CS 452 Operating Systems

Processes

Dr. Denton Bobeldyk

History

- Early computers allowed only one program to execute at a time
  - Complete control of the system
  - Access to all the systems resources (memory, disk, etc)
- Modern computers allow for more than one program (process) to operate at a time

Process

- Definition: A program in execution
    - A program itself is a passive entity
    - A process is an active entity

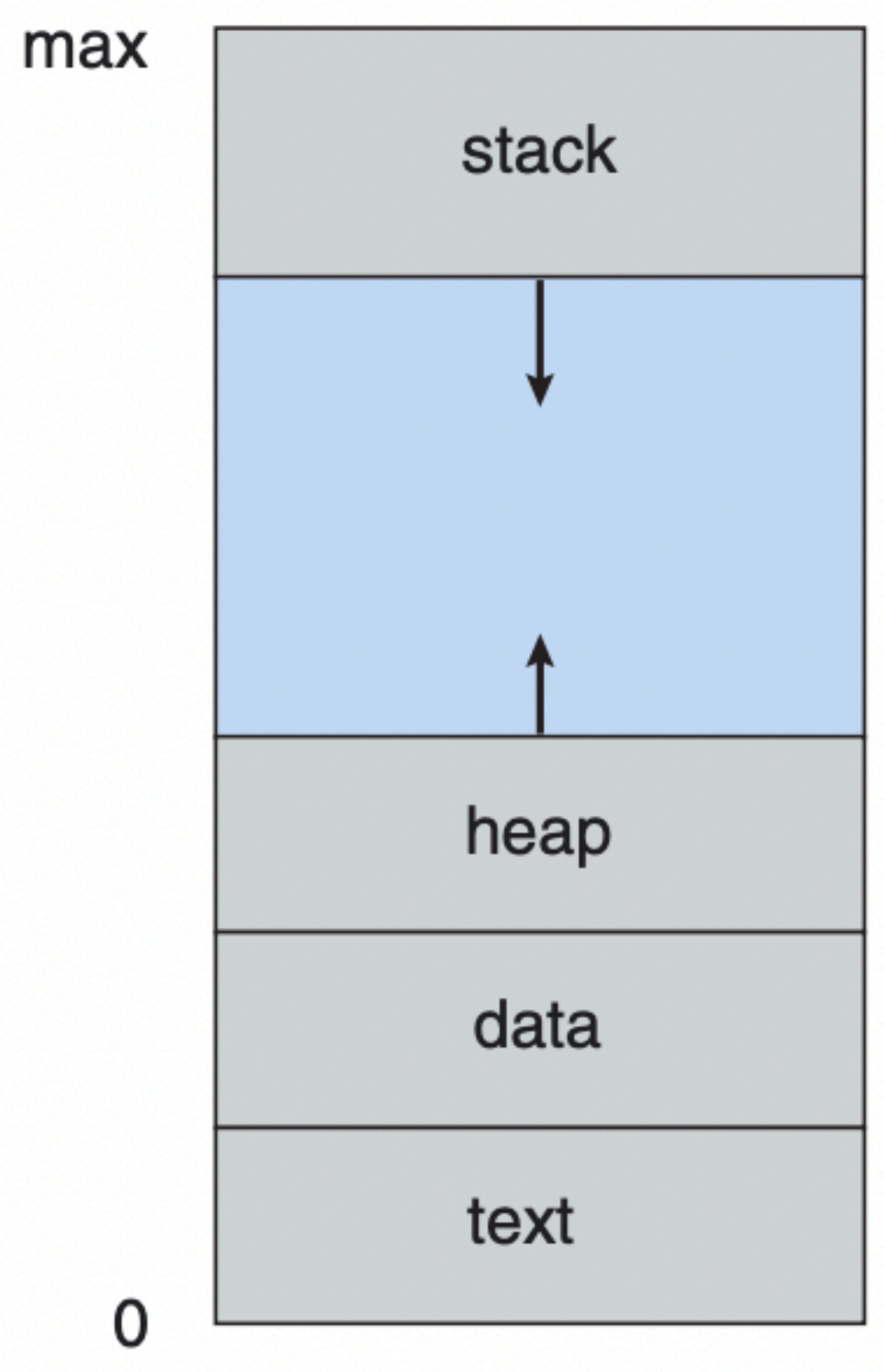

Figure 3.1 Process in memory.

