Processes

CS 241

Announcements

- About 1/4th Done!
  - 2/8 MPs are complete
  - 2/8 weeks of lecture are complete
- Wade's Office Hours Moved
  - **Mondays**, 1pm-2pm (after class)
- MP3

fork()

- You already know about fork()…
  - fork(): Create a new process. The child process is nearly an exact copy of the parent process.

  - **Parent**: Returns PID of the child (value >0)
  - **Child**: Returns 0.
    - Can get PPID by calling getppid().

fork() Example

```
void main()
{
  int k = 3;
  pid_t pid = fork();
  if (pid == 0) { k += 1; }
  else { k += 2; }
  printf("%d\n", k);
}
```

fork() Example #2

```
void main()
{
  int k = 3;
  pid_t pid = fork();
  if (pid == 0) { k += 5; }
  k += 10;
  printf("%d: %d\n", getpid(), k);
}
```

Parent ID: 100
Child ID: 200

wait()

- You already know about wait()…
  - wait(): Wait for a child process to terminate.

- Another variant, waitpid()…
  - waitpid(): Waits for a specific child process to terminate.

fork()+wait() Example

```
void main()
{
  int k = 3;
  pid_t pid = fork();
  if (pid == 0) { k += 1; }
  else { k += 2; wait(); }
  printf("%d\n", k);
}
```

fork()+wait() Example #2

```
void main()
{
  int k = 3;
  pid_t pid = fork();
  if (pid > 0) {
     pid = fork();
     k += 10;
     if (pid > 0) { k += 20; wait(); }
  }
  printf("%d\n", k);
}
```

fork()+wait() Example #3

```
void main()
{
  int k = 3;
  pid_t pid = fork();
  if (pid == 0) {
     pid = fork();
     wait();
     k += 10;
  }
  printf("%d\n", k);
}
```

exec()

- exec(): Execute a file
  - The exec() family of functions replaces the current process image with a new process image.
  - The exec() function call never returns if successful.
  - exec() broadly refers to a set of six functions that do the same with different parameters.

fork()+exec()+wait()

```
void run(char *command_line)
{
  pid_t pid = fork();
  if (pid == 0) {       /* Child */
     exec(command_line);
     perror("Failed to exec()");
     exit(1);
  } else if (pid > 0) { /* Parent */
     waitpid(pid);
  } else {              /* fork() Error */
     perror("Failed to fork()");
  }
}
```

