Processes & Threads

CS 256/456

Dept. of Computer Science, University of Rochester

Recap of the Last Class

- Hardware protection
  - kernel and user mode
- System components
  - process management, memory management, I/O system, file and storage, networking, …
- Operating system architectures
  - monolithic architecture, microkernel
  - … …

Microkernel System Architecture

- Microkernel architecture:
  - Moves as much from the kernel into "*user*" space (still protected from normal users).
  - Communication takes place between user modules using message passing.
- Benefits:
  - More reliable (less code is running in kernel mode)
  - More secure (less code is running in kernel mode)
- Disadvantage:
  - More overhead in inter-domain communications

Layered Structure

- Layered structure
  - The operating system is divided into a number of layers (levels), each built on top of lower layers.
  - The bottom layer (layer 0), is the hardware.
  - The highest (layer N) is the user interface.
  - Decreased privileges for higher layers.
- Benefits:
  - more reliable
  - more secure
- Disadvantage:
  - Weak integration results in performance penalty (similar to the microkernel structure).

Outline

- Process
  - Process concept
  - A process's image in a computer
  - Operations on processes
- Thread
  - Thread concept
  - Multithreading models
  - Types of threads

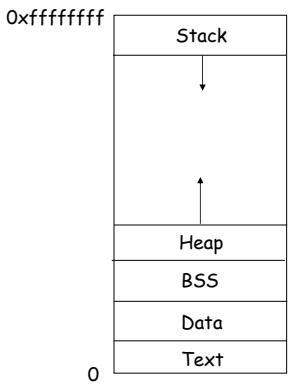
Process and Its Image

- An operating system executes a variety of programs:
  - A program that browses the Web
  - A program that serves Web requests
- Process – a program in execution.
- A process's state/image in a computer includes:
  - User-mode address space
  - Kernel data structure
  - Registers (including program counter and stack pointer)
- Address space and memory protection
  - Physical memory is divided into user memory and kernel memory
  - Kernel memory can only be accessed when in the kernel mode
  - Each process has its own exclusive address space in the user-mode memory space (sort-of)

User-mode Address Space

User-mode address space for a process:

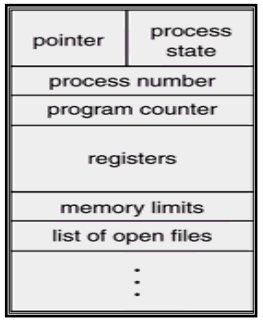
- Text: program code, instructions
- Data: initialized global and static variables (those data whose size is known before the execution)
- BSS (block started by symbol): uninitialized global and static variables
- Heap: dynamic memory (those being malloc-ed)
- Stack: local variables and other stuff for function invocations



Process Control Block (PCB)

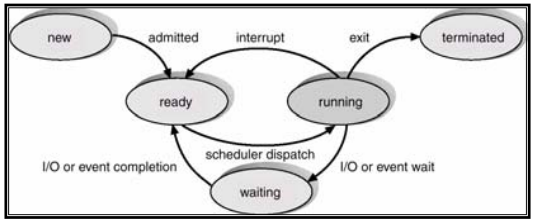
OS data structure (in kernel memory) maintaining information associated with each process.

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- Information about open files
- maybe kernel stack?



Process State

- As a process executes, it changes *state*
  - **new**: The process is being created.
  - **ready**: The process is waiting to be assigned to a process.
  - **running**: Instructions are being executed.
  - **waiting**: The process is waiting for some event to occur.
  - **terminated**: The process has finished execution.

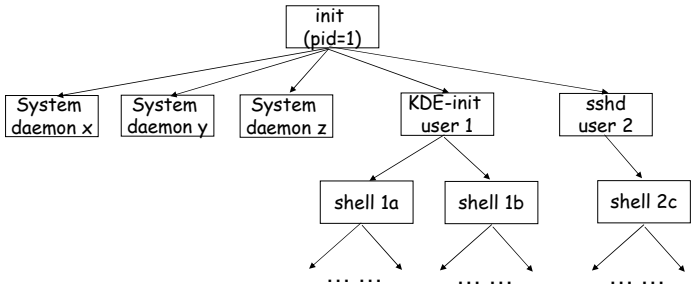


Process Creation

- When a process (parent) creates a new process (child)
  - Execution sequence?
  - Address space sharing?
  - Open files inheritance?
  - … …
- UNIX examples
  - **fork** system call creates new process with a duplicated copy of everything.
  - **exec** system call used after a **fork** to replace the process' memory space with a new program.
  - child and parent compete CPU like two normal processes.
- Copy-on-write

Process Tree on a Linux System

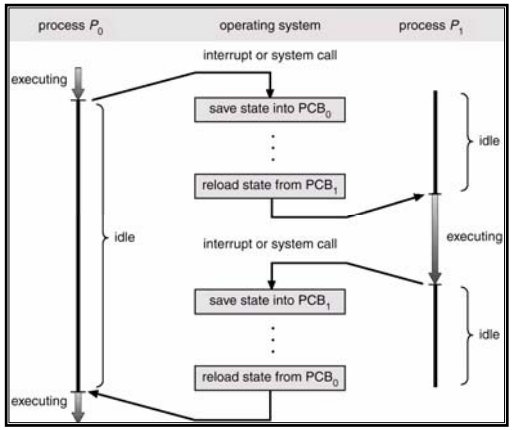
- Parent process creates children processes, which, in turn create other processes, forming a tree of processes.



