Lecture Topics

- Today: Processes
  (Stallings, chapter 3.1-3.6)
- Next: Threads
  (Stallings, chapter 4.1-4.2, 4.6)
Announcements

- Consulting hours  2-4PM, Rm 3353 Eng.
- Self-Study Exercise #1, #2, #3
- Project #1 (Due Wed, 24th May)
- Project #2 (Soon to be released, Due Wed, 31st May)
Processes

- All multiprogramming operating systems are built around the concept of the *process*

- **Main issues:**
  - different process states, transitions among them
  - process description (data structures maintained)
  - process control (switching between processes)
OS Requirements

- Interleave the execution of multiple processes to maximize processor utilization while providing reasonable response time
- Allocate resources to processes, based on policies and priorities
- Support inter-process communication
Background

- Computer platform consists of a collection of hardware resources
- OS provides a convenient, feature-rich, secure, and consistent interface for applications
- OS provides a uniform, abstract representation of resources that can be requested and accessed by applications
OS Manages Execution of Applications

- Resources made available to multiple applications
- Processor is switched among multiple applications
- The processor and I/O devices used efficiently
Process Definitions

- An instance of a program running on a computer
- The entity that can be assigned to and executed on a processor
- A unit of activity characterized by the execution of a sequence of instructions, a current state, and an associated set of system resources
Processes

- A process consists of (at least):
  - code and data for executing program
  - program counter (PC) and status register (PSW)
  - general-purpose registers
  - set of OS resources (open files, signals received, network connections, …)

- In other words, everything the OS needs to run the program -- or to re-start it, if it’s interrupted at some point
Process Elements

Process Control Block

- Identifier
- State
- Priority
- Program counter
- Memory pointers
- Context data
- I/O status info
- Accounting info
Process States

- Behavior of *process*: sequence of instructions executed by that process (trace)

- Behavior of *processor*: interleaved sequences of instructions from individual processes

- **Dispatcher** (part of OS) switches the processor from one process to another
Example

Assume no virtual memory: dispatcher and three processes are fully loaded into memory.

Program counter currently points to an address in Process B.
Example (cont)

<table>
<thead>
<tr>
<th>5000</th>
<th>8000</th>
<th>12000</th>
</tr>
</thead>
<tbody>
<tr>
<td>5001</td>
<td>8001</td>
<td>12001</td>
</tr>
<tr>
<td>5002</td>
<td>8002</td>
<td>12002</td>
</tr>
<tr>
<td>5003</td>
<td>8003</td>
<td>12003</td>
</tr>
<tr>
<td>5004</td>
<td></td>
<td>12004</td>
</tr>
<tr>
<td>5005</td>
<td></td>
<td>12005</td>
</tr>
<tr>
<td>5006</td>
<td></td>
<td>12006</td>
</tr>
<tr>
<td>5007</td>
<td></td>
<td>12007</td>
</tr>
<tr>
<td>5008</td>
<td></td>
<td>12008</td>
</tr>
<tr>
<td>5009</td>
<td></td>
<td>12009</td>
</tr>
<tr>
<td>5010</td>
<td></td>
<td>12010</td>
</tr>
<tr>
<td>5011</td>
<td></td>
<td>12011</td>
</tr>
</tbody>
</table>

(a) Trace of Process A  (b) Trace of Process B  (c) Trace of Process C

5000 = Starting address of program of Process A  
8000 = Starting address of program of Process B  
12000 = Starting address of program of Process C
Assume timer interrupt after 6 clock cycles

Assume dispatcher is at addresses 100-105

Assume dispatcher selects next process using "round robin"
Two-State Process Model

(a) State transition diagram

(b) Queuing diagram
Process Creation

- Submission of a batch job
- User logs onto system
- Created by OS to provide a service
  (ex: control print job)
- One process creates another process
  (ex: parent process spawns child process)
Process Termination

- Batch job terminates
- User logs off system
- Application halts normally
- Error (ex: segmentation fault)
- Parent process terminates child process
Execution patterns

Execution of a process is usually a pattern of CPU burst followed by I/O burst (repeated)
Execution patterns

- CPU-bound processes have few bursts, but long bursts
- I/O-bound processes have many bursts, but short bursts
- Pattern varies within a process – depends on actions within the process
Most processes: many short CPU bursts, few long CPU bursts
Refinement: Five-State Process Model
Previous Example

- Process A
- Process B
- Process C
- Dispatcher

- Running
- Ready
- Blocked
Process Switch

- Process currently in Running state
  - save context (PC, other registers)
  - move PCB to appropriate queue
  - update PCB (state, accounting info)

- Process selected by OS
  - update PCB (state, accounting info)
  - move PCB out of Ready queue
  - restore context (PC, registers)
Queueing Models

(a) Single blocked queue
Queueing Models

(b) Multiple blocked queues
Process Control Block

- PCB assigned to process when created
- OS maintains info in PCB while process is in existence
- OS moves PCBs between queues to manage system
Suspended Processes

