CMSC 321: Operating Systems

Lecture 3

Processes
Operating System Services

- What services must an OS provide?
  - program development
  - program execution
  - access to I/O
  - controlled access to files
  - system access (shared)
  - error detection & response
  - accounting (stats)
Hardware support for OS Features

How many programs can I run?

• The beginning: single program gets the whole computer – hand loaded
• Batch systems
• Multiprogramming
  – non-preemptive
  – time-sliced preemptive
• Interactive timesharing
• Scheduling needed because a preemptive multiprogramming system allows programs to be interrupted (suspended) and restarted at any time.
Hardware support for OS Features

The Interrupt Mechanism

- Interrupts occur due to hardware events (completion of disk I/O, etc.) or software events (expiration of a timer, reception of a message, etc.).
- Instruction cycle is modified to include a check for pending interrupts after each instruction execution.
Hardware support for OS Features

The Interrupt Mechanism

- *Interrupt vector* - an area of system memory containing a pointer to the interrupt handler for each type of interrupt.
- Interrupt handler first preserves processor state for the suspended process, then performs actions necessary to service the interrupt.
- Different type of interrupts may have different priorities: interrupt handlers may themselves be interruptable!
Peripheral Handling: Direct Memory Access

- High speed peripheral devices have the potential to swamp the CPU with interrupts.
- Transfers to or from such devices can be grouped into blocks.
- The device driver (OS component that services the device) provides the memory addresses of the start/end of the memory area the device reads from/writes to, then gives the device permission to directly access that range of locations without further CPU involvement (hence Direct Memory Access or DMA).
Hardware support for OS Features

Peripheral Handling: Direct Memory Access

• All the work to complete the transfer occurs between the device and the memory controller.

• Cycle stealing - since the memory controller can handle only one request at a time, some memory requests from the CPU may be delayed by requests from the device performing DMA.
Hardware support for OS Features

• Dual mode operation
  – Monitor mode for execution of OS (also called Supervisor mode or Kernel mode)
  – User mode for execution of user programs

• Time slicing - prevents a "runaway" process from dominating CPU usage. Hardware timer needed to implement this feature.
Hardware support for OS Features

Memory protection

- CPU provides Base Register and Limit Register.
- When a process runs on the CPU, these registers contain the addresses of the range of memory locations that the process is allowed to access.
- All memory references are compared against the hardware registers.
- References outside the allowed range cause a trap. (Segmentation fault, in Unix.)
Process

• Most fundamental concept in OS

• Process: a program in execution
  – one or more threads (units of work)
  – associated system resources

• Program vs. process
  – program: a passive entity
  – process: an active entity

• For a program to execute, a process is created for that program
Process Management

• Fundamental task of any OS

• For processes, OS must:
  – allocate resources
  – facilitate *multiprogramming*
  – allow sharing and exchange of info
  – protect resources from other processes
  – enable synchronization

• How?
  – data structure for each process (coming soon...)
  – describes *state* and resource ownership
A Simple Two-State Model

• What are the two simplest states of a process?
  – Running
  – Not running

• When a new process created: “not running”
  – memory allocated, enters waiting queue

• Eventually, a “running” process is interrupted
  – state is set to “not running”

• Dispatcher chooses another from queue
  – state is set to “running” and it executes
The Two-State Model

(a) State transition diagram

(b) Queuing diagram
Two States: Not Enough

• Running process makes I/O syscall
  – moved to “not running” state
  – can’t be selected until I/O is complete!

• “not running” should be two states:
  – blocked: waiting for something, can’t be selected
  – ready: just itching for CPU time...

• Five states total
  – running, blocked, ready
  – new: OS might not yet admit (e.g., performance)
  – exiting: halted or aborted
    • perhaps other programs want to examine tables & DS
The Five-State Model
Process State Diagram

- Process A
- Process B
- Process C

Legend:
- ■ = Running
- □ = Ready
- • = Blocked
Problem: processes blocked on different events must search entire queue

Solution: one queue for each type of event
Queuing Diagram

Event 1 Queue

Event 2 Queue

Event n Queue

Timeout

Event 1 Wait

Event 2 Wait

Event n Wait

Multiple blocked queues
Are Five States Enough?

• Problem: Can’t have *all* processes in RAM

• Solution: swap some to disk
  – i.e., move all or part of a process to disk

• Requires new state: *suspend*
  – on disk, therefore not available to CPU
Six-State Model

(a) With One Suspend State
Process Description...

• What info does OS need:
  - to control processes?
  - to manage their resources?

