CPU Mode

Three major subsystems of an OS

-User space: application , library

-Kernel : Process management , memory management, device management

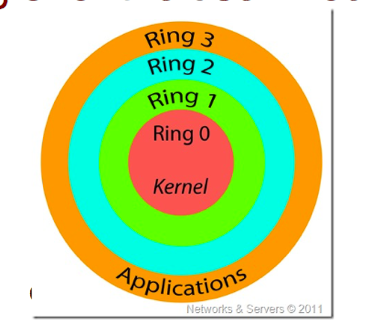
-Hardware: CPU, memory, device

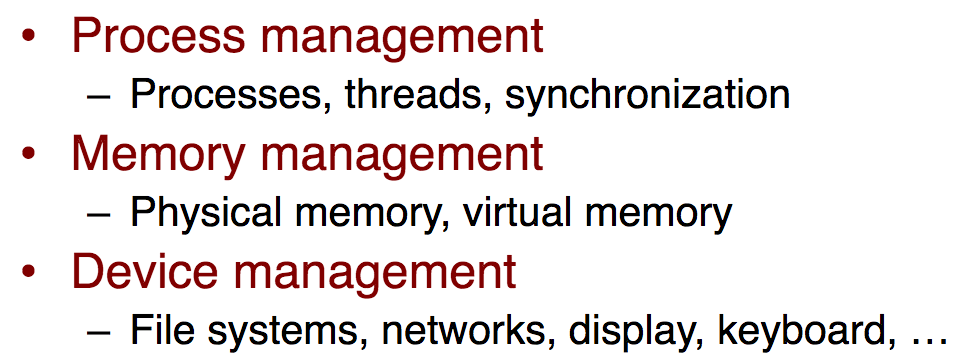
How CPU Modes are implemented

-Implemented through protection rings

A ring with a lower number has higher privileges

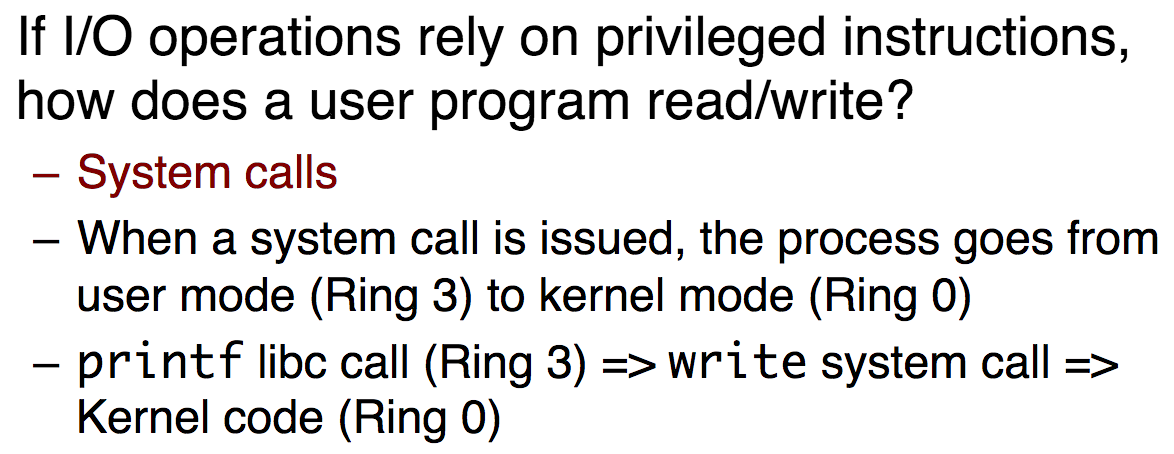
Ring 0 for the kernel mode and Ring 3 for the user mode





Privileged instructions can only be issued in the

kernel mode



In Kernel mode, the executing code has complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address.

In User mode, the executing code has no ability to *directly* access hardware or reference memory. Code running in user mode must delegate to system APIs to access hardware or memory.

**root” refers to the kernel mode**

**no!**

Long answer: the root user and non-root user refer to the

user account types

; in Linux/Unix, the root user can

access any files, while a non-root user only has access

to some files

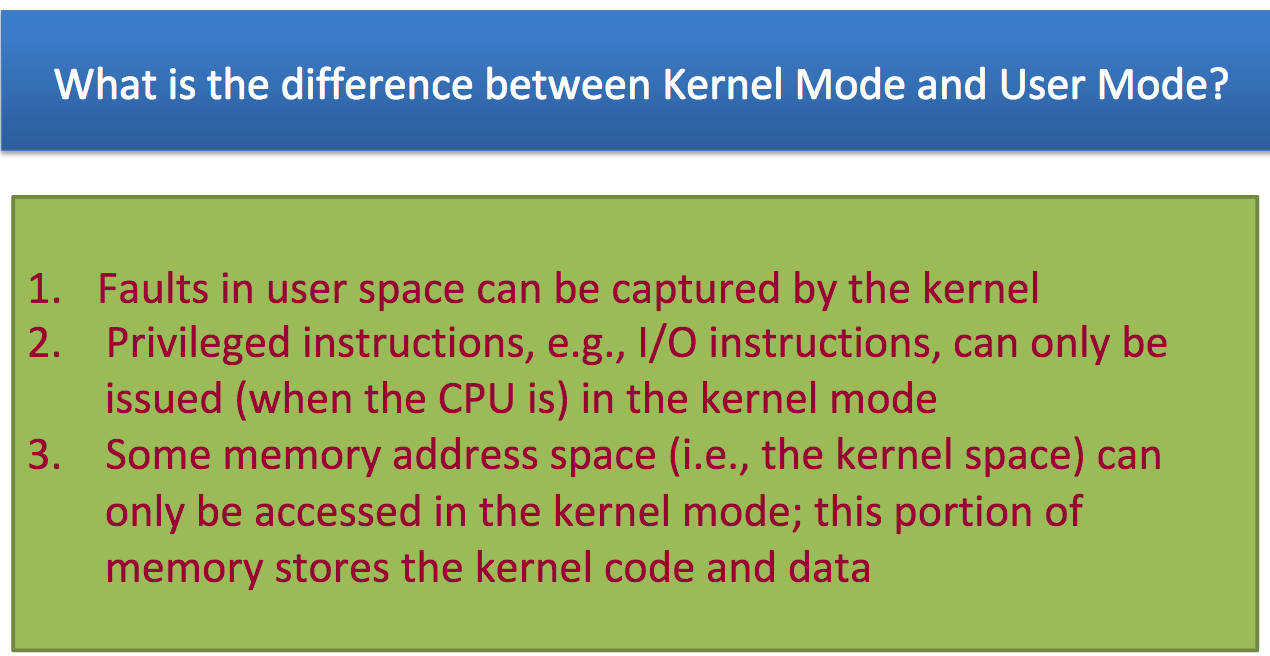
-Kernel Mode and User Mode refer to the processor

