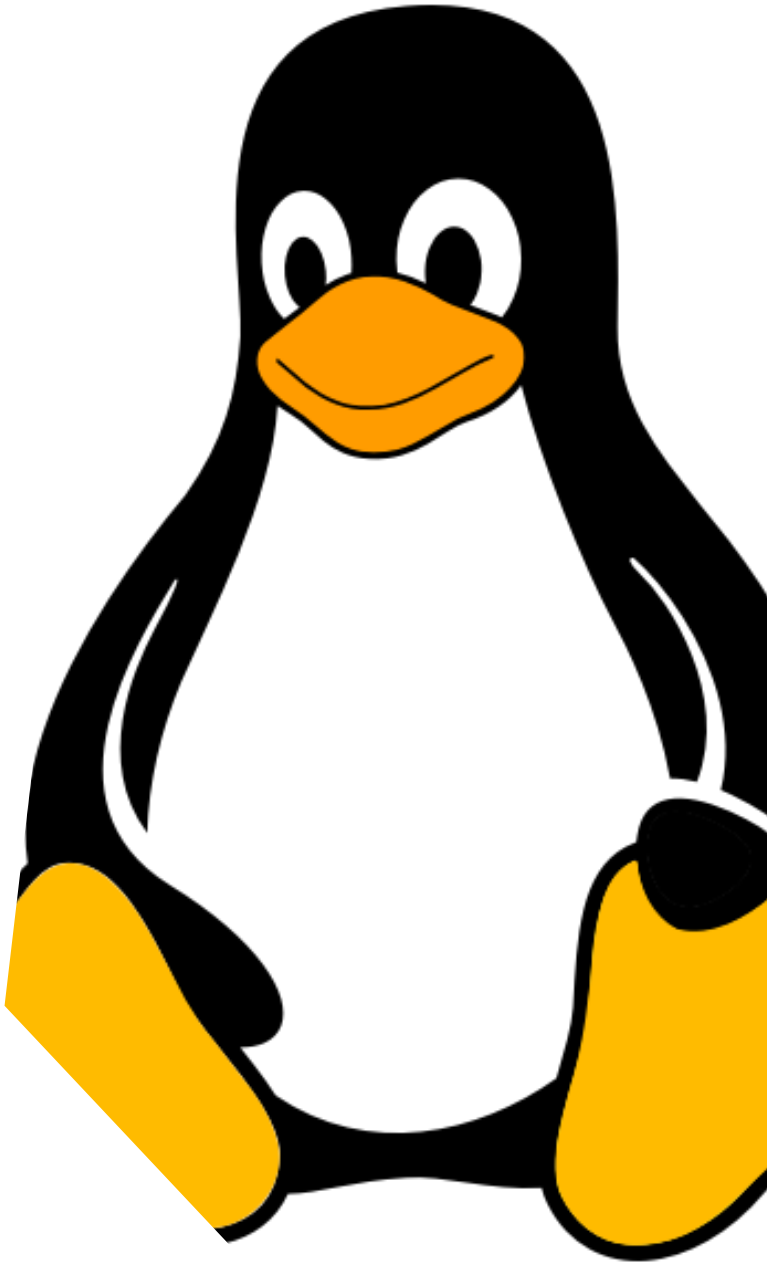
CS4500
Operating Systems

Week 3

Course Content

Unit	Module	Week	Topic
1 – Introduction	1 & 2	1	Welcome to OS & Introduction
	2	2	OS Overview
2 – Processes and Threads	3	3	Processes
	4	4	Threads
	5	5	CPU Scheduling
	6	6	Multiprocessor Scheduling
	7	7	Interprocess Communication
	8	8	Pthread
3 – Deadlocks & Memory	9	9	Deadlock
	10	10	Memory Management
	--	11	Midterm Exam
4 – Additional Topics	11	11	Page Replacement
	12	13	File Systems
	13	14	IO Devices
	14	15	Security
	--	16	Final Exam

Introduction to Operating Systems: Processes



What to expect today:

