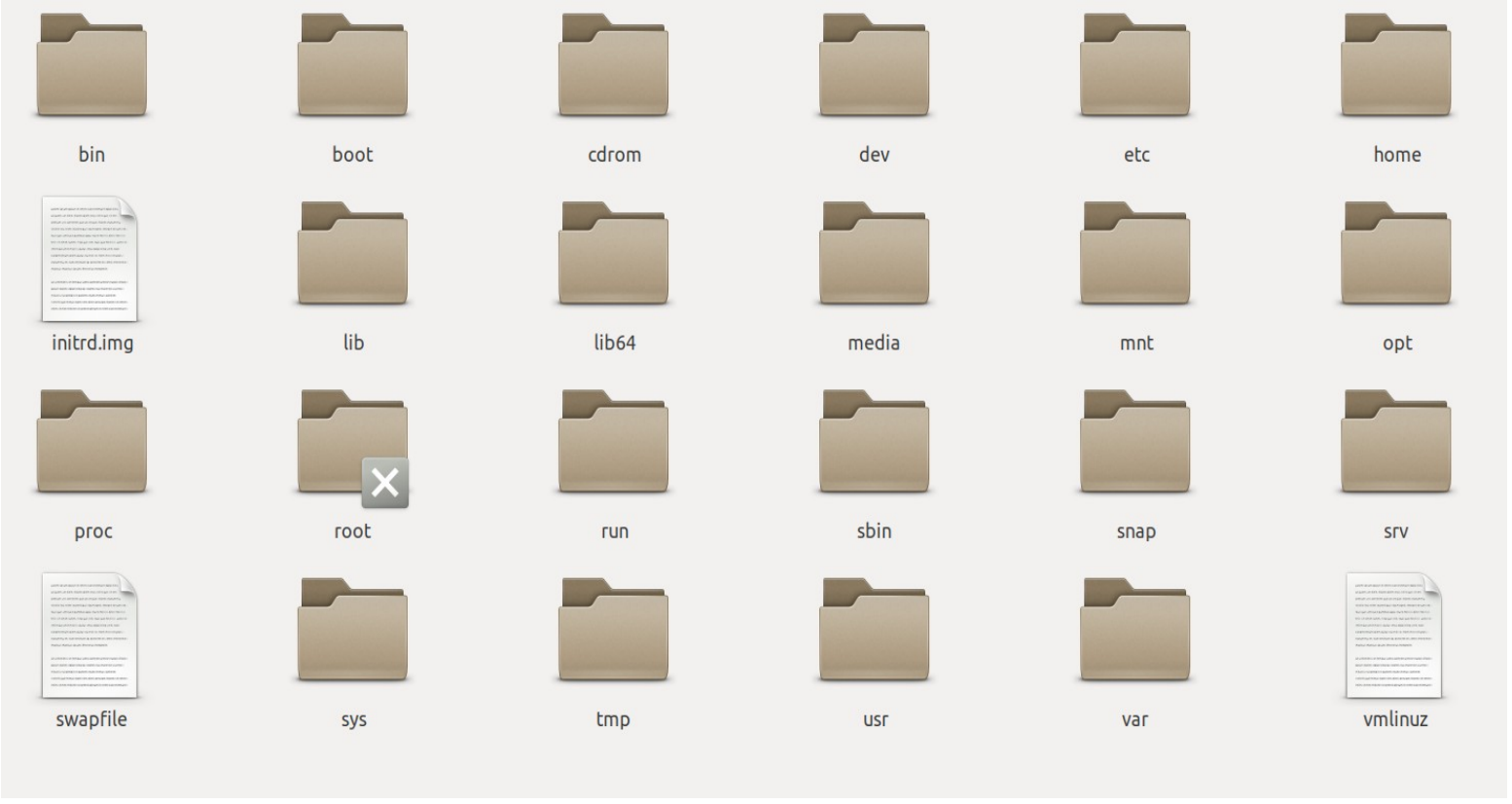
Root directory



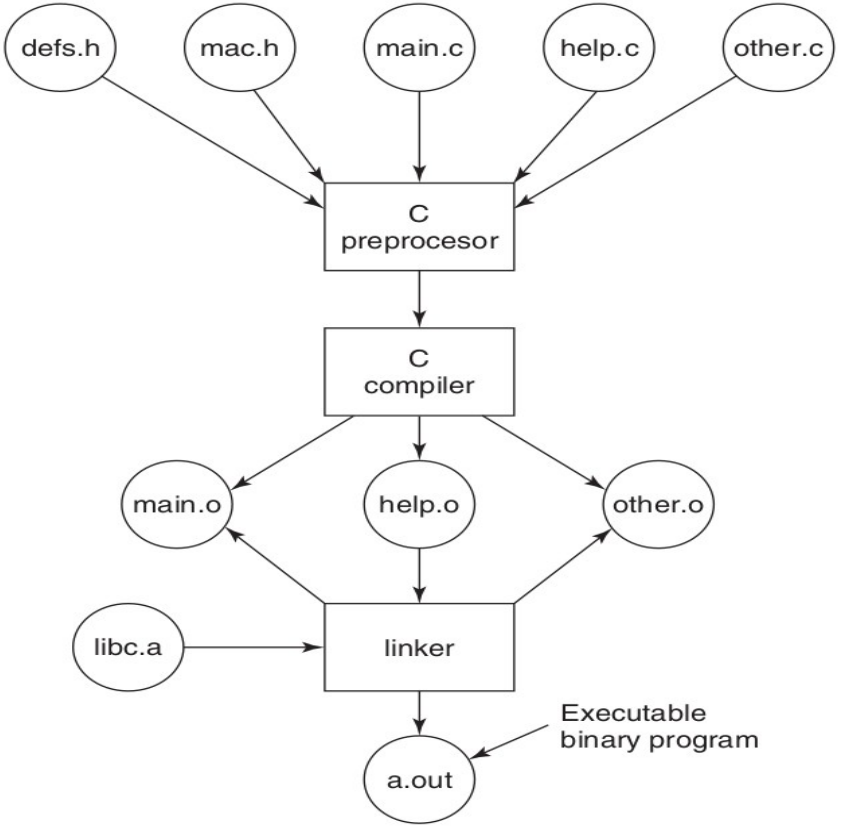
Directories in root (1)

- **/bin** - **Bin**aries.
- **/boot** - Files required for **boot**ing.
- **/dev** - **Dev**ice files.
- **/etc** - ***Et c****etera*. The name is inherited from the earliest Unixes, which is when it became the spot to put config-files.
- **/home** - Where **home** directories are kept.
- **/lib** - Where code **lib**raries are kept.
- **/media** - A more modern directory, but where removable **media** gets mounted.
- **/mnt** - Where temporary file-systems are **mount**ed.
- **/opt** - Where **opt**ional add-on software is installed. This is discrete from `/usr/local/` for reasons I'll get to later.

Directories in root (2)

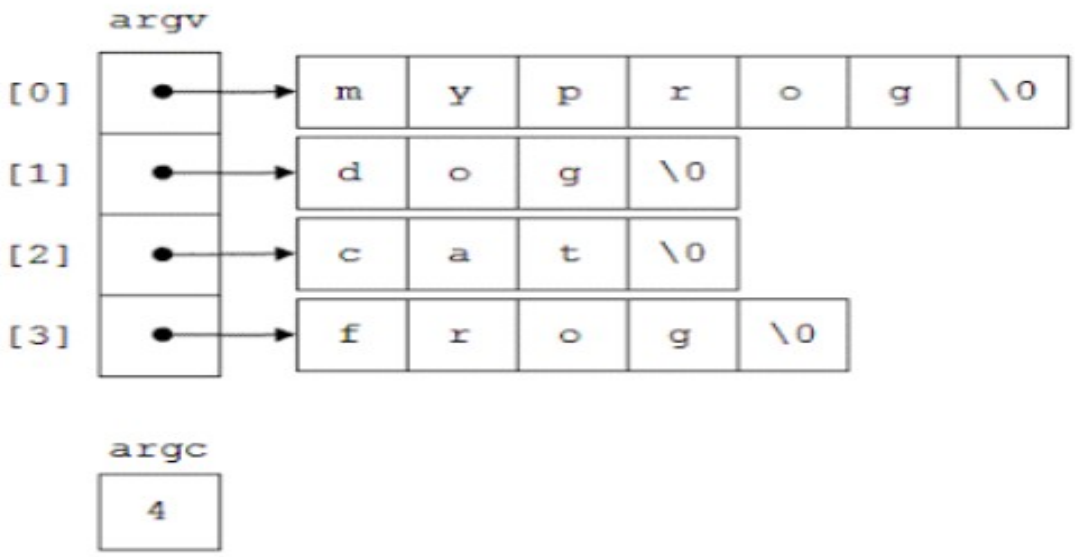
- **/run** - Where **run**time variable data is kept.
- **/sbin** - Where **s**uper-**bin**aries are stored. These usually only work with root.
- **/srv** - Stands for "**s**er**v**e". This directory is intended for static files that are served out. `/srv/http` would be for static websites, `/srv/ftp` for an FTP server.
- **/tmp** - Where **tem**porary files may be stored.
- **/usr** - Another directory inherited from the Unixes of old, it stands for "**U**NIX **S**ystem **R**esources". It does *not* stand for "user" (see the Debian Wiki). This directory should be sharable between hosts, and can be NFS mounted to multiple hosts safely. It can be mounted read-only safely.
- **/var** - Another directory inherited from the Unixes of old, it stands for "**var**iable". This is where system data that varies may be stored. Such things as spool and cache directories may be located here. If a program needs to write to the local file-system and isn't serving that data to someone directly, it'll go here.