mode

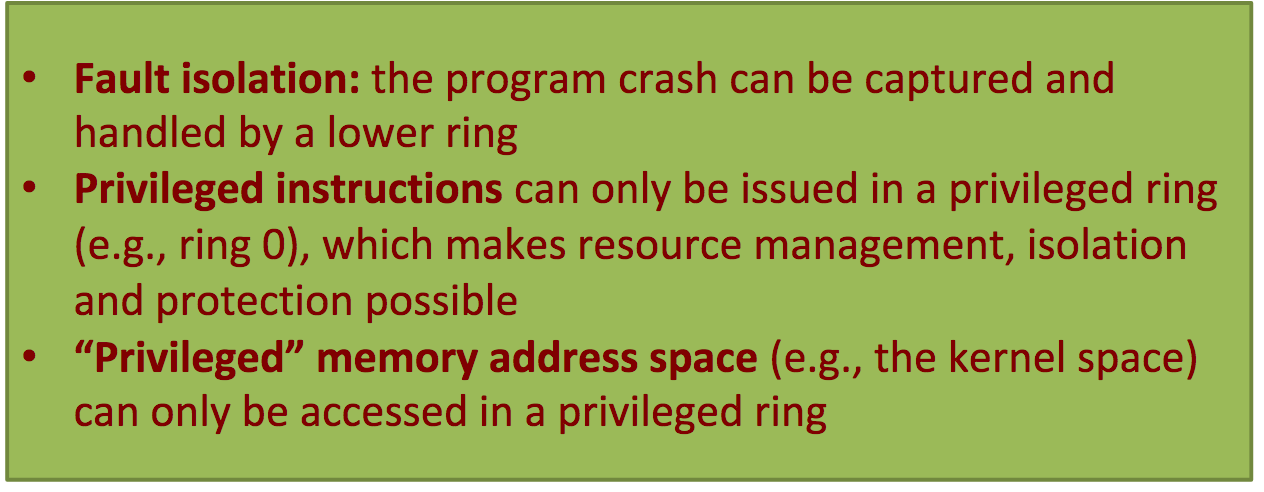
**Program Counter (PC)** register in

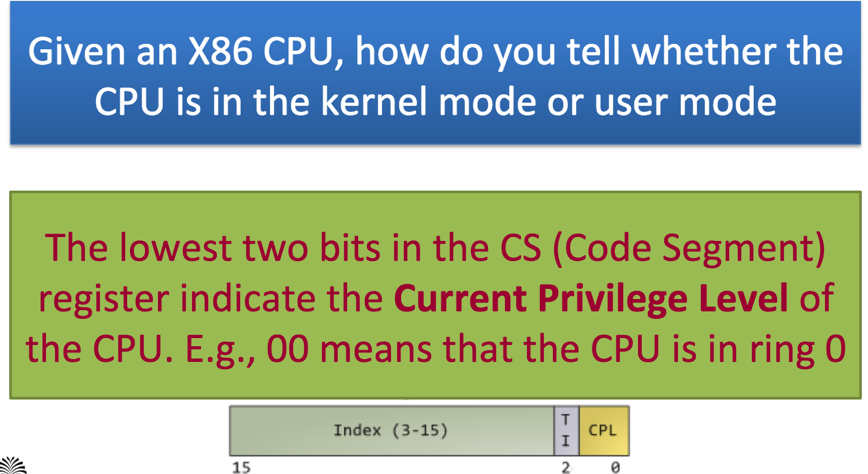
a processor stores the address of the instruction to

