Today's Plan

Upcoming:

- Assignment 1

Last time:

- (Almost) finished introduction

Today's topics:

- From last time:
  - Virtual Machines
- The Process Concept
- Process States
- Process Control Blocks
- Precedence & Concurrency
- Race Conditions

Processes

- Process Concept
- Concurrency
- Race Conditions
- Process Creation
- Interprocess Communication
- Examples of IPC Systems

What is a Process?

- Fundamental building block of modern operating systems is the notion of a *process*
- A process is a running program (a program in execution). This includes:
  - All programs running on behalf of users (application programs)
  - Some operating system functions are also implemented using processes
- A process is a single thread of execution under control of the OS

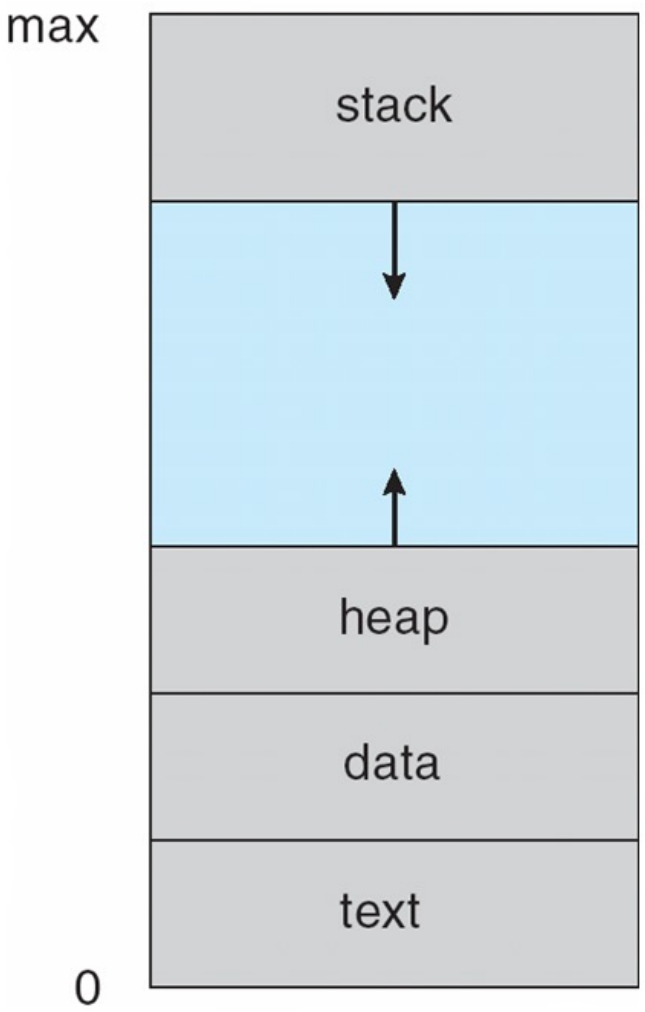
Process Details

- Much of the functionality of a modern OS is the work required to manage processes
- OS may have hundreds of processes active at the same time
  - Although only a small number of them executing at a given time on a multi-core CPU system
- Processes are not found in the operating system *kernel*

What is not a Process?

- A program by itself is not a process
- There is no one-to-one correspondence between programs and processes
  - E.g. there may be 10 people using emacs at the same time, i.e. 10 processes running emacs, but only one copy of the emacs program on disk
  - E.g. there may be many programs on disk that are not executing at the present time → these are not processes!
  - Programs are passive entities, while processes are active

