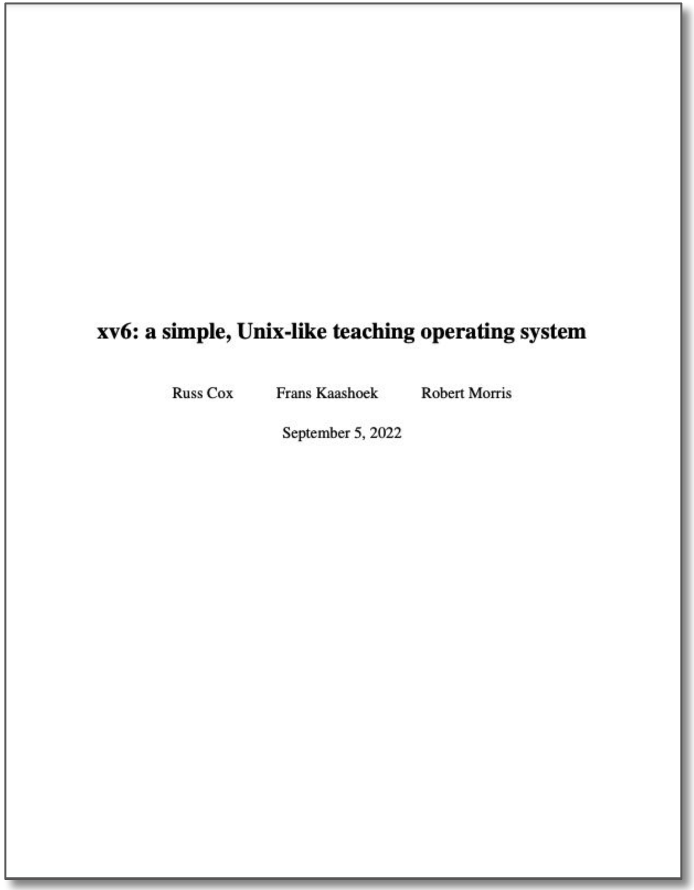
Since the last "cblock" in Figure 23.3 was full, a new one has been obtained from "cfreelist" and linked into the list of "cblock"s. The character count

UNIX Operating System | 23-1 | Character Handling

https://archive.org/details/bitsavers_attunix6thtaryontheUnixOperatingSystem197705_12314928

https://cs3210.cc.gatech.edu/r/unix6.pdf


Unix-like systems
Open source
Mixed/shared source
Closed source
Unnamed PDP-7 operating system
Unix Version 1 to 4
Unix Version 5 to 6
BSD 1.0 to 2.0
Unix Version 7
Unix/32V
BSD 3.0 to 4.1

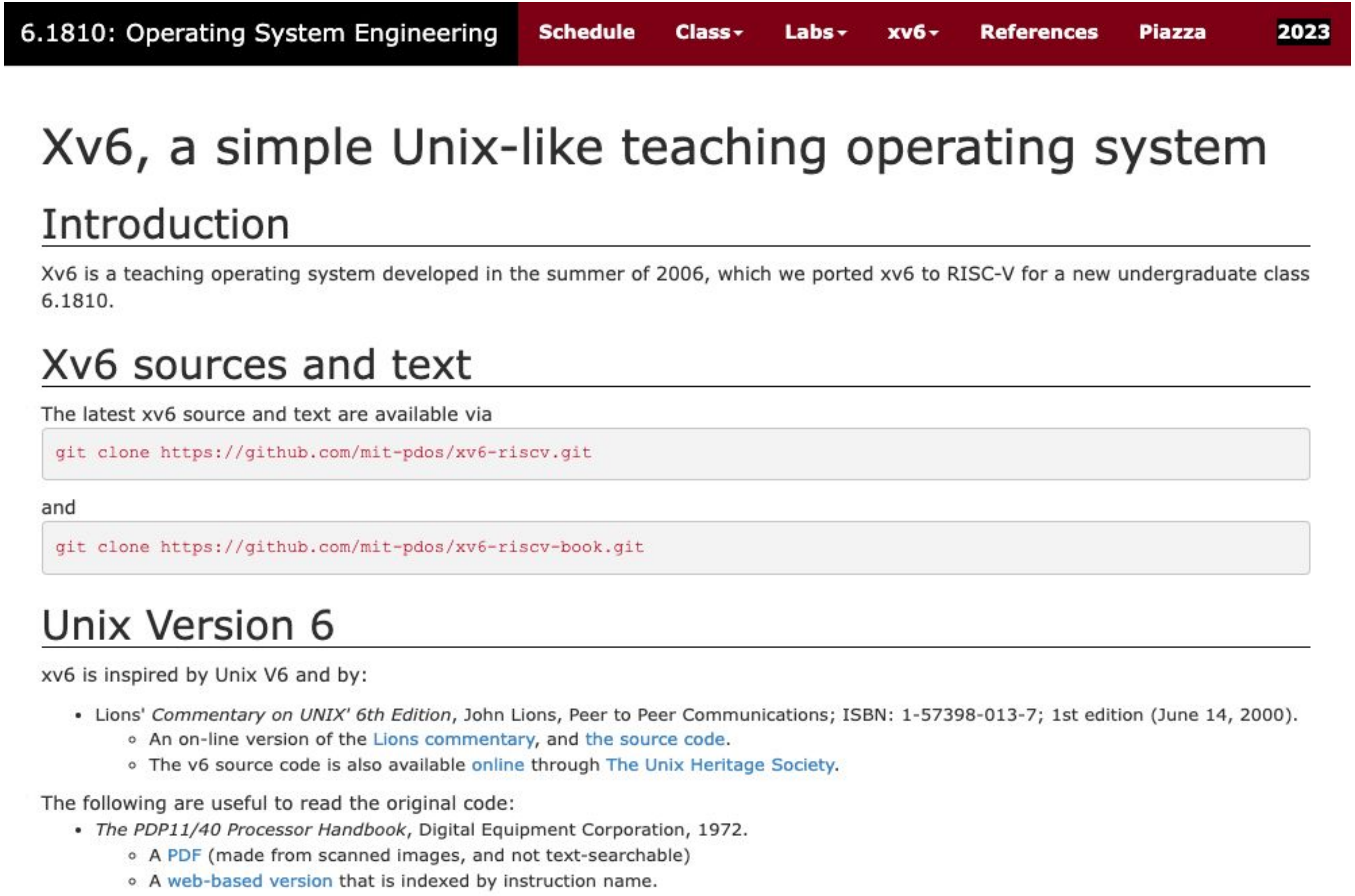


xv6: a simple, Unix-like teaching operating system

Russ Cox   Frans Kaashoek   Robert Morris

September 5, 2022

https://pdos.csail.mit.edu/6.828/2023/xv6.html



Xv6, a simple Unix-like teaching operating system

Introduction

Xv6 is a teaching operating system developed in the summer of 2006, which we ported xv6 to RISC-V for a new undergraduate class 6.1810.

