

Reminder

- Lab 2 will be due tomorrow
- Today we will announce Lab 3
 - 6% of your final grade
 - Cache and (virtual) memory operations
 - Involving OS programming
 - Due on October 22

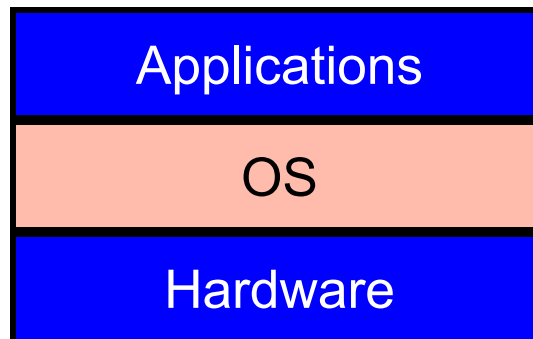
ECE 1175
Embedded Systems Design
Operating Systems - I

Wei Gao

What is an Operating System?

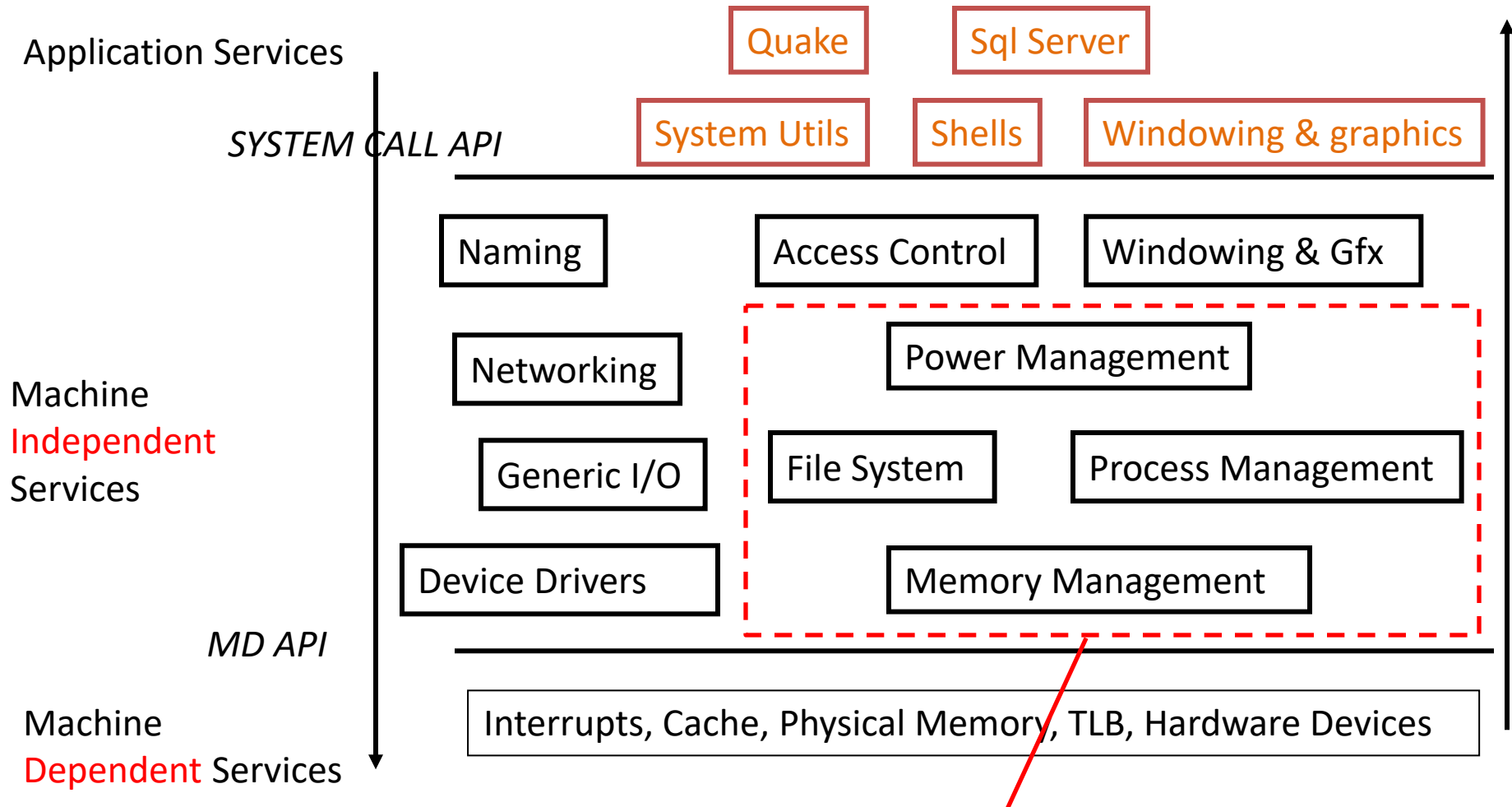
- Operating System

- A **software** layer to **abstract away** and **manage details** of hardware resources
