CISC 3320 MW3

Process Management

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Acknowledgement

- These slides are a revision of the slides by the authors of the textbook

Outline

- Process Concept
- Process Scheduling
- Operations on Processes

CPU Activities

- Batch systems execute jobs
- Time-shared system has multiple user programs and tasks
  - Example: a user can run multiple programs or multiple instances of a program on Windows or UNIX systems
- Process management: to support these, we need to manage and share memory and CPU
- Use job and process interchangeably

Two Concepts

- Program
- Process

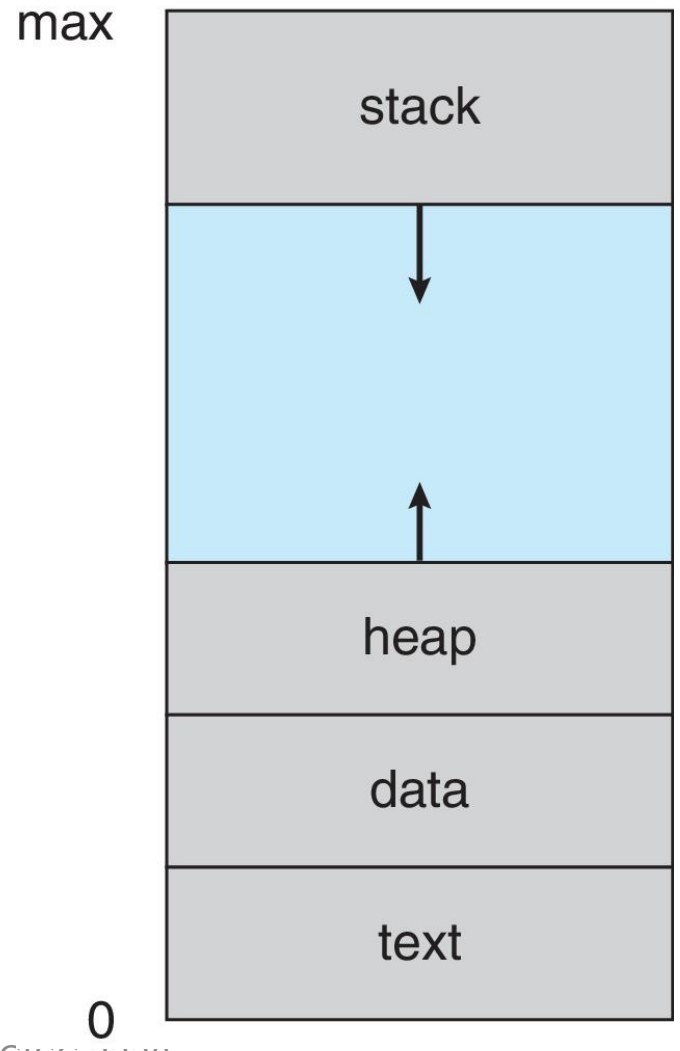
Program and Executing a Program

- A program is *passive* entity stored on disk in the form of *executable file*
- An operating system provides means to execute a program
  - e.g., execution of program started via GUI mouse clicks, command line entry of its name

Process

- A process is a program in execution, as such, a program is a passive entity while a process is an active one
- A program becomes a process when executable file loaded into memory
- One program can be several processes
  - Consider multiple users executing the same program