C-preprocess,compile and link

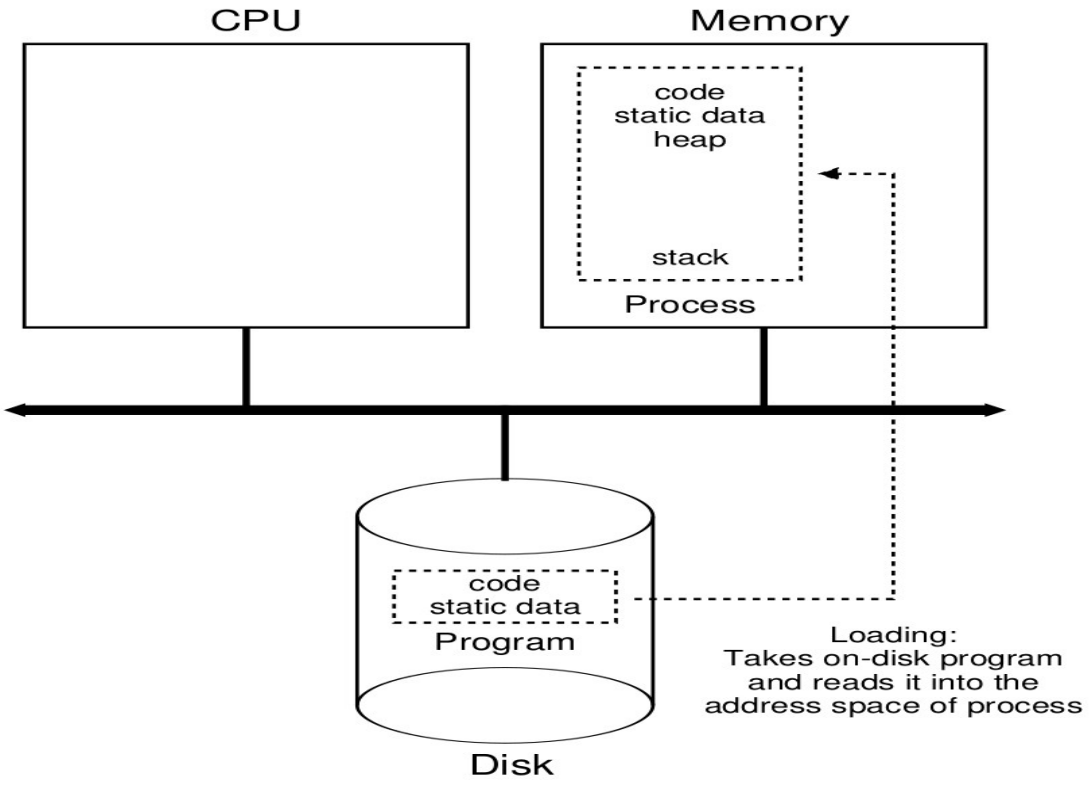


C main arguments

```
z123456@turing:~$ myprog dog cat frog
```



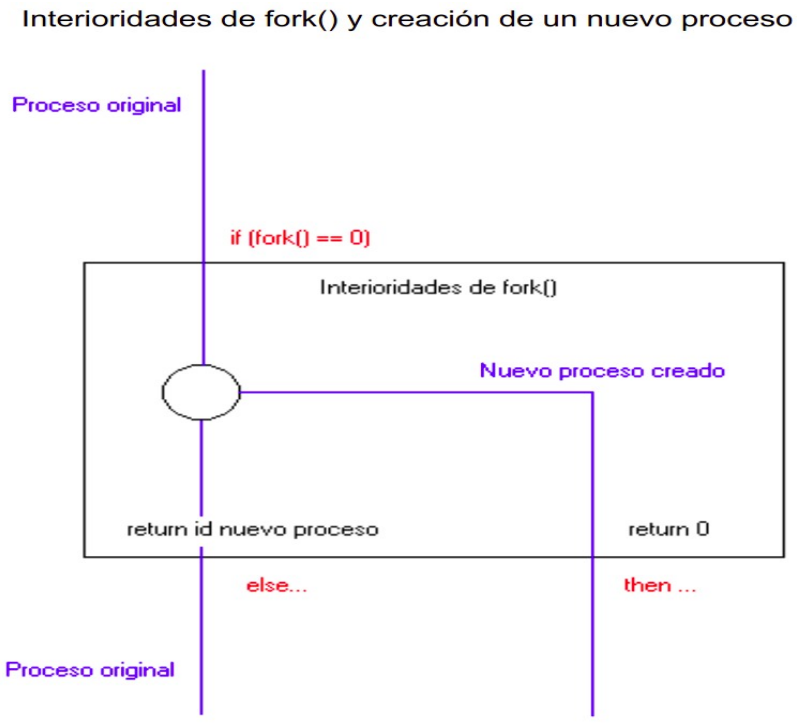
Unix process loading



Direct execution no limits

OS	Program
Create entry for process list	
Allocate memory for program	
Load program into memory	
Set up stack with argc/argv	
Clear registers	
Execute **call** main()	
	Run main()