A Process in Memory



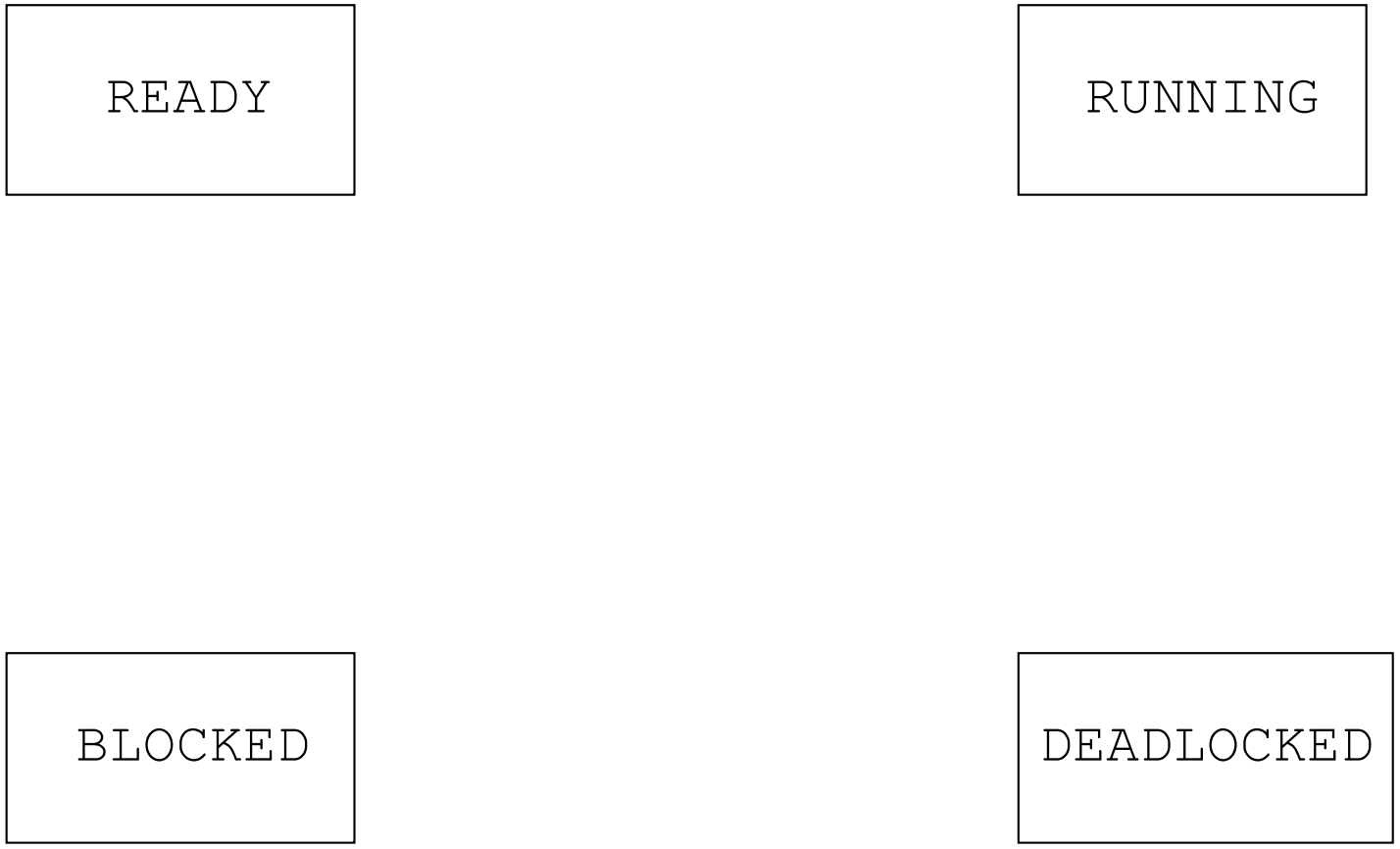
- **Text**: the instructions that make up the program
- **Data**: the data the program uses
- **Heap**: used for dynamic memory
- **Stack**: used for function calls

Process States

Modern OSes allow for more than one process to exist at the same time, and since there is usually only one processor, processes must assume different states during their lifetime:

- **Running**: currently being executed by the processor
- **Blocked**: waiting for some external event, e.g. I/O operation
- **Ready**: waiting for a turn at the processor
- **Deadlocked**: waiting for an event that will never happen
  - OS must recognize this and deal with it