An overview of processes

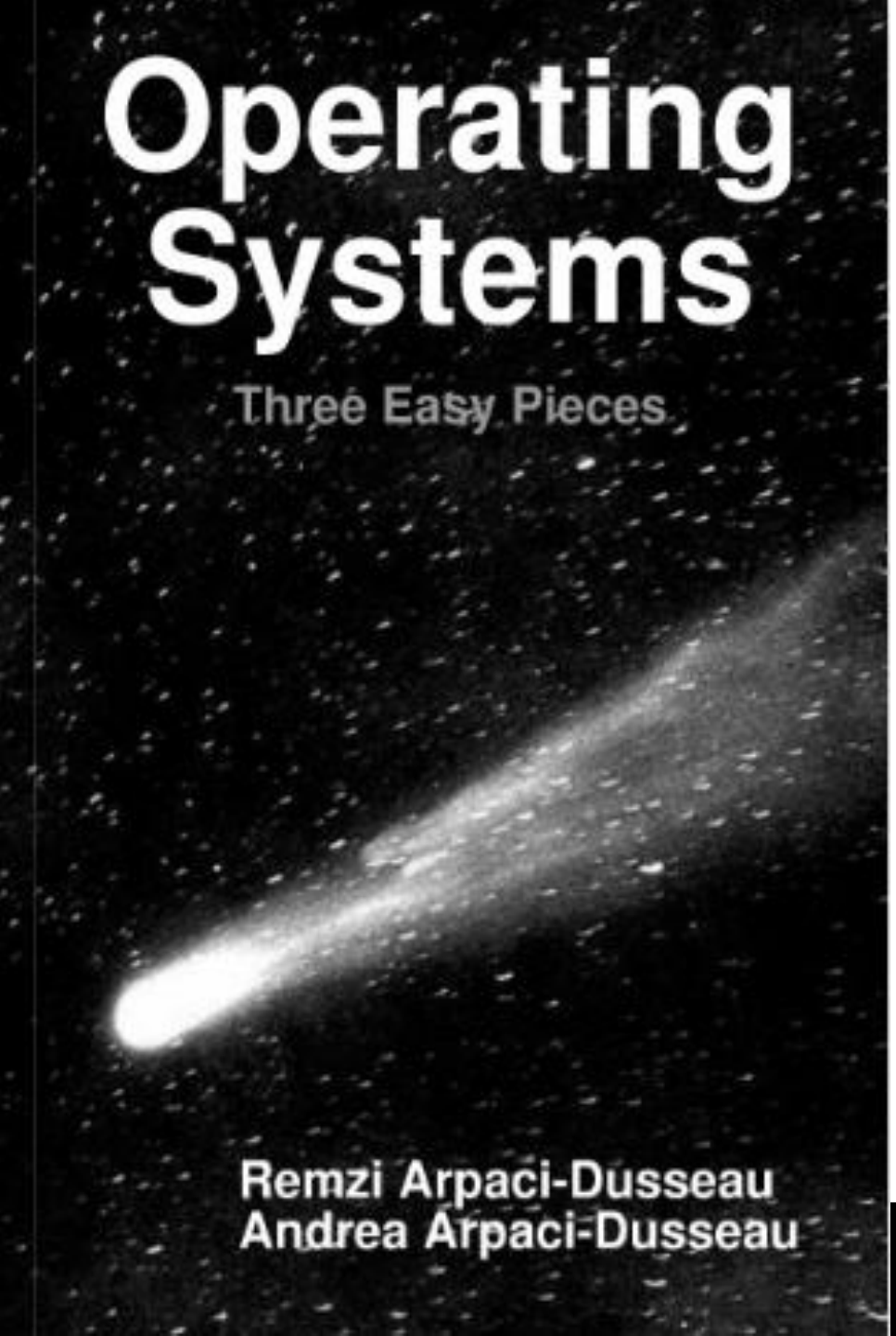
Assigned reading

- Andrew S. Tanenbaum, Modern Operating Systems, 4thd edition, 2014, Prentice Hall Chapter 2

Lecture Resources

Andrew S. Tanenbaum, Modern Operating Systems, 5th edition, 2022, Prentice Hall

Peter Jay Salzman, Michael Burian, Ori Pomerantz, Bob Mottram, Jim Huang. The Linux Kernel Module Programming Guide, 2022. https://sysprog21.github.io/lkmpg/



Additional Resource

Remzi H. Arpaci-Dusseau, R. H., Arpaci-Dusseau, A.C. *Operating systems: Three easy pieces: Chapter 10 and 28.* (2018). Arpaci-Dusseau Books.

https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched-multi.pdf

Copy of .pdf available in Canvas

Recap of the Last Class

Computer hardware

- Time-multiplexed – resource not divided into units
- Space-multiplexed – resource split in pieces, then assigned a process

OS components

- Process management
- Memory management
- File and storage management

Which of the following are likely components of an operating system?