Process

Stack: Contains temporary data such as function parameters, return addresses, local variables
Heap: memory dynamically allocated during process run time
Data Section: Global variables
Text section: Program code

Multiple Processes

- A single program could run as multiple processes
  - For example, running two copies of Word or a command line app
  - These processes aren't associated with each other and are separate copies

Process States

- New - Process is being created
- Running - Instructions are executed
- Waiting - The process is waiting for some event to occur
- Ready - The process is waiting to be assigned to a processor
- Terminated - The process has finished execution

Image from: https://en.wikipedia.org/wiki/Terminator_(character)

Process States

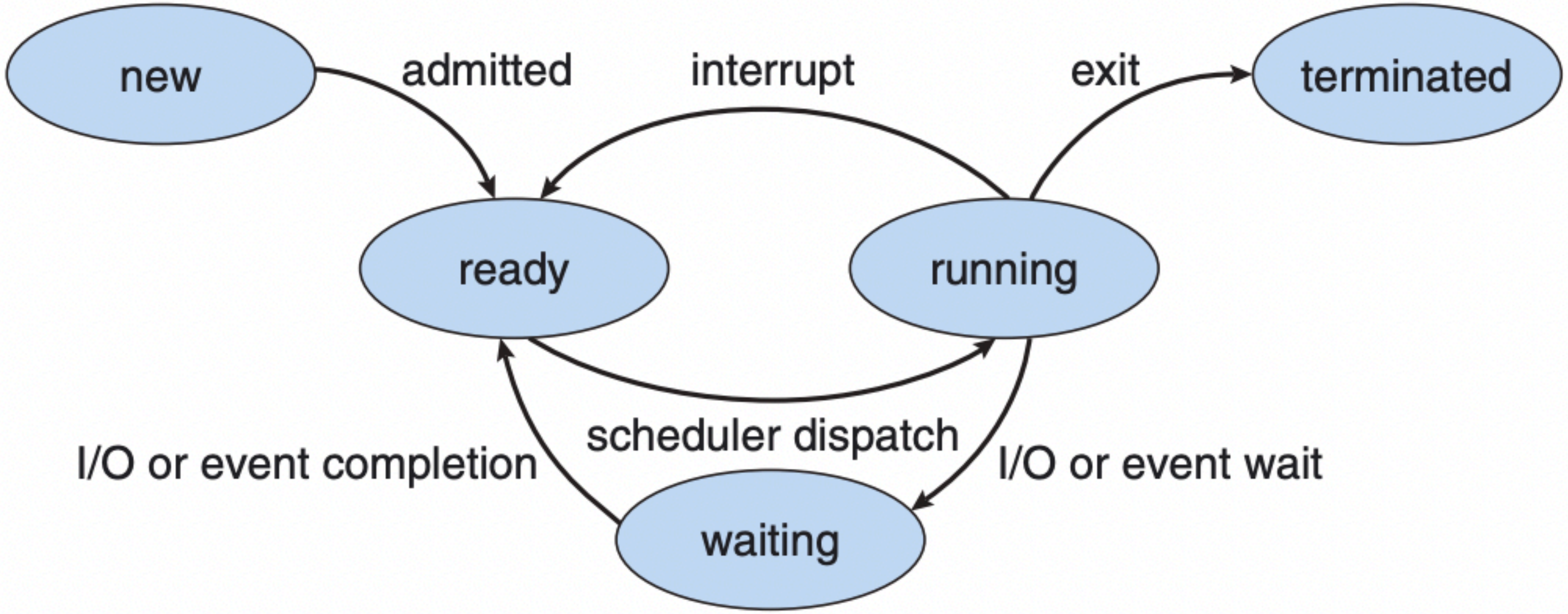

Figure 3.2 Diagram of process state.

Process Control Block

- Process Control Block - each process represented in the operating system by a process control block

Process Control Block

- Process State - {New, Ready, Running, Waiting, Terminated}
- Program Counter - indicates the address of the next instruction to be executed
- CPU registers - saves state of cpu registers for when an interrupt occurs. May include accumulators, index registers, stack pointers, …

Process Control Block

- CPU-scheduling information - process priority, pointers to scheduling queues, …
- Memory-management information - base and limit registers, page tables, segment tables (covered in chapter 7)
- Accounting Information - CPU time used, time limits, account numbers, process numbers
- I/O status information - list of I/O devices allocated, list of open files, …