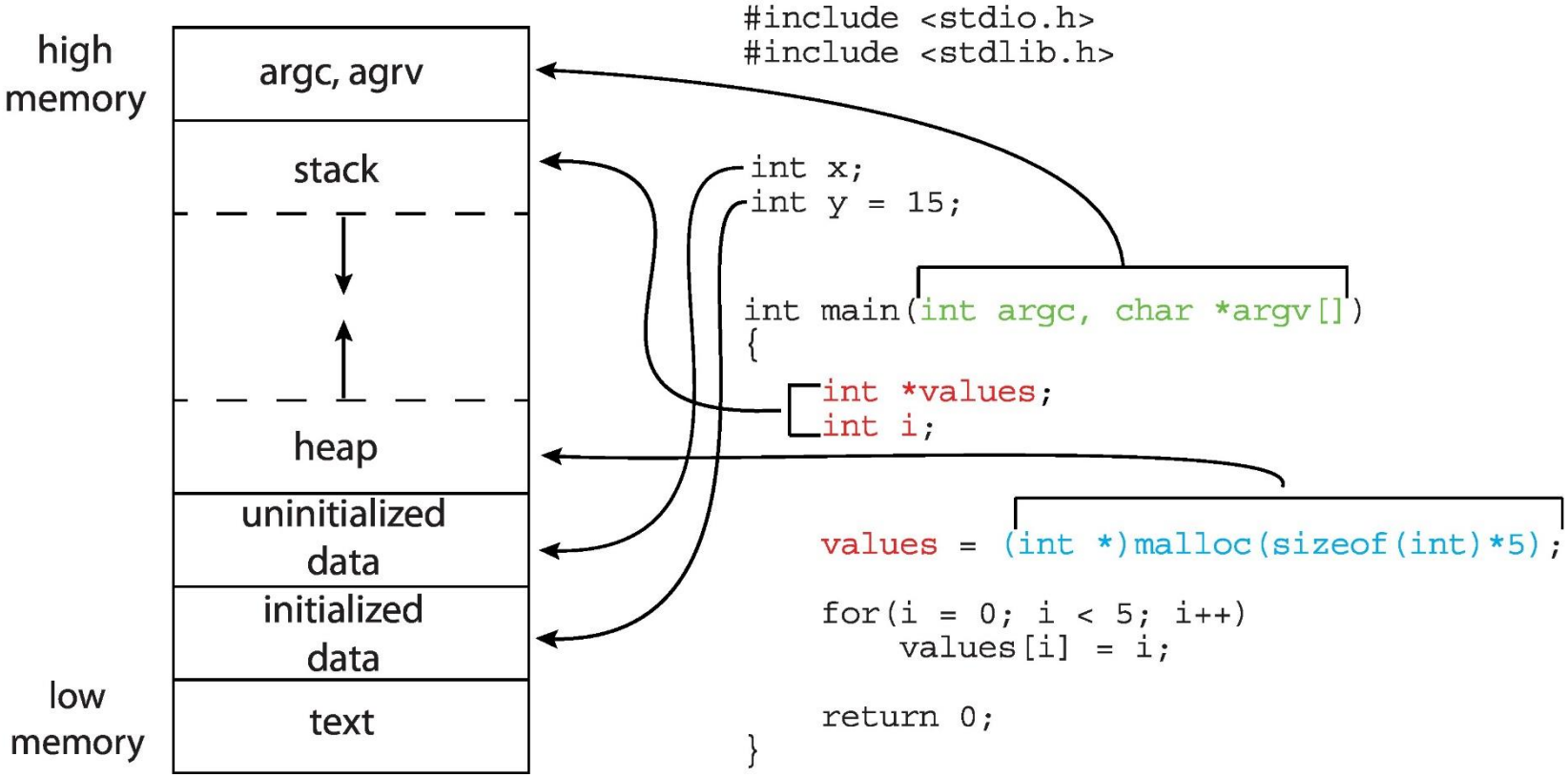
Process in Memory



Process: Memory Layout

- A process consists of multiple parts, generally,
  - The program code, also called ***text section***
  - Current activity including ***program counter***, processor registers
  - ***Stack*** containing temporary data
    - Function parameters, return addresses, local variables
  - ***Data*** section containing global variables
  - ***Heap*** containing memory dynamically allocated during run time

Memory Layout of a C Program



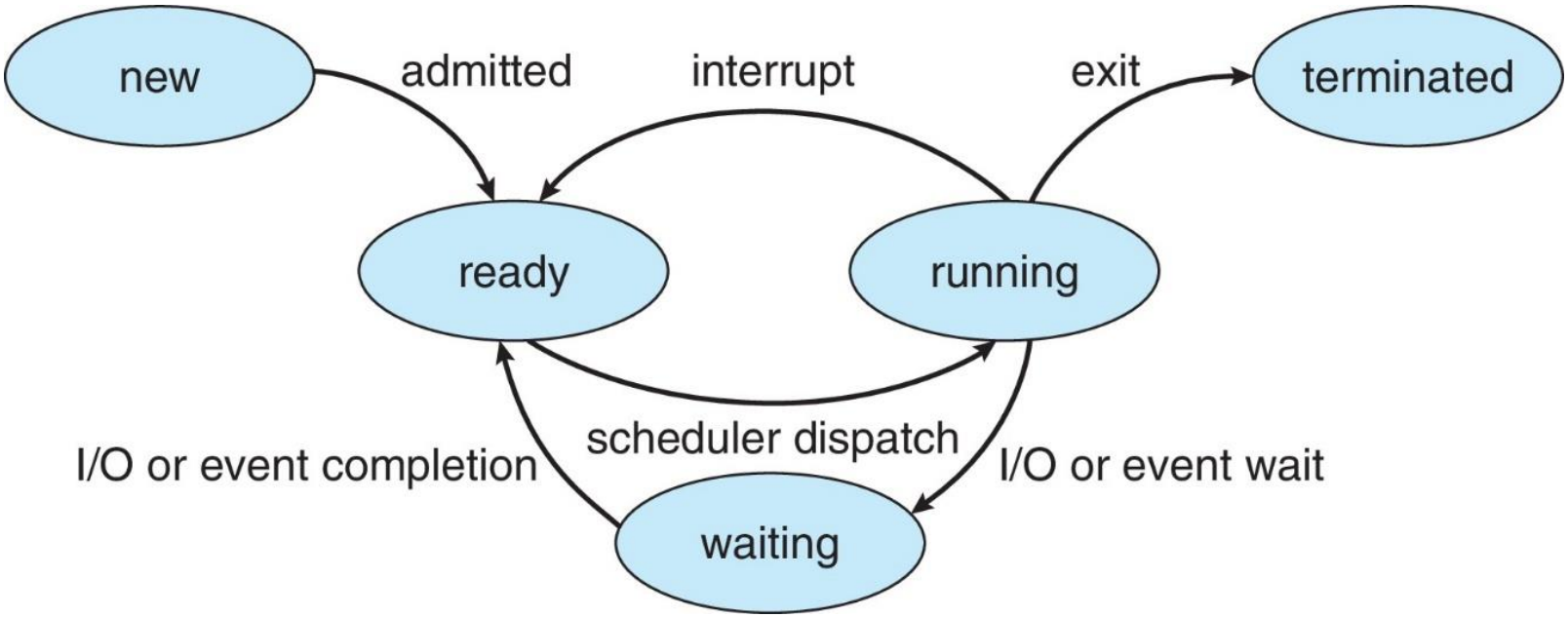
Questions?