	Execute **return** from main
Free memory of process	
Remove from process list	

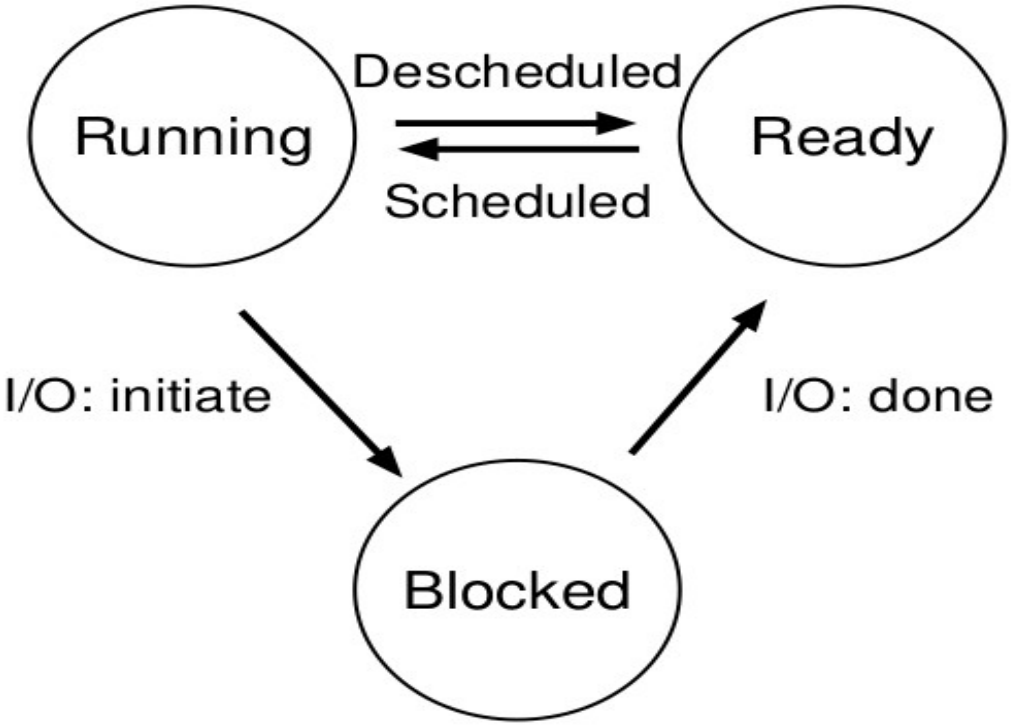
Fork process creation



Fork program layout

```
switch (fork())
{
    case -1:
        /* Código de error */
        ...
        break;
    case 0:
        /* Código del proceso hijo */
        ...
        break;
    default:
        ...
        /* Código del proceso original */
}
```