- A set of utilities to **simplify application development**



- “all the code you didn’t write” in order to implement your application

Logical OS Structure



Crucial to Embedded Systems!

Key OS Issues

■ Multi-Programming

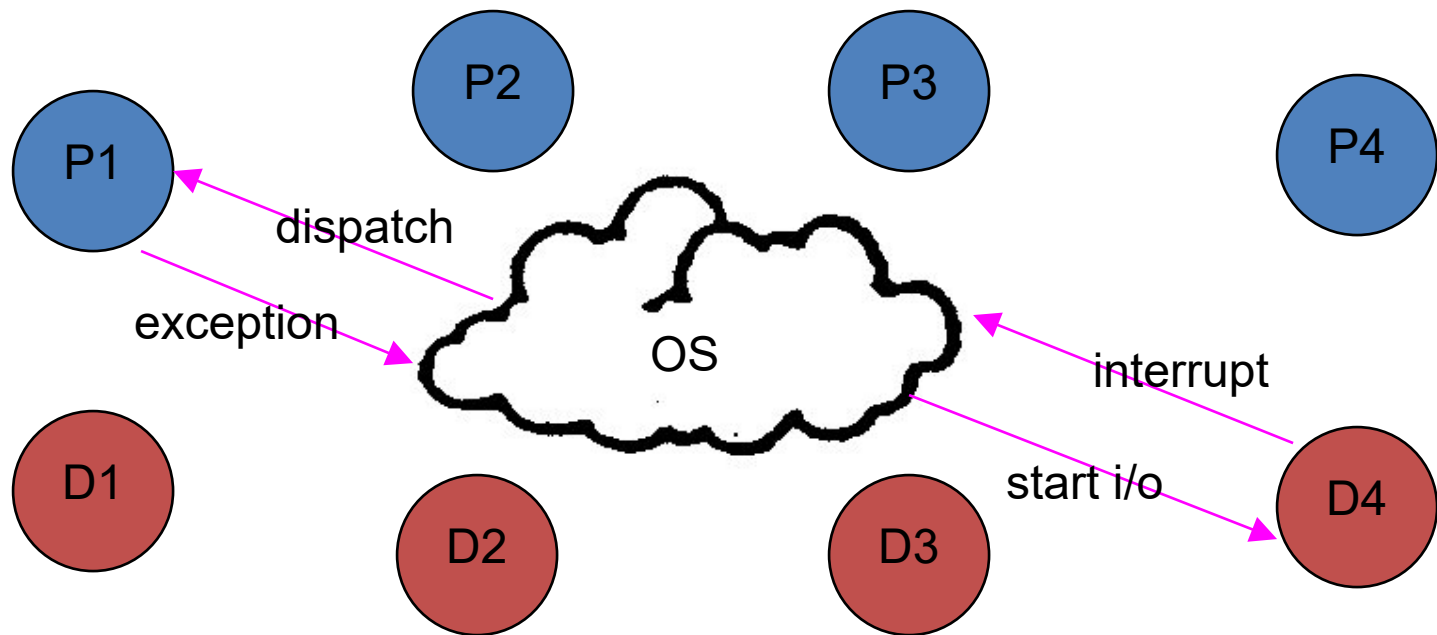
- keeps multiple runnable jobs loaded in memory at once
- overlaps I/O of a job with computing of another
 - while one job waits for I/O completion, OS runs instructions from another job
- goal: **optimize system throughput**
 - perhaps at the cost of response time...