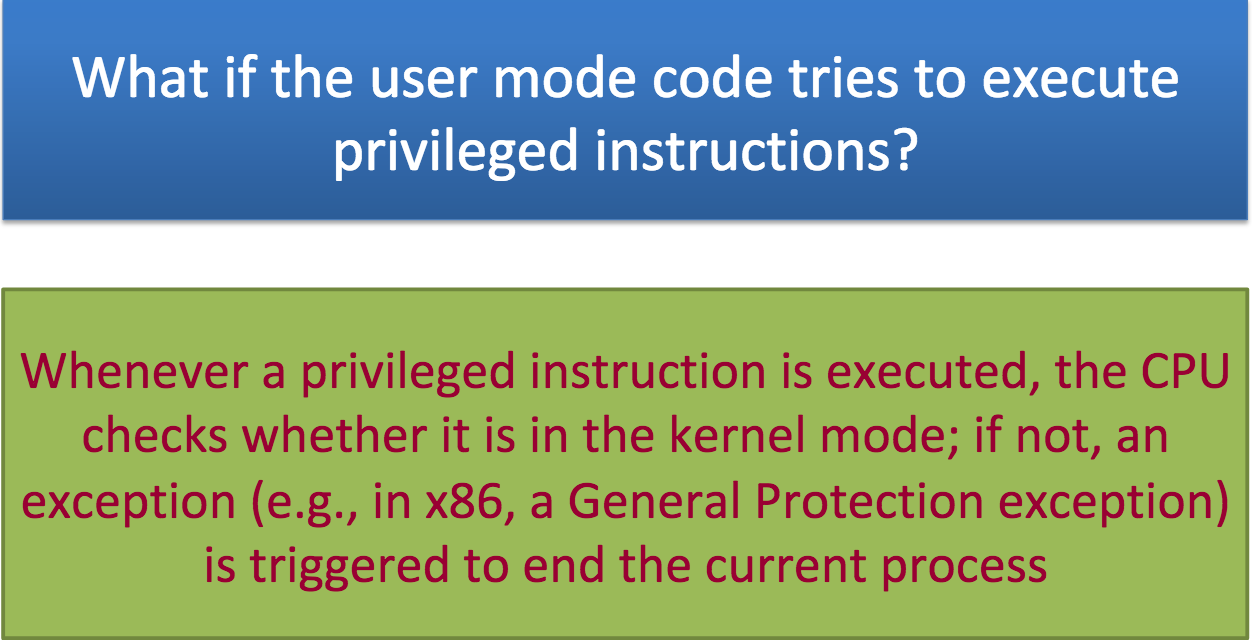
be executed

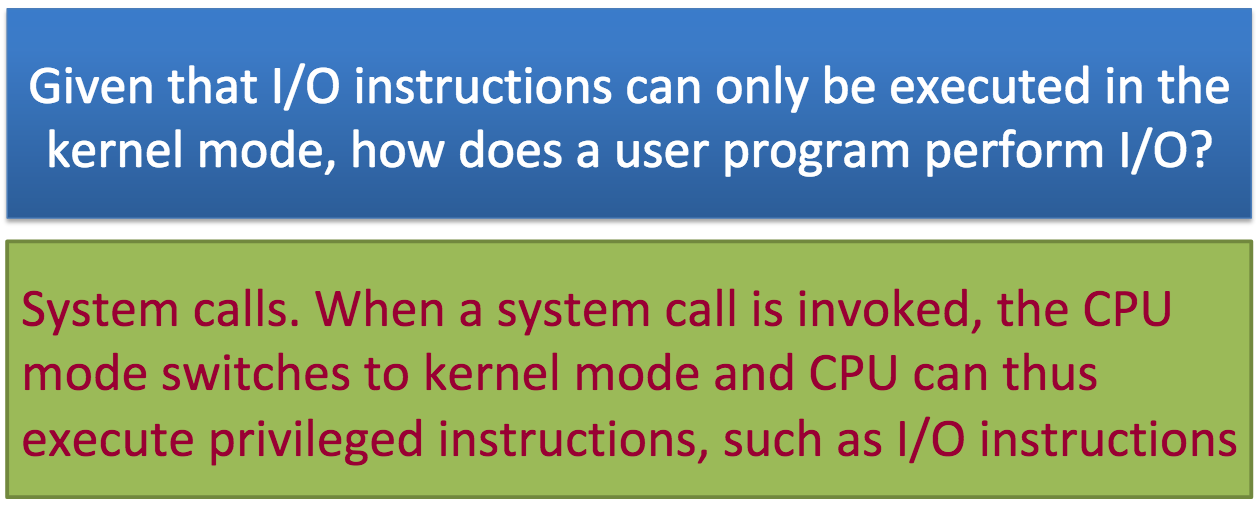


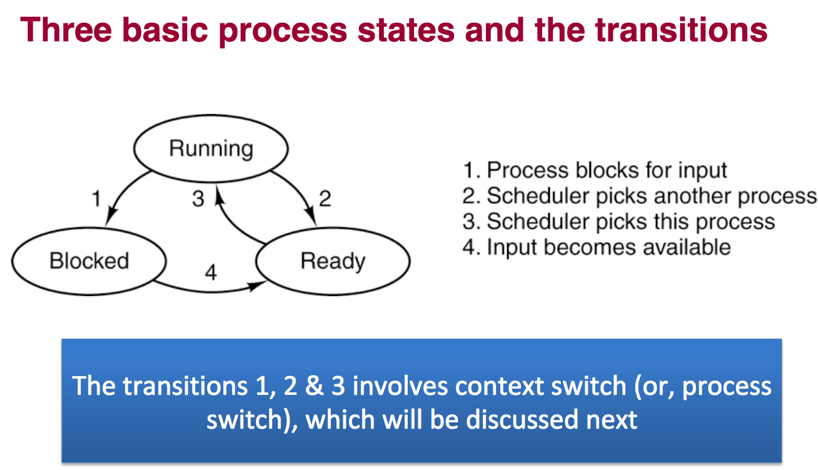
Why are Protection Rings needed











Where is the context information stored

-Process Control Block containing

the information needed to manage a particular process (scheduling state, process ID)

**A Process Switch :**(or Task Switch) occurs when

the OS scheduler suspends the execution of one

process on a CPU and runs another

The effect is that the first process is scheduled

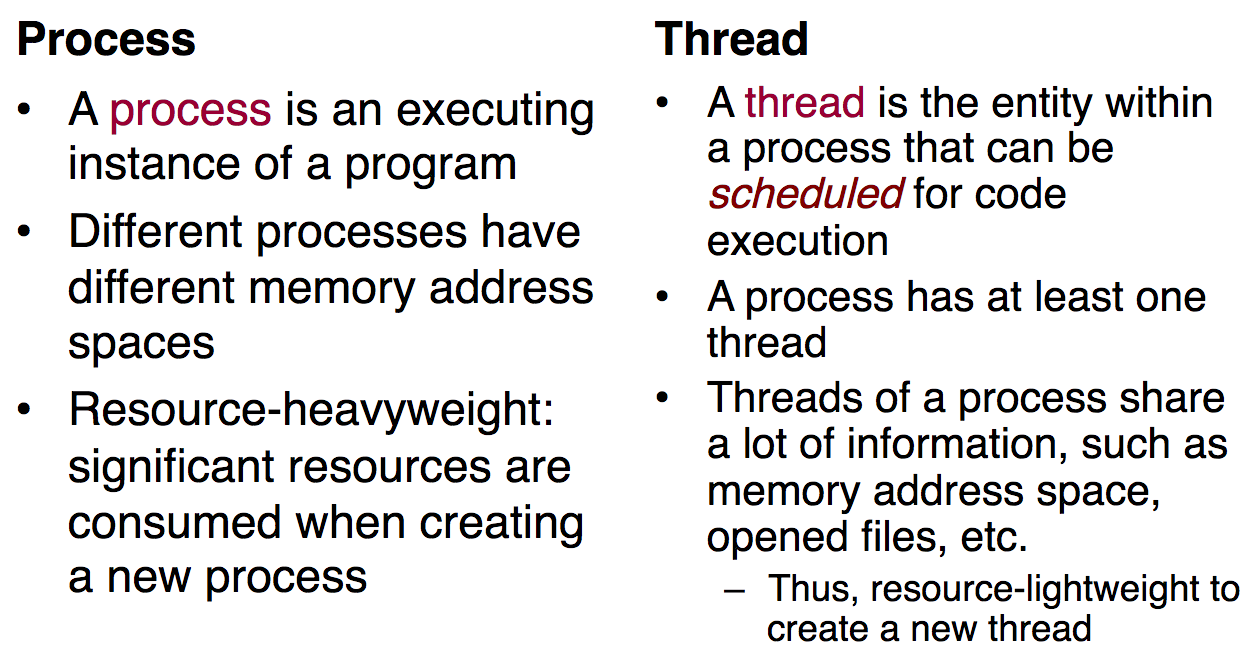
Out and the second process is scheduled in

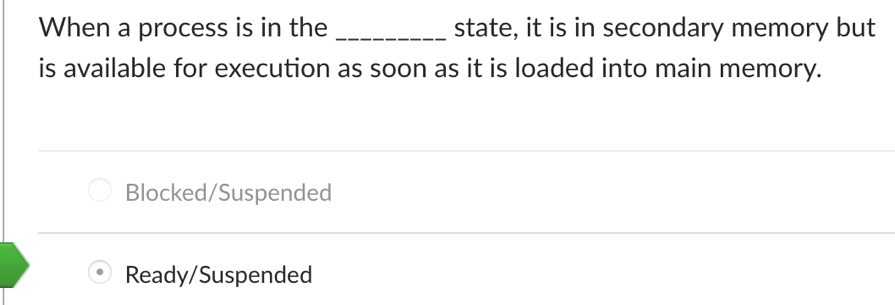
When does a Process Switch occur?

