From last time

Which of the following operations would you expect to be privileged (available only in System mode) & which available in User mode?

– halt the processor?
– system call?
– write an absolute memory location?
– load register from memory?
– disable interrupts?
– load stack pointer?
– write to segment or page not present in memory?
– change memory management register value?
– write to Program Status Register?
– write to interrupt vector table?

MANCHESTER 1824

The University of Manchester

COMP25111: Operating Systems

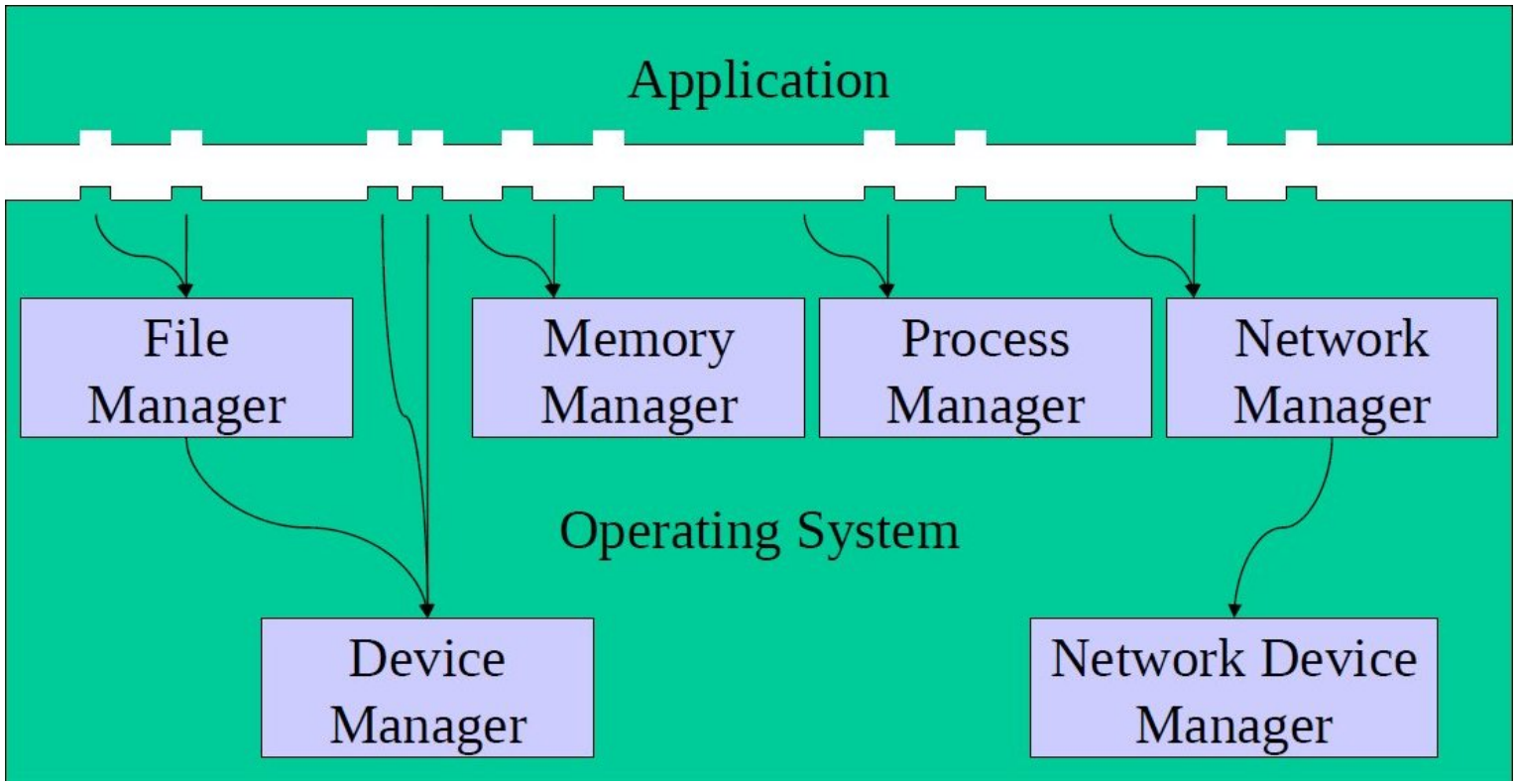
Lecture 5: The Process Manager – Processes & Threads