Xv6 sources and text

The latest xv6 source and text are available via

```
git clone https://github.com/mit-pdos/xv6-riscv.git
```

and

```
git clone https://github.com/mit-pdos/xv6-riscv-book.git
```

Unix Version 6

xv6 is inspired by Unix V6 and by:

- Lions' *Commentary on UNIX' 6th Edition*, John Lions, Peer to Peer Communications; ISBN: 1-57398-013-7; 1st edition (June 14, 2000).
    - An on-line version of the Lions commentary, and the source code.
    - The v6 source code is also available online through The Unix Heritage Society.

The following are useful to read the original code:

- *The PDP11/40 Processor Handbook*, Digital Equipment Corporation, 1972.
    - A PDF (made from scanned images, and not text-searchable)
    - A web-based version that is indexed by instruction name.

Repaso de la clase 1

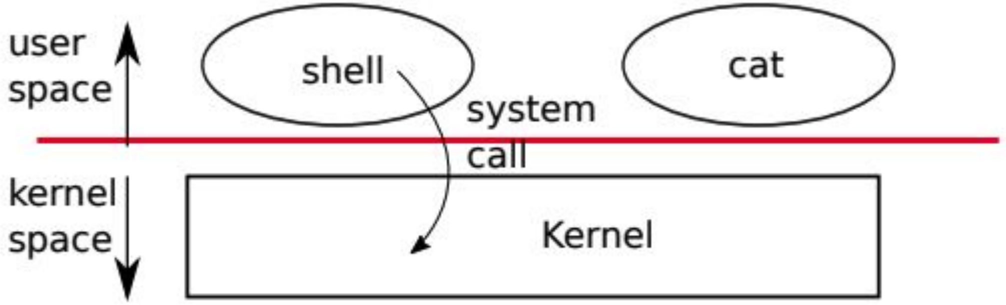


Figure 1.1: A kernel and two user processes.

System call	Description
int fork()	Create a process, return child's PID.
int exit(int status)	Terminate the current process; status reported to wait(). No return.
int wait(int *status)	Wait for a child to exit; exit status in *status; returns child PID.
int kill(int pid)	Terminate process PID. Returns 0, or -1 for error.
int getpid()	Return the current process's PID.
int sleep(int n)	Pause for n clock ticks.
int exec(char *file, char *argv[])	Load a file and execute it with arguments; only returns if error.
char *sbrk(int n)	Grow process's memory by n bytes. Returns start of new memory.
int open(char *file, int flags)	Open a file; flags indicate read/write; returns an fd (file descriptor).
int write(int fd, char *buf, int n)	Write n bytes from buf to file descriptor fd; returns n.
int read(int fd, char *buf, int n)	Read n bytes into buf; returns number read; or 0 if end of file.
int close(int fd)	Release open file fd.
int dup(int fd)	Return a new file descriptor referring to the same file as fd.
int pipe(int p[])	Create a pipe, put read/write file descriptors in p[0] and p[1].
int chdir(char *dir)	Change the current directory.
int mkdir(char *dir)	Create a new directory.
int mknod(char *file, int, int)	Create a device file.
int fstat(int fd, struct stat *st)	Place info about an open file into *st.
int stat(char *file, struct stat *st)	Place info about a named file into *st.
int link(char *file1, char *file2)	Create another name (file2) for the file file1.
int unlink(char *file)	Remove a file.

Figure 1.2: Xv6 system calls. If not otherwise stated, these calls return 0 for no error, and -1 if there's an error.