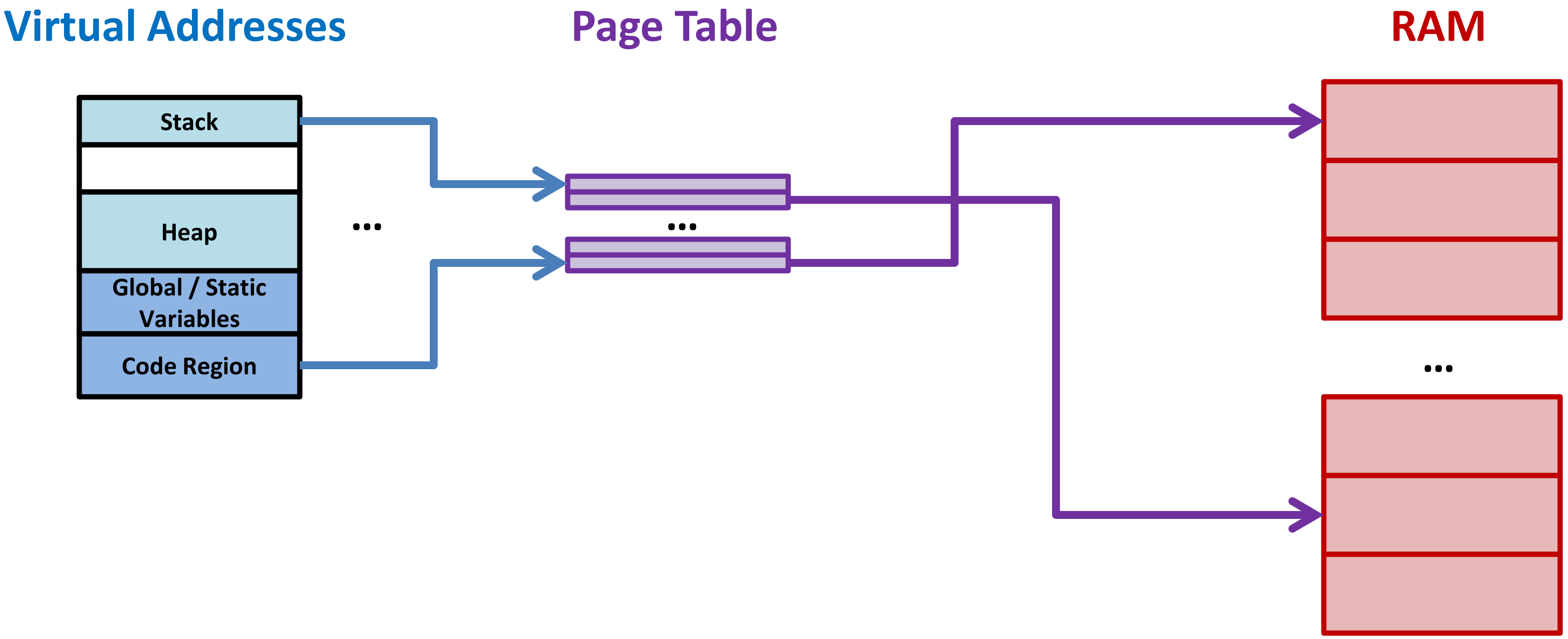
Launch a new process…

- The fork()+exec()+wait() sequence is so common there is a library call to do it for you!
  - **system()**: executes a shell command
  - Use this on MP3!

How does this work in memory?

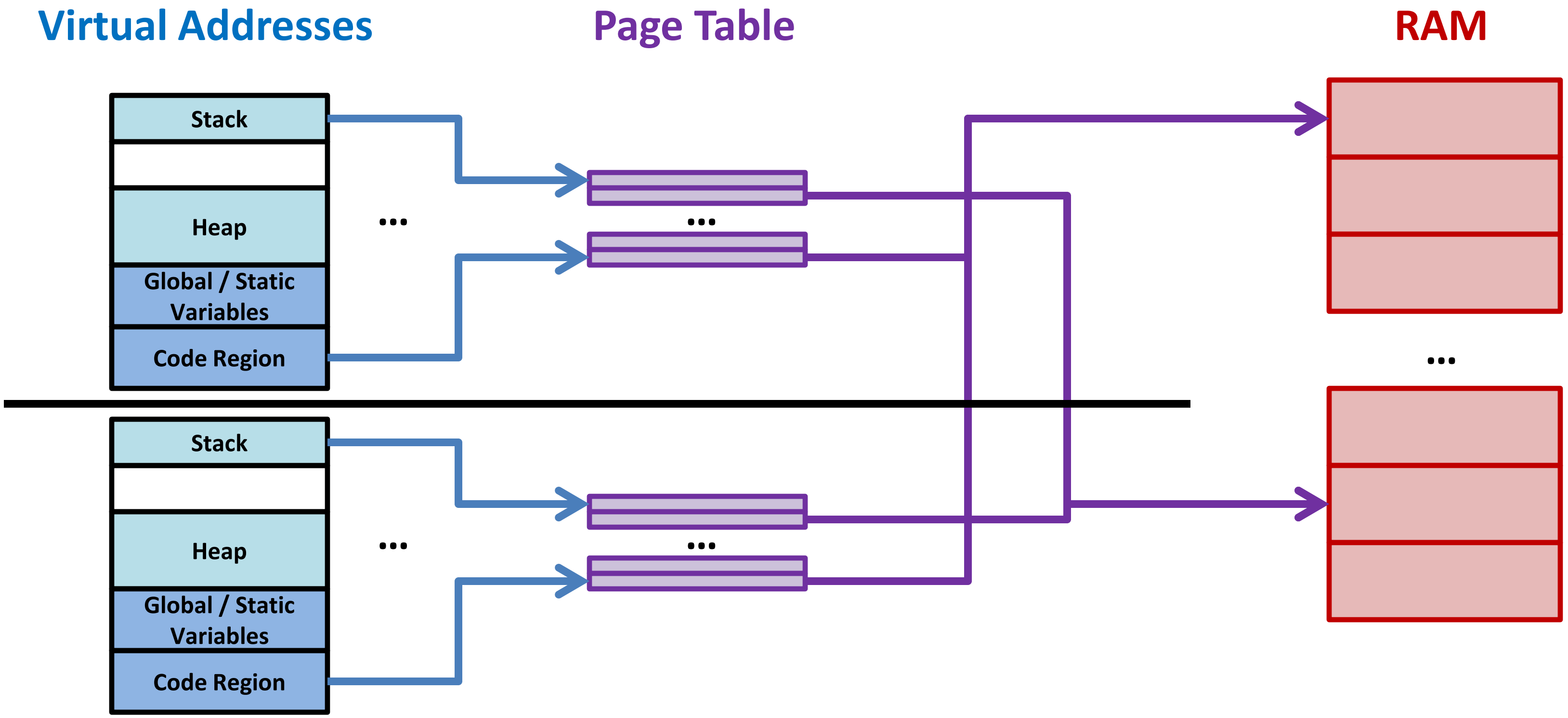
- A process footprint is quite large…
  - A small process is easily several MB in RAM.