■ Timesharing

- multiple terminals into one machine
- each user has illusion of entire machine to him/herself
- **optimize response time**, perhaps at the cost of throughput
- Timeslicing
 - divide CPU equally among the users

How OS Works?

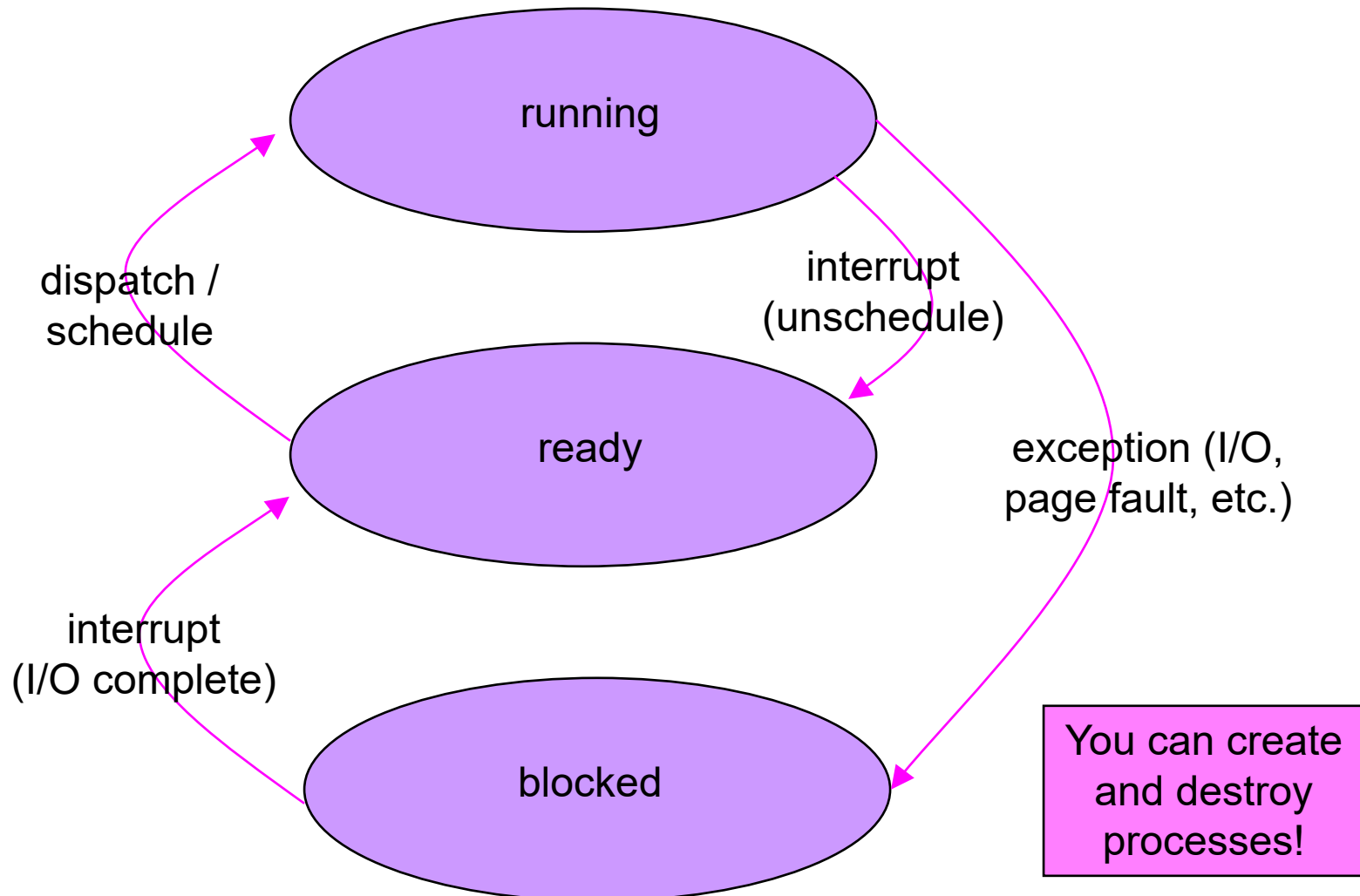
- The OS sits between application programs and the hardware
 - it mediates access and abstracts away ugliness
 - programs request services via exceptions (**traps** or **faults**)
 - devices request attention via **interrupts**



