CSE3320
Operating Systems
Processes

Jia Rao

Department of Computer Science and Engineering

http://ranger.uta.edu/~jrao

Recap of the Last Class

- Computer hardware
  - Time-sharing
  - Space-sharing
  - Characteristics
    - Locality, multiple working modes, load-balancing
- OS components
  - Process management
  - Memory management
  - File and storage management

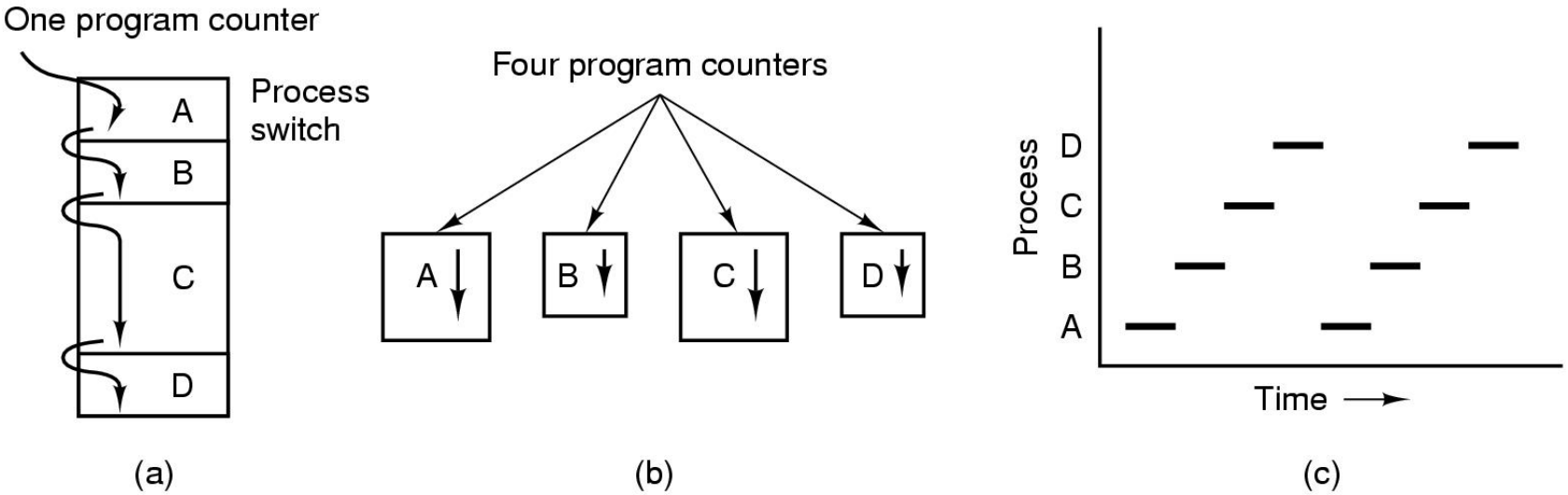
Process

- Definition
  - 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*
- Two parts of a process
  - Sequential execution of instructions
  - Process state
    - registers: PC, SP,…
    - Memory: address space, code, data, stack, heap …
    - I/O status: open files …