Process states



Context switch

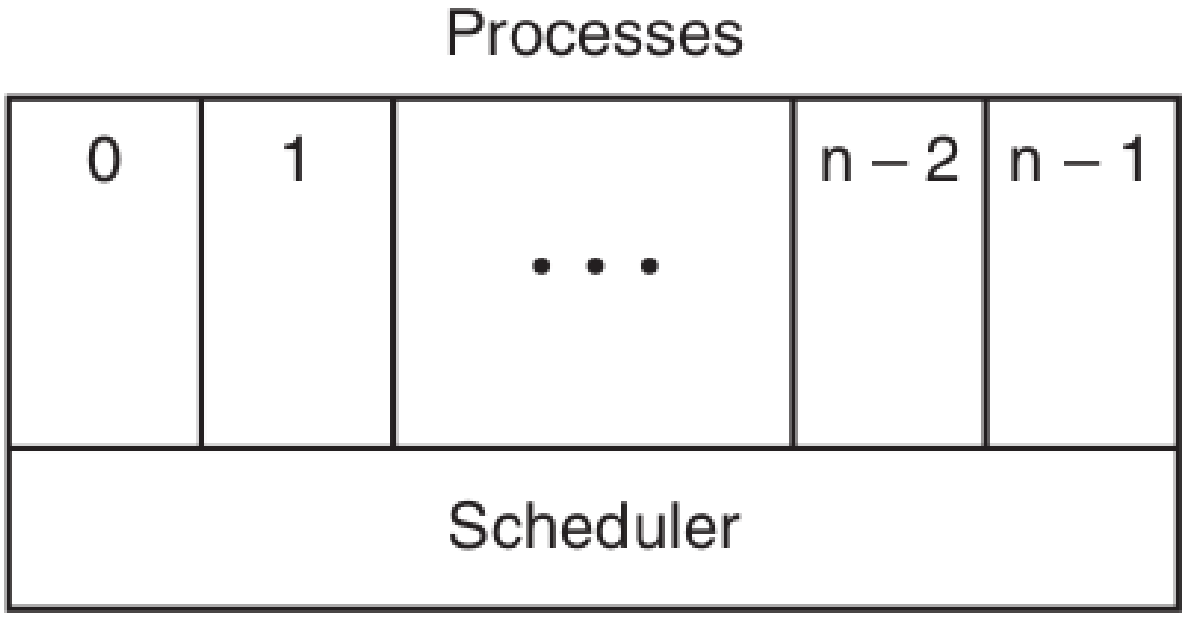
OS @ boot (kernel mode)	Hardware
initialize trap table	
	remember addresses of...
	syscall handler
	timer handler
start interrupt timer	
	start timer
	interrupt CPU in X ms

OS @ run (kernel mode)	Hardware	Program (user mode)
		Process A
		...
	timer interrupt	
	save regs(A) to k-stack(A)	
	move to kernel mode	
	jump to trap handler	
Handle the trap		
Call `switch()` routine		
save regs(A) to proc-struct(A)		
restore regs(B) from proc-struct(B)		
switch to k-stack(B)		
return-from-trap (into B)		
	restore regs(B) from k-stack(B)	
	move to user mode	
	jump to B's PC	
		Process B
		...

Kernel interrupt service

1. Hardware stacks program counter, etc.
2. Hardware loads new program counter from interrupt vector.
3. Assembly-language procedure saves registers.
4. Assembly-language procedure sets up new stack.
5. C interrupt service runs (typically reads and buffers input).
6. Scheduler decides which process is to run next.
7. C procedure returns to the assembly code.
8. Assembly-language procedure starts up new current process.