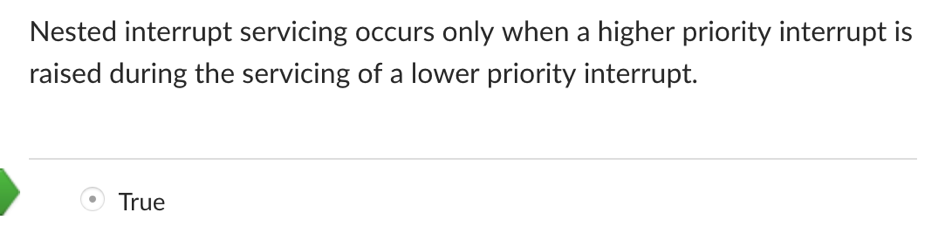
•A process blocks (e.g., due to I/O)

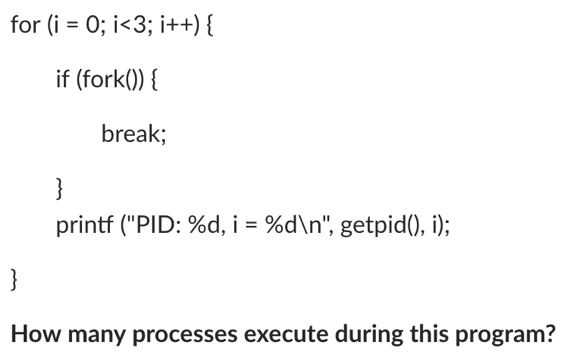
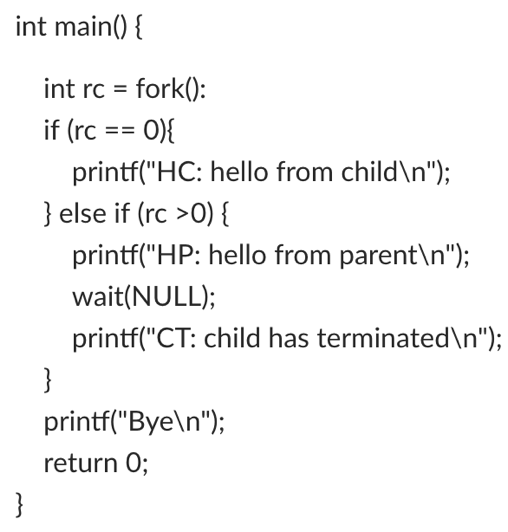
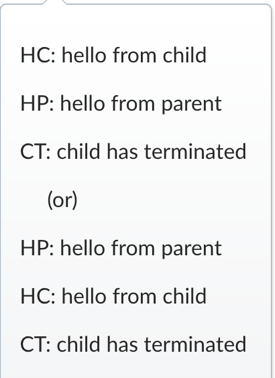
•A process exits

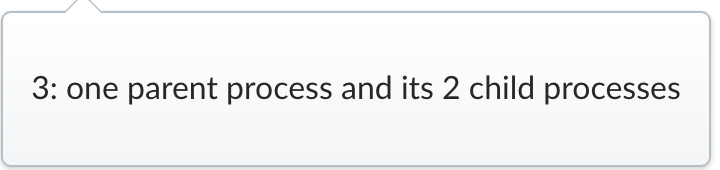
•The CPU time slice is used up

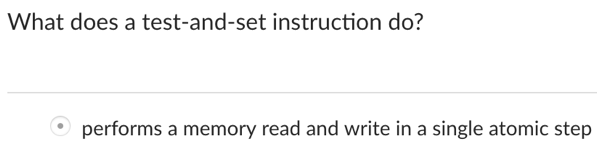
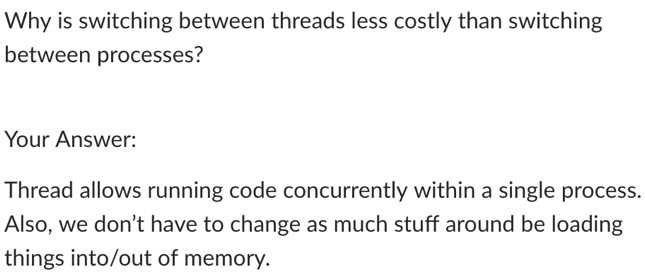