Process State Diagram

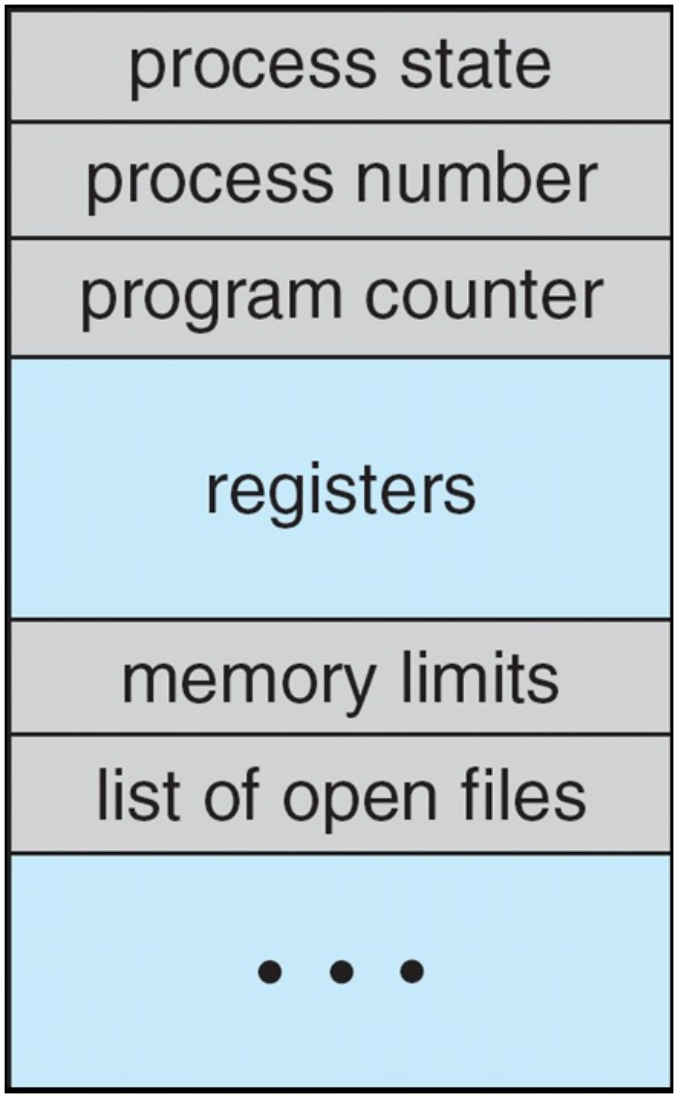


Process Control Block (PCB)

Information associated with each process

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

Process Control Block (PCB)



Process Model of an OS

- Modern OSes are a collection of cooperating processes that run on top of (and are supported by) an OS kernel
- The kernel is responsible for the following services:
  - Creation and destruction of processes
  - CPU scheduling, memory management, device management
  - Process synchronization tools
  - Process communication tools
- OS services provided by the kernel are invoked using *system calls*

Precedence & Concurrency

- Logical concurrency is achieved on a uni-processor system by quickly switching the CPU from one process to the next
- Consider the following two processes which share data:

```
P1: A = B + C
P2: D = A * 2
```

  - P1 must precede P2!
- In general, the issues of precedence and concurrency are the same for logical or physical concurrency
- When is it okay for two ore more processes to execute concurrently so that we always get consistent results?
  - Depends on what data is shared between the processes

Race Conditions

Consider the following example, where P1 and P2 share all data, and we alternate executing one line of code between P1 and P2

```
P1: z = 0;
    B = 1;
    C = 2 + B;
```

```
P2: z = 1;
    B = 2;
    D = z + B;
```

- If P1 goes first:
- If P2 goes first:

Race Conditions

- When there is a dependency on the exact execution order of statements between two or more processes, it is called a *race condition*
- Race conditions occur because of the relationships between the read and write sets of the processes

Read & Write Sets

- A process' *read set* is the set of all data in RAM, secondary storage, or other existent data that a process reads (uses during execution)
  - Read set for process P denoted `R(P)`
- A process' *write set* is the set of all data that a process writes (changes during execution)
  - Write set for process P denoted `W(P)`
- E.g.:

```
P1: z = x + y          P2: a = a + 3
```

Bernstein's Concurrency Conditions

- Used to dictate when two processes are able to execute concurrently, and always produce consistent results
- Bernstein's Concurrency Conditions are as follows:
  In order for two processes P1 and P2 to run concurrently, the following 3 conditions must hold:
  1. `R(P1)` cannot intersect `W(P2)`
  2. `W(P1)` cannot intersect `R(P2)`
  3. `W(P1)` cannot intersect `W(P2)`
- If two processes do not satisfy BCC, then they are said to have a *critical section* problem
  - The critical sections are the sections of code that cause violation of BCC