Step 1: Before Fork



- Standard mapping of process memory to RAM via a page table.

Step 2: After Fork



- The page table is copied, but the entries **initially** point to the same pages in RAM.

Copy on Write

- Copy on Write (CoW) prevents the unnecessary coping of RAM pages until **either** process writes to a RAM page.

- Particularly efficient in the case when **exec()** is immediately use in the child process.
  - Remember: **exec()** replaces the entire contents of the process memory with a new program

Processes: System View

CS 241

Managing Processes

- An operating system typically has tens or hundreds of processes running.

- Each process is managed by information contained in a **Process Control Block (PCB)**.
  - Information available only for the OS, not used by the process itself.

Process Control Block

- The PCB Contains:
  - **Identifiers**
    - Process ID (PID), Parent Process ID (PPID)
  - **State Information**
    - Registers (program counter, stack pointer)
    - Pointer to the page table, handles to open files
    - Lots of other stuff (signals, privileges, resources)
  - **Scheduling Information**
    - Priority
    - Accounting Information (when was it last ran?, how long?)
    - Current State (waiting for I/O?)

Who runs?

- Each CPU may only run one process at a time.

- How do we decide when someone else gets to run?
  - Modern systems use a hybrid of many strategies!

CPU Scheduling Strategy #1

- **Time Slicing**
  - Give each process an equal-sized slice the CPU. Kick the process off when its quantum expires.
  - **Advantages?**
  - **Disadvantages?**

CPU Scheduling Strategy #2

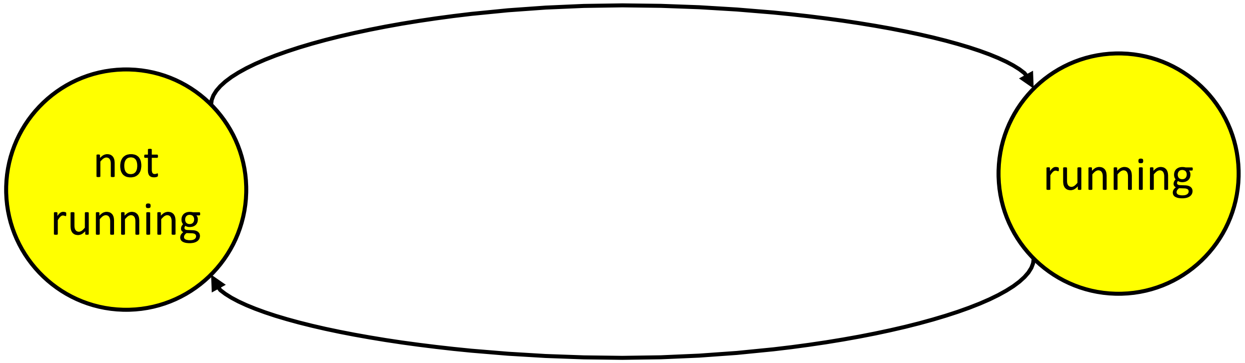
- **Cooperative Multi-tasking**
  - Each process will cooperate, **yield()**ing every so often to allow other processes to run.
  - **Advantages?**
  - **Disadvantages?**

CPU Scheduling Strategy #3

- **Multi-programming**
  - During every system call, determine if the process should be swapped out.
  - **Advantages?**
  - **Disadvantages?**