- Processor is faster than I/O devices, so all active processes could be waiting for I/O

- Swap some processes to disk to free up more memory (activate other processes)

- Two new states:
  - Blocked/Suspend
  - Ready/Suspend
Refinement: Seven-State Process Model
Reasons for Process Suspension

- Swapping: OS needs to free up memory
- Interactive user request (ex: debugging)
- Timing (ex: periodic process)
- Parent process request (ex: suspend child)
- Other OS reasons (ex: misbehaving process)
Operating System Control Structures

- Information about the current status of each process and resource
- Table for each resource the OS manages
- Tables in kernel memory
Process Table

- List of processes currently in the system
- Table entry contains pointer to *process image*
  - user text segment (machine language)
  - user data segment (static data objects)
  - user stack segment (dynamic data objects)
Typical PCB Contents

- **Process Identifiers**
  - ID of this process
  - ID of the parent process
  - User ID

- **Processor State Information**
  - User-visible registers
  - Control and status registers

- **Process Control Information**
  - Supplemental info maintained by OS
Processor State Information

- All information needed to resume execution of the process
- Contents of user-visible registers
- Contents of control and status registers
  - program counter
  - program state word (PSW)
  - condition codes (if present)
PSW: x86 EFLAGS Register

<table>
<thead>
<tr>
<th></th>
<th>31</th>
<th>21</th>
<th>16</th>
<th>15</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>ID</td>
<td>VIP</td>
<td>VIP</td>
<td>ACM</td>
<td>VF</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IO</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ODF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ITSF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ZSZ</td>
</tr>
<tr>
<td>A</td>
<td>AF</td>
<td>PF</td>
<td>CF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- ID = Identification flag
- VIP = Virtual interrupt pending
- VIF = Virtual interrupt flag
- AC = Alignment check
- VM = Virtual 8086 mode
- RF = Resume flag
- NT = Nested task flag
- IOPL = I/O privilege level
- OF = Overflow flag
- DF = Direction flag
- IF = Interrupt enable flag
- TF = Trap flag
- SF = Sign flag
- ZF = Zero flag
- AF = Auxiliary carry flag
- PF = Parity flag
- CF = Carry flag
PSW: SPARC Processor Status Register

<table>
<thead>
<tr>
<th>Impl</th>
<th>Ver</th>
<th>ICC</th>
<th>Reserved</th>
<th>EC</th>
<th>EF</th>
<th>PIL</th>
<th>S</th>
<th>PS</th>
<th>ET</th>
<th>CWP</th>
</tr>
</thead>
</table>

PIL: processor interrupt level
S: supervisor mode? (vs. user mode)
PS: previous S bit
ET: traps enabled?
Process Control Information

- Scheduling and state information
  - Process state: running, ready, blocked, etc.
  - Priority: one or more fields to describe the scheduling priority of the process.
  - Scheduling-related information: depends on the specific scheduling algorithm used (ex: the amount of time that the process has been waiting and the amount of time that the process executed the last time it was running).
  - Event: event the process is awaiting
Process Control Information

- Inter-process Communication: flags, signals, and messages may be associated with communication between two independent processes.
- Process Privileges: access restrictions for process
- Memory Management: pointers to page tables that describe the virtual memory assigned to process.
- Resource Ownership and Utilization: resources controlled by process
Memory Tables

- Allocation of main memory to processes
- Allocation of secondary memory to processes
- Protection attributes for access to shared memory regions
- Information needed to manage virtual memory
I/O Tables

- I/O device is available or assigned
- Status of I/O operation
- Location in main memory being used as the source or destination of the I/O transfer
File Tables

- Existence of files
- Location on secondary memory
- Current status
- Attributes

- Sometimes this information is maintained by a file management system
Modes of Execution

- **User mode**
  - Less-privileged mode
  - User programs execute in this mode

- **Kernel mode (control mode, system mode)**
  - More-privileged mode
  - OS kernel operates in this mode

- Bit in status register specifies current mode
Mode Switch

- User mode to kernel mode
  - interrupt: during execution of current instruction; reaction to asynchronous external event
  - trap: caused by execution of current instruction; reaction to exceptional condition
  - system call: explicit request by current instruction; request service from OS

- Kernel mode to user mode
Process Switch

- Process currently in Running state
  - save context (PC, other registers)
  - move PCB to appropriate queue
  - update PCB (state, accounting info)

- Process selected by OS
  - update PCB (state, accounting info)
  - move PCB out of Ready queue
  - restore context (PC, registers)
Mode and Process Switch

Approximate timing on "arctic.cse.msu.edu":

Mode switch: 51-58 ns
Process switch: 2200-2800 ns

Approximate timing on "cse410.cse.msu.edu":

Mode switch: 235-240 ns
Process switch: 5500-6000 ns
Execution of the Operating System