John Gurd

School of Computer Science, University of Manchester

Autumn 2015

Overview & Learning Outcomes



Process manager supports:

– Processes & Multi-processing

– Threads & Multi-threading

Processes

Early computers: one program at a time

Time-sharing → more control & protection

share 1 CPU & 1 Program Counter register

Process = executing program, in its own virtual CPU

Real CPU switches back and forth from process to process

e.g.
4 processes

(MOS fig2.1c)

Process: a program in execution

Not a program on a disk (= a file)

Process-switching keeps CPU busy

OS = collection of processes

“Process = Thread + Address space”
+ Register values
+ External interfaces

Thread (flow of control)
= abstraction of instruction-sequence obeyed by CPU

Process Creation

– System initialisation

– Running process executes process-creation system-call

– User request to create a new process

Parent process creates new **child** process(es)

via a “create-process” system call:
– UNIX: `fork()` and `execve()`
– Win32: `CreateProcess`

Process Hierarchy

Unix-based OSs: process & descendants associated

```
rpc-rizos-> ps -ef
UID        PID  PPID  CMD 0
root         1     0  init [3]
root         2     1  [keventd]
root         3     1  [kapm-idled]
...
root       563     1  /usr/sbin/sshd
root       585     1  xinetd -stayalive -pidfile /v
...
root      1991   563  /usr/sbin/sshd
rizos     1992  1991  -ksh
root      2234   585  in.rlogind
root      2235  2234  login -- rizos
rizos     2236  2235  -ksh
rizos     2380  2236  /bin/bash /usr/local/bin/nets
rizos     2392  2380  /usr/lib/netscape/netscape-co
```

Process Termination

Normal exit

Error exit

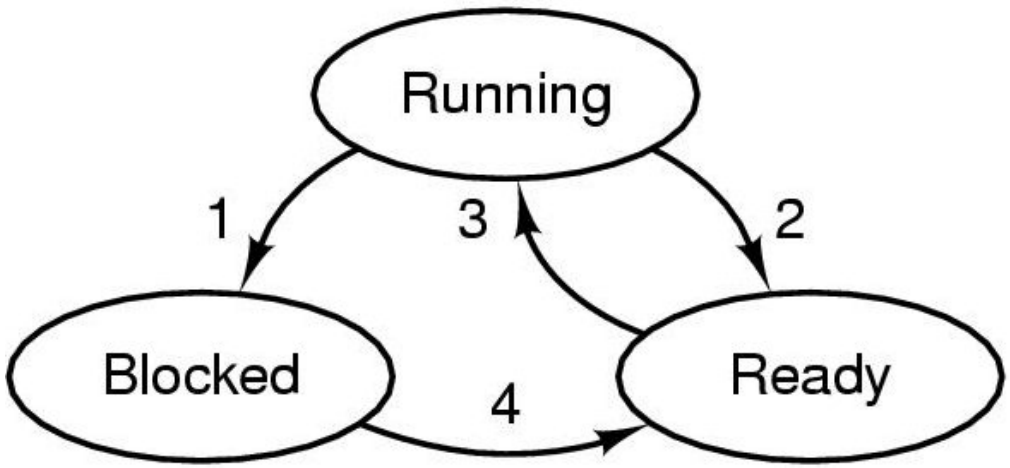
Fatal error

Killed by another process
– Unix: `kill`
– Win32: `TerminateProcess`

(in some systems) Parent process terminates

Process States

Newly created → Ready - **admitted**



(OSC/J fig3.2 (older - fig4.1); MOS fig2-2)

3: Scheduler selects process to run - **dispatch**

2: Process forcibly preempted - **interrupt / relinquish CPU / time-slice expired**

1: Process needs to wait for I/O or event - **block**

4: I/O or event occurs - **ready**

Running → Terminated - **exit**

Important Issues

Scheduling:

– which process to pick?