A. Cache memory
B. Scheduler
C. File editor
D. File system
E. Device driver
F. Web browser

Which of the following are likely components of an operating system?

A. Cache memory
B. Scheduler
C. File editor
D. File system
E. Device driver
F. Web browser

Process

Process == Program??

Process

Definitions

- An instance of a program running on a computer
- An abstraction that supports running programs
- An *execution stream* in the context of a particular *process state*
- A *sequential* stream of execution in its *own address space*

Ref. MOS3E, OS@Austin, Columbia, Rochester

Process

Two parts of a process

- Sequential execution of instructions
- Process state
  - Registers: PC, SP,…
  - Memory: address space, code, stack, heap …
  - I/O status: opened files …

Program vs. Process

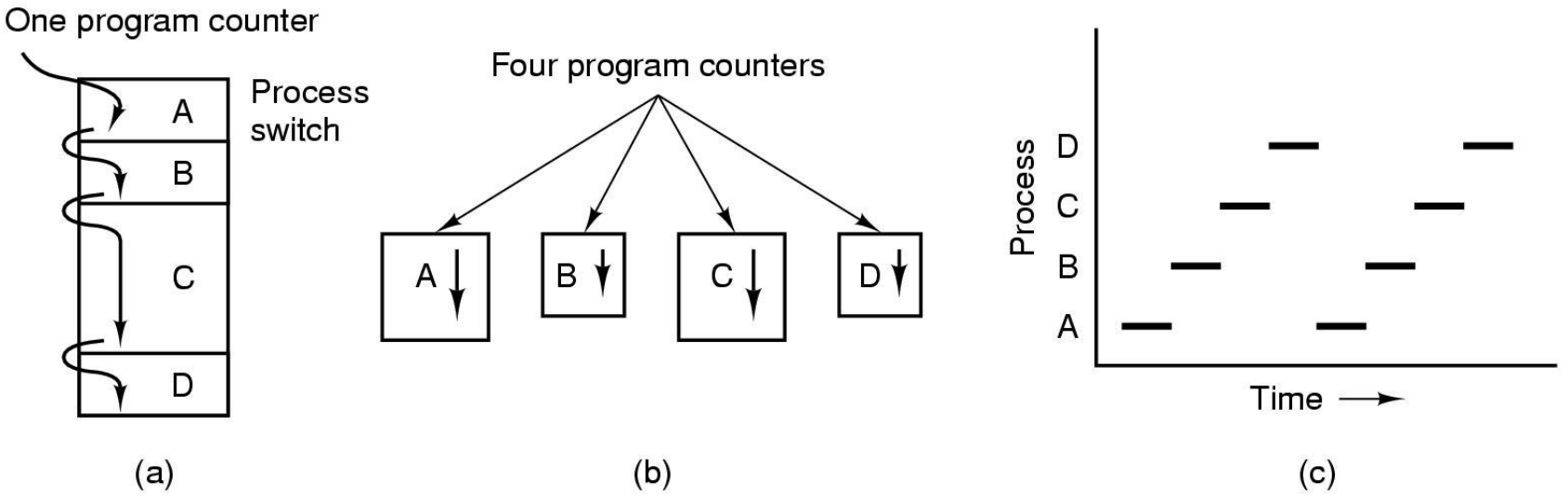
Program != Process

- Program = static code + data
- Process = dynamic instantiation of code + data + files
  …

No 1:1 mapping

- A program can invoke many processes
  - Running the same program twice
  - A program contains fork()

The Process Model



- (a) Multiprogramming of four programs
- (b) Conceptual model of 4 *independent*, *sequential processes*
  - Sequential process mode: hiding the effects of interrupts, and support blocking system calls
- (c) Only one program active at any instant

Process Creation

Principal events that cause process creation

- System initialization; foreground and background
- Execution of a process creation system call
- User request to create a new process; interactive systems
- Initiation of a batch job

Process Creation

UNIX example

- ***fork*** system call creates an **exact copy** of calling process
  - Same memory image, environment settings, and opened files
  - After fork, caller is parent, newly-created process is child