- Concept of process
- Parts of a process
- Memory layout of a process
- Demo program

Process State

- As a process executes, it changes ***state***, *generally,*
  - **New**: The process is being created
  - **Running**: Instructions are being 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

Transition of Process States



Process State: Example in Linux (ps)

```
cisc3320@debian:~$ ps aux | head
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  2.6  0.5   9524  6132 ?        Ss   17:29   0:00 /sbin/init
root         2  0.0  0.0      0     0 ?        S    17:29   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    17:29   0:00 [ksoftirqd/0]
root         4  0.0  0.0      0     0 ?        S    17:29   0:00 [kworker/0:0]
root         5  0.0  0.0      0     0 ?        S<   17:29   0:00 [kworker/0:0H]
root         6  0.0  0.0      0     0 ?        S    17:29   0:00 [kworker/u2:0]
root         7  0.2  0.0      0     0 ?        S    17:29   0:00 [rcu_sched]
root         8  0.0  0.0      0     0 ?        S    17:29   0:00 [rcu_bh]
root         9  0.0  0.0      0     0 ?        S    17:29   0:00 [migration/0]
cisc3320@debian:~$
```

Process State: Example in Linux (man ps)

PROCESS STATE CODES

Here are the different values that the s, stat and state output specifiers (header "STAT" or "S") will display to describe the state of a process:

D uninterruptible sleep (usually IO)

R running or runnable (on run queue)

S interruptible sleep (waiting for an event to complete)

T stopped by job control signal

t stopped by debugger during the tracing

W paging (not valid since the 2.6.xx kernel)

X dead (should never be seen)

Z defunct ("zombie") process, terminated but not reaped by its parent

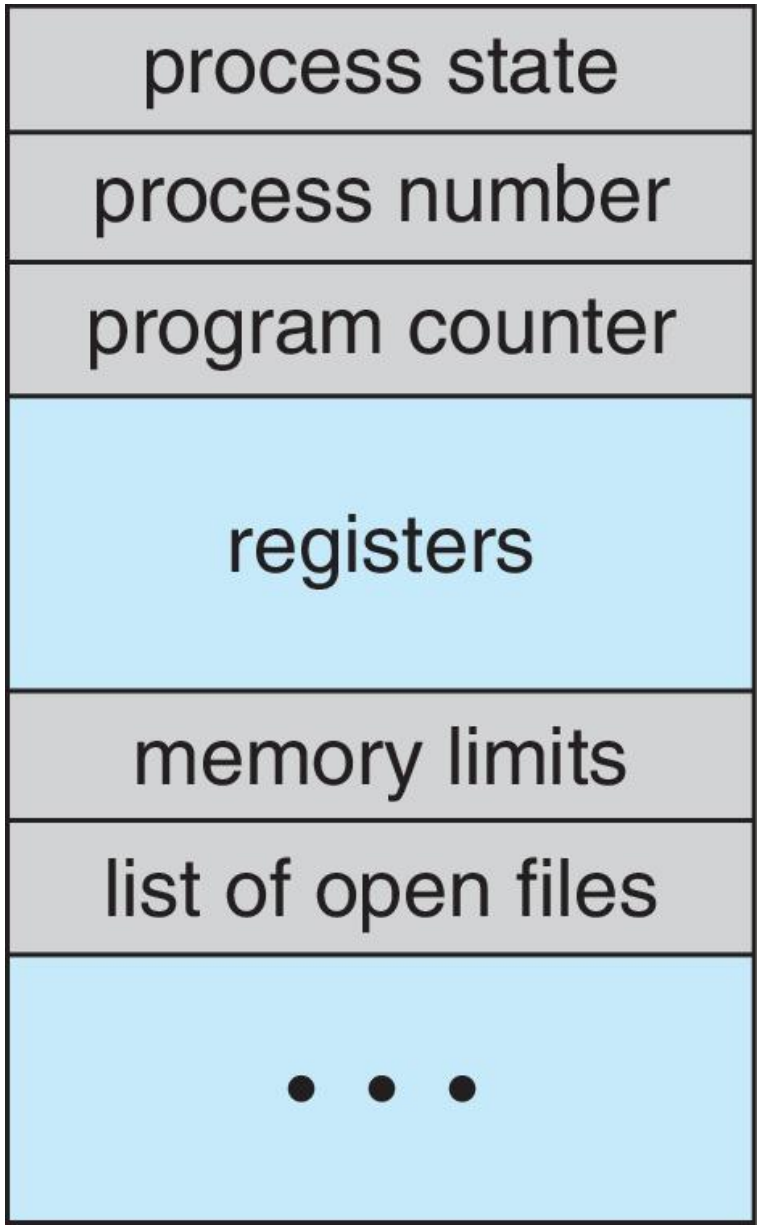
Questions?