Process Control Block

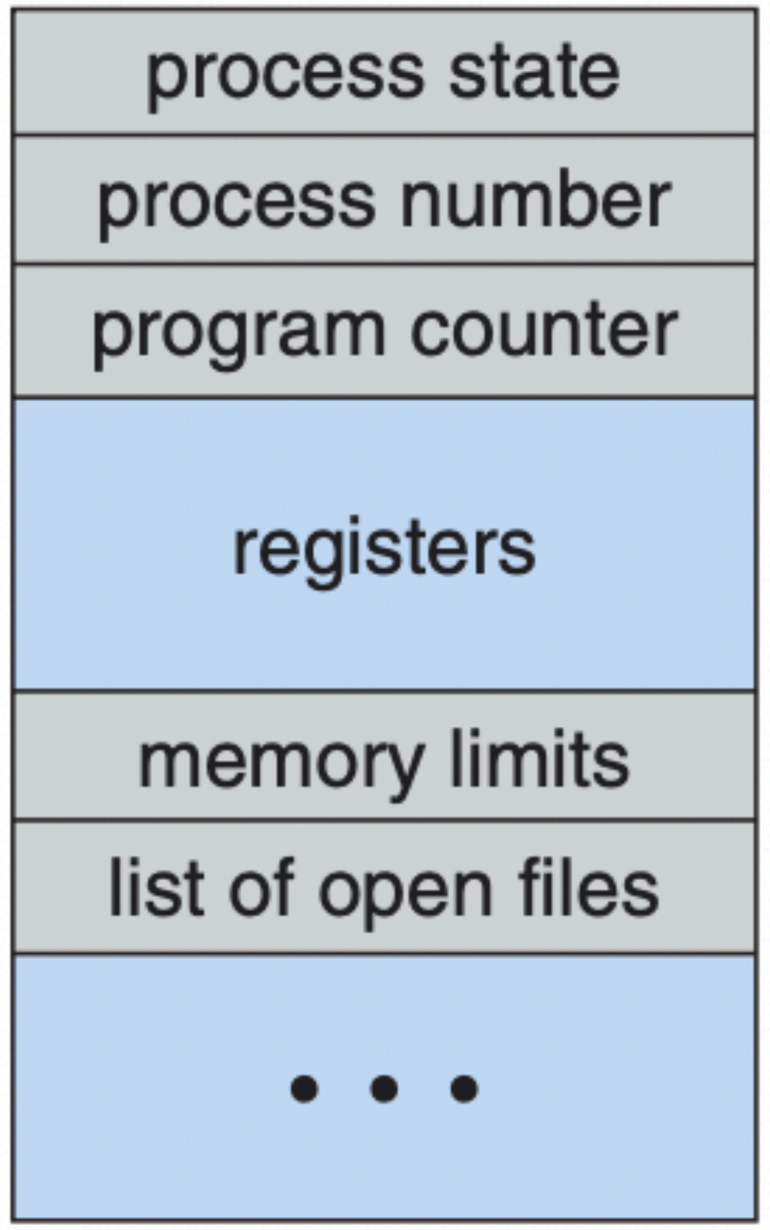

Figure 3.3 Process control block (PCB).