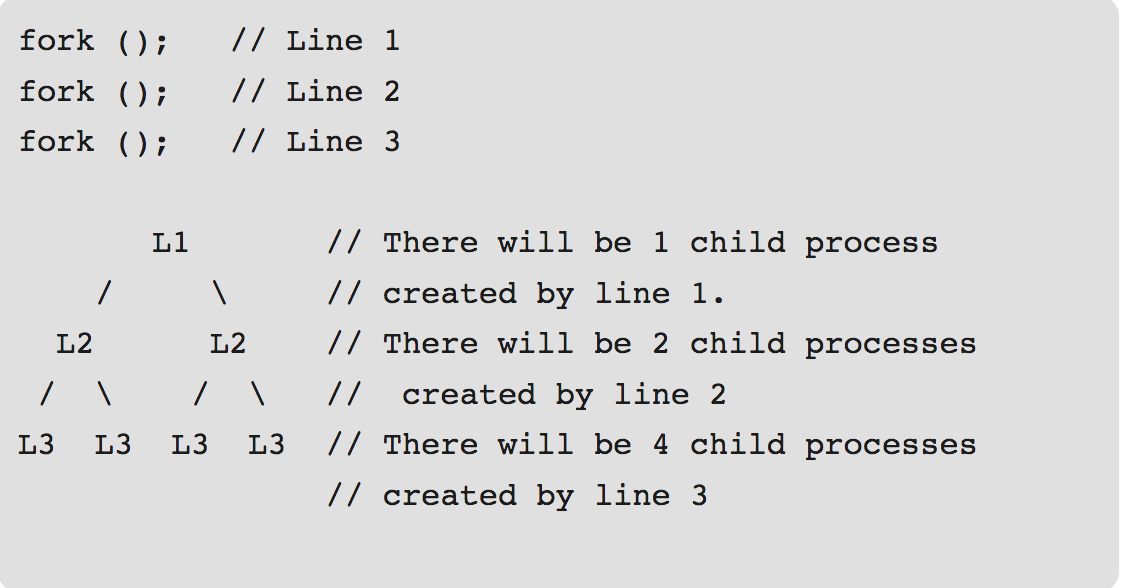


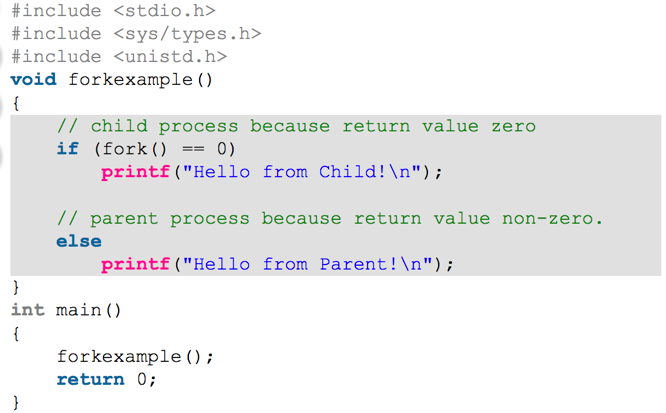
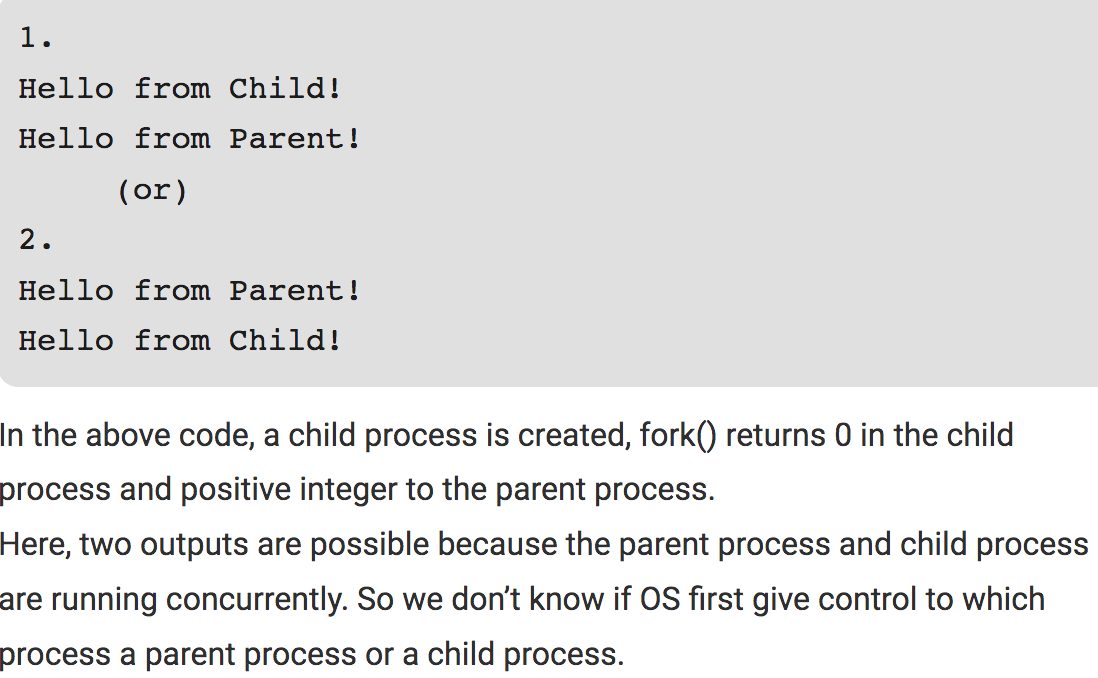


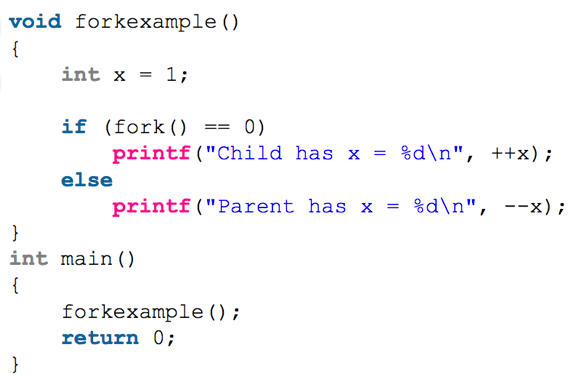
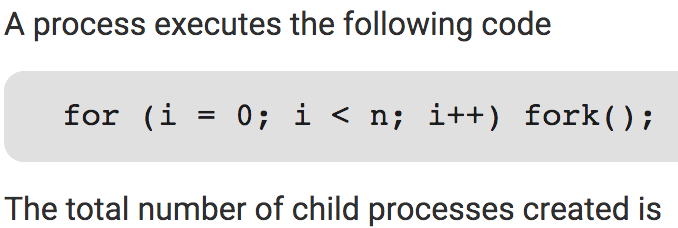
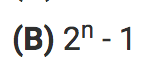


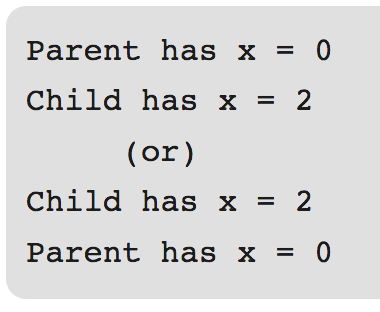
Fork(): 21 = 2 process, 1 child

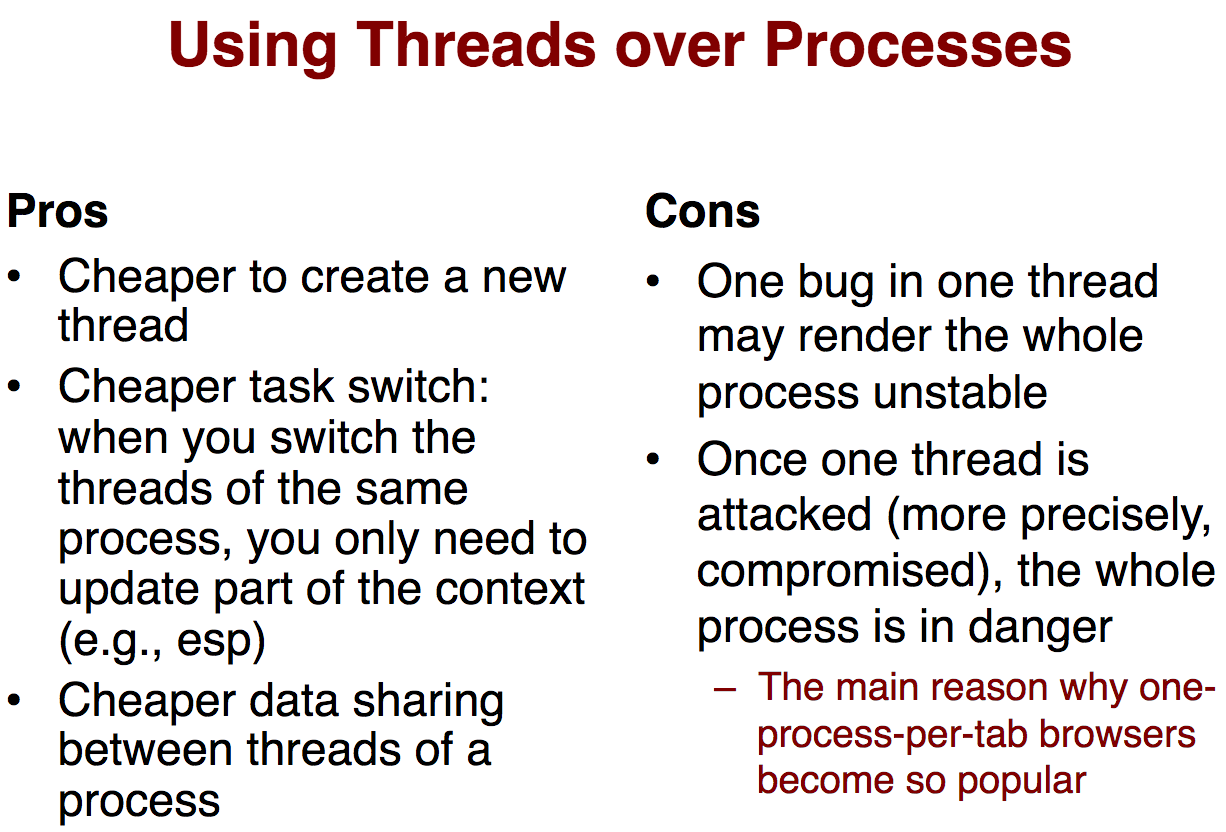
Fork (), Fork(), Fork() : 23  = 8 process, 7 child

