CISC 7310X
C03: Process

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Recap & Issues

- Topics
  - Tools of the trade
    - SCM (e.g., Git), Virtualization (e.g., Virtual Box), Operating Systems (e.g., Linux)
  - A glimpse of systems research
  - OS overview
    - Kernel mode, user mode, kernel, shell
  - Computer systems hardware overview
    - Architecture and organization, interrupts and I/O
- Assignments
  - Practice and Project 1

Project 1 Observations

- What should be in the repository?
  - Derived artifacts generally should not be in the repository
    - Can be reproduced, with difference without substance
    - Occupy space
    - Binary files, "diff" is difficult
- Experiences?
  - OS kernels, shells, terminals, modules, device drivers
  - Systems programming
  - System calls
- Question: how are these related to the discussion in the lectures?

Questions?

- Recap & issues
  - Topics discussed
  - Assignments

Program and Process

- Program
  - A sequence of instructions
- Process
  - Representing an activity consisting of a program, input, output, and a state

User's Perspective: Creating Process

- In Unix, use system call fork()
  - Unix is interpreted as any POSIX-based systems
    - Linux, FreeBSD, OS X, Solaris, Android, iOS
  - What fork() does?
- Example
  - In Sample Programs

User's Perspective: Creating Process

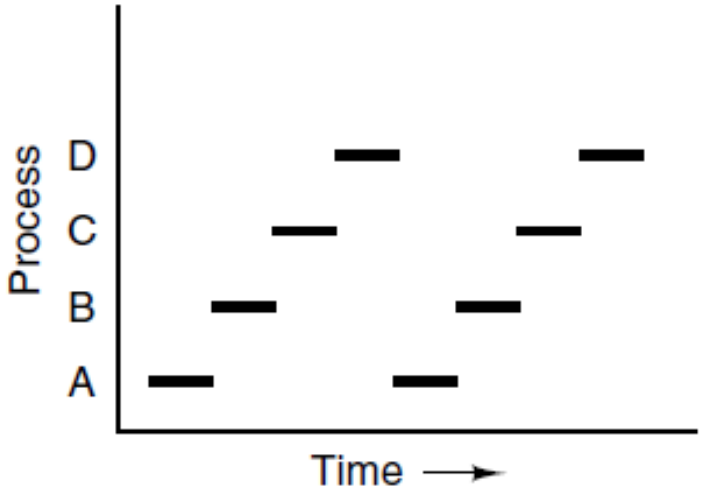
- On Windows, use Win32 API CreateProcess
  - Must load a specified program
- Example
  - In Sample Programs

User's Perspective: Process Status

- In Linux, use ps
- In Windows, use Task Manager (taskmgr.exe) or TaskList (tasklist.exe)

Multiprogramming

- Process rapidly switching back and forth to share the CPU time

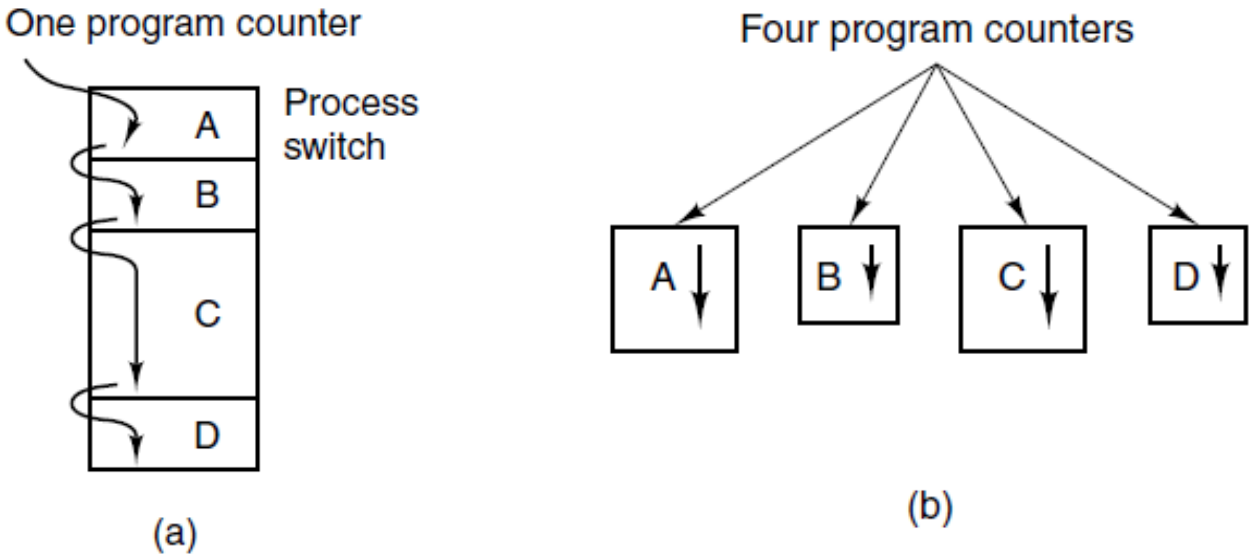


(c)