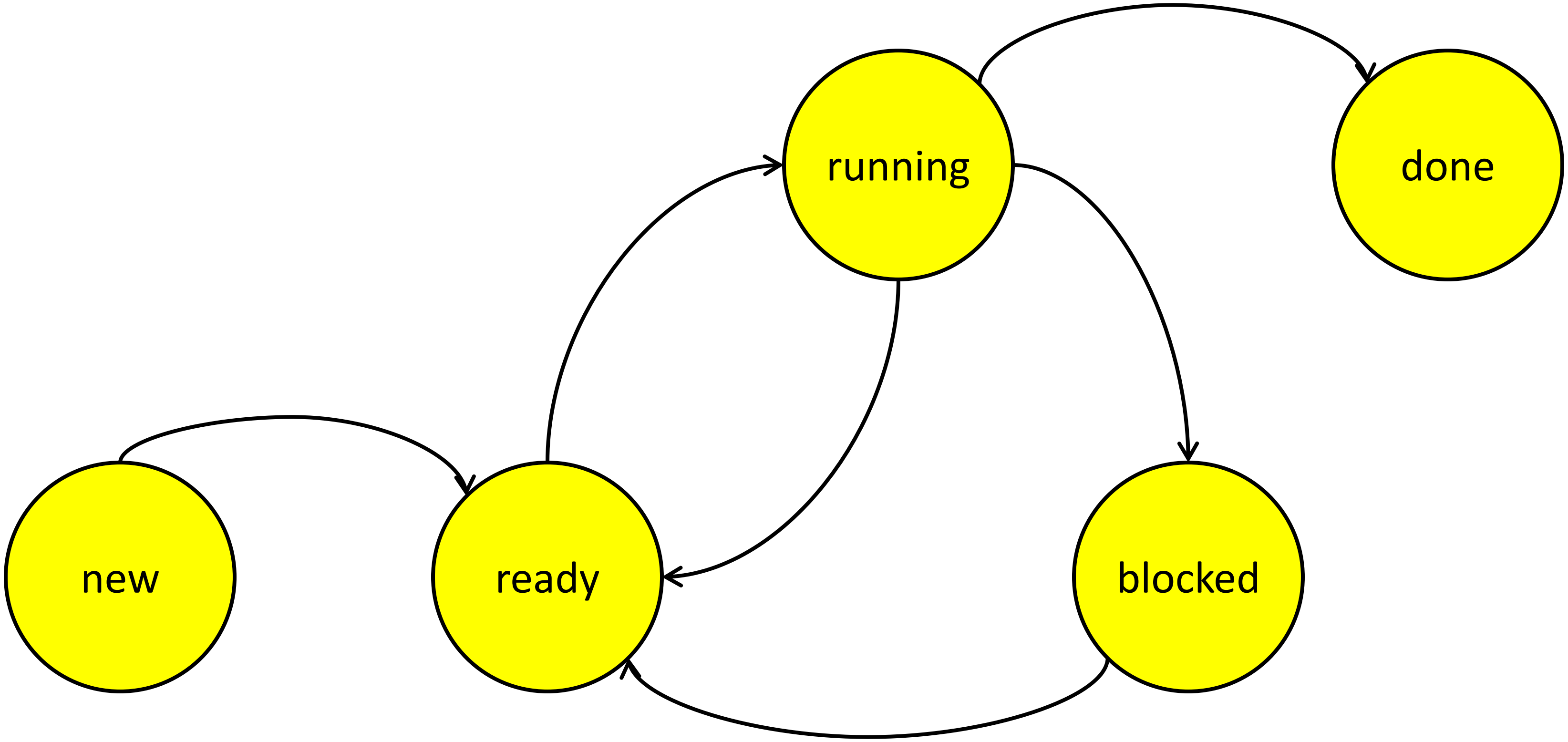
Modeling Processes

- Two-state process diagram:



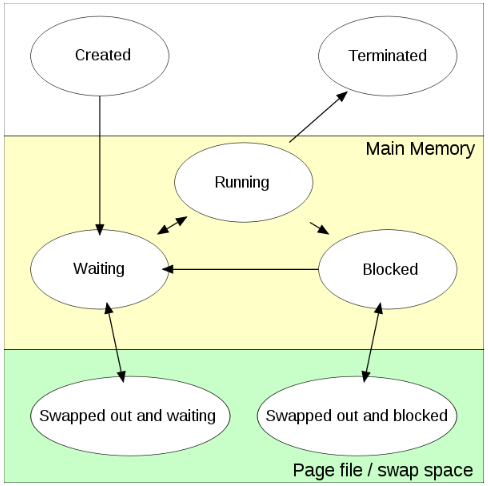
- Is it important to know why something is **not** running?