• OS uses tables about managed entities
  - memory tables
  - I/O tables
  - file tables
  - processor tables
Memory → Memory Tables
Devices → I/O Tables
Files → File Tables
Processes → Primary Process Table

Primary Process Table:
- Process 1
- Process 2
- Process 3

Process 1 → Process Image

Process 2 → Process Image

Process 3 → Process Image

Process n → Process Image
Process Image

- **Must know:**
  - where process is located
  - attributes for managing

- *Process image*: physical manifestation of process
  - program(s) to be executed
  - data locations for vars and constants
  - stack for procedure calls and parameter passing
  - PCB: info used by OS to manage
Process Image

- At least small portion must stay in RAM
- To execute, entire image must be in RAM
PCB: Process Control Block

- AKA process table
- Most important data structure in OS
- Contains info about the process

- Collectively, PCBs define state of OS

- 3 categories of PCB info:
  - process identification
  - processor state info
  - process control info

- Must protect PCB! (use handler to modify)
PCB (Process Table)

1. Process identification
   - unique number: index into process table
   - user identifier (perhaps)

2. Processor state info
   - contents of processor *registers*
   - save register values when interrupted for restoring *exactly*

3. Process Control Info
   - scheduling and state info (process state, priority, event waiting for)
   - resources, privileges
Register Sets

- **User-visible registers:**
  - for compiler optimization
  - in C, you can suggest the use of registers!

- **Control, status registers:**
  - e.g., PC, IR

- **Stack pointers, RA**

- **PSW (Program [Process] Status Word):**
  - contains status info
  - e.g., bit for interrupts enabled/disabled
    - bit for user/kernel mode
    - bit to signify arithmetic overflow
Pentium 4 Registers

- Four general-purpose registers: EAX, EBX, ECX, EDX.
  - EAX is the main arithmetic registers.
  - EDX is needed for multiplication/division.
    - EAX and EDX hold 64-bit products/dividends.
  - Each register holds 16-bit register and 8-bit registers.
    - Compatibility with 8088 and 80286.

- Special-purpose registers.
  - ESI and EDI: string manipulation instructions (source and destination).
  - EBP: points to base of current stack frame (frame pointer).
  - ESP: points to top of stack (stack pointer).
  - EIP: program counter.
  - EFLAGS: program status word.

- Segment registers: CS, SS, DS, ES, FS, GS.
Pentium II PSW

EFLAGS Register

- 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
Process Control...

- Two modes of execution for processor
  - *user* mode
  - *kernel* mode

- Privileged instructions in kernel mode
  - e.g., read/modify PSW, primitive I/O, memory management, ...
  - syscalls

- How? bit in PSW
  - processor changes bit when syscall is made
Process Creation

1. Assign a new process ID
   • new entry in process table

2. Allocate space for process image
   • space for PCB
   • space for address space and user stack

3. Initialize PCB
   • ID of process, parent
   • PC set to program entry point
   • typically, “ready” state

4. Linkages and other DS
   • place image in list/queue
   • accounting DS
Switching Processes

• When to switch: interrupt or trap or syscall

• interrupt: due to event external to process
  – first, generic interrupt handler (housekeeping)
  – then, passed to interrupt-specific routine
  – examples:
    • clock: for time slicing
    • I/O: move processes blocked on event to “ready”
    • memory fault: requested addy not in RAM

• trap: error or exception in executing process

• system call: request by process for OS routine
  – e.g., open a file, read from a device
  – generally, user process becomes “blocked”
How is CPU aware of interrupts?

- interrupt cycle included as part of fetch/execute

- check if interrupt pending before fetching

- if pending:
  - save context of current process
  - set PC to addr of interrupt handler (IH) routine
  - switch from user to kernel mode
    (IH can access privileged instructions)
What *Context* Should be Saved?

- Any info that may be altered by IH
  - processor state info: PC, processor regs, stack info

- What about other info in PCB?
  - typically, IH performs only a few basic tasks
    - e.g., reset bit in PSW, notify signalling entity
    - currently running process usually resumes

- So, interrupt does not necessarily mean process (context) switch
  - handling interrupt can be referred to as *mode switching*
Process Switch (Context Switch)

1. Save processor context (PC, other regs)
2. Update PCB: change of state, accounting
3. Put process in appropriate queue (ready, blocked, etc.)
4. Select another process
5. Update PCB of new process
6. Update memory management DS
   (may need to bring part of image from disk)
7. Restore processor context
   (reflect state @ time new process was last switched out)
Context Switching

- Requires more effort than a mode switch

- Pure overhead!
  - no useful work can be done during
  - use threads when possible
Current Assignments

- Project 1 due Thursday
- Read Ch. 1, Ch. 2 (up to 2.2)

Next lecture:
- threads
- intro to Nachos

Lab 2: intro to Valgrind