Basic Application Unit: Process

- A process is the application's
 - Unit of execution
 - Unit of scheduling
 - Unit of ownership
 - Dynamic (active) execution context
- Process Control Block (PCB)
 - Identified by an integer process ID (PID)
 - Keeps all of a process's hardware execution states
 - ready: waiting to be assigned to CPU
 - running: executing on the CPU
 - waiting: waiting for an event, e.g., I/O

A Process's Lifecycle



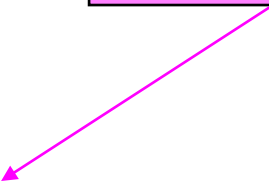
PCB Control

- Running state:
 - Hardware states are in the CPU
- Ready & blocked states:
 - PCB stores the hardware states from registers
- Context switch = PCB reloading

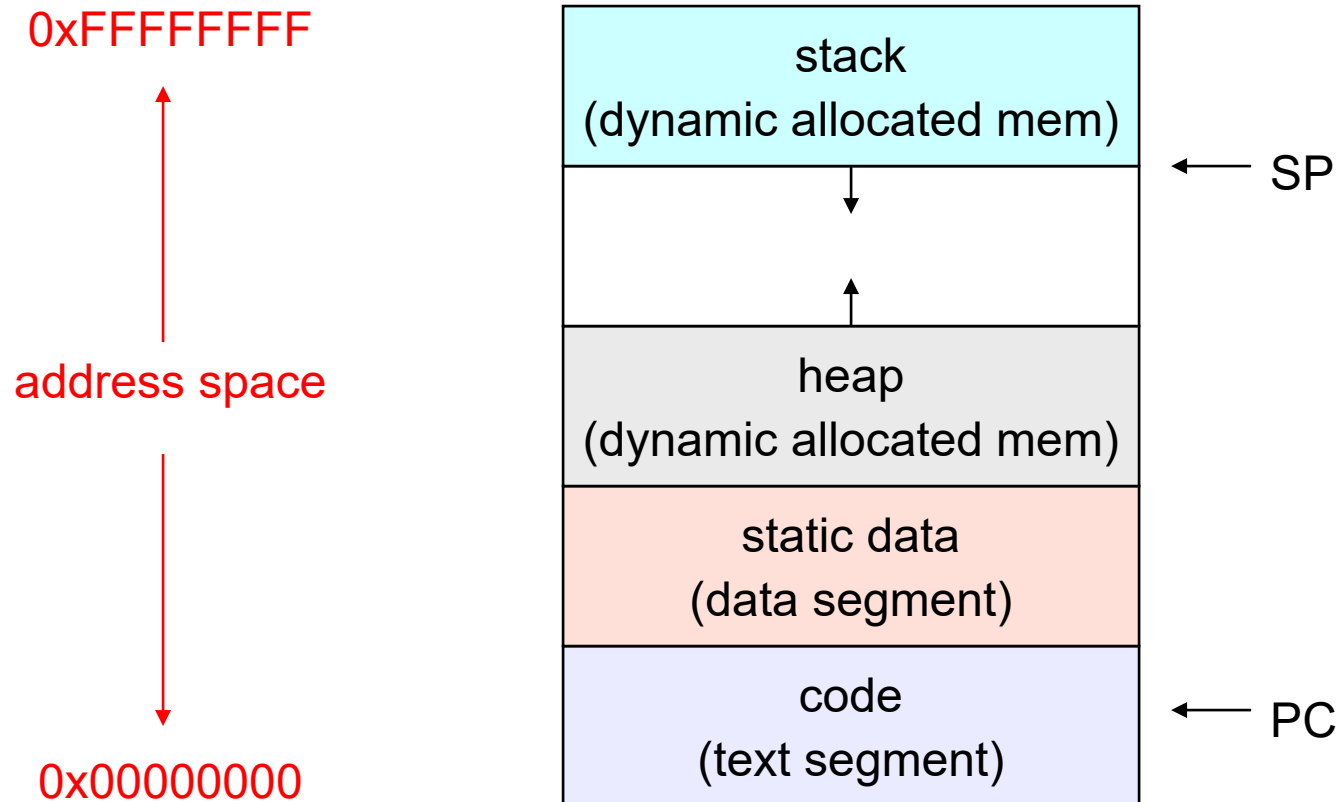
PCB Control

Process ID
Pointer to parent
List of children
Process state
Pointer to address space descriptor
Program count stack pointer (all) register values
uid (user id) gid (group id) euid (effective user id)
Open file list
Scheduling priority
Accounting info
Pointers for state queues
Exit ("return") code value

This is (a simplification of) what each of those PCBs looks like inside!



A Process's Address Space



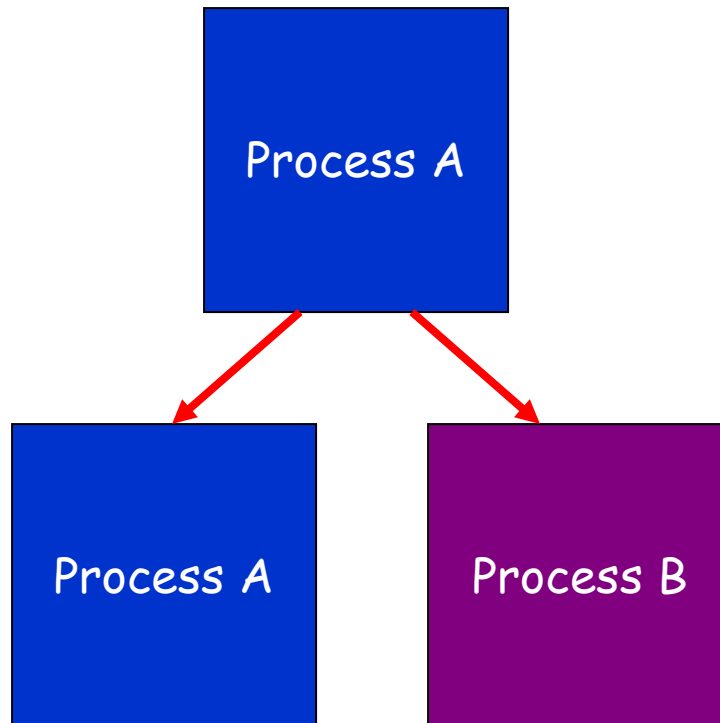
Process Creation

- New processes are created by existing processes
 - creator is called the **parent**
 - created process is called the **child**
 - what creates the first process, and when?
- In some systems, parent defines or donates resources and privileges for its children
 - UNIX: child inherits parent's uid, environment, open file list, etc.
- When child is created, parent may either wait for it to finish, or may continue in parallel, or both!

Process Creation

Create a process with fork:

- parent process keeps executing the old program;
- child process executes a new program.



Create a New Process using fork()

- A process can create a child process by making a **copy** of itself
- Parent process is returned with the child process ID
- Child process gets a return value of 0

```
child_id = fork();  
if (child_id == 0) {  
    /* child operations */  
} else {  
    /* parent operations */  
}
```

Overlay a Process using execv()

- Child process usually runs different code
- Use execv() to overlay the code of a process
- Parent uses wait() to wait for child process to finish and then release its memory

```
childid = fork();  
if (childid == 0) {  
    execv("mychild",childargs);  
    exit(0);  
} else {  
    parent_stuff();  
    wait(&csstatus);  
    exit(0);  
}
```