- Process state
- Transition of process state
- Process state in Linux

Process Control Block (PCB)

Information associated with each process (also called **task control block**)

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files


process state
process number
program counter
registers
memory limits
list of open files
• • •

Threads

- So far, a process has a single thread of execution
- Consider having multiple program counters per process
  - Multiple locations can execute at once
    - Multiple threads of control -> **threads**
- Must then have storage for thread details, multiple program counters in PCB
- Explore in detail next week

Questions?

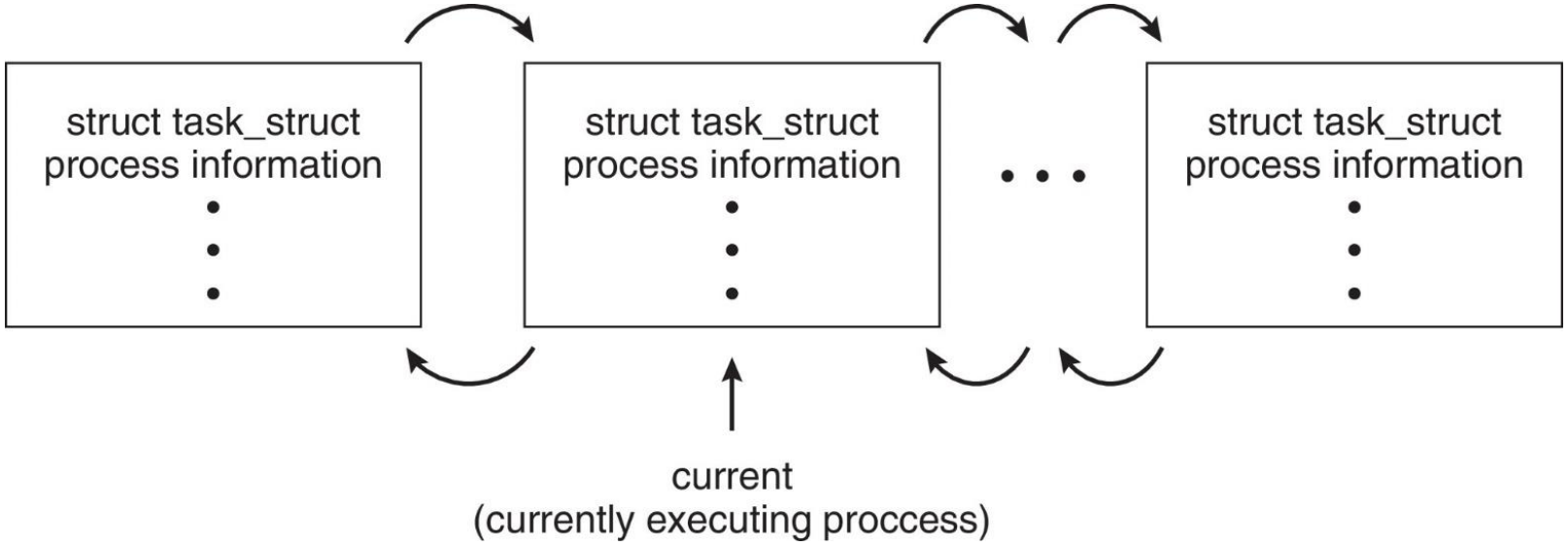
- Data structure for managing and representing process
- Concepts of thread of control/execution and thread.

Process Representation in Linux

- Represented by the C structure `task_struct`

```
pid t_pid;                    /* process identifier */
long state;                   /* state of the process */
unsigned int time_slice       /* scheduling information */
struct task_struct *parent;/* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files;/* list of open files */
struct mm_struct *mm;         /* address space of this process
*/
```

Process Representation in Linux: Task List




```
pid t_pid;                      /* process identifier */
long state;                     /* state of the process */
unsigned int time_slice         /* scheduling information */
struct task_struct *parent;/* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files;/* list of open files */
struct mm_struct *mm;           /* address space of this process
*/
```