Context Switch:

– current process's state saved
– next process's state loaded

Process Control Block (PCB) (Process Descriptor)

OS maintains PCB table, 1 entry per process

PCB = all info needed to restart process as if it had never stopped (varies from system to system)

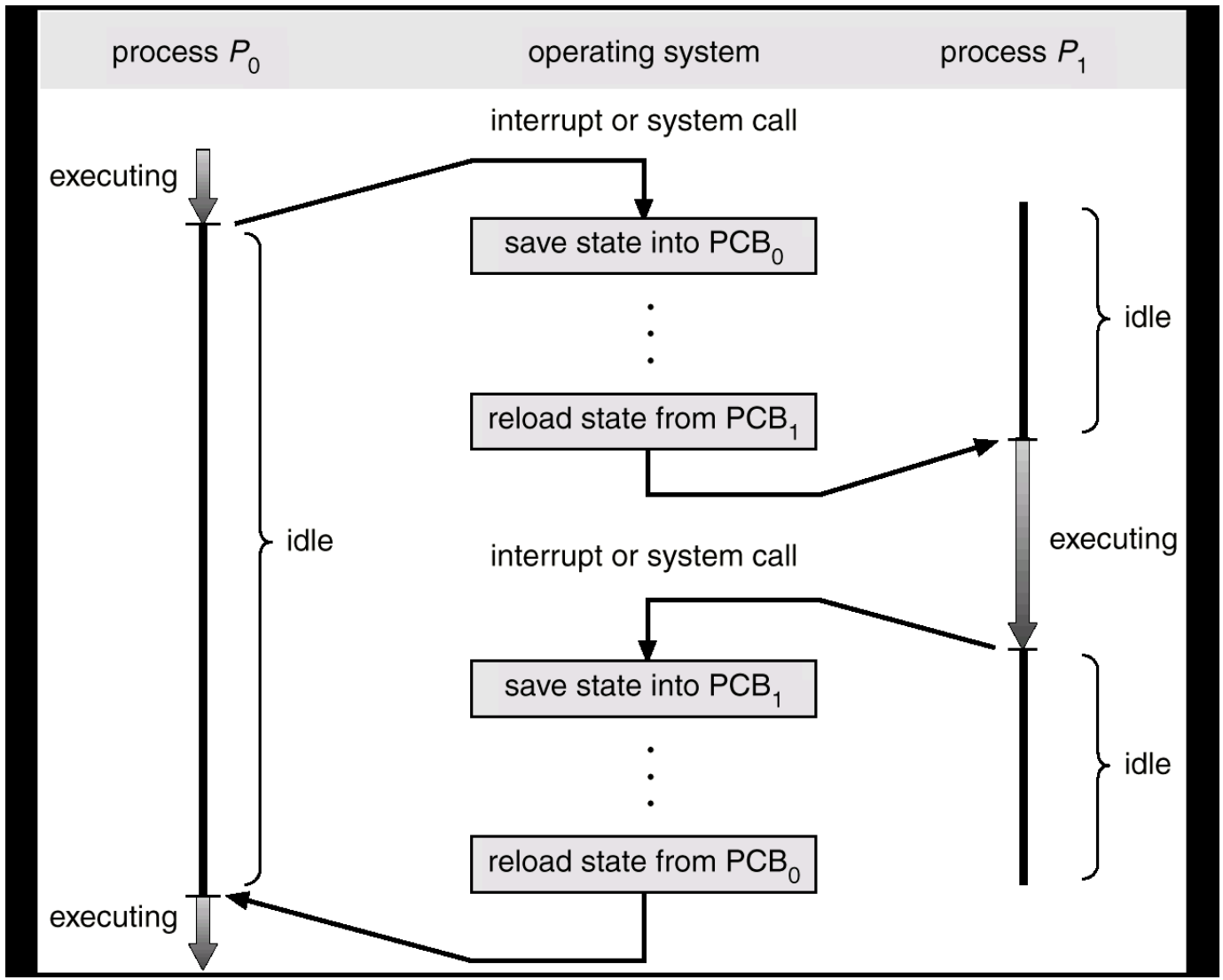
– PID (Process IDentification number)
– PPID (parent PID)
– Process State e.g. saved registers
– Memory Management info
– File & I/O Management info
– CPU Scheduling info
– Accounting information
...