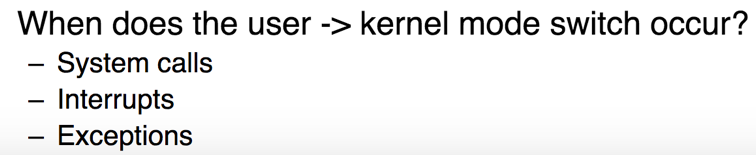


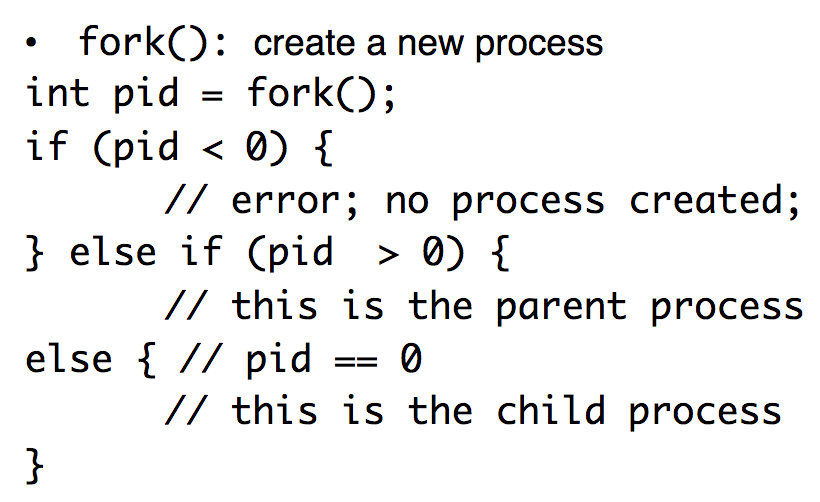


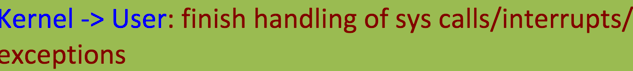










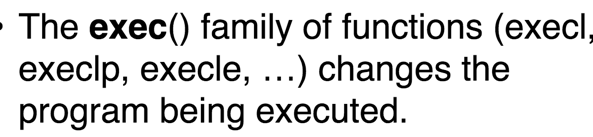


Why could the fork fail

-not able to allocate memory for process, lack of available resource

what happen when fork call fail

-

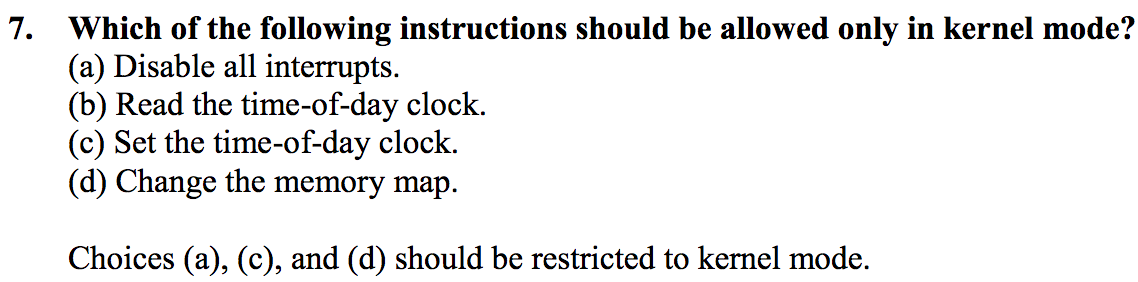


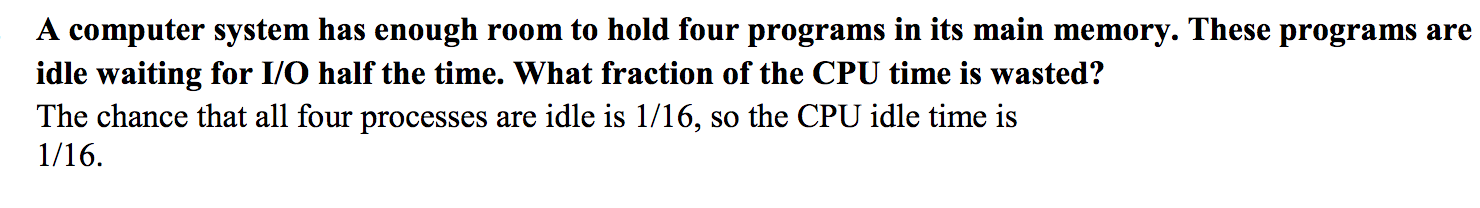
Screen%20Shot%202018-12-11%20at%204.28.55%20PM.png

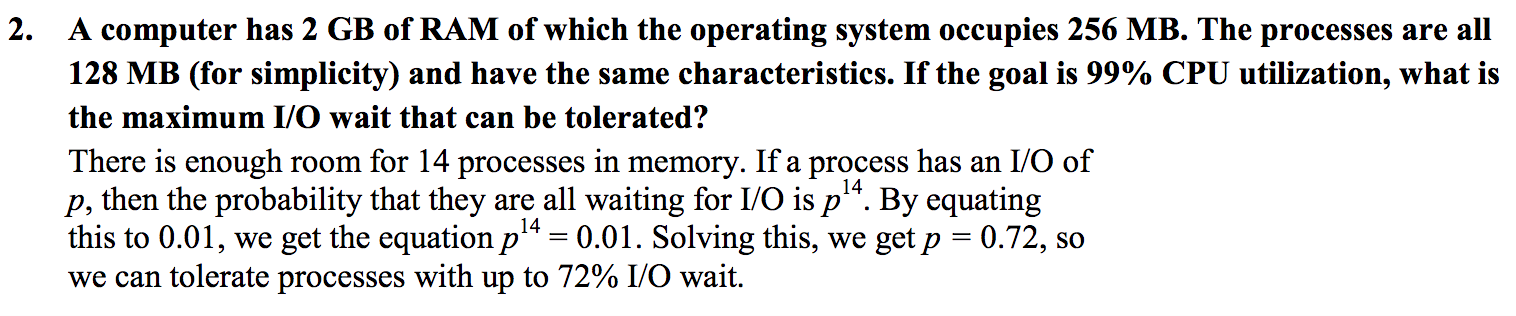
wait() system call suspends execution of