CPU Switch From Process to Process

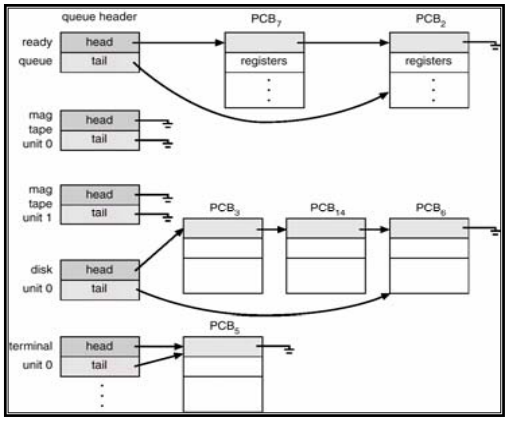
When can the OS switch the CPU from one process to another?

Which one to switch to? - scheduling



Queues for PCBs

- Ready queue – set of all processes ready for execution.
- Device queues – set of processes waiting for an I/O device.
- Process migration between the various queues.

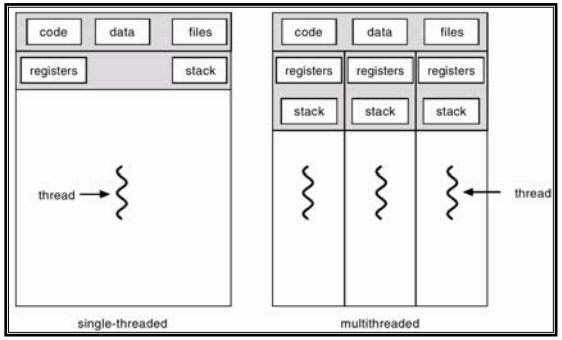


Process Termination

- Process executes last statement and gives the control to the OS (**exit**).
  - Notify parent if it is **wait**-ing.
  - Deallocate process' resources.
- The OS may forcefully terminate a process.
  - Software exceptions
  - Receiving certain signals

Processes and Threads

- Thread – a program in execution; without a dedicated address space.
- OS memory protection is only applied to processes.



Why Use Threads?

- Multithreading is used for parallelism/concurrency. But why not multiple processes?
- Memory sharing.
- Efficient synchronization between threads
- Less context switch overhead

User/Kernel Threads

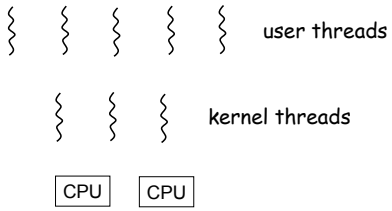
- What is really happening when the execution of one thread switches to another?
  - save the registers (including the SP) of the old thread;
  - restore the registers of the new thread;
  - set the new PC appropriately for the new thread;
- Does this need help from the OS kernel?
- Can you do it in C/Java?
- Does process switching need help from the OS kernel?
- User threads
  - Thread data structure is in user-mode memory
  - scheduling/switching done at user mode
- Kernel threads
  - Thread data structure is in kernel memory
  - scheduling/switching done by the OS kernel

User/Kernel Threads (cont.)

- Benefits of user threads
  - lightweight – less context switching overhead
  - more efficient synchronization??
  - flexibility – allow application-controlled scheduling
- Problems of user threads
  - can't use more than one processor
  - oblivious to kernel events, e.g., all threads in a process are put to wait when only one of them does I/O

Mixed User/Kernel Threads

- M user threads run on N kernel threads (M≥N)
  - N=1: pure user threads
  - M=N: pure kernel threads
  - M>N>1: mixed model



Solaris/Linux Threads

- Solaris
  - supports mixed model
- Linux
  - No standard user threads on Linux
  - Processes are treated similarly with threads (both called tasks)
  - Processes are tasks with exclusive address space
  - Tasks can also share the address space, open files, …

Pthreads

- Different OS has its own thread package with different Application Programming Interfaces ⇒ poor portability.
- Pthreads
  - A POSIX standard API for thread management and synchronization.
  - API specifies behavior of the thread library, not the implementation.
  - Commonly supported in UNIX operating systems.

Disclaimer

- Parts of the lecture slides contain original work of Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Andrew S. Tanenbaum, and Gary Nutt. The slides are intended for the sole purpose of instruction of operating systems at the University of Rochester. All copyrighted materials belong to their original owner(s).