- Non-process Kernel
  - Kernel executes as a separate entity in privileged mode (not a process)
  - Typical of older batch-oriented operating systems
Execution of the Operating System

- Execution Within User Processes
  - Kernel executes within context of a user process
  - Process executes in privileged mode when executing operating system code
  - Process switching code is separate

![Diagram showing execution within user processes and process switching functions]
Execution of the Operating System

- Process-Based Operating System
  - Implement OS as a collection of system processes
  - Useful in multi-processor or multi-computer environment
  - Process switching code is separate
UNIX Process Management

- Two categories of processes
  - System processes
    - run in kernel mode for housekeeping functions (memory allocation, process swapping, etc)
  - User processes
    - run in user mode for user programs
    - run in kernel mode for system calls, traps, interrupts
UNIX Process Management

1. **Created**
   - fork
   - not enough memory (swapping system only)
   - enough memory

2. **Ready to Run In Memory**
   - swap in
   - swap out
   - wake up

3. **Sleep, Swapped**
   - swap out
   - wake up

4. **Asleep in Memory**
   - swap out
   - wake up

5. **Kernel Running**
   - interrupt, return
   - exit
   - sleep
   - reschedule process
   - preempt
   - return to user
   - system call, interrupt

6. **Zombie**
   - interrupt, interrupt return
   - return to user

7. **Preempted**
   - return to user
## UNIX Process States

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Running</td>
<td>Executing in user mode.</td>
</tr>
<tr>
<td>Kernel Running</td>
<td>Executing in kernel mode.</td>
</tr>
<tr>
<td>Ready to Run, in Memory</td>
<td>Ready to run as soon as the kernel schedules it.</td>
</tr>
<tr>
<td>Asleep in Memory</td>
<td>Unable to execute until an event occurs; process is in main memory (a blocked state).</td>
</tr>
<tr>
<td>Ready to Run, Swapped</td>
<td>Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.</td>
</tr>
<tr>
<td>Sleeping, Swapped</td>
<td>The process is awaiting an event and has been swapped to secondary storage (a blocked state).</td>
</tr>
<tr>
<td>Preempted</td>
<td>Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.</td>
</tr>
<tr>
<td>Created</td>
<td>Process is newly created and not yet ready to run.</td>
</tr>
<tr>
<td>Zombie</td>
<td>Process no longer exists, but it leaves a record for its parent process to collect.</td>
</tr>
</tbody>
</table>
UNIX Process States

- Similar to seven state model
- Two running states (User and Kernel): transitions to other states (blocked, ready) must come from Kernel Running
- Sleeping states (in memory/ swapped) correspond to blocked states
- Preempted state is distinguished from Ready state (but they form one queue)
- Preemption can occur only when a process is about to move from kernel to user mode
### UNIX Process Image

#### User-Level Context
- **Process Text**: Executable machine instructions of the program
- **Process Data**: Data accessible by the program of this process
- **User Stack**: Contains the arguments, local variables, and pointers for functions executing in user mode
- **Shared Memory**: Memory shared with other processes, used for interprocess communication

#### Register Context
- **Program Counter**: Address of next instruction to be executed; may be in kernel or user memory space of this process
- **Processor Status Register**: Contains the hardware status at the time of preemption; contents and format are hardware dependent
- **Stack Pointer**: Points to the top of the kernel or user stack, depending on the mode of operation at the time of preemption
- **General-Purpose Registers**: Hardware dependent

#### System-Level Context
- **Process Table Entry**: Defines state of a process; this information is always accessible to the operating system
- **U (user) Area**: Process control information that needs to be accessed only in the context of the process
- **Per Process Region Table**: Defines the mapping from virtual to physical addresses; also contains a permission field that indicates the type of access allowed the process: read-only, read-write, or read-execute
- **Kernel Stack**: Contains the stack frame of kernel procedures as the process executes in kernel mode
UNIX Process Creation

- Every process (except process 0) is created by the fork() system call
  - fork() allocates entry in process table and assigns a unique PID to the child process
  - child gets a copy of the process image of the parent; both child and parent are executing the same code after fork()
  - fork() returns the PID of the child process to the parent process and returns 0 to the child process
UNIX System Processes

- Process 0 is created at boot time and becomes the "swapper" process after forking process 1 (the "systemd" process) and process 2 (the "kthreadd" process).

- The "kthreadd" process is the ancestor of all kernel processes.

- The "systemd" process is the ancestor of all user processes.
Examples

- Use of "ps" command
- Use of "fork" and related functions
Self Study

- “man proc”
- Login into your cse410 account, go to “/proc” i.e. cd /proc”, use “man proc” as a guide into the actual components of a process. It is strongly recommended that you don’t try to overwrite data within any given process.
- [https://www.bottomuppcs.com/elements_of_a_process.xhtml](https://www.bottomuppcs.com/elements_of_a_process.xhtml)
- [https://en.wikipedia.org/wiki/Call_stack](https://en.wikipedia.org/wiki/Call_stack)