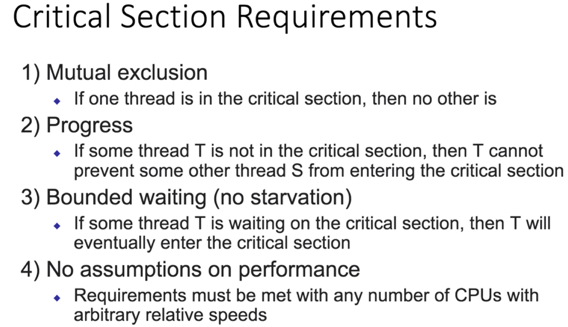
the calling process until one of its children

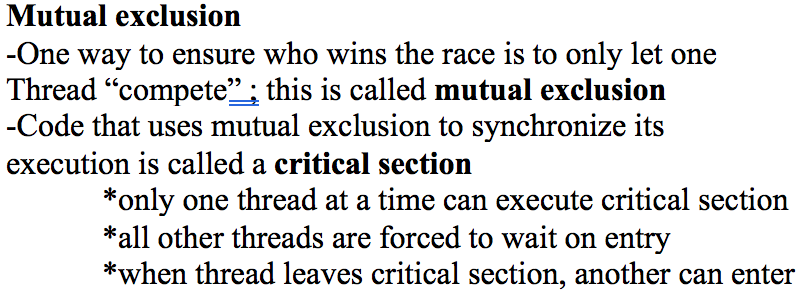
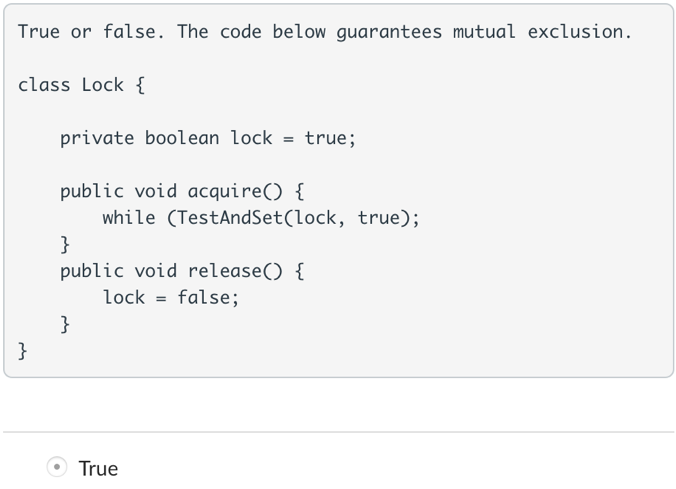
terminates.