Context switch CPU from process to process

Context switch is overhead

speed varies e.g. 1μs-1ms

(OSC/J fig3.4, older fig4.3)



Multiple flows of control within a process

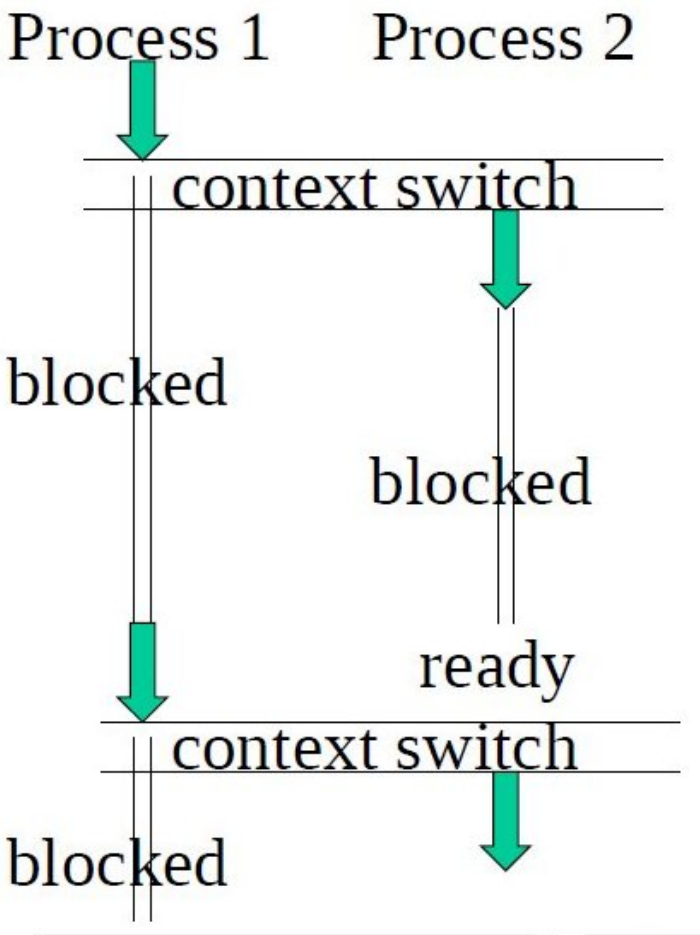
So far assumed 1 **thread** (flow of control) per process