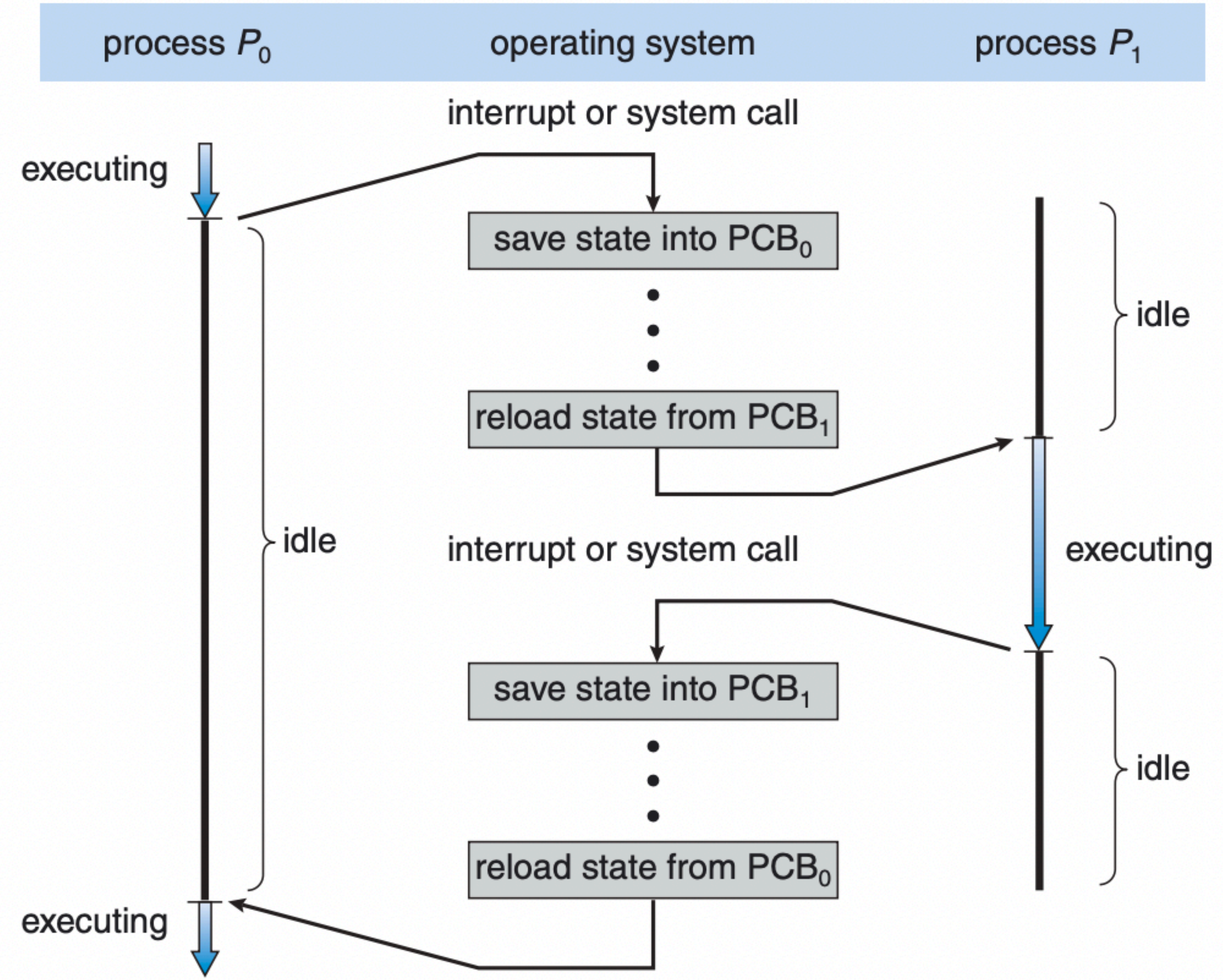

Figure 3.4 Diagram showing CPU switch from process to process.

Threads

- A single thread allows a process to perform only one task at a time
- A multithreaded process allows a GUI to be responsive and to perform a task at the same time.
- Threads will be covered in Chapter 4

Process Scheduling

- Multiprogramming - goal is to have a process running at all times
- Time Sharing - switch processes so frequently that users are able to interact with each

Process Scheduling

- Multiprogramming - goal is to have a process running at all times
- Time Sharing - switch processes so frequently that users are able to interact with each
- Process scheduler helps us accomplish this!

Scheduling Queues

- Job Queue - all processes in a system
- Ready Queue - processes residing in main memory, ready and waiting to execute
- Device Queue - processes waiting for an I/O device. Each device has it's own queue