Program v.s. Process

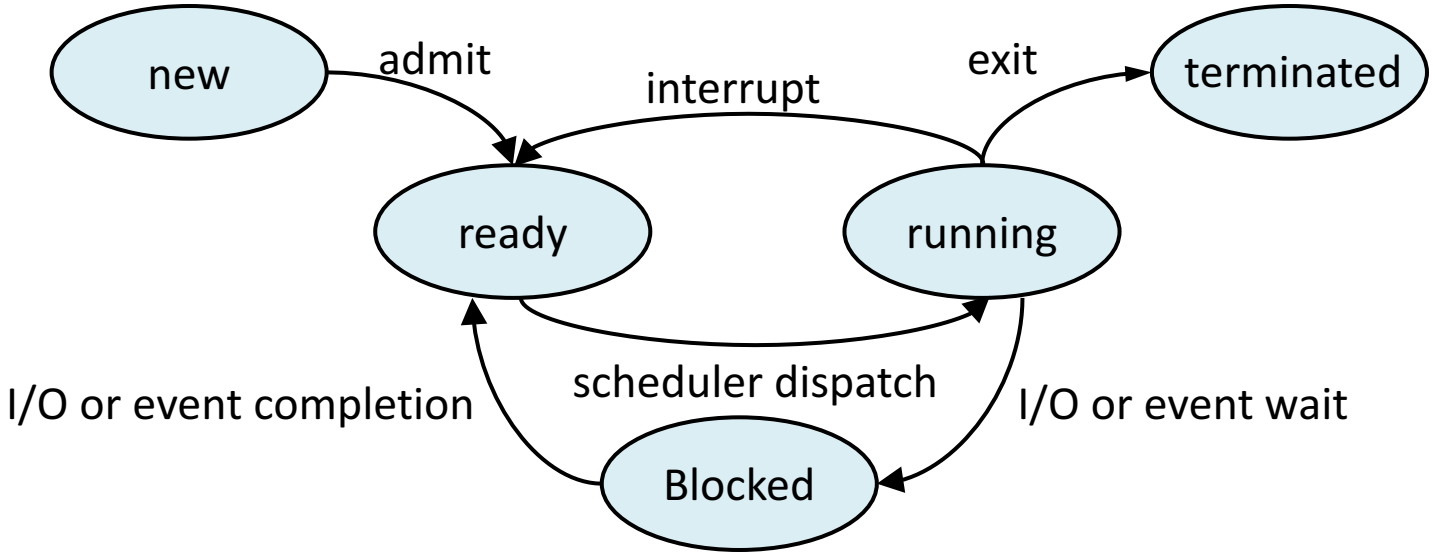
- Program != Process
  - Program = static code + data
  - Process = dynamic instantiation of code + data + files …
- No 1:1 mapping
  - A program can invoke many processes
  - Many processes of the same program

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 Life Cycle



Blocked: unable to run, wait for an event
Ready: willing to run, wait for the CPU

Process Creation

- Principal events that cause process creation
  - System initialization; foreground and background
  - Execution of a process creation system
  - User request to create a new process; interactive systems
  - Initiation of a batch job
- UNIX example
  - ***fork*** system call creates an exactly copy of calling process
    - Same memory image, environment settings, and open files
  - Child process calls ***execve*** to change its memory image and run a new program

Process Termination

- Conditions which terminate processes
  - Normal exit (voluntary)
  - Error exit (voluntary)
  - Fatal error (involuntary)
  - Killed by another process (involuntary)