After fork()

The child process returns executing at the exact same point after its parent called fork()

- fork() returns **twice**: the new PID to the parent, and 0 to the child

```
pid= fork();
if (pid == 0) {
    /* I am the child (0: invalid PID)
*/
} else {
   /* I am the parent */
}
```

Two processes execute the code! (parent/child share same text)

- All memory contents of parent/child are identical
- Both have the same files open at the same position (point to the same file objects)

Putting it Together

```
/* now create new process */
pid = fork();
char *const parmList[] = {"./Helloworld", NULL};
if (pid == 0) /* fork() returns 0 to the child process */
    {
        sleep(1);
        printf("CHILD: My parent's PID: %d\n", getppid());
       execve("./Helloworld", parmList);
       printf("retval of Helloworld: %d\n", retval);
        exit(retval);
    }
else /* fork() returns new pid to the parent process */
    {
        printf("PARENT: my child PID: %d\n", pid);
        wait(&status);
        printf("PARENT: Child's exit code is: %d\n", WEXITSTATUS(status));
       exit(0);
    }
```

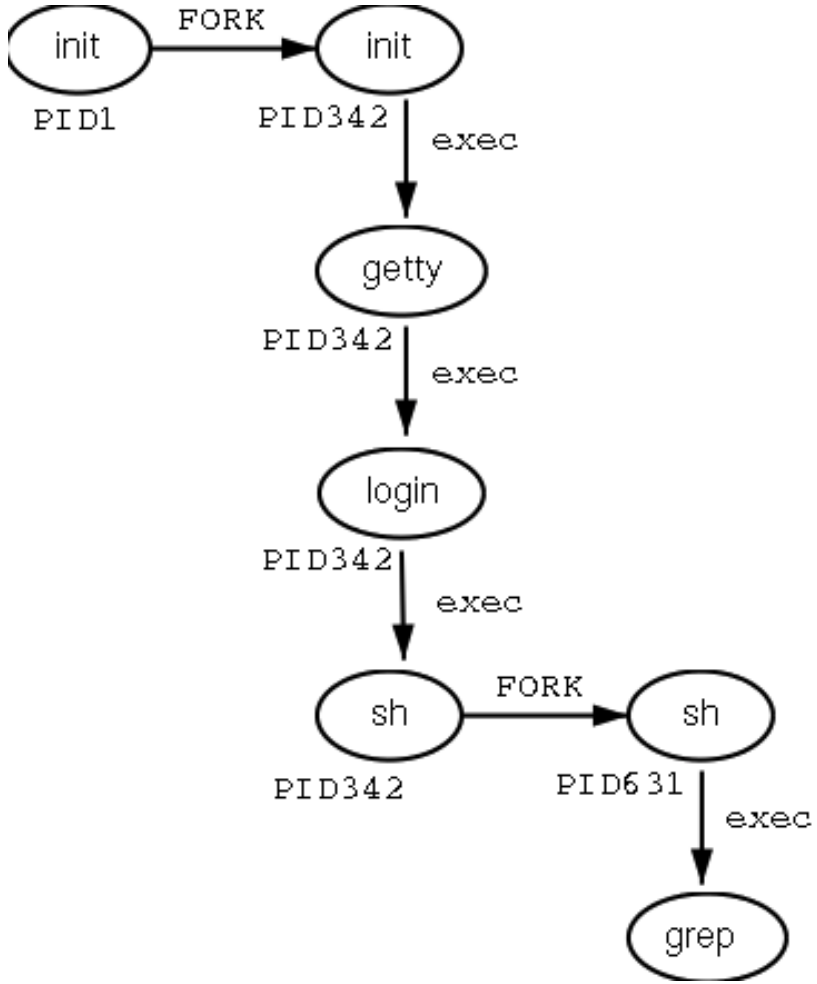
Process Termination

Conditions which terminate processes

- Normal exit (voluntary)
- Error exit (voluntary)
- Killed by another process (involuntary)

Process Hierarchies (Trees)