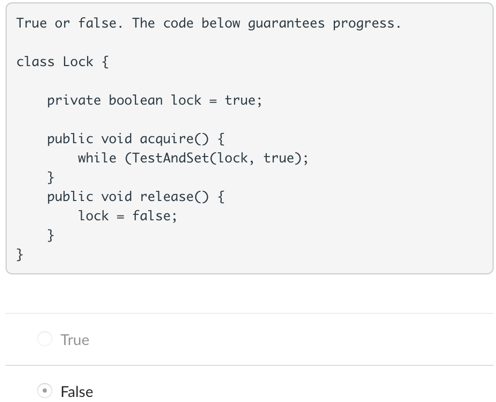


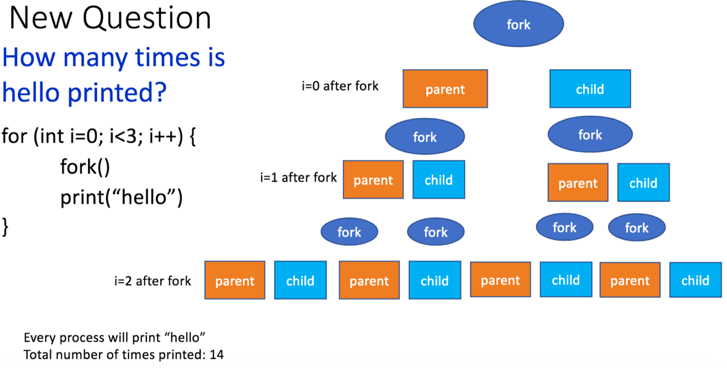


****





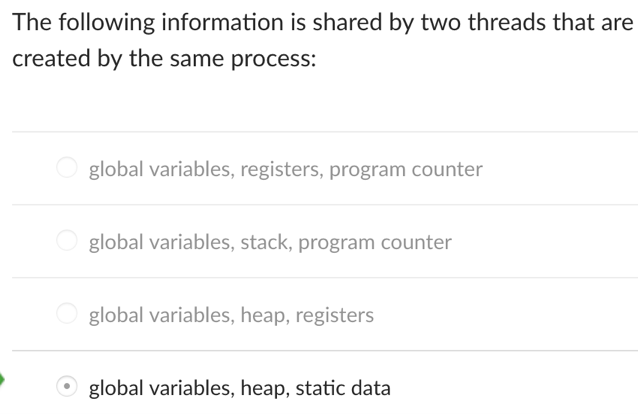


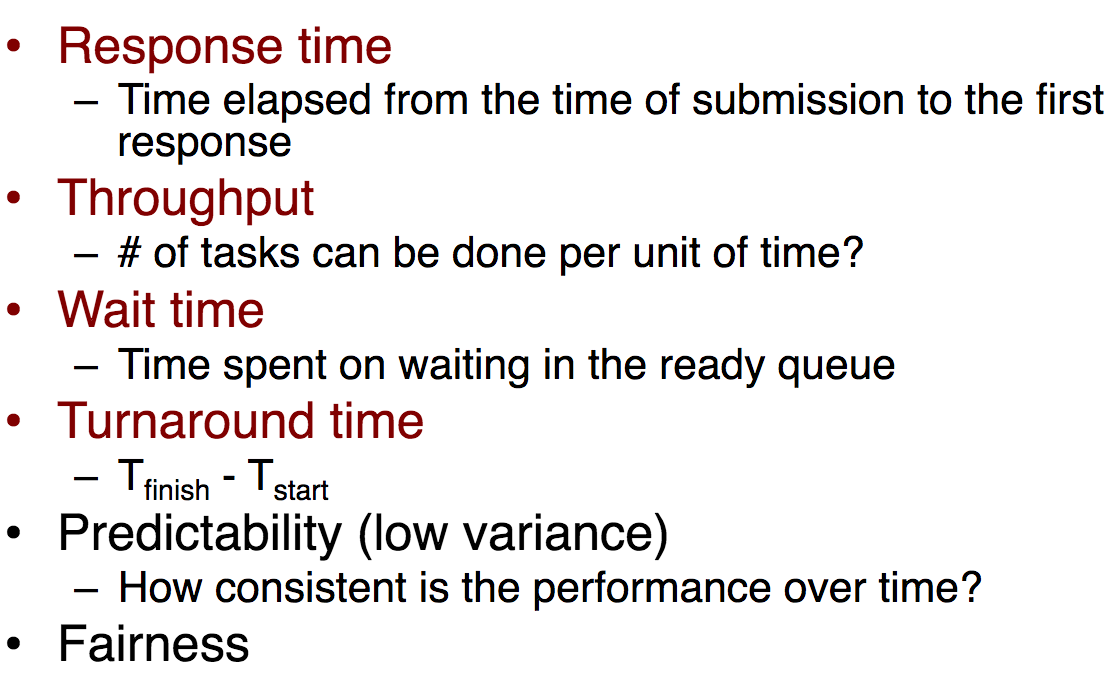


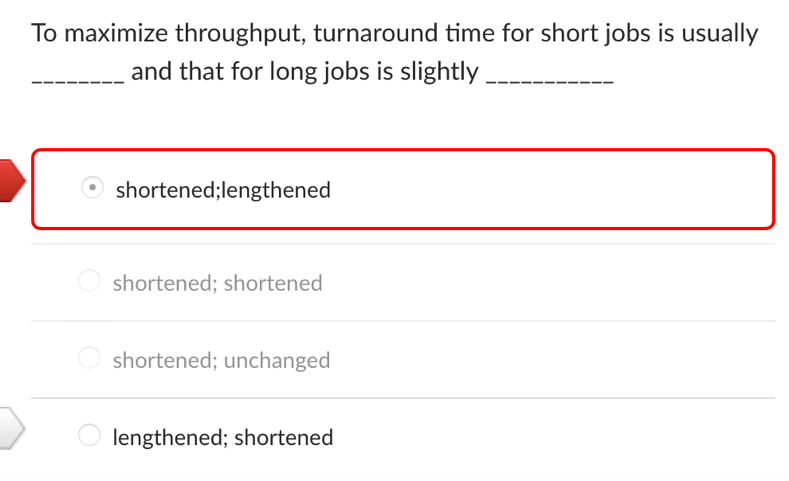
**Threads are made to share**

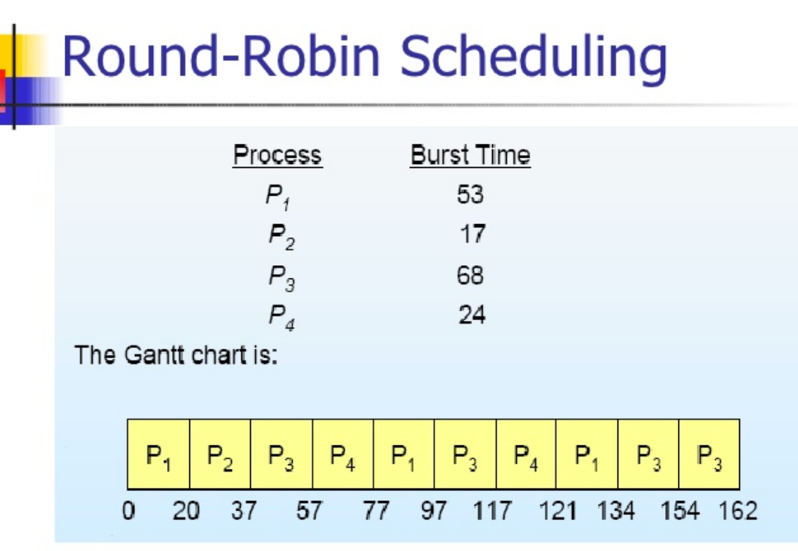
**-**shared: global variable and static object, dynamic object and other heap object

-not shared: local variable

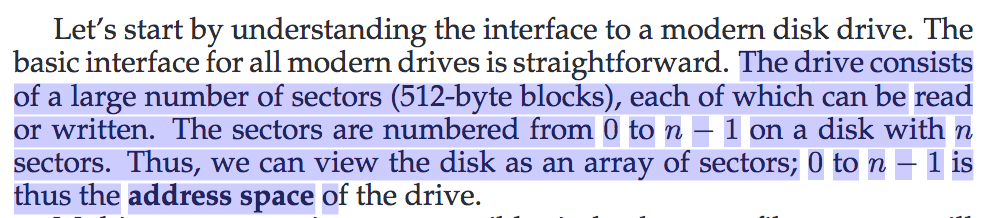


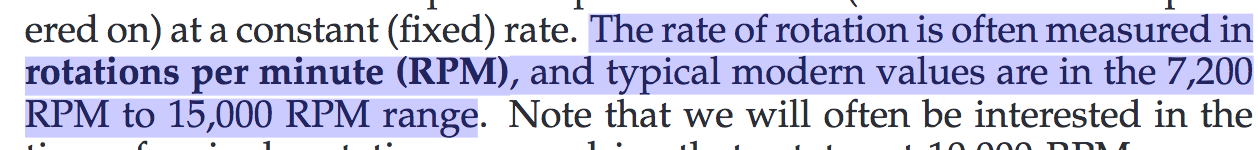




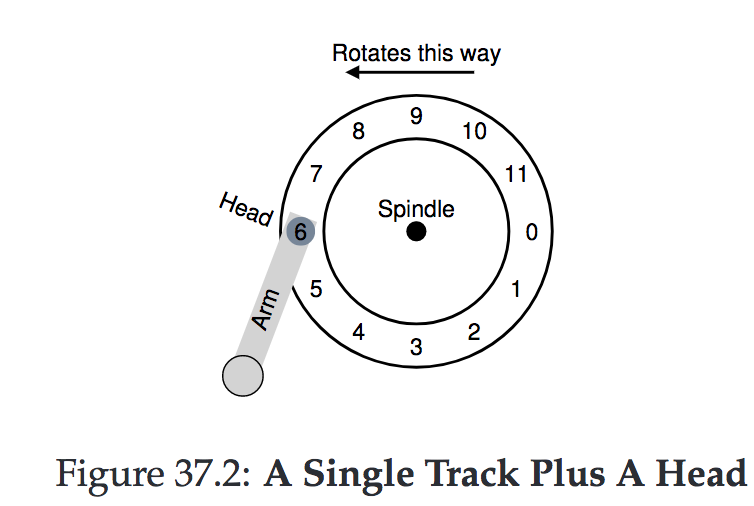


Hard Disk Drive





a drive that rotates at 10,000 RPM means that a single rotation takes about 6 milliseconds (6 ms).



This track has just 12 sectors, each of which is 512 bytes in size

(our typical sector size, recall) and addressed therefore by the nu

mbers 0 through 11.

I/O Time: Doing The Math