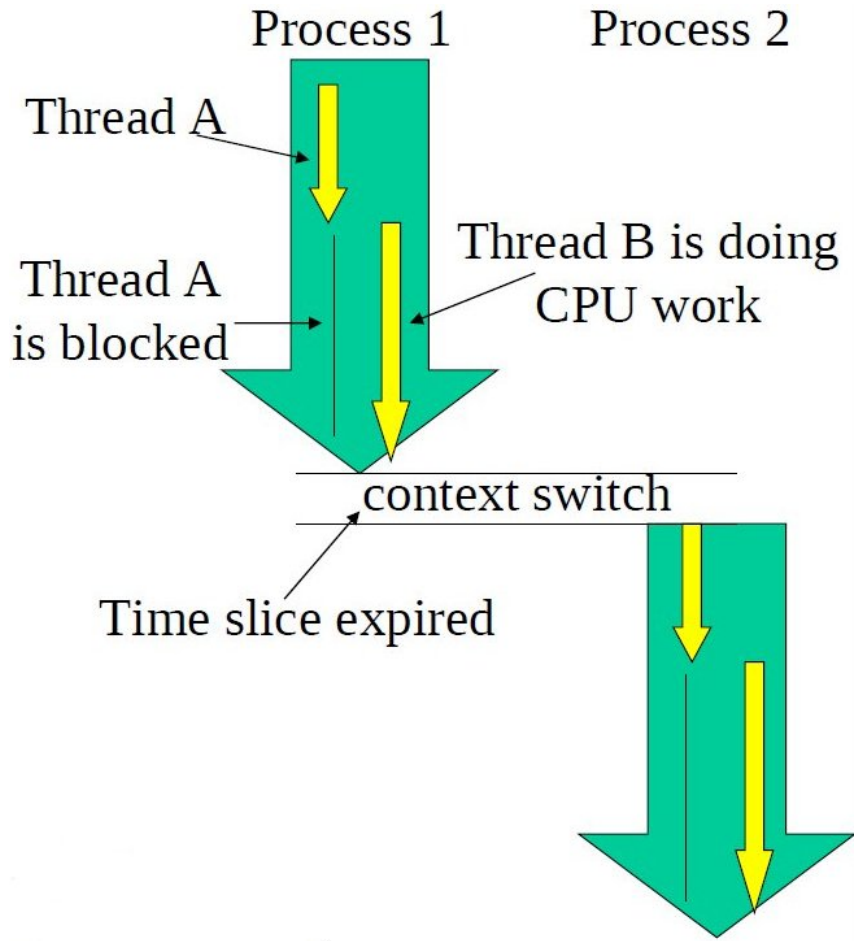
Multi-threading (multiple threads in one process)
– can improve user experience

Word processor: thread for UI + thread for time-consuming tasks

Web browser: thread to display images or text + thread to receive data from network

The Argument in Favour of Threads





Other benefits

(OSC/J sec.4.1.2 (older sec. 5.1.2), MOS sec.2.2.2)

Reduce context-switching
– process can do something even if part is blocked

Economy: thread creation much faster than process creation

Useful on systems with multiple CPUs

Threads (“lightweight processes”)

(OSC/J section 4.1 (older 5.1) MOS section 2.2.1)