Parent creates a child process, child processes can create its own process

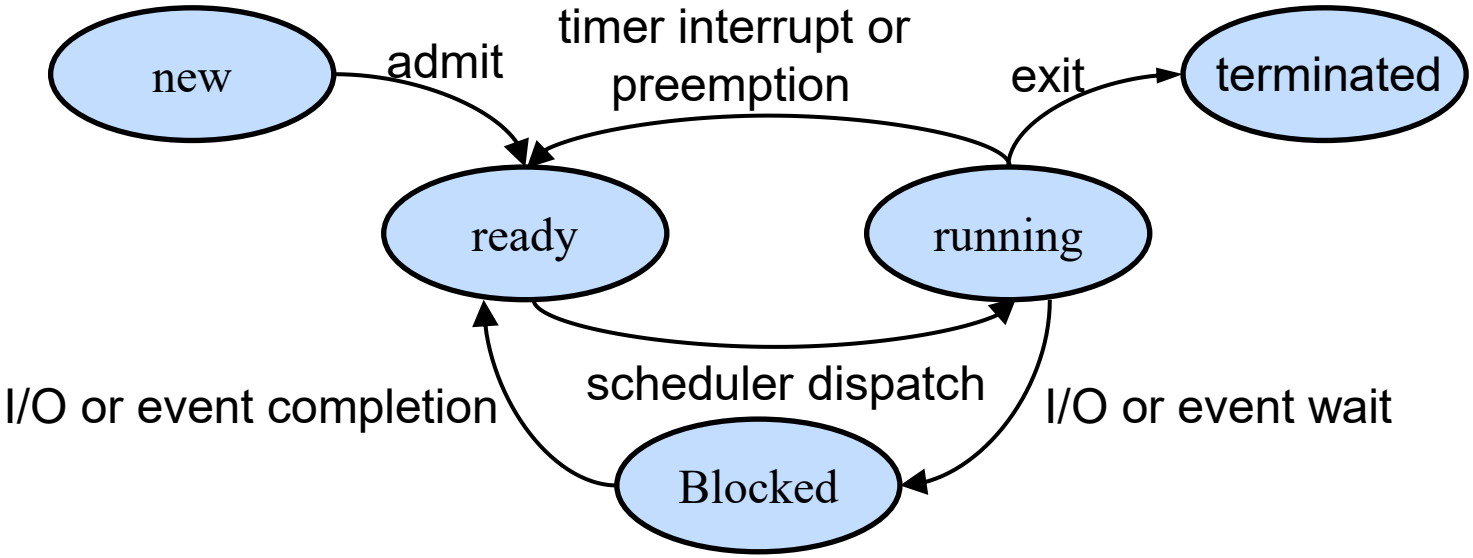


Forms a hierarchy

- UNIX: a process and all its children and further descendants form a "process group“
- ***init***, a special process present in the boot image

- Try: **`pstree -h`**

Process Life Cycle



Process Life Cycle Stages

- **New** - The process is in the stage of being created.
- **Ready** - The process has all the resources available that it needs to run, but the CPU is not currently working on this process's instructions.
- **Running** - The CPU is working on this process's instructions.
- **Waiting** - The process cannot run because it is waiting for some resource to become available or for some event to occur. **Terminated -** The process has completed.

Implementation of Processes

Process table

- One entry per process
- Each entry is called a process control block (PCB)

Process control block (PCB)

- OS data structure containing data associated with processes

Process Elements

- Identifier
- State
- Priority
- Program counter
- Memory pointers
- Context data
- I/O status
- Accounting information