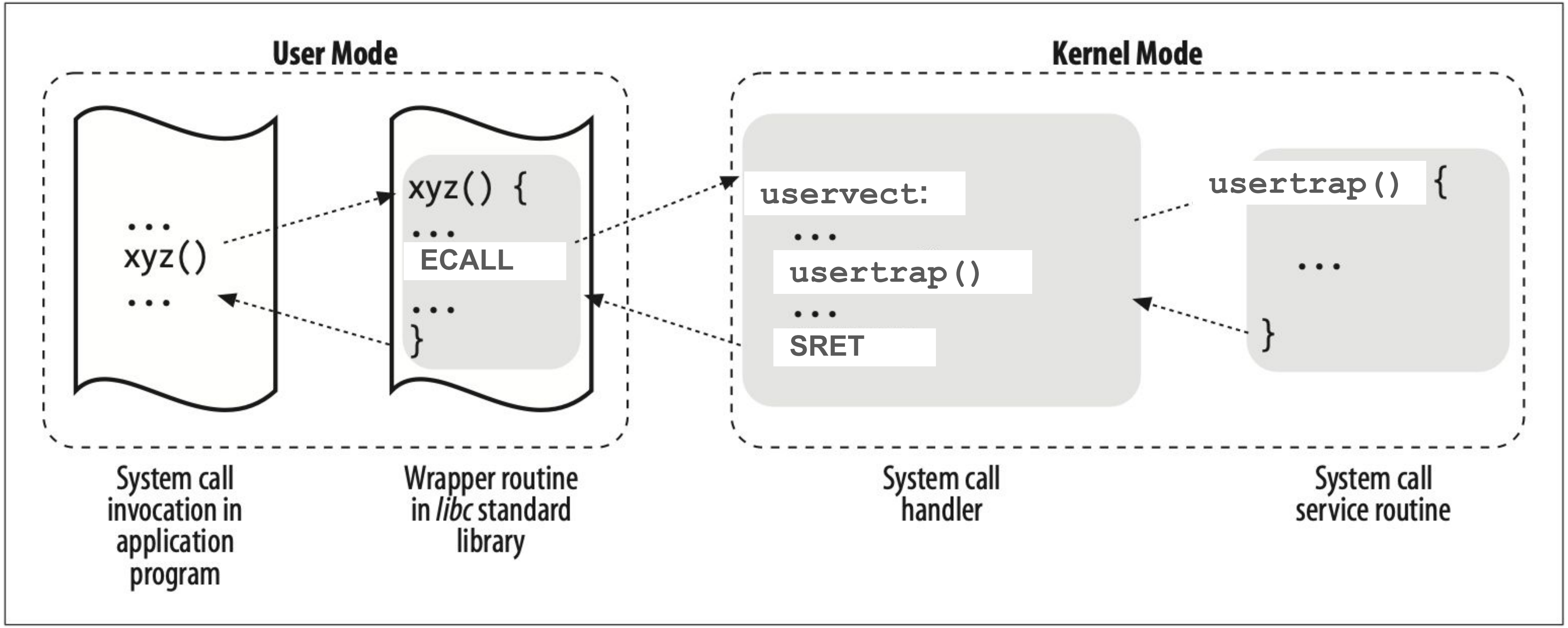
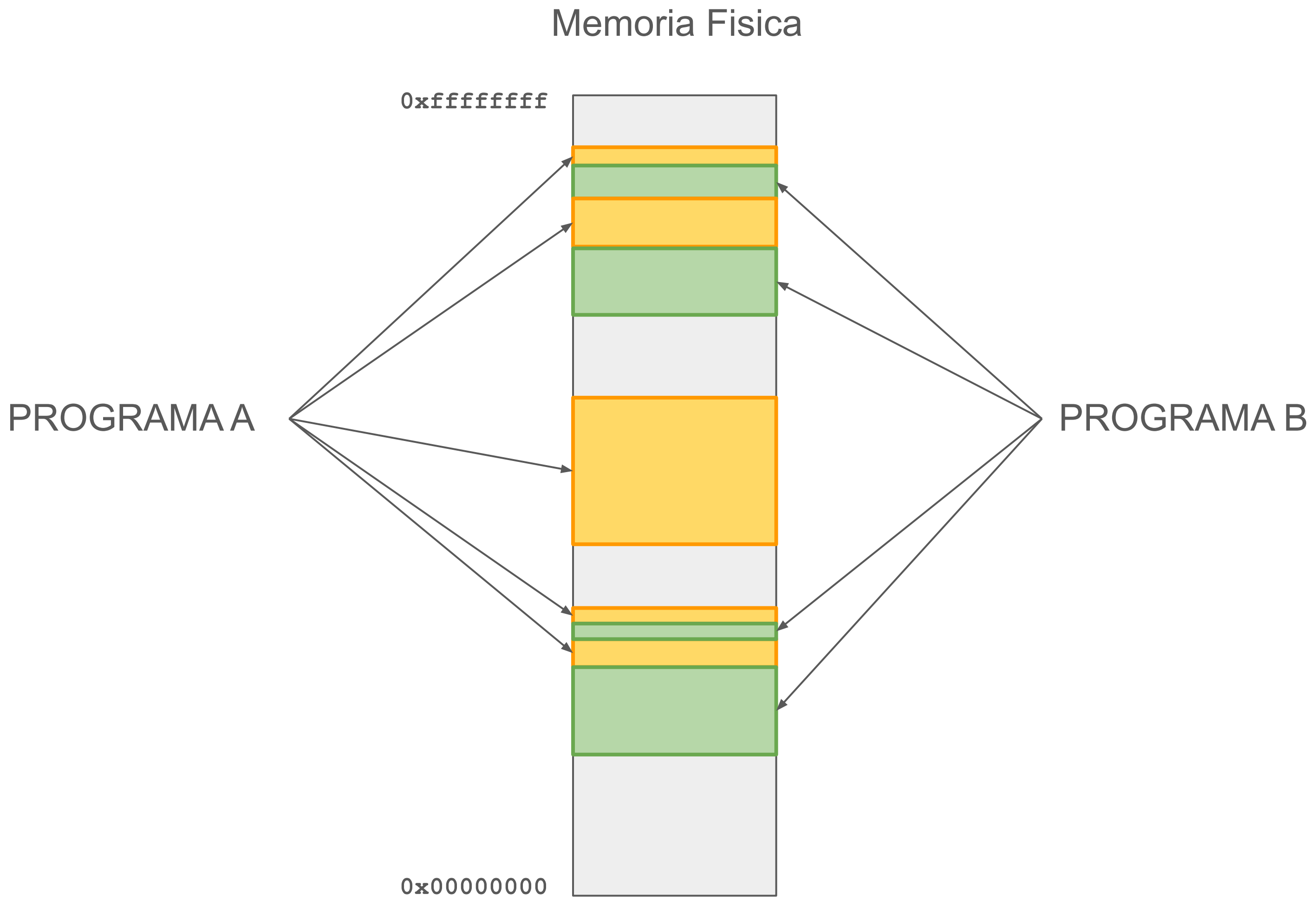
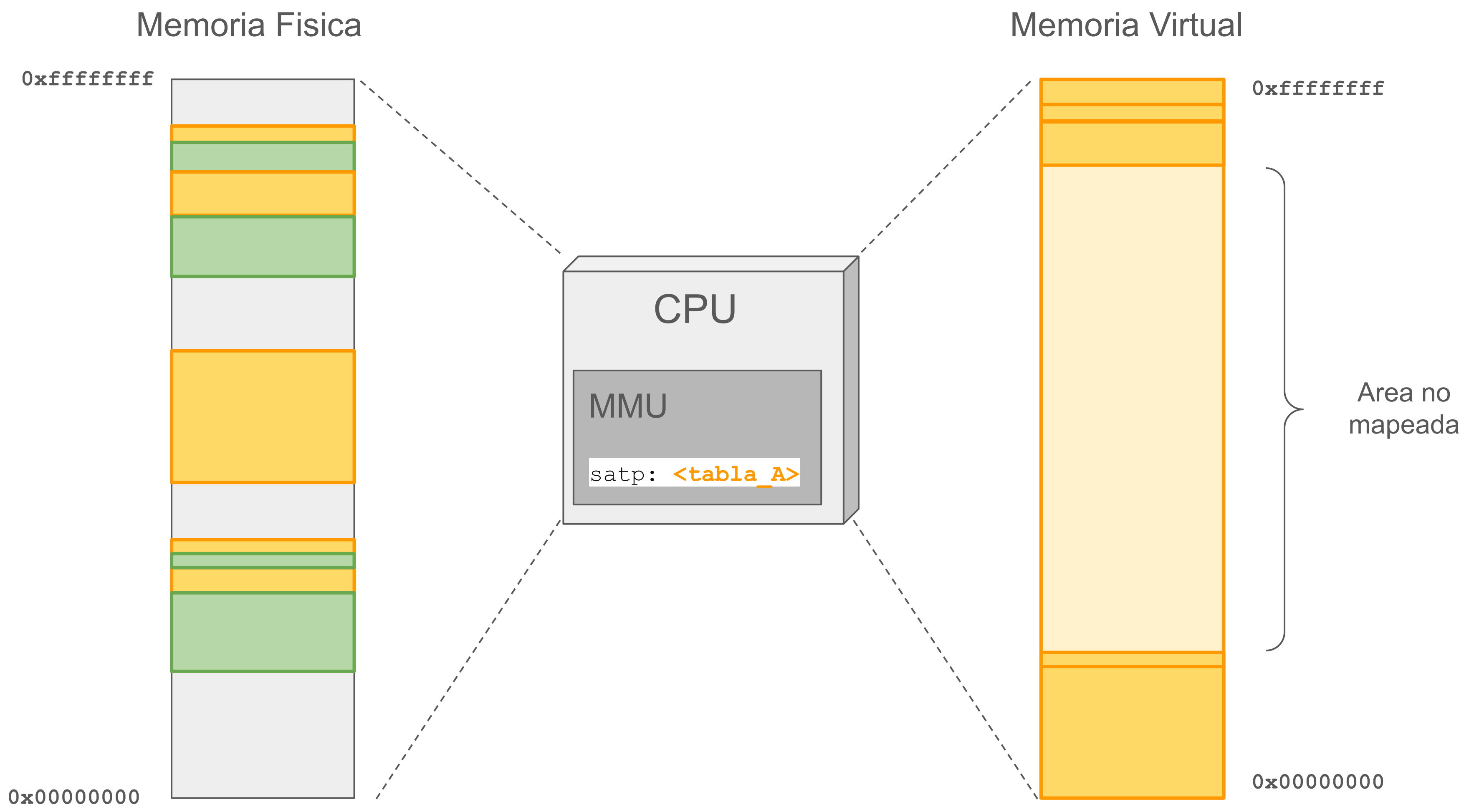