Scheduling Queues

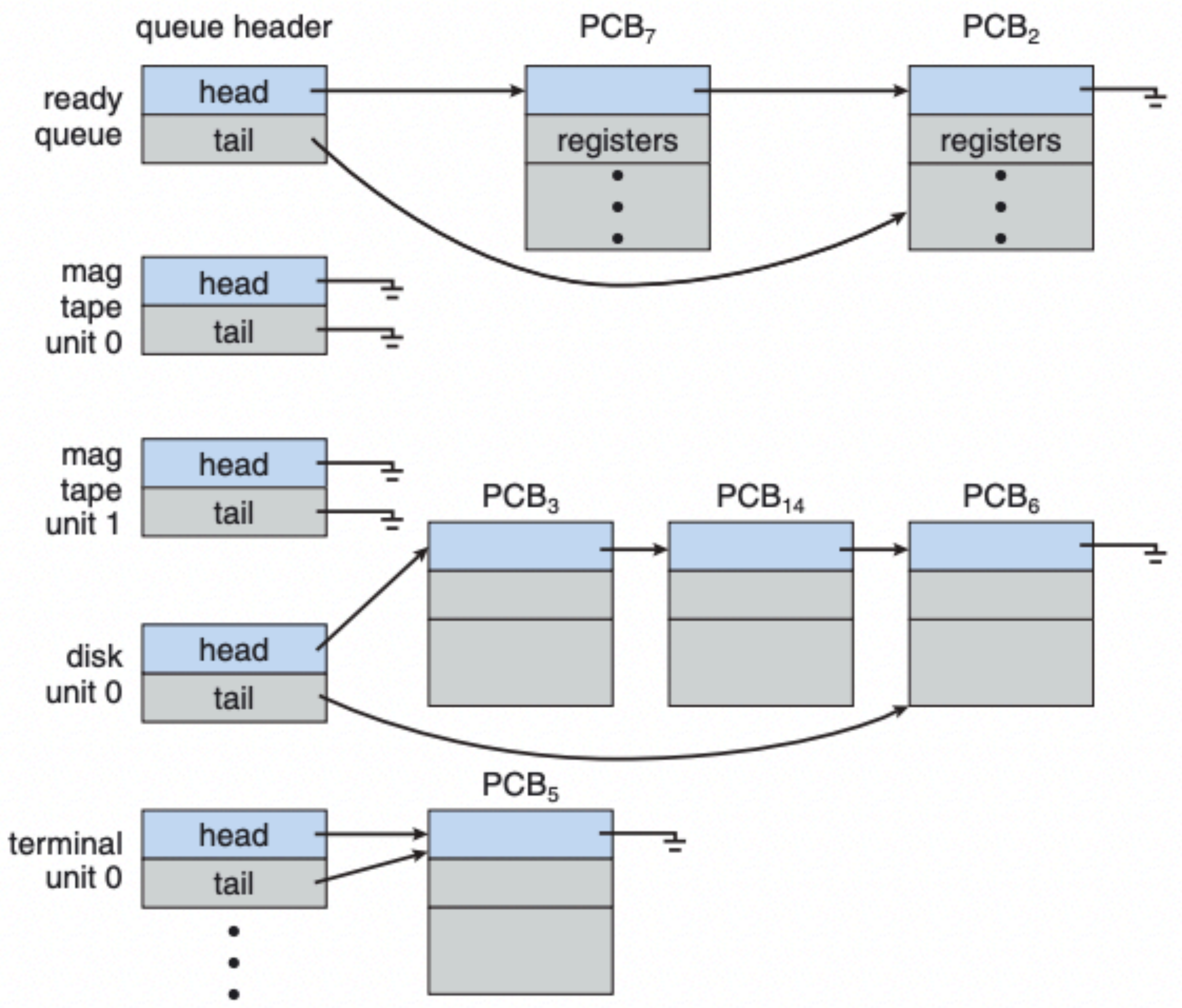

Figure 3.5 The ready queue and various I/O device queues.