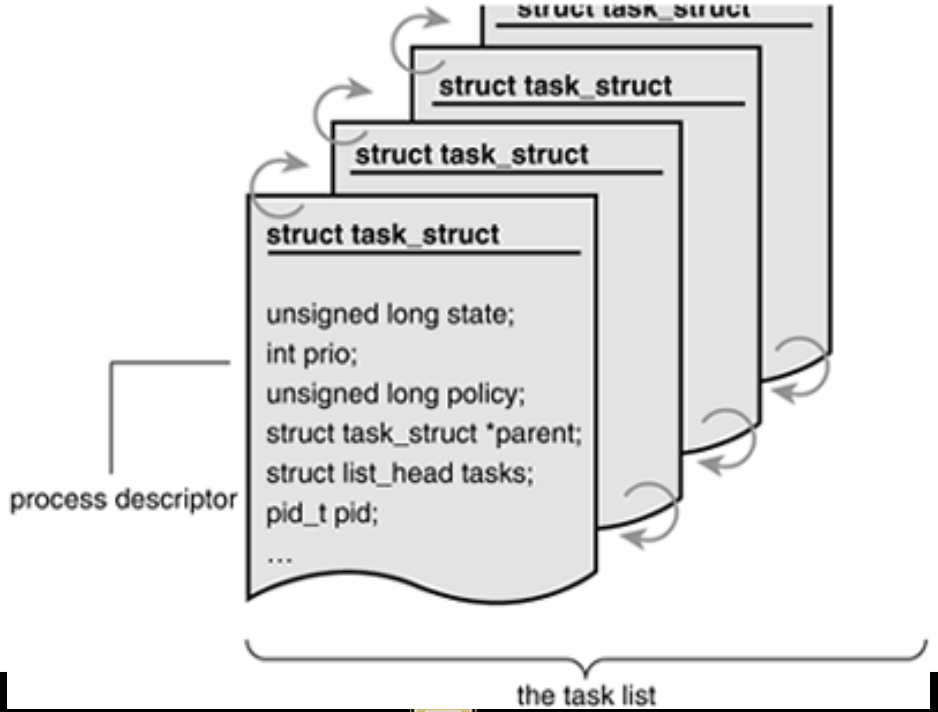
Process Control Block (PCB)

- Contains the elements of the process
- Created and maintained by OS
- Supports multiple processes

Linux Processes

Process table: implemented as a linked list/ hashtable

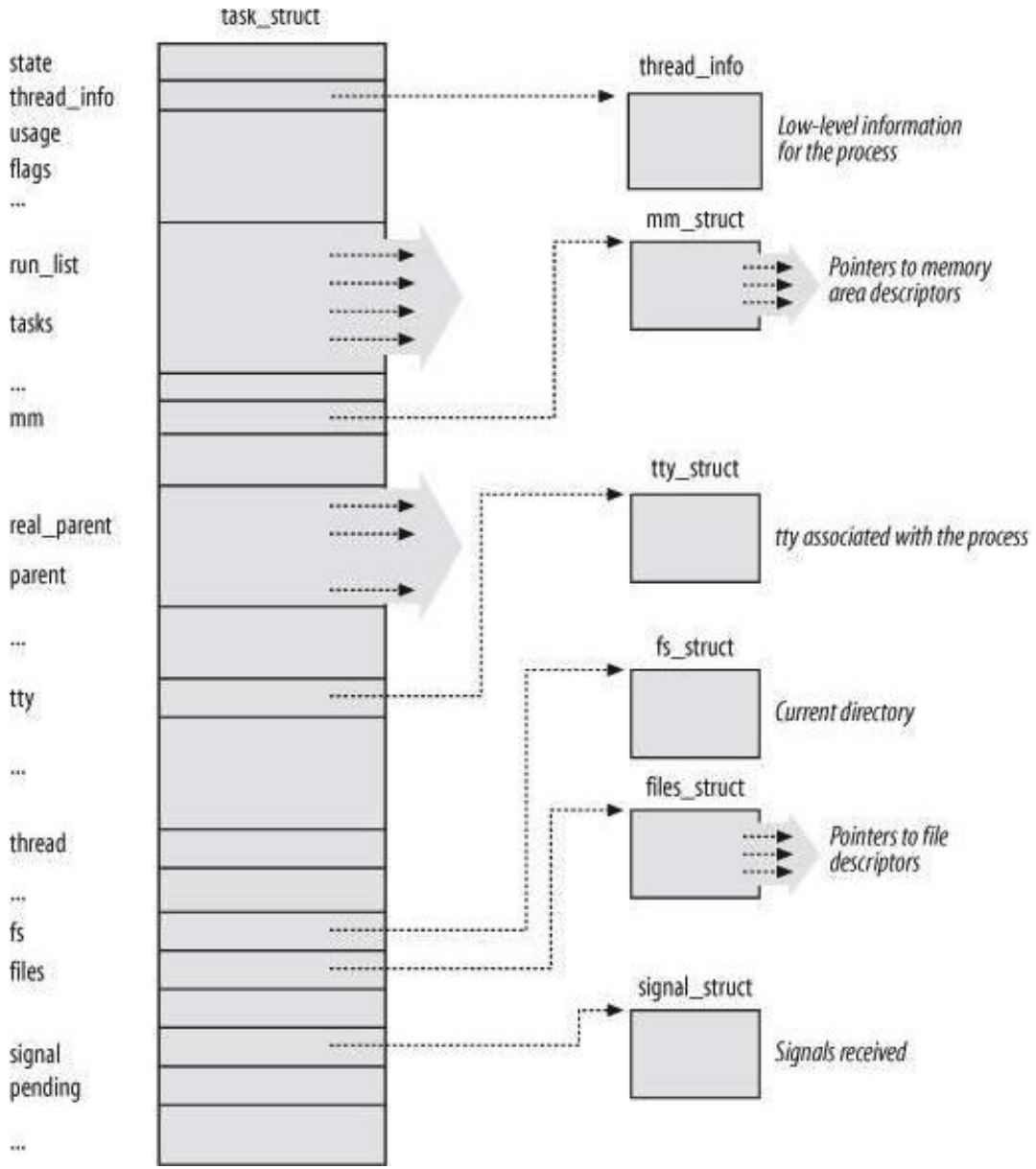
- Each element: a process descriptor of type **task_struct**
- Dynamically allocated for each process
- Process descriptor
  - Contains all info about a specific process