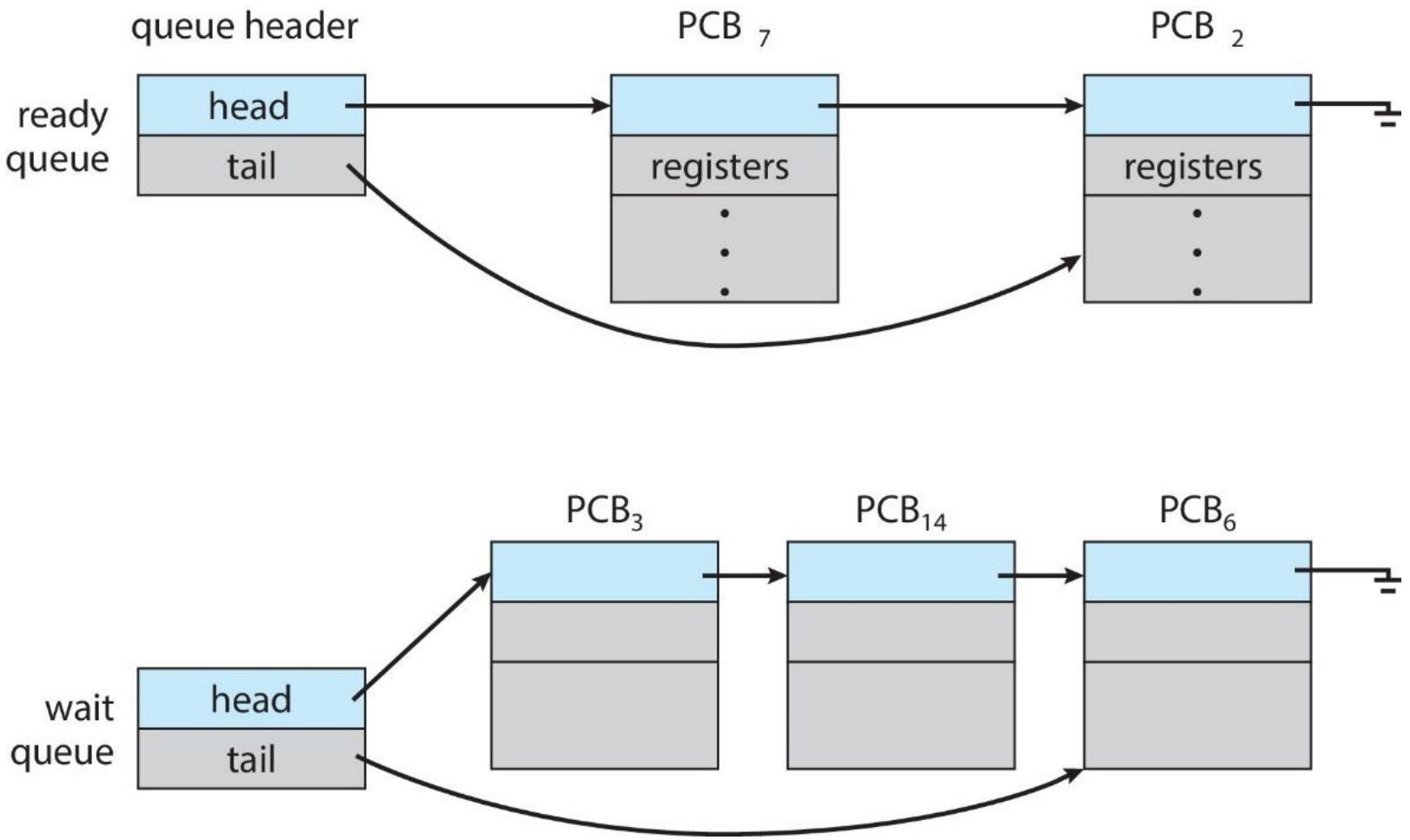
Questions?

- Process representation in Linux?