Five State Process Diagram



Other Process Models Exist

- Seven State:

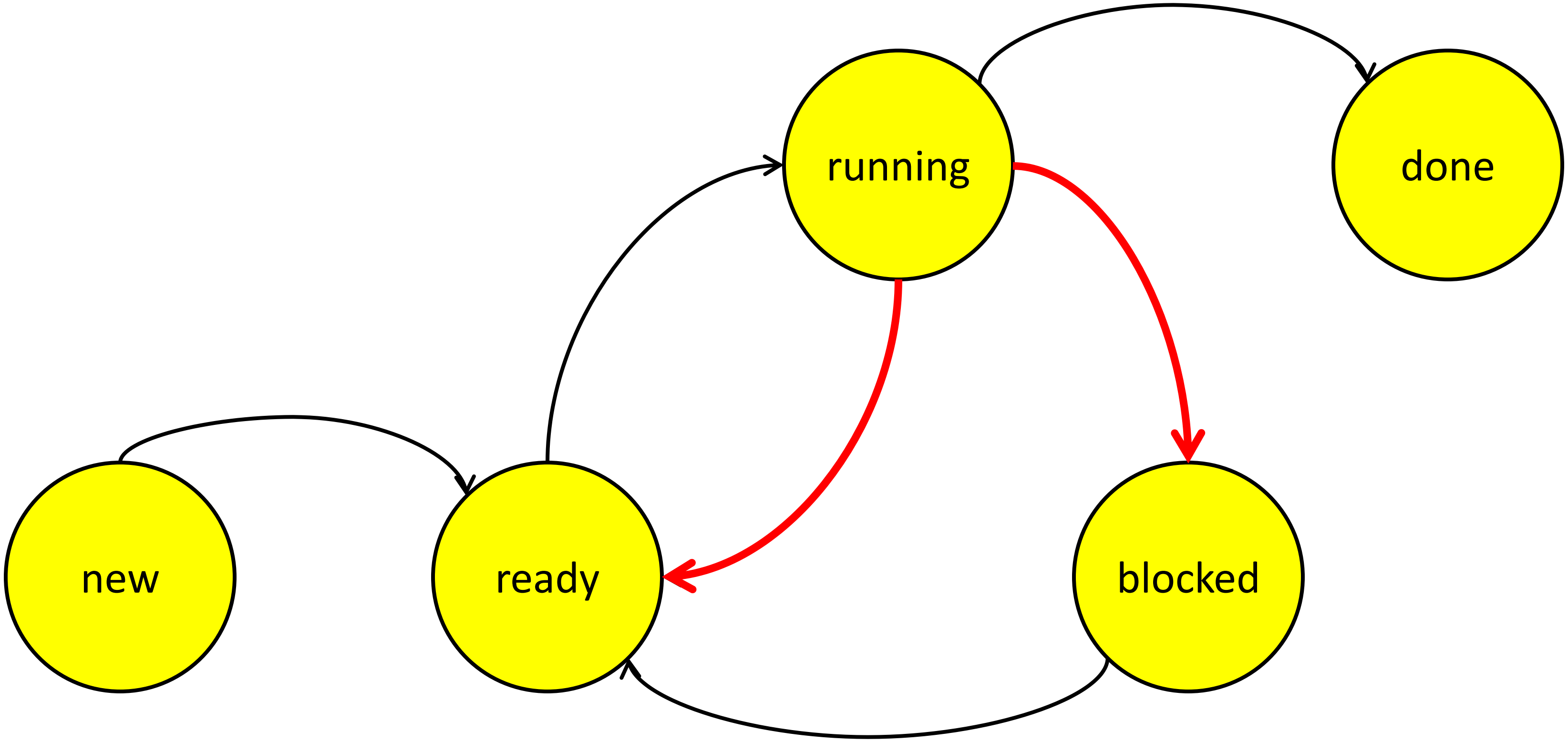


…also 9 and 11 state diagrams.

- The more states will more completely describe each process. In CS 241, we will only worry about five.

Image Source: http://en.wikipedia.org/wiki/File:Process_states.svg

Context Switch



Content Switch

- A **content switch** is the system event when a CPU switches from one process to another.

- Significant overhead:
  - – Save CPU state (registers) and PCB
    - Page table (4 KB), etc
  - – Scheduling Overhead
    - Save accounting information
    - Decide the next process to run, queue the old process
  - – Load the new process state and PCB

Tomorrow

- Threads!