Linux Processes

Process descriptor (PCB)

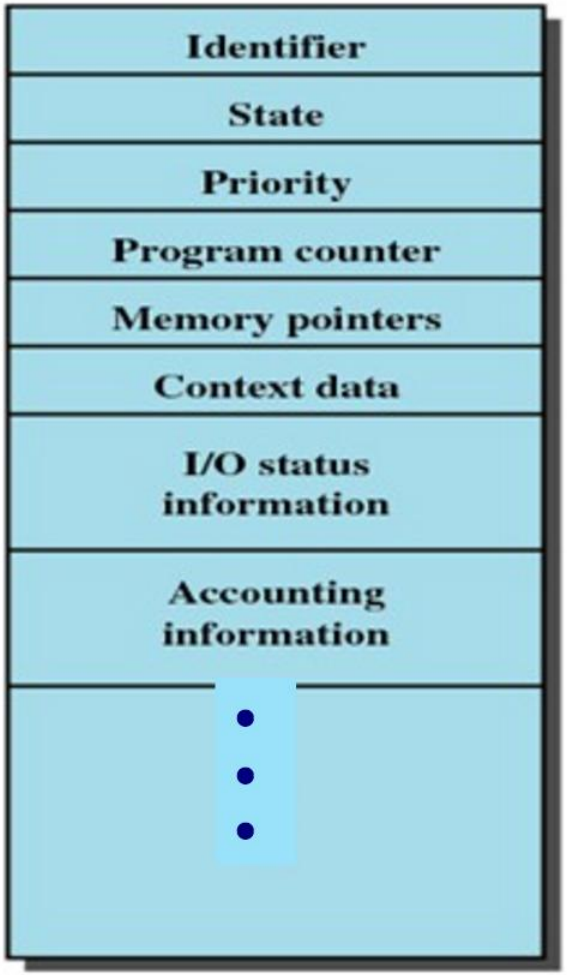
- State
- Identifiers
- Scheduling info
- File system
- Virtual memory
- …



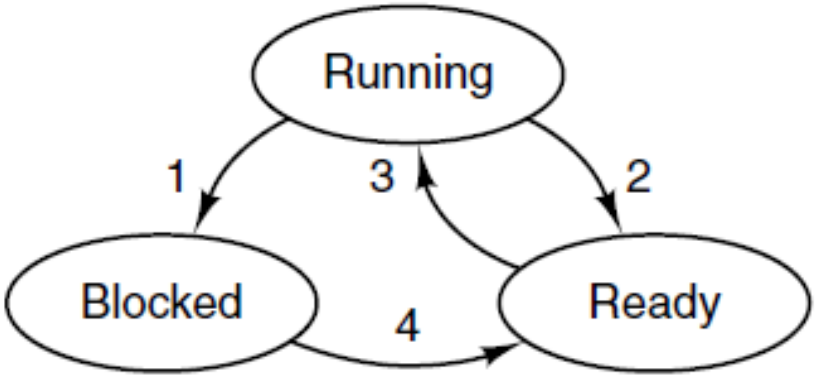
Linux Process Descriptor

State

- TASK_RUNNING
  - Running
- TASK_INTERRUPTABLE
  - Blocked
- EXIT_ZOMBIE
  - Terminated by not deallocated
- EXIT_DEAD
  - Completely terminated



Process States



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Figure 2-2. A process can be in running, blocked, or ready state. Transitions between these states are as shown.