Scheduler processes



Process table fields

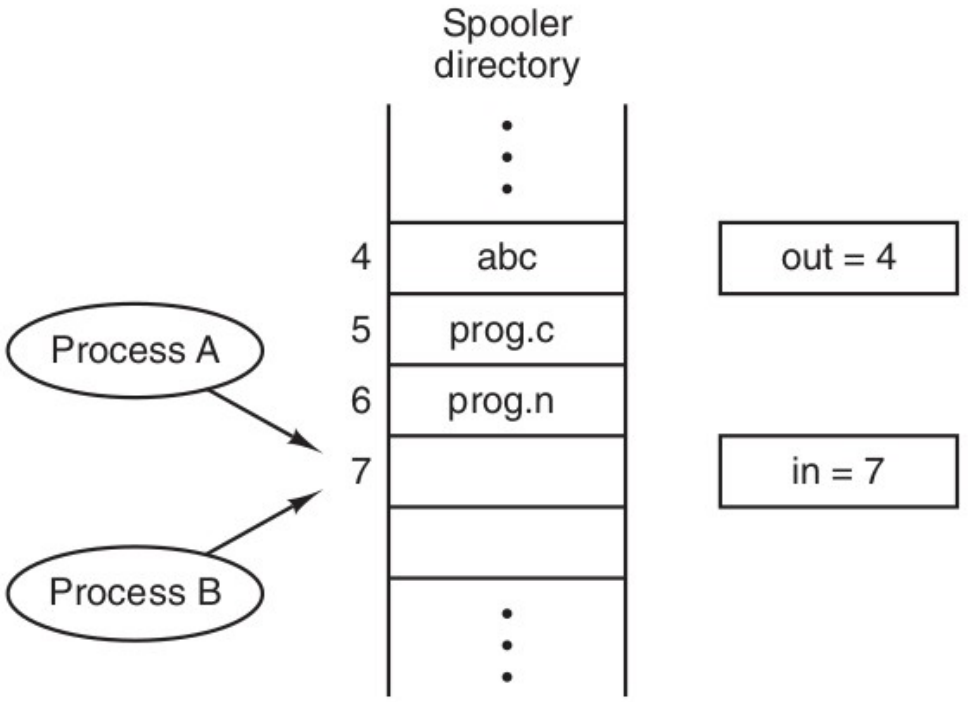
Process management	**Memory management**	**File management**
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Threads items

Per-process items	Per-thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

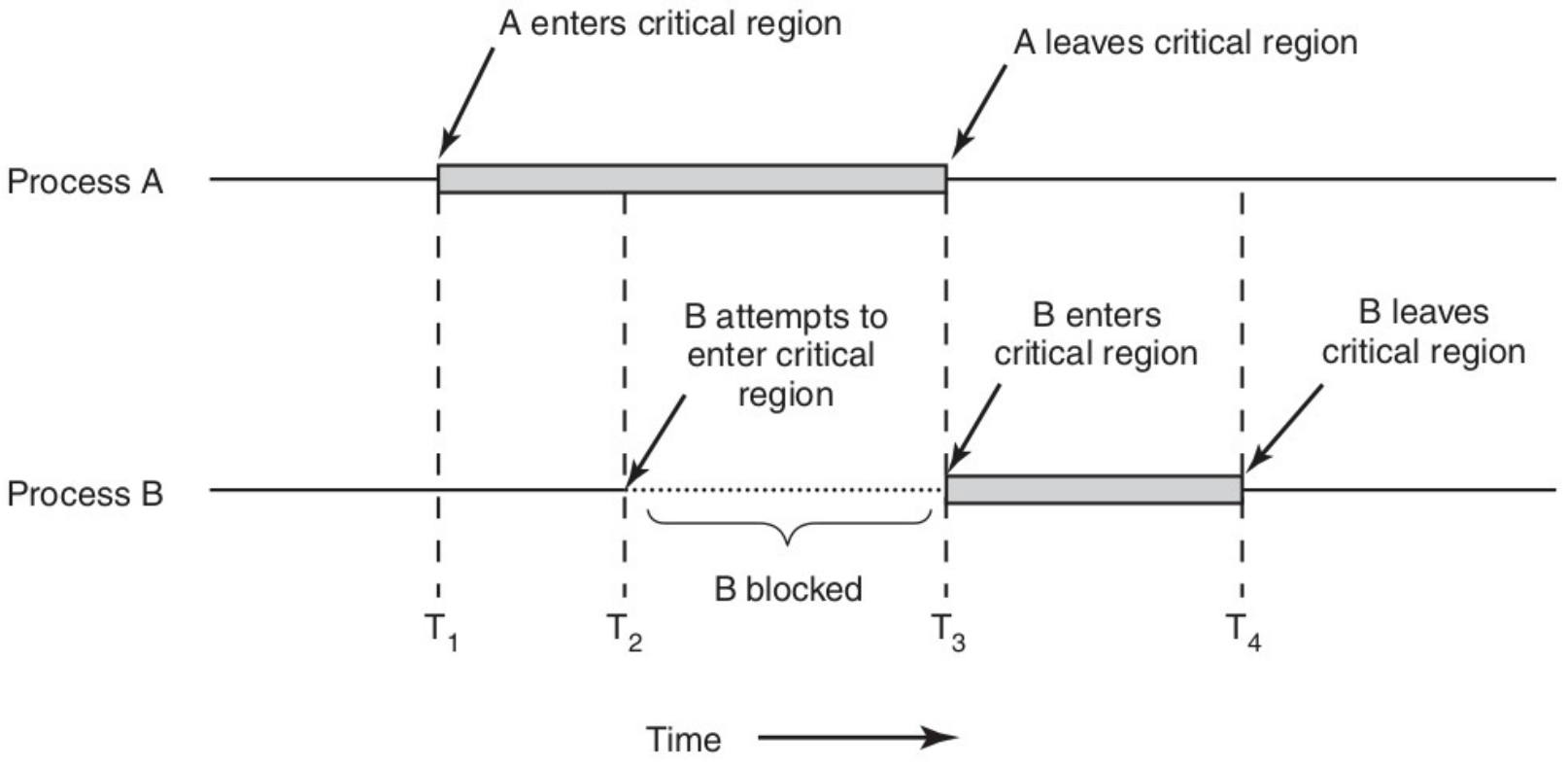
Figure 2-12. The first column lists some items shared by all threads in a process. The second one lists some items private to each thread.