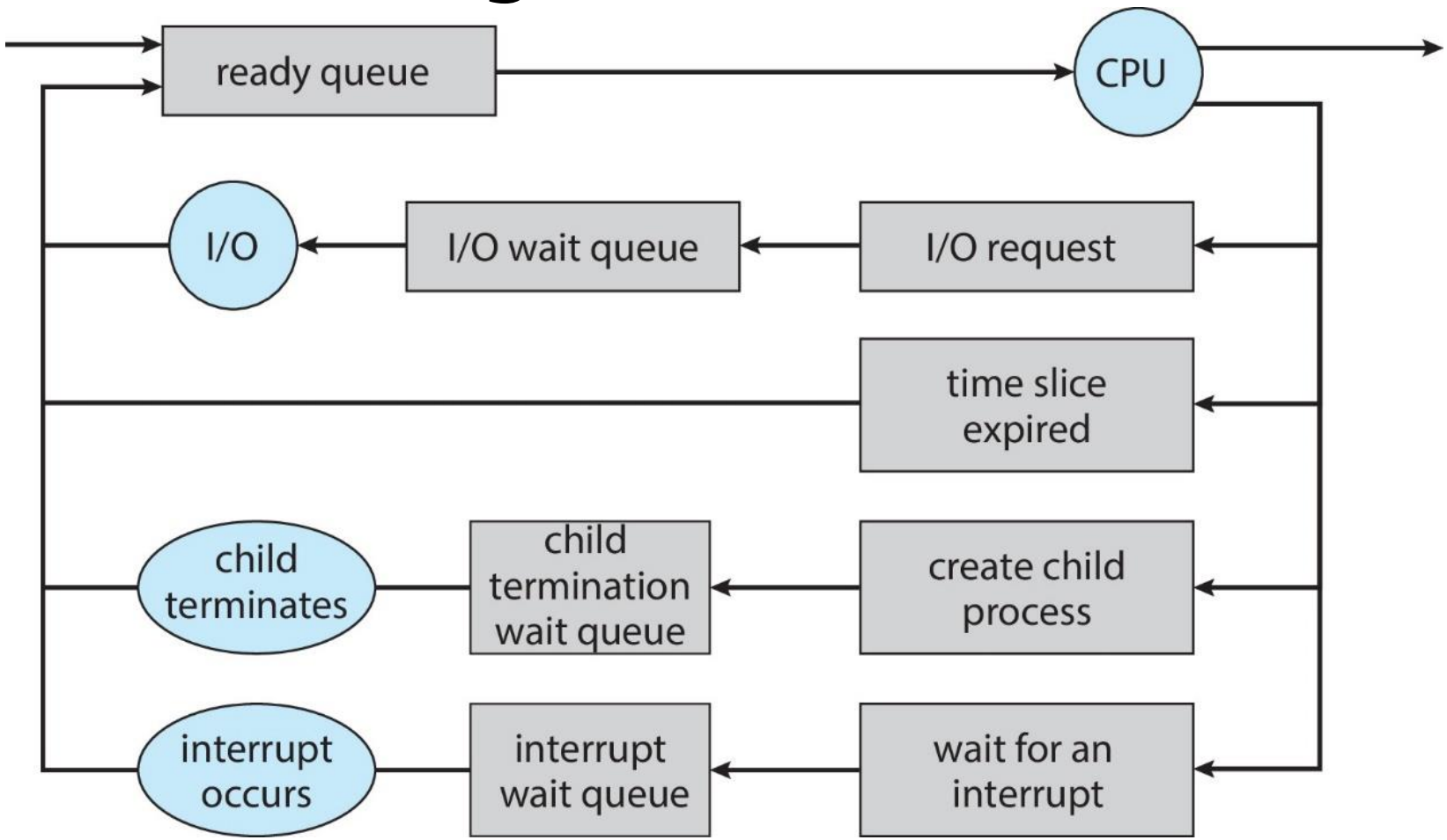
Process Scheduling

- Maximize CPU use, quickly switch processes onto CPU core
- ***Process scheduler*** selects among available processes for next execution on CPU core
- Maintains scheduling queues of processes
  - Ready queue – set of all processes residing in main memory, ready and waiting to execute
  - Wait queues – set of processes waiting for an event (i.e. I/O)
  - Processes migrate among the various queues