Process States

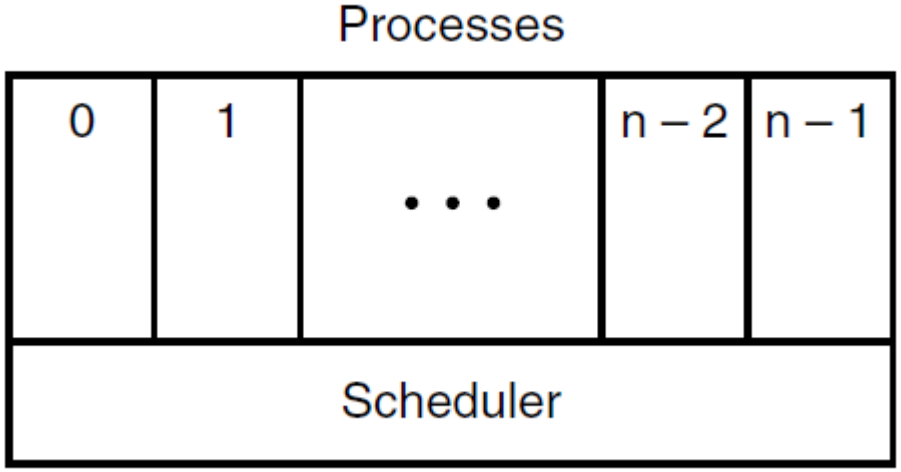


Figure 2-3. The lowest layer of a process-structured operating system handles interrupts and scheduling. Above that layer are sequential processes.

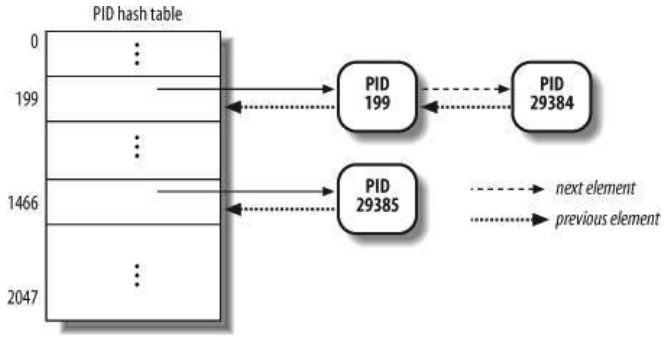
Linux Process Descriptor (cont')

Identifiers

- pid: PID of the process/thread
- pgrp: PID of the group leader

How to get the pointer to a specific process?

- The **`current`** macro
- The **`init_task`** macro
- find_task_by_vpid(pid_t pid)




```
pid* pid_struct = find_get_pid(int pid);
   task_struct* task = pid_task(pid_struct,PIDTYPE_PID);
```

Linux Process Descriptor (cont')

Files

- fs_struct
  - file system information: root directory, current directory
- files_struct
  - Information on opened files

Summary

What is a process?

- An instantiation of a program

Program life cycle

- Ready, running, blocked, new, terminated

Process implementation

- Process table, PCB

Additional practice

- Download Linux kernel source to your VM, find the following fields in structure task_struct (PCB) in LINUX_SRC_FOLDER/include/linux/sched.h
  - Program counter (try to google)
  - Stack pointer
  - Process ID
  - Opened file descriptors

For Next Time

Read: Andrew S. Tanenbaum, Modern Operating Systems, 4thd edition, 2014, Prentice Hall Chapter 2

Course Content

Unit	Module	Week	Topic
1 – Introduction	1 & 2	1	Welcome to OS & Introduction
	2	2	OS Overview
2 – Processes and Threads	3	3	Processes
	4	4	Threads
	5	5	CPU Scheduling
	6	6	Multiprocessor Scheduling
	7	7	Interprocess Communication
	8	8	Pthread
3 – Deadlocks & Memory	9	9	Deadlock
	10	10	Memory Management
	--	11	Midterm Exam
4 – Additional Topics	11	11	Page Replacement
	12	13	File Systems
	13	14	IO Devices
	14	15	Security
	--	16	Final Exam