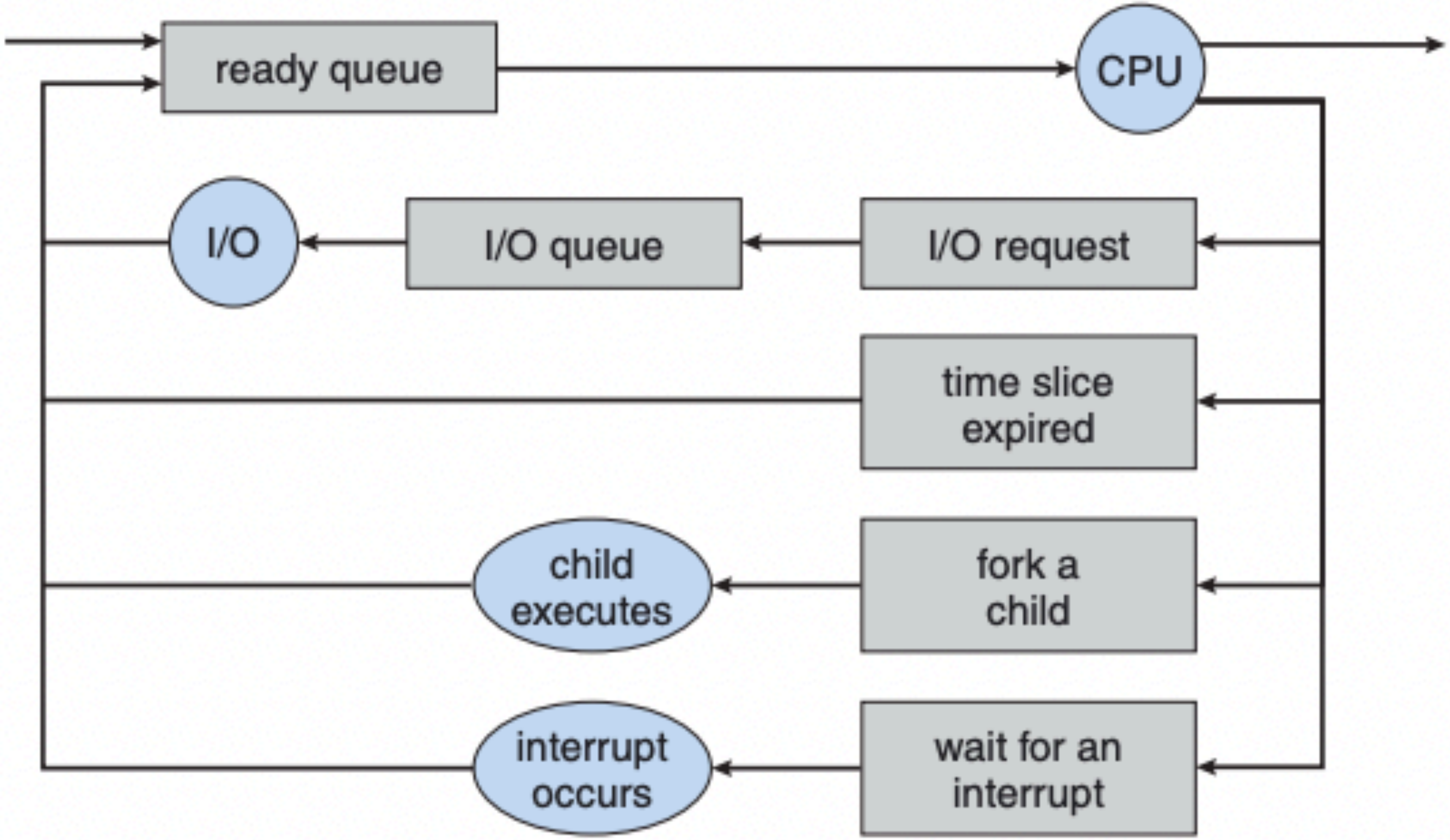

Figure 3.6 Queueing-diagram representation of process scheduling.

Schedulers

- Long-term scheduler or job scheduler - selects processes from a pool to be executed. The processes selected are loaded into memory
- Short-term scheduler - selects from processes already in memory

Schedulers

- Long-term scheduler or job scheduler - runs fairly infrequently, can take longer times to make decisions, not available on all computers
- Short-term scheduler - runs frequently, needs to make decisions faster

Processes

- I/O bound - spends more time doing I/O than doing computations
- CPU bound - spends more time doing computations than waiting for I/O

Context Switch

- Interrupt occurs, need to save context of current process.
- Context is saved as a Process Control Block (PCB)
- Switching the CPU to another process, and saving the state of the current process is known as a **context switch**.

Context Switch

- What type of hardware might help speed up a context switch?

Context Switch

- What type of hardware might help speed up a context switch?
  - Hardware support
    - Multiple sets of registers and the context switch is simply changing a pointer to the running process.

Process Creation

- A process (sometimes called a task) can create another process.
  - Parent process - the creating process
  - Child process - the process created
- Tree - parent and its children