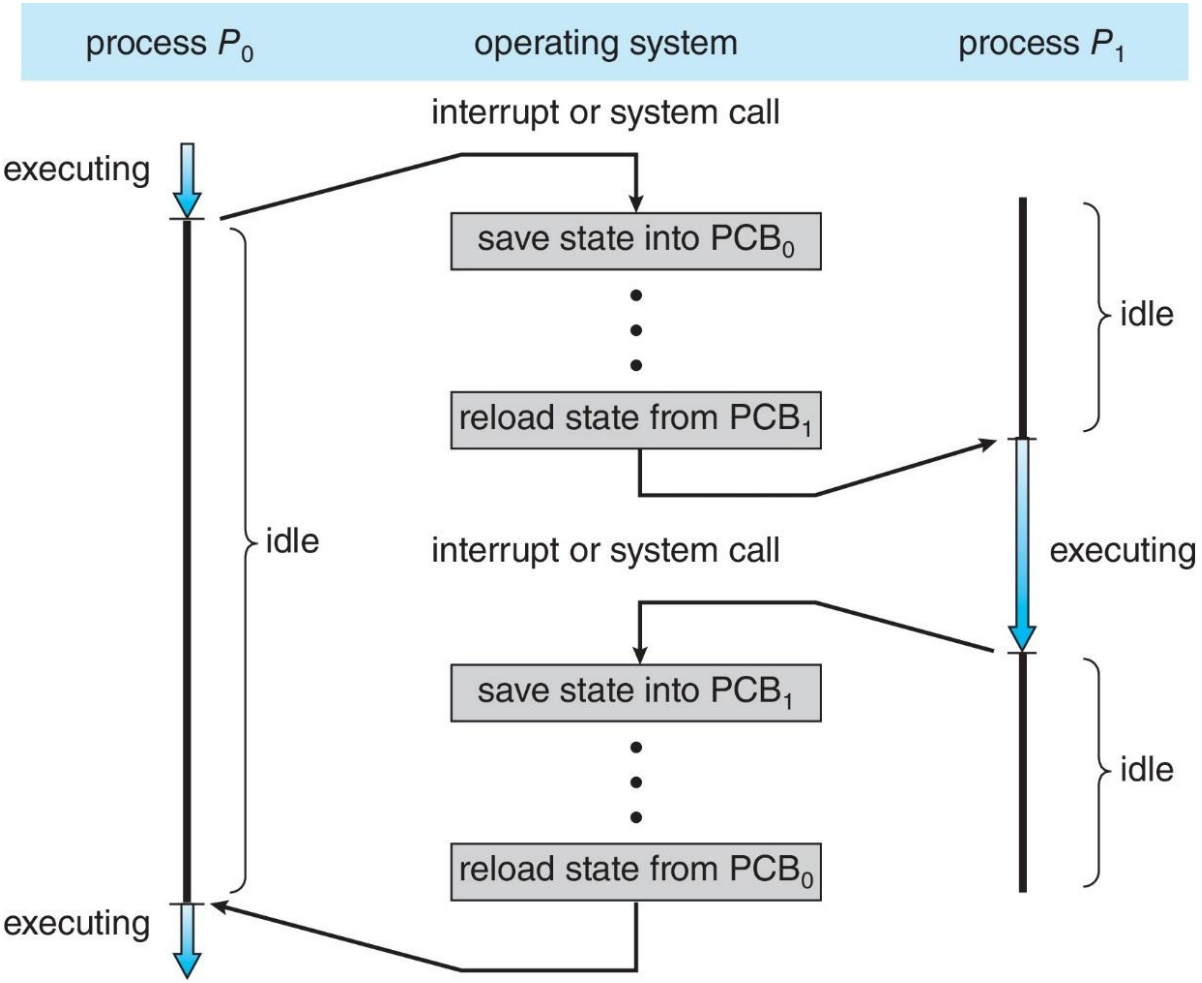
Ready and Wait Queues



Representation of Process Scheduling



CPU Switch From Process to Process



Context Switch

- A context switch occurs when the CPU switches from one process to another.

Context Switch: What must Happen?

- When CPU switches to another process, the system must ***save*** the state of the old process and ***load*** the saved state for the new process.
- ***Context*** of a process represented in the PCB
- Context-switch time is ***overhead***; the system does no useful work while switching
  - The more complex the OS and the PCB ➔ the longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU ➔ multiple contexts loaded at once

Questions?

- Concept of process scheduling
- Concept of context switch
- When must happen during a context switch?