Races

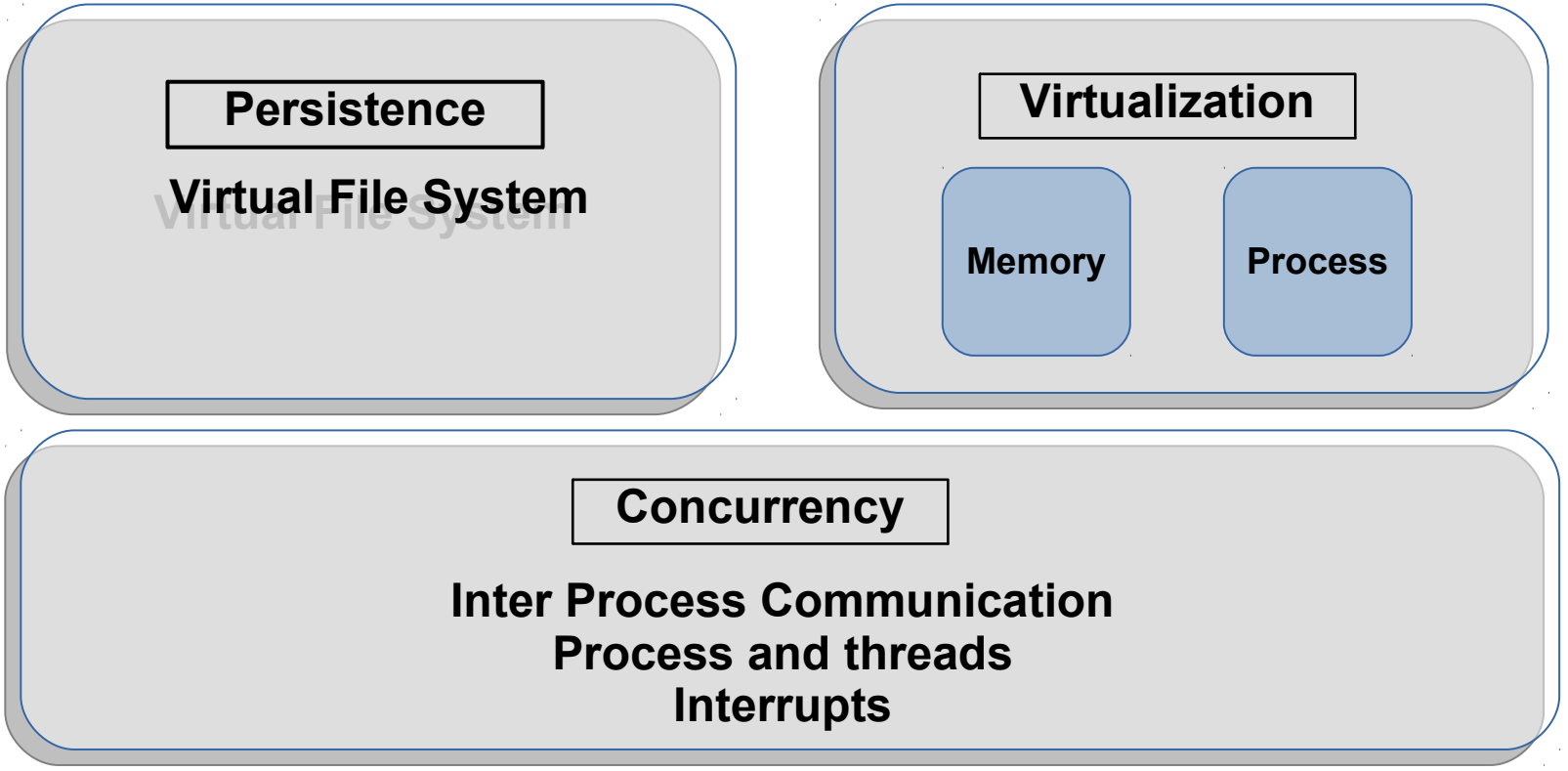


21. Two processes want to access shared memory at the same time.

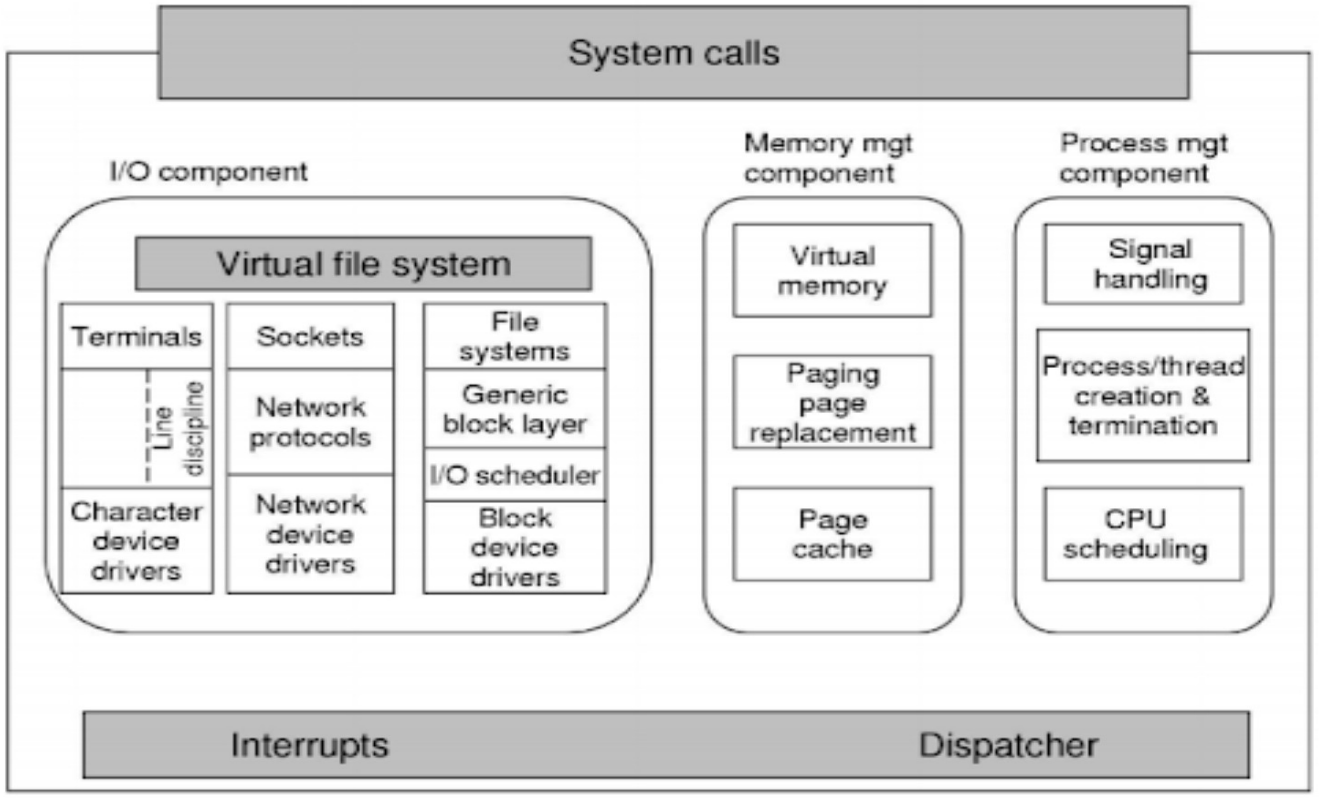
Mutual exclusion (critical region)



Operating Systems abstractions



Linux layered structure



Structure of the Linux kernel