Computer Systems Lecture 20

Protecting System Resources

The OS must guarantee safe and orderly access to critical system resources

* Hardware (processor, networking, I/O)
* Program memory (including page tables)

The OS is the ultimate arbiter of what’s allowed

* TLB miss -> Ok (but must access page table to service)
* Arithmetic overflow -> may be OK (depends on what we’re doing)
* Illegal opcode -> not OK (kill the program)

Exceptions are used to hand control over to the OS

* We need a separate mechanism to limit capabilities of user programs

Kernel vs User Mode Protection

Exceptions (including system calls) are handled by the OS

* The CPU has two modes of operation: user and kernel (OS)
* The current mode is identified by a bit in a special status register

There are a bunch of privileged instructions that can only be executed in kernel mode

* For example accessing I/O devices, handling page table accesses and TLB updates, halt or reset the processor or change its voltage ect…

Kernel mode can only be entered through an exception

* User programs cannot jump to OS instruction space

The eret instruction sets the mode back to what it previously was.

Advantages of Dual Mode Architecture

* Dual mode architecture guarantees that control is transferred to the OS when a user program attempts to perform potentially dangerous tasks.
* It allows the OS to ensure that programs do not interfere with each other
  + E.g. each program is able to get its share of physical memory
* Allows the OS to ensure that programs don’t have access to resources for which they don’t have permission
  + E.g. files or another program’s memory
* Ensures that user programs don’t have indefinite control of the processor
  + Time-sharing of the CPU

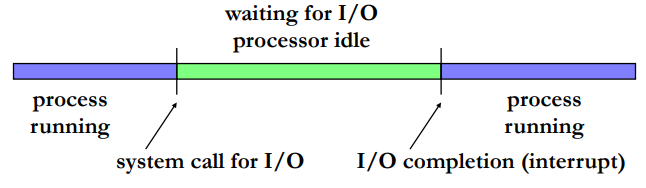
Time-Sharing the CPU

* Problem:
  + I/O takes too long -> processor idle
  + User programs can crash or monopolize the CPU (either unintentionally or maliciously)
* Solution:
  + Multiplex or time-share the CPU and other resources among several user processes
  + Switch from on process to another when it performs I/O or when its time allocation (time slice) expires

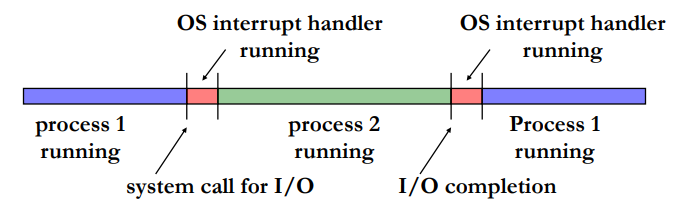
Process: “a program in execution”

Multi-Tasking

Single-task system:



Multi-tasking system:



Managing Processes

Processes are manged by the OS kernel

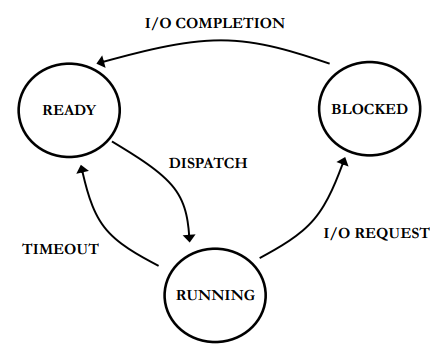
* Kernel: the core of the operating system that controls all software and hardware resources
  + First to be loaded when the computer boots
  + Manages interrupts, processes, memory, I/O
* The kernel’s scheduler chooses which process to run next from the pool of active processes

New processes can be explicitly created by the user, or implicitly by another process (through forking)

* Original process -> parent
* New process -> child

Process States

States:

* Running: process is currently running in the CPU
* Ready: process is not running, but could run if brought into CPU
* Blocked: process is not able to run because it is waiting for I/O to finish

Transition:

* I/O request: process initiates I/O
* I/O completion I/O finishes
* Dispatch: OS moves process into CPU and it starts executing
* Timeout: process’s time slice is over

Step 1: the process calls (or traps into) the OS, or interrupt occurs (e.g. because of timer)

Step 2: OS’s dispatcher performs context switch:

* Process’s context is saved (registers, PC, etc..) in process control block (PCB)
* Dispatcher chooses new process to run
* Processes’ states are updated

PCD: OS data structure containing each process’s information:

* Process id (PID)
* Process state (blocked running, etc)
* Process priority
* Process permissions
* Etc…

Suspending and Resuming Processes

Problem:

* Might not have enough physical memory for all processes
* Some processes have higher priority and must get more processor & memory resources (e.g. a high-res game)

Solution:

* Processes can be “swapped out” from memory to disk
* Such processes are moved into an “inactive” state
  + 2 new process states
* PCB of inactive processes are still kept in the OS’s memory
* Inactive processes are resumed by “swapping in” the data from disk back to memory

Suspending and Resuming Processes