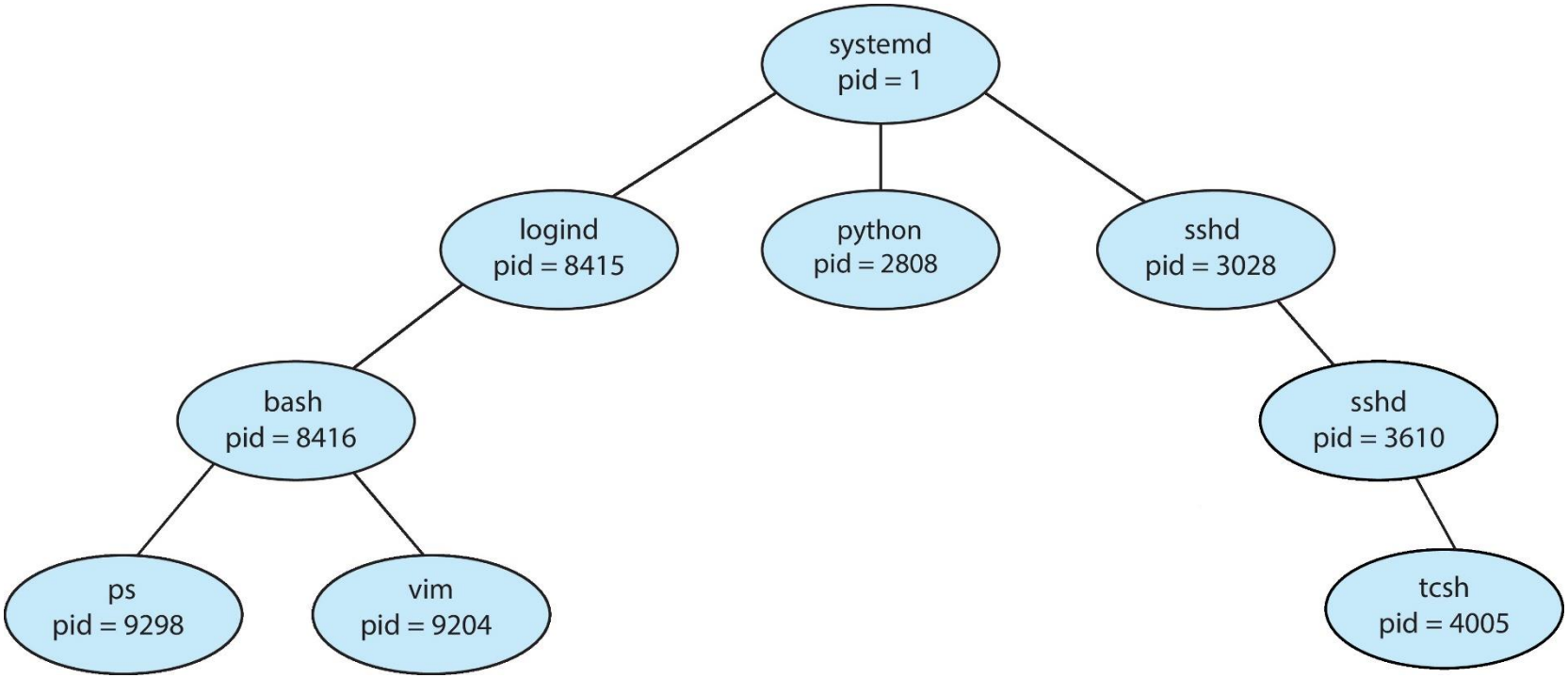
Operations on Processes

- System must provide mechanisms for:
  - process creation
  - process termination

Process Creation

- ***Parent*** process create ***children*** processes, which, in turn create other processes, forming a tree of processes
- Generally, process identified and managed via a ***process identifier (pid)***
- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate

A Tree of Processes in Linux (Examples: pstree or ps –ajxf)



Process Creation: Design Consideration

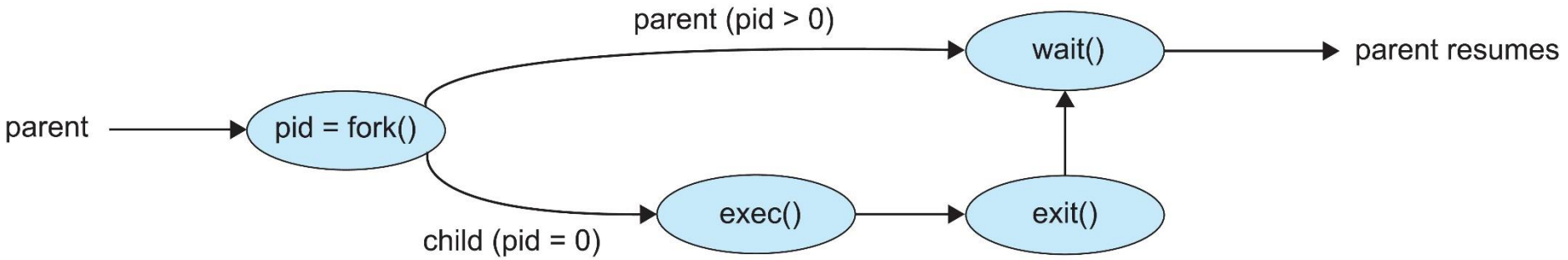
- Physical and logical resources
  - CPU time, memory, files, I/O devices
  - Child obtains from the OS
  - Child is constrained to a subset of the parent process's resources
- Program data
  - Parent process may pass initialization data to child process
- Address space
  - Child duplicate of parent
  - Child has a program loaded into it

Process Creation in UNIX

- **`fork()`** system call creates new process
- **`exec()`** system call used after a **`fork()`** to replace the process' memory space with a new program
- Parent process calls **`wait()`** for the child to terminate

Example Application in Linux

- See the example program



Example Application in Windows

- See the example program

Questions?

- Process creation
- Using system calls to create processes
- Demo programs (Linux and Windows)

Process Termination

- Processes executes last statement (normal process termination)
- Parent terminates child process (abort the child process)

Normal Process Termination

- Process executes last statement and then asks the operating system to delete it
  - e.g., in UNIX, using the **`exit()`** system call.
  - Returns status data from child to parent (e.g., via **`wait() in UNIX`**)
  - Process' resources are deallocated by operating system

Abort Child Process

- Parent may terminate the execution of children processes.
  - e.g., using the **`abort()`** system call
  - Some reasons for doing so:
    - Child has exceeded allocated resources
    - Task assigned to child is no longer required
    - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates

Terminate Children or Wait for Children?

- Allow child process to exist without the existence of the parent?
- Allow parent to wait for child to complete?

Terminate All Children

- Some operating systems do not allow child to exists if its parent has terminated.
- If a process terminates, then all its children must also be terminated.
  - **cascading termination.** All children, grandchildren, etc. are terminated.
  - The termination is initiated by the operating system.

Wait for Children

- The parent process may wait for termination of a child process
  - e.g., in UNIX, by using the **`wait()`** system call. The call returns status information and the pid of the terminated process

```
pid = wait(&status);
```

Zombie Process

- A process that has terminated, but whose parent has not yet called wait(), is known as a zombie process.
- All processes transition to this state when they terminate, but generally they exist as zombies only briefly.
- Once the parent calls wait(), the process identifier of the zombie process and its entry in the process table are released.

Orphan Process

- A parent did not invoke wait() and instead terminated, thereby leaving its child processes as orphans.
- In UNIX, assign new parent (initd/systemd)

Terminating Processes: Linux Examples

- Example programs

Questions?

- Process termination?
- Process termination in UNIX?
- Zombie?
- Orphan?
- Demo programs