- In a single processor core system, only one program is active at once (pseudoparallelsim) [Figure 2-1(c) in Tanenbaum & Bos, 2014]

Logical Program Counter

- A processor has only one program counter register.
- Need to save and restore program counters



- Multiprogramming of 4 processes [Figure 2-1(a) and (b) in Tanenbaum & Bos, 2014]

Systems Perspective

- Process creation
- Process scheduling
- Process termination
- Interprocess communication

Process Creation

- 4 principal events cause process creation
  - Created at system initialization
  - Requested by a running process
  - Requested by a user
  - Initialized in a batch job
- Essentially, one method, i.e., via a process creation system call
  - fork(), CreateProcess(…)

System Initialization

- Numerous processes are created when the OS boots
  - Including many background processes (services, daemons)

Running Processes

- Often create processes to do the work
  - Example
    - One receives and stores emails
    - Another check if an email is a SPAM
  - Easier to divide the work into several related processes
  - Easier to take advantage of processors

User Request and Batch System

- Particularly in an iterative system, users can start a program
  - Enter a command
  - Double-click an icon
- Batch system
  - Run jobs in a job queue

Process Termination

- Typically a process is terminated due to one of the 4 conditions,
  - Normal exit (voluntarily)
  - Error exit (voluntarily)
  - Fatal error (involuntarily)
  - Killed by another process (involuntarily)

Normal Exit

- Upon completion its work, the compiler issues a system call
  - Example
    - Unix: exit(…), exit_group(…)
      - Need to differentiate _EXIT(2) and EXIT(3)
      - In Sample Programs
    - Windows: ExitProcess(…)

Error Exit

- The process discovers an error and quits
  - Developers programmed it in
  - The program issues the system call to exit
  - Example:
    - In Sample Programs

Fatal Error

- A process caused unrecoverable error
  - Example:
    - In Sample Programs
- In some systems (e.g., Unix), a process may inform the OS that it wishes to handle itself
  - Example
    - In Sample Programs

Terminated by Another Process

- A process can request to terminate another process
  - By issue a system call
  - Example
    - Unix: kill(…)
      - In Sample Programs
    - Win32: TerminateProcess(…)

Process Hierarchy

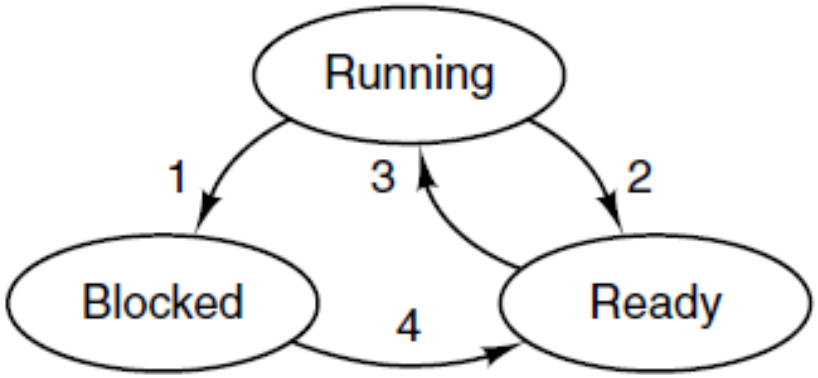
- In some systems, process and child process continued to be associated in certain ways
- Process group
  - In Unix, a process and all of its descendent form a group
    - Implication: when a user signals a process, the signal is delivered to all members of the process group
    - Unix cannot disown descendants
- In others, there may not be a hierarchy
  - In Windows, all processes are equal and hierarchy can be invalidated
    - Parent is given a handle (a special token, a data structure) to use to control the processes it created
    - But the token can be passed to other processes

Questions?

- Concept of process creation
- Concept of process termination

Process States

- A process can be one of a set of states
- Example



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

- Process states [Figure 2-2 in Tanenbaum & Bos, 2014]

Process States: Examples

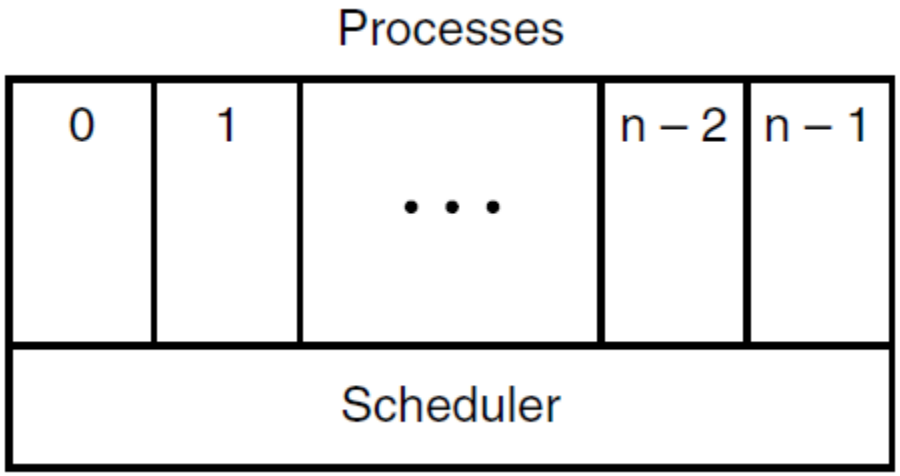
- Running
  - Actually using the CPU at the instant
- Ready
  - Runnable, temporarily stopped to let another process run
- Blocked
  - Unable to run until some external event happens