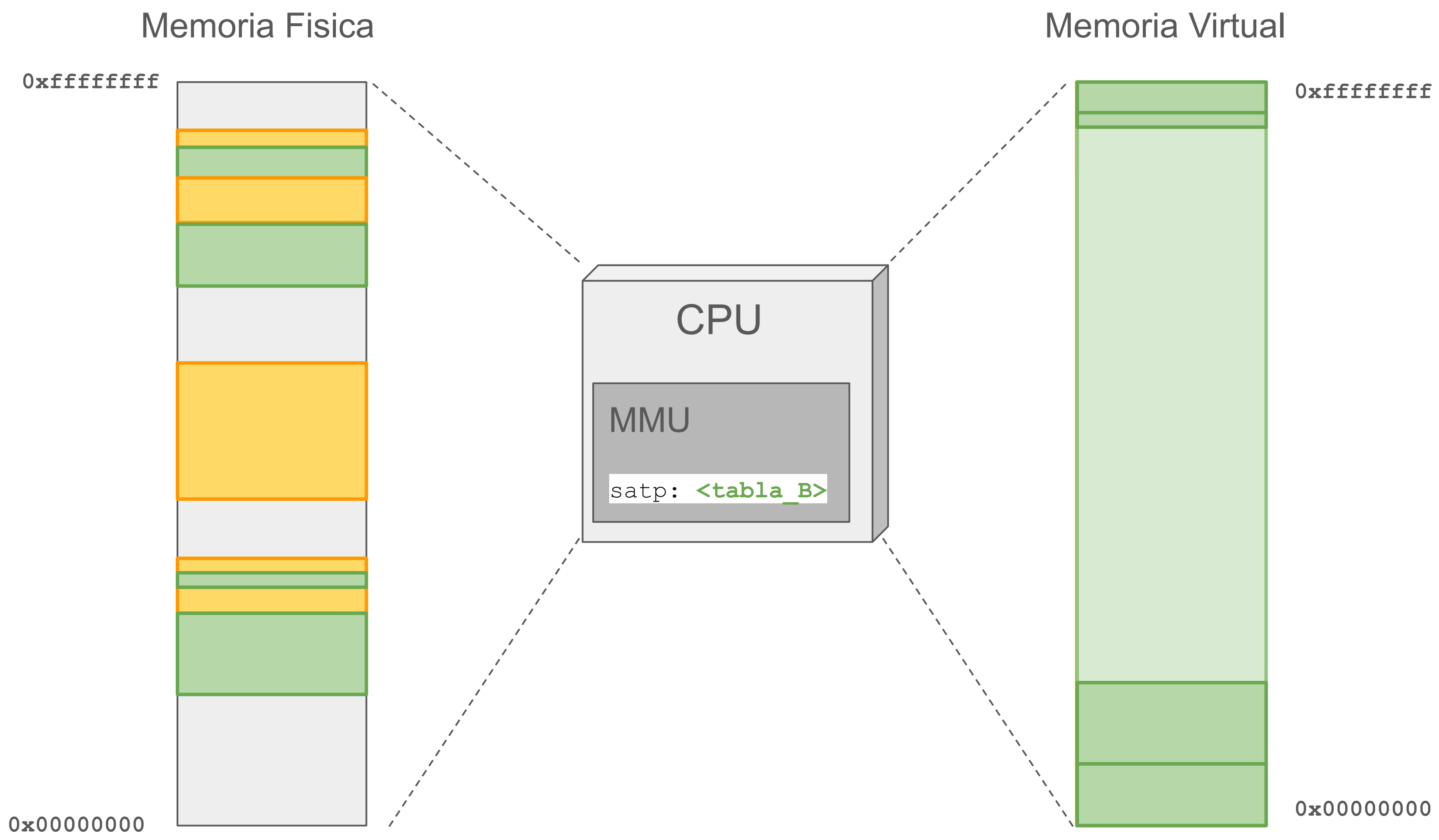
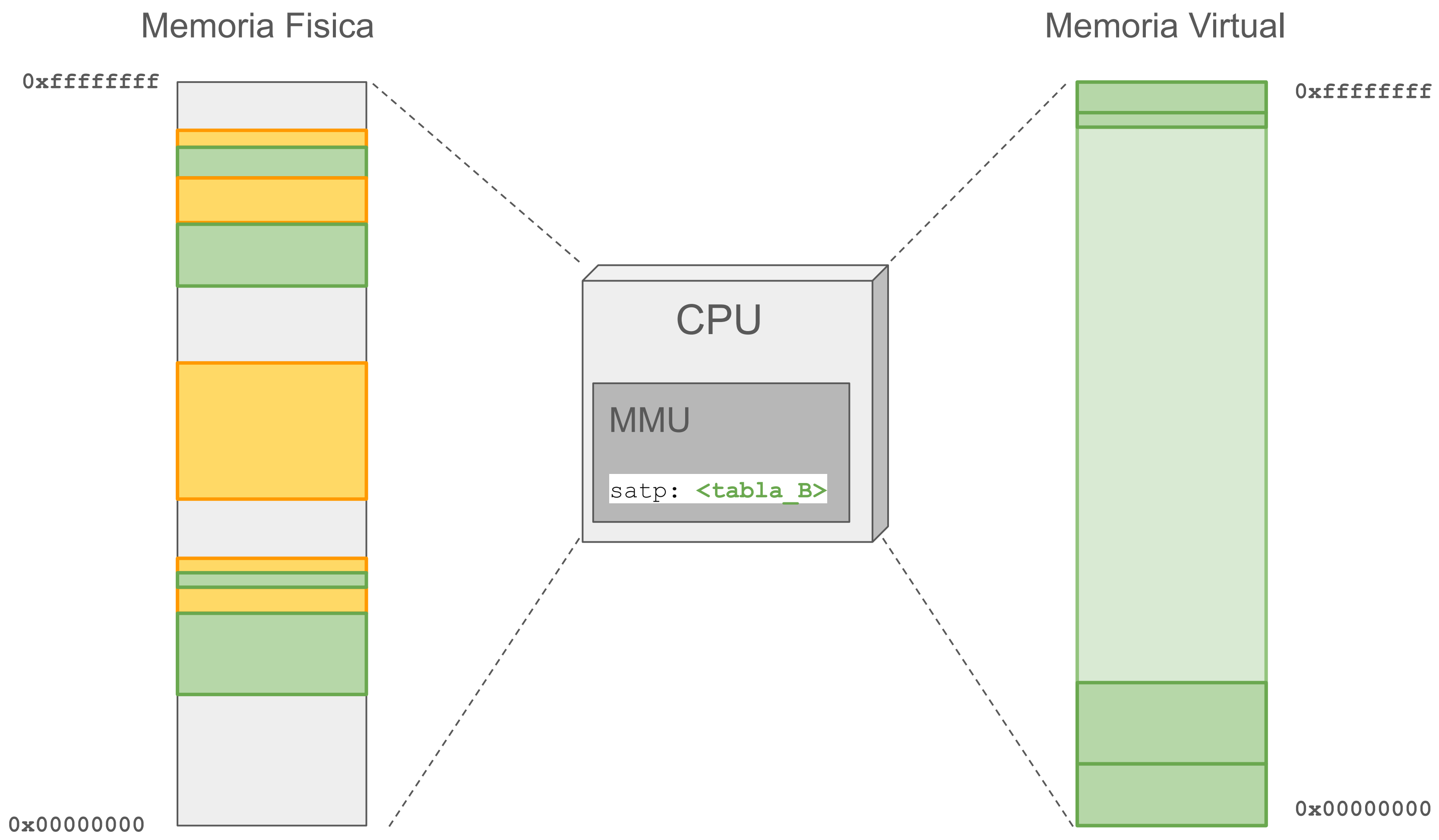