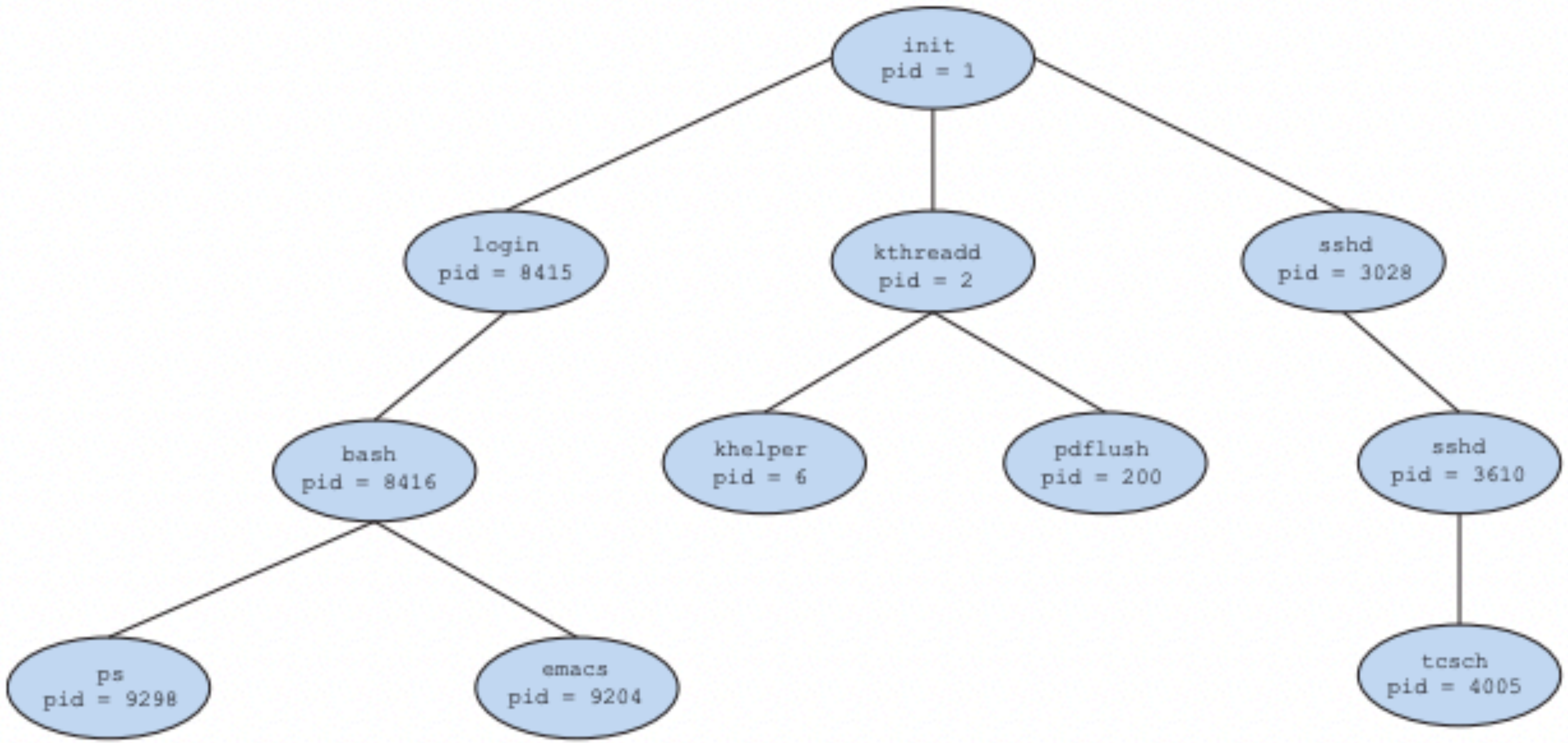

Figure 3.8 A tree of processes on a typical Linux system.

Process Creation

- Child process
  - Access resources (memory, files, I/O devices, CPU time) directly from the Operating System
  - Access resources partitioned by the parent

Process Creation

- What happens to the parent when it spawns a child?
  - Wait for child to execute and terminate?
  - Continue to execute concurrently with its children

Process Creation

- Child process can be:
  - Duplicate of the parent process (same program/data)
  - New program loaded into it (using exec)
    - Replaces process memory space with the new program

Process Creation In-Class Assignment

- Create a new process using the fork method

- To complete after the lecture (time permitting)

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h> /*textbook doesn't have this, wait doesn't work without it*/


int main(void)
{

    pid_t pid;
    pid_t childProcessID;
    pid_t parentProcessID;

    parentProcessID = getpid();

    printf("\n[%d]This gets run once\n", getpid());

    /* fork a child process */
    childProcessID = fork();


    if(childProcessID < 0){ /* error occurred */
        printf("Fork Failed\n");
        return 1;
    }
```

Process Creation

- How many children can you fork from a process?

Process Creation

- How many children can you fork from a process?
  - Limited based on the amount of memory, maximum value for pid.
  - On a 64 bit system could be ~4 million, but typically related to resources

Process Termination

- A process terminates when it finishes executing its final statement and asks the OS to delete it by using the exit() system call.
- A return value can be sent to it's parent via the wait() system call
- All system resources are deallocated.

Process Termination

- Parent typically is the only one that can terminate a child process
- Prevent users from terminating other users' processes

Process Termination

- Possible reasons for child termination:
  - Child has exceeded usage of resources allocated
  - Task assigned to the child is no longer required
  - The parent is exiting and the OS does not allow the child to continue if it's parent is terminated
    - Cascading termination

Process Termination

- Parent process may wait for a child process using the wait() system call

```
pid_t pid;

int status;