Process Cannot Run

- A process may be blocked due to a few conditions
  - Locally it cannot continue, typically because it is waiting for input and is not yet available
    - Example
      - In Unix, cat helloworld.txt | grep "Hello"
      - In Windows, more helloworld.txt | find "Hello"
  - The process is conceptually ready and able to run, but the OS has decided to allocate the CPU to another process

Scheduler

- OS scheduler handles interrupts, starts and stops processes



- Scheduler and processes [Figure 2-3 in Tanenbaum & Bos, 2014]

Scheduler Examples

- xv6:
  - https://github.com/mit-pdos/xv6-public/blob/master/proc.c#L323
- Linux scheduler
  - https://github.com/torvalds/linux/blob/master/kernel/sched/core.c#L3311

Questions

- Process states and scheduler?

Implementation: Process Table

- Process table
  - An entry per process
    - Commonly referred to as process control blocks
  - An array of the process entries

Process Table Entry: Examples

- Process table entry or process control block
- xv6: struct proc
  - https://github.com/mit-pdos/xv6-public/blob/master/proc.h#L38
- Linux: struct task_struct
  - https://github.com/torvalds/linux/blob/master/include/linux/sched.h#L524

Process Table Entry

- Common attributes

Process management	Memory management	File management
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

- Process table entry [Figure 2-4 in Tanenbaum & Bos, 2014]

Implementation: Handling Interrupts

- May be triggered by clocks, and others

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.

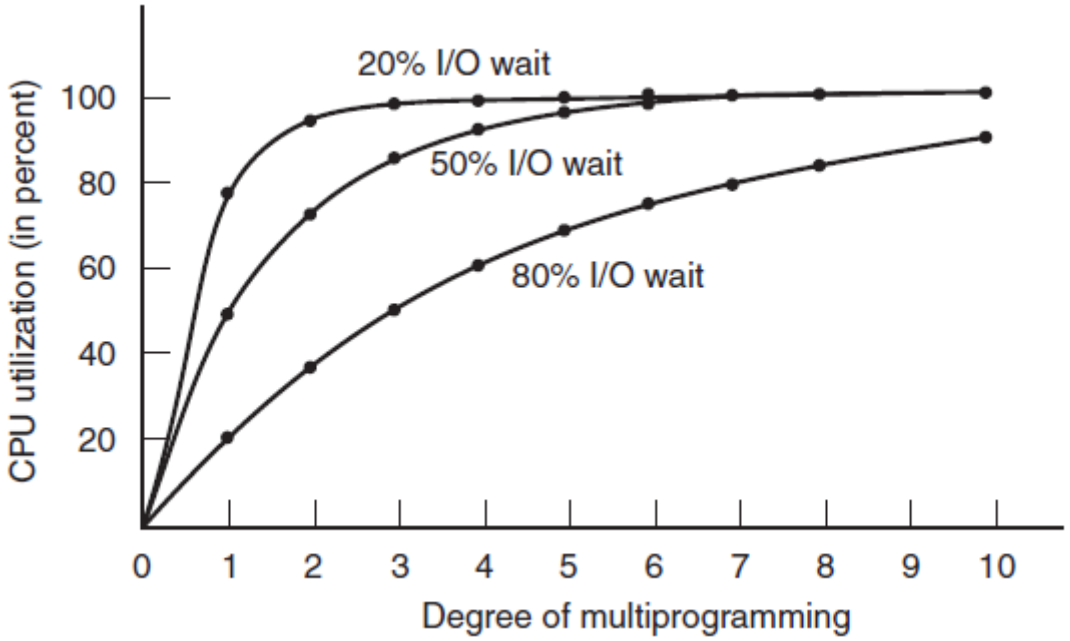
- Handling interrupts for processes [Figure 2-5 in Tanenbaum & Bos, 2014]

Benefit of Multiprogramming

- CPU utilization can be improved due to multiprogramming
- Intuition
  - When one process is waiting for I/O, another can be scheduled to CPU
- How much do we benefit?
- Future lesson
  - In more realistic scenarios, how much can we benefit more from a better scheduling system?

Modeling Multiprogramming

- CPU utilization



- CPU utilization [Figure 2-6 in Tanenbaum & Bos, 2014]

Simple Multiprogramming Model

- Assumptions
  - n processes in the main memory;
  - a process spends a fraction p of its waiting for I/O independent of the others
- Analysis
  - CPU is idle when all processes are waiting for I/O
  - The probability that all n processes waiting for I/O is p^n
  - CPU Utilization = $1 - p^n$

Questions

- Implementation
- Modeling?

Assignment

- Practice assignment in Blackboard