Figure 10-1. Invoking a system call

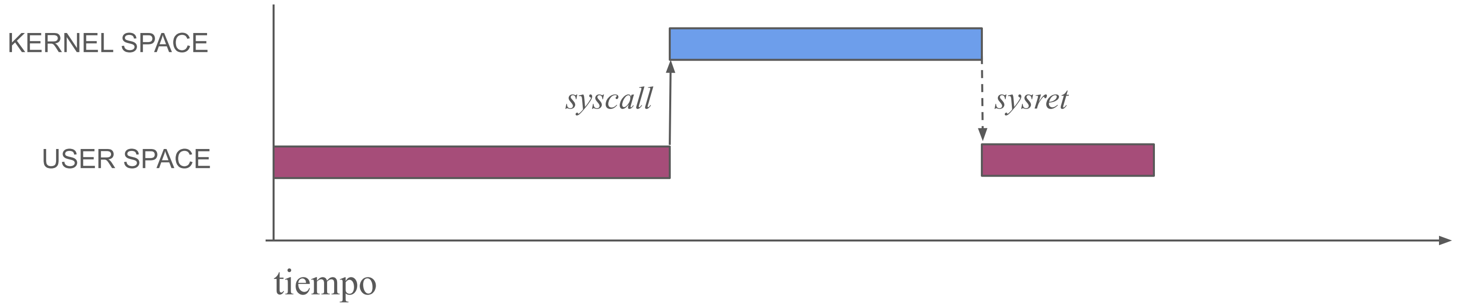

Memoria Fisica
0xffffffff
0x00000000
PROGRAMA A
PROGRAMA B


Memoria Fisica
Memoria Virtual
0xffffffff
0x00000000
CPU
MMU
satp: <tabla_A>
0xffffffff
0x00000000
Area no mapeada