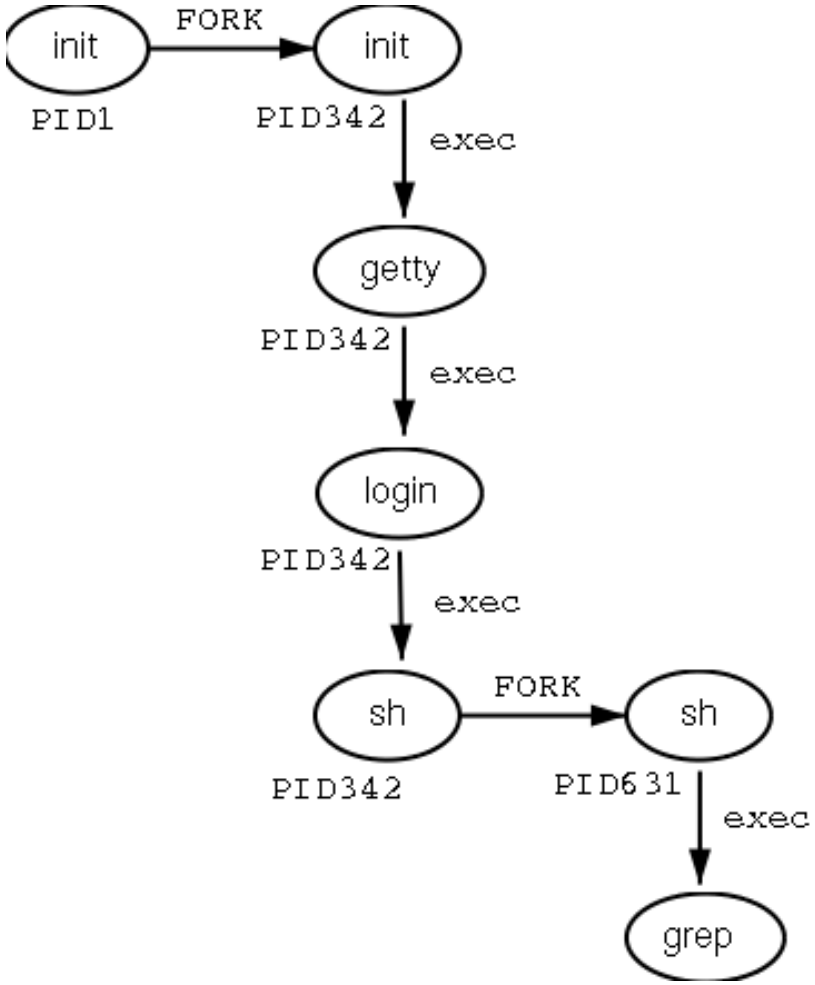
Put it Together

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

Process Hierarchies (Trees)

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

- Forms a hierarchy
  - UNIX calls this a "process group“
  - ***init***, a special process is present in the boot image
  - Try: **`pstree -h`**



Implementation of Processes

- Process table
  - One entry per process
  - Each entry is called a process control block (PCB)
- Process control block
  - OS data structure containing info associated with processes
    - Process state (ready, running, blocked)
    - Program counter
    - CPU registers
    - Scheduling info (priorities)
    - Memory management info
    - Accounting info (elapsed runtime)
    - Opened files

Multiprogramming

- Rapid switching between processes gives the illusion of running multiple programs in parallel
- When to switch to a process ?
  - Interrupts
    - I/O interrupt
    - Timer interrupt
  - Memory fault
  - Trap

OS Interrupt Handling

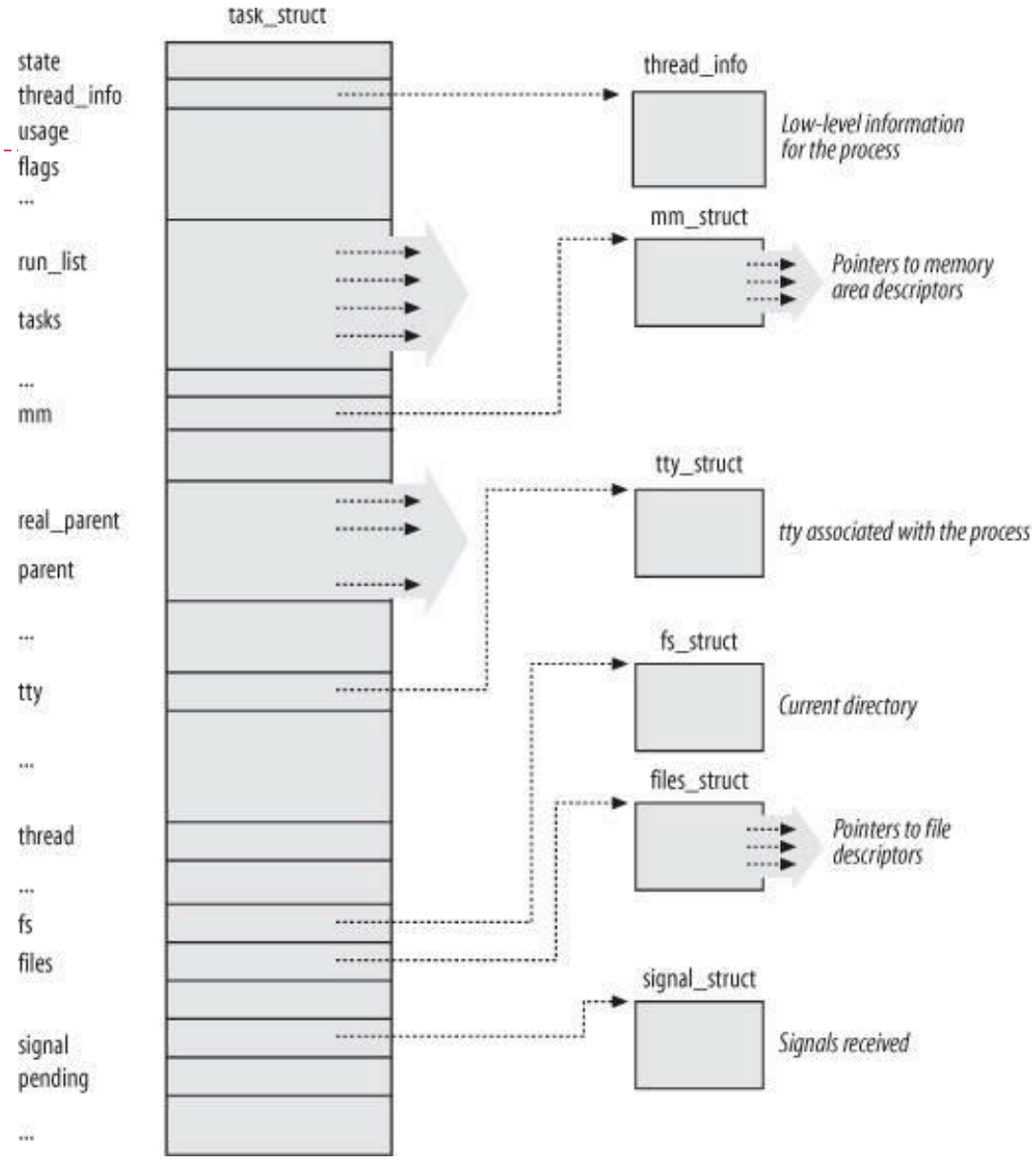
- Interrupt vector
  - contains the address of the interrupt service procedures
  - Jump table