pid = wait(&status)
```

Process Termination

- A process who has terminated, but the parent has not yet called wait() is called a zombie process
- Once the parent calls wait() the process identifier and **zombie** process and it's entry in the process table is released
- If the parent did not invoke wait and instead terminates, the child then becomes an **orphan**

Interprocess Communication

- Cooperating - any process that can be affected by another process. Any process that shares data with another process is a cooperating process
- Independent - any process that cannot affect or be affected by another process

Interprocess Communication

- Why cooperate?
  - Information Sharing (e.g., shared file)
  - Computation Speedup (e.g., run in parallel)
  - Modularity (e.g., system functions in separate processes)
  - Convenience (e.g., work on multiple tasks at the same time)

Interprocess Communication

- Two fundamental models for IPC
  - Shared Memory
    - Reading/Writing data to a shared region of memory
  - Message Passing
    - Useful for exchanging smaller amounts of data

Interprocess Communication

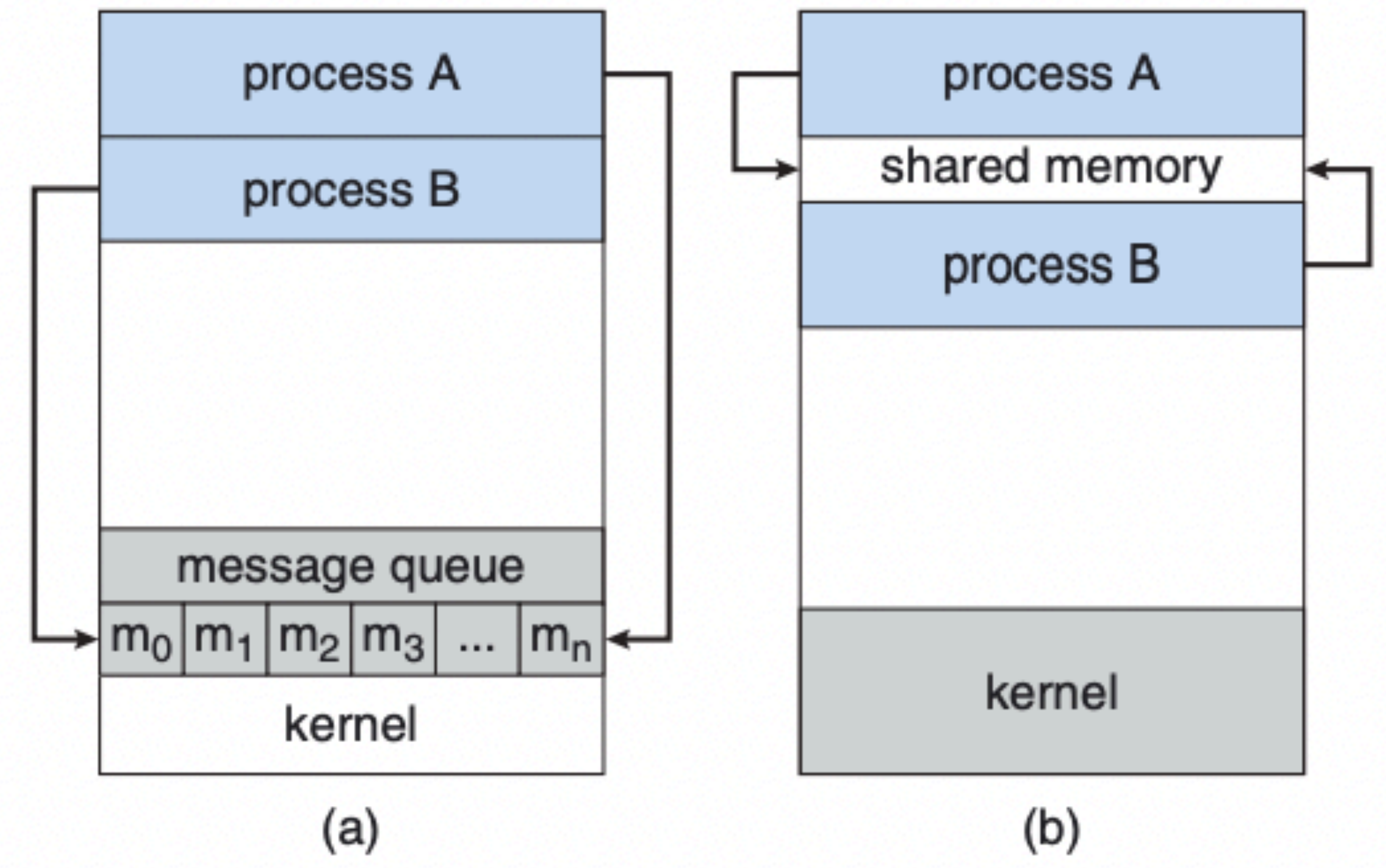

Figure 3.12 Communications models. (a) Message passing. (b) Shared memory.

Shared-Memory Systems

- Create shared memory region
- Processes attach to the shared address space

Shared-Memory Systems

- Producer/Consumer : One process produces, the other process consumes
  - What are some examples of a Producer/Consumer scenario

Shared-Memory Systems

- Unbounded buffer - no practical limit on the size of the buffer
    - Producer can always produce new items
- Bounded buffer - fixed buffer size
    - Producer waits if buffer is full
    - Consumer waits if buffer is empty

Message-Passing Systems

- Two operations:
  - send(message)
  - receive(message)

Message-Passing Systems

- A communication link is established between every pair of processes that want to communicate.
- A link is associated with exactly two processes

Message-Passing Systems

- Symmetry - sender and receiver name each other to communicate
- Asymmetry - sender names the recipient but not vice versa
  - send(P, message) - send a message to P
  - receive(id, message) - receive a message from any process, the variable id is set to the name of the process communicated with

Pipe Creation

- Create an array with 2 elements:
  - int fd[2]
- Create a pipe and return the result of the creation to a variable:
  - pipeCreationResult = pipe(fd)
- Check to see if the pipe was created successfully, if not, return an error:
  - if(pipeCreationResult < 0){
    - perror("Failed pipe creation\n");
    - exit(1);
  - }

Pipe Creation

Create a new process using fork, then:

```
if(pid == 0){
  write(fd[1], &output, sizeof(int));
  printf("Child wrote [%d]\n", output);
}
else{
  read(fd[0], &input, sizeof(int));
  printf("Parent received [%d] from child process\n", input);
}
```

In-Class Assignment

Create a pipe as demonstrated in today's lecture.

The program should:

1. Fork off a child
2. Request an input string up to 500 characters
3. Output that the child sent a message (and what it was)
4. Output that the parent received a message (and what it was)