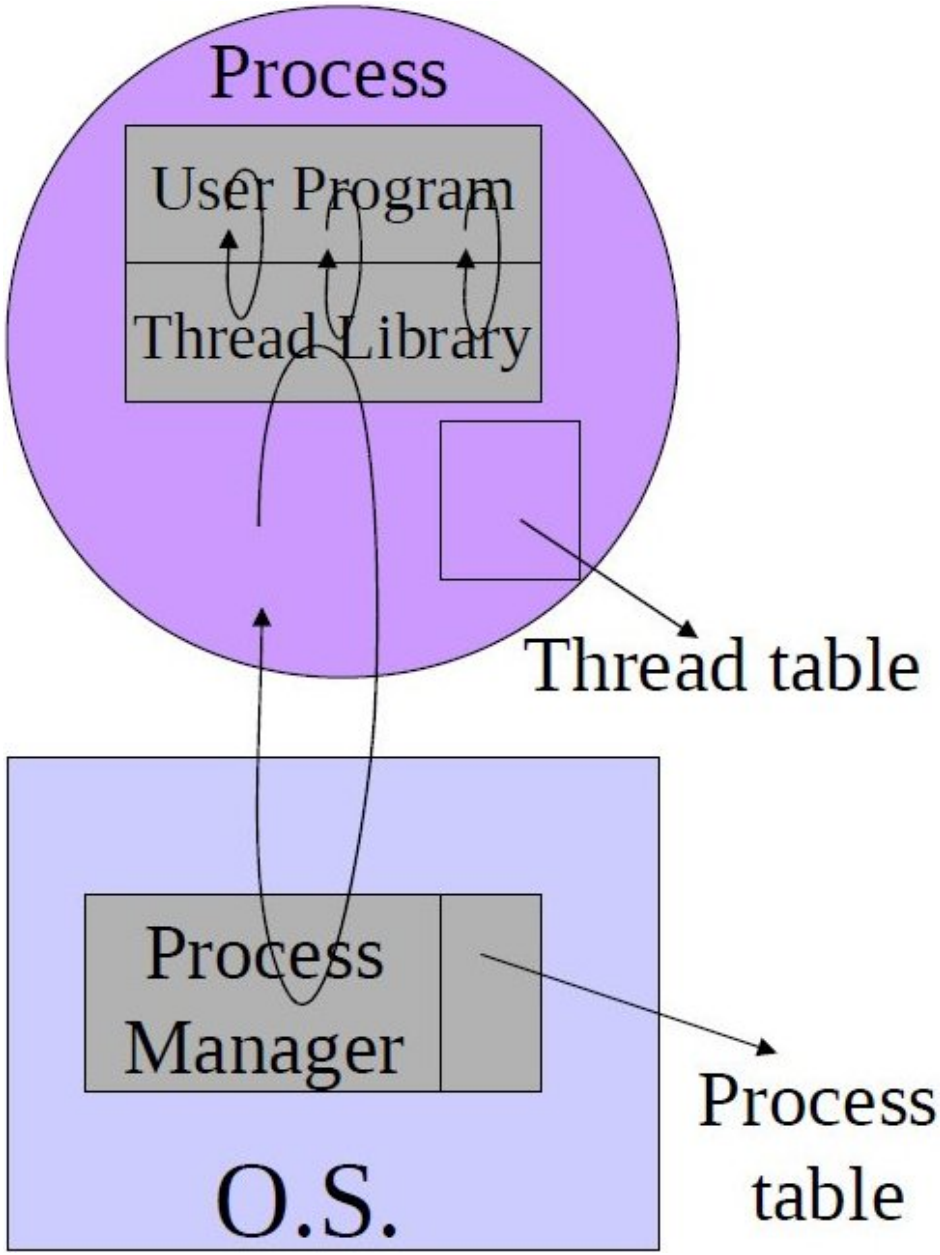
– multiple flows of control in one address space

– each needs program counter, registers, stack

– (in the same process) share code, global variables, open files, network connections

Harder to code!

User-Level Threads (Library)