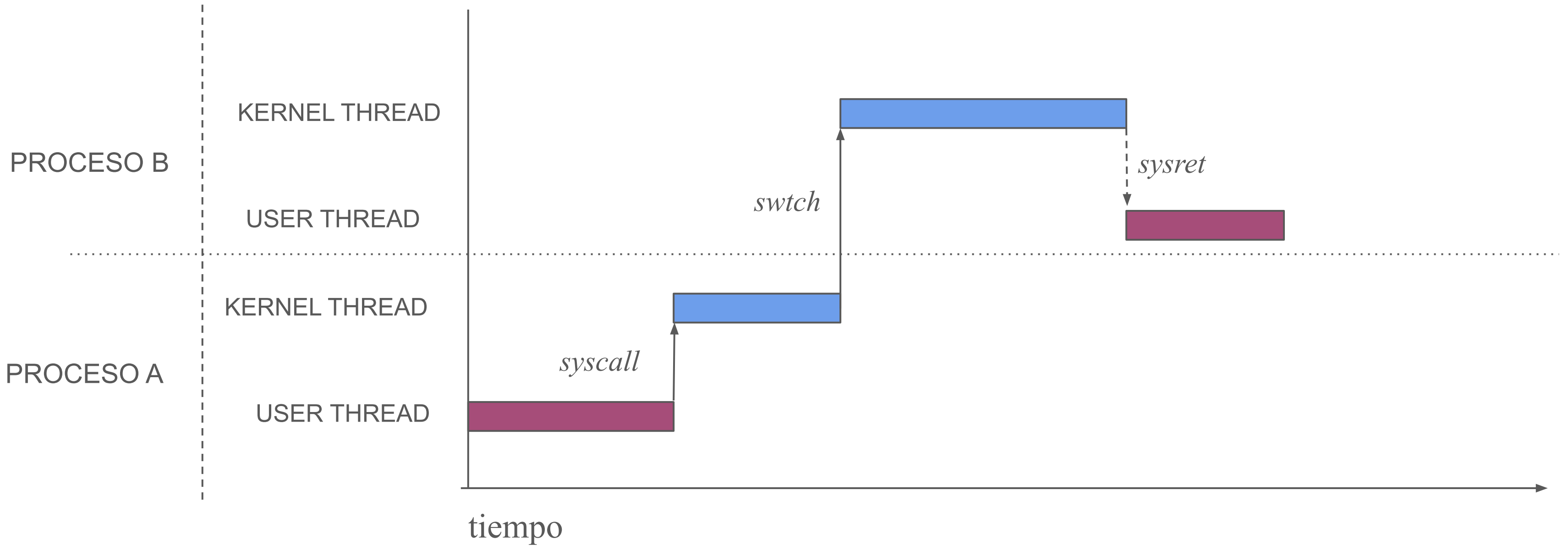

Memoria Fisica
0xffffffff
0x00000000
CPU
MMU
satp: <tabla_B>
Memoria Virtual
0xffffffff
0x00000000


Memoria Fisica
Memoria Virtual
0xffffffff
0x00000000
CPU
MMU
satp: <tabla_B>
0xffffffff
0x00000000