1. Hardware stacks program counter, etc.
2. Hardware loads new program counter from interrupt vector.
3. Assembly language procedure saves registers.
4. Assembly language procedure sets up new stack.
5. C interrupt service runs (typically reads and buffers input).
6. Scheduler decides which process is to run next.
7. C procedure returns to the assembly code.
8. Assembly language procedure starts up new current process.

Skeleton of what lowest level of OS does when an interrupt occurs when a process is running

Linux Processes

- Process descriptor (PCB)
  - State
  - Identifiers
  - Scheduling info
  - Links
  - File system
  - Virtual memory
  - Processor specific context

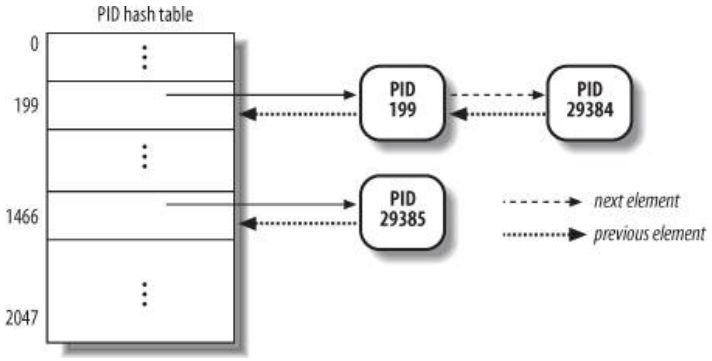


Linux Process Descriptor

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

Linux Process Descriptor (cont')

- Identifiers
  - pid: PID of the process/thread
  - tgid: PID of the thread group leader
  - pgrp: PID of the group leader
  - Session: PID of the session leader
- How to get the pointer to a specific process ?
  - The **`current`** macro
  - PID hash table



Linux Process Descriptor (cont')

- Scheduling information
  - prio, static_prio, normal_prio
  - rt_priority
  - sched_class
    - Task->sched_class->pick_next_task(runqueue)

Linux Process Descriptor (cont')

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

Linux Process Descriptor (cont')

- Virtual memory
  - mm_struct: describes the content of a process's virtual memory
    - The pointer to the page table
    - Pointers to the virtual memory areas

Summary

- What is a process ?
  - An instantiation of a program
- Program life cycle
  - Ready, running, blocked, new, terminated
- Process implementation
  - Process table, PCB
- Multiprogramming
- 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