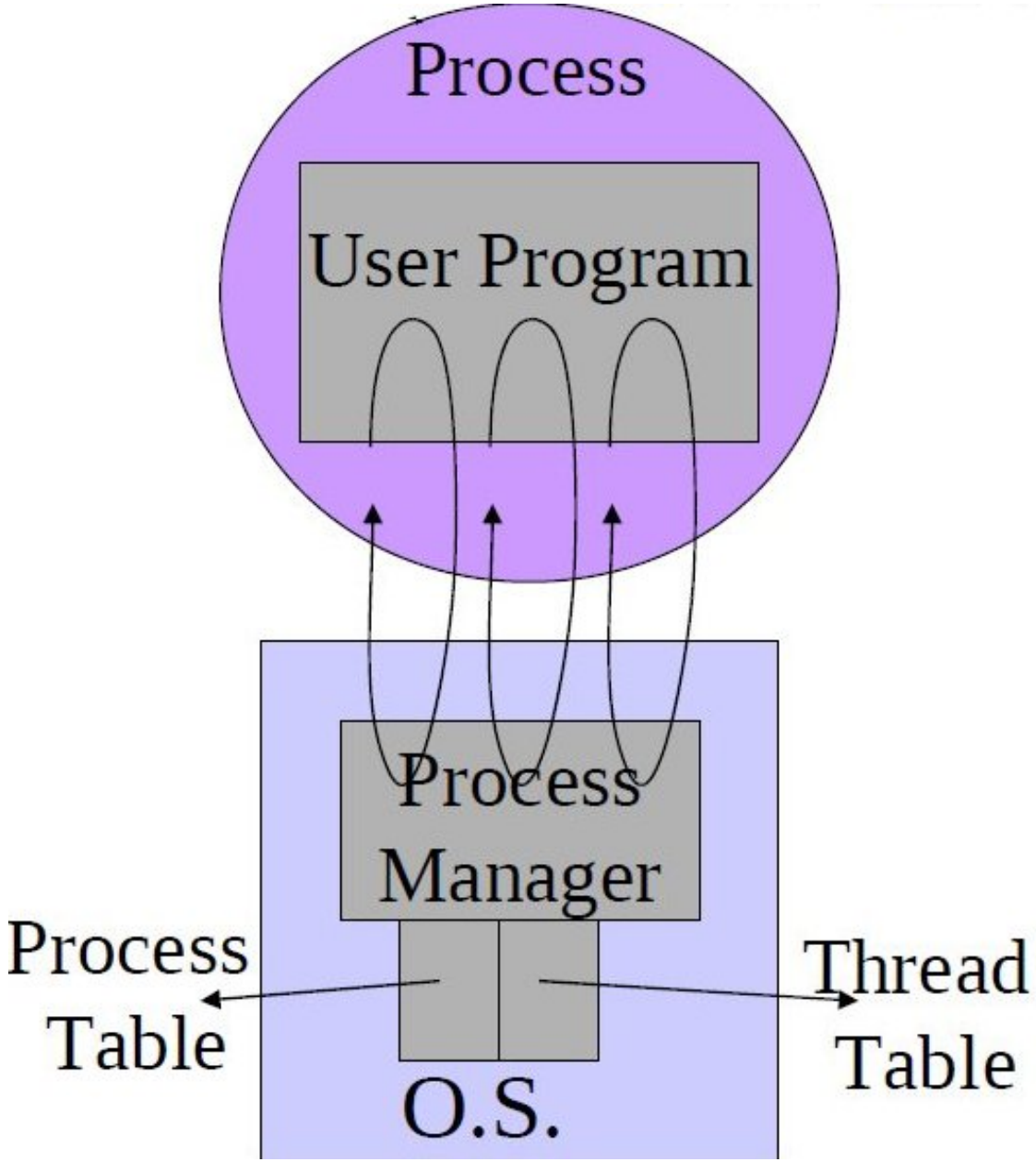


Thread creation & scheduling fast

Only on 1 CPU

Thread blocked → process blocked

Native or Kernel Threads (OS)



OS creates & schedules threads

Useful if multiple CPUs

Thread creation slower (system call)

Summary of key points

Process: a program in execution
– in one of 3 main states
– context switch, PCB

Thread: a flow of control within a process
– benefits
– User-level v. Native/Kernel

Next: Process (& Thread) Scheduling

Your Questions

For next time

Does each of the following appear in processes, programs, both, or neither?

– instructions
– read-only data
– registers
– a stack
– a heap
– network connections
– system calls
– a shared data area

Exam Questions

Explain briefly what is mean by the term ”multiprogramming” (2 marks)

Draw a diagram showing the various states of a process in an OS, and label the transitions between the states, and entry to and exit from the set of states, with comments explaining what causes a process to make that transition. (4 marks)

Of the three basic states that a process can be in, in which state does the number of processes at any given time depend on the number of CPUs available? Justify your answer.
(2 marks)

Describe the actions that occur when a context switch happens in an OS. (3 marks)

Glossary

Process
Multi-processing
Parent & child processes
PID, PPID
Running
Ready
Blocked
Context switch
PCB
Thread
Multi-threading
Lightweight & Heavyweight processes
User-level threads
Native/Kernel threads

Reading

OSC/J: Sections 3.1, 3.2.3, 3.3, 3.7, 4.1 (and skim thru rest of Ch3)

older OSC/J: sections 4.1, 4.2.3, 4.3, 4.7, 5.1 (and skim thru rest of Ch4)

MOS2: section 2.1 up to and including 2.2.4
MOS3: section 2.1 up to and including 2.2.5 but omit 2.2.3