Transición User-Kernel

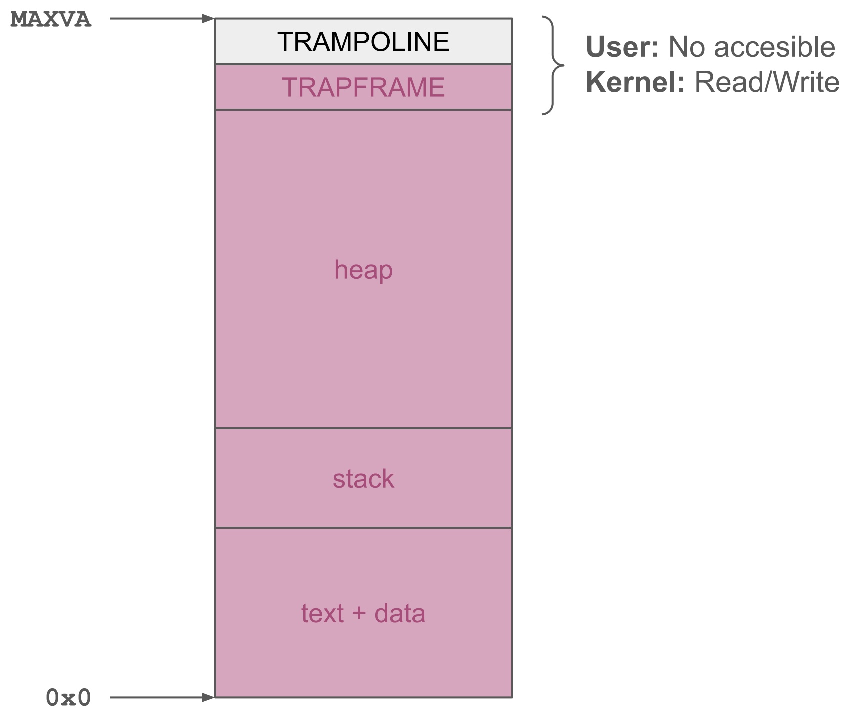


Context Switch

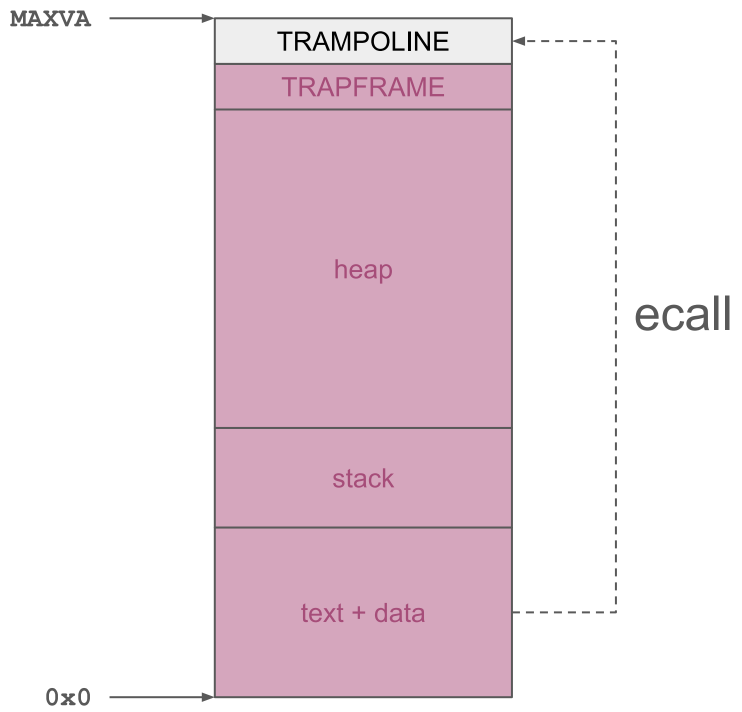


Transición User-Kernel

Memoria virtual en user space

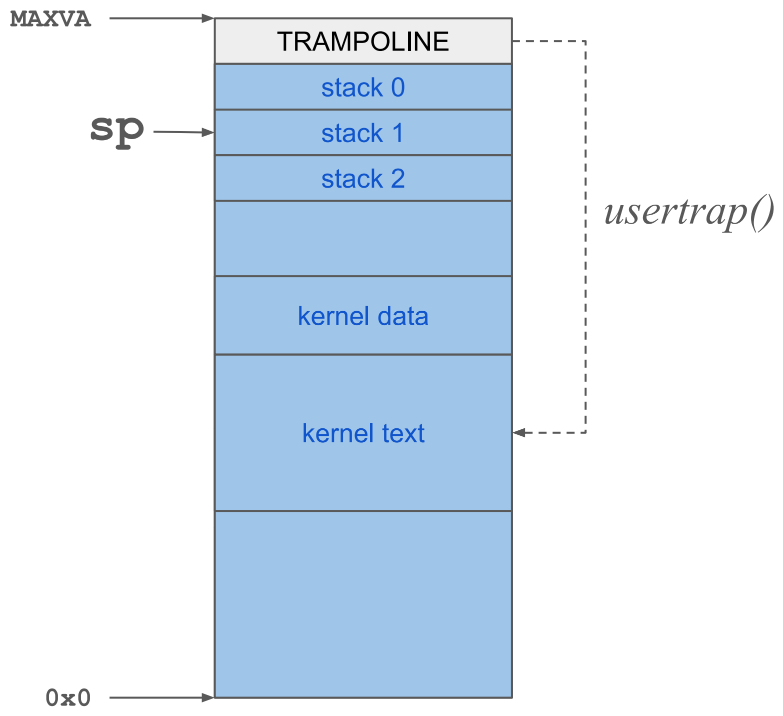


Transición User-Kernel

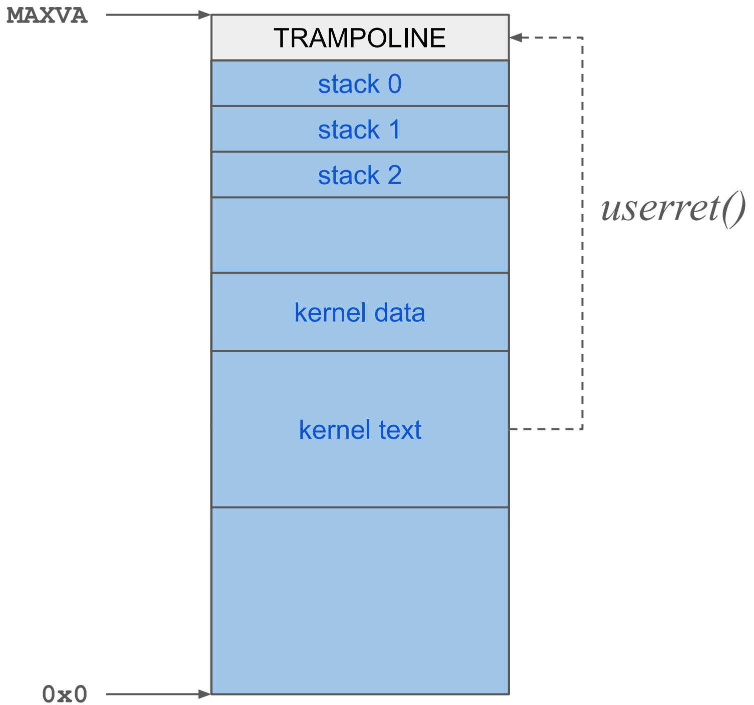


Transición User-Kernel

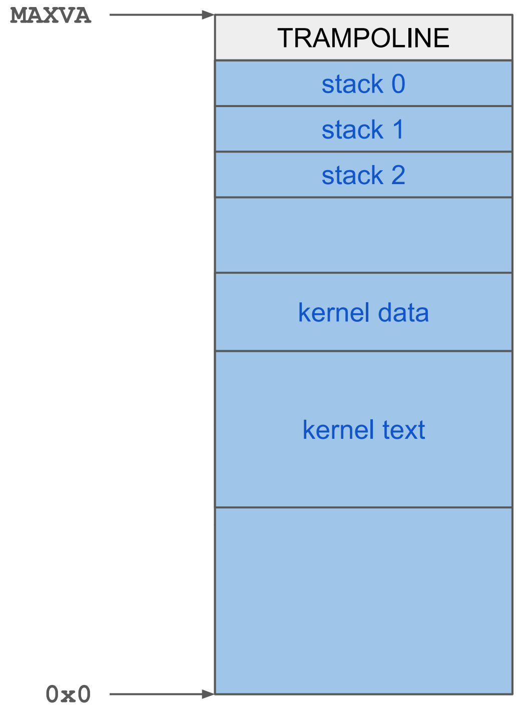
Memoria virtual en kernel space



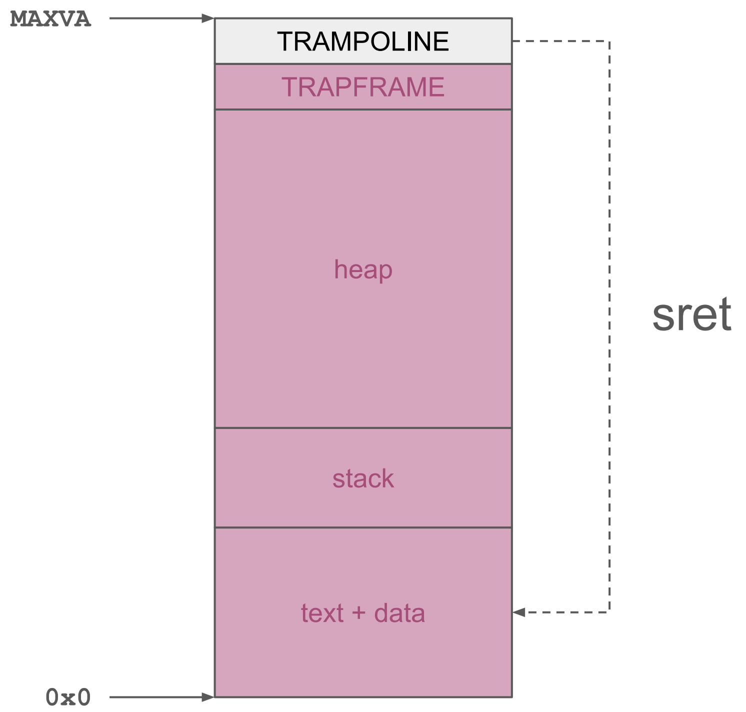
Transición User-Kernel



Transición User-Kernel



Transición User-Kernel



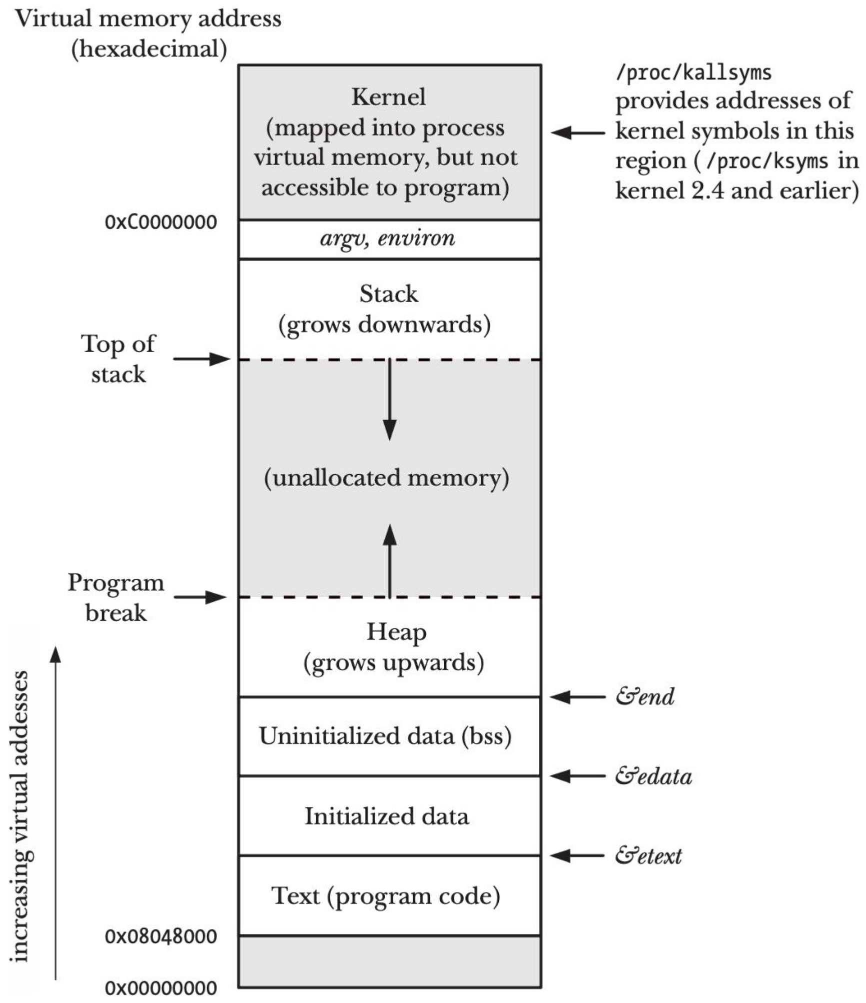
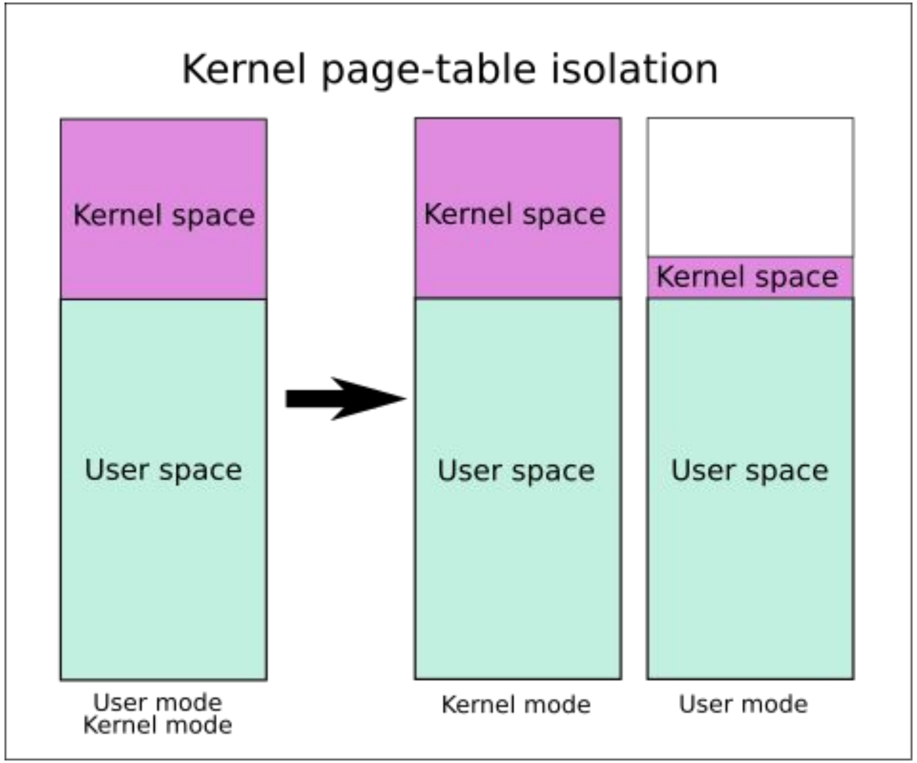

Virtual memory address (hexadecimal)
Kernel (mapped into process virtual memory, but not accessible to program)
/proc/kallsyms provides addresses of kernel symbols in this region (/proc/ksyms in kernel 2.4 and earlier)
0xC0000000
argv, environ
Stack (grows downwards)
Top of stack
(unallocated memory)
Program break
Heap (grows upwards)
&end
Uninitialized data (bss)
&edata
Initialized data
&etext
Text (program code)
0x08048000
0x00000000
increasing virtual addresses

Figure 6-1: Typical memory layout of a process on Linux/x86-32

Vulnerabilidad Meltdown y su mitigación

MELTDOWN



Meltdown: https://medium.com/@mattklein123/meltdown-spectre-explained-6bc8634cc0c2
KPTS: https://en.wikipedia.org/wiki/Kernel_page-table_isolation