Lecture 06
프로세스와 스레드

Processes

- A process is an instance of a computer program that is being executed
  - A stream of instructions being executed
  - Abstraction used by the operating system

- A process consists of:
  - Registers
  - Memory (code, data, stack, heap, etc.)
  - I/O status (open file tables, etc.)
  - Signal management information

Supervisor Mode vs. User Mode

- Modern processors provides two different modes of execution:
  - Supervisor (kernel) mode
    - All instructions can be executed in supervisor mode (also known as protected mode, system mode, monitor mode, or privileged mode)
    - For the operating system kernel
  - User mode
    - The processor is allowed to execute only a subset of the instructions
    - For all other software (including the remaining part of the operating system) than the kernel
- Example:
  - I/O instructions are privileged instructions
    - An application needs to request an I/O service to the operating system to perform I/O operations

System Calls

- A system call is the way how a program running in user mode requests a service to the operating system
  - Typically implemented with a trap (a.k.a. an exception or a fault)
  - A trap instruction invoked by the program triggers a trap, resulting in a switch to kernel mode
  - The kernel performs some action to handle the trap before returning control to the program

Uniprogramming vs. Multiprogramming

- Uniprogramming
  - Only one process at a time
    - DOS
  - Poor resource utilization

- Multiprogramming
  - Multiple processes at a time
    - Modern operating systems, such as Windows, Unix, Linux, etc.
  - Increases resource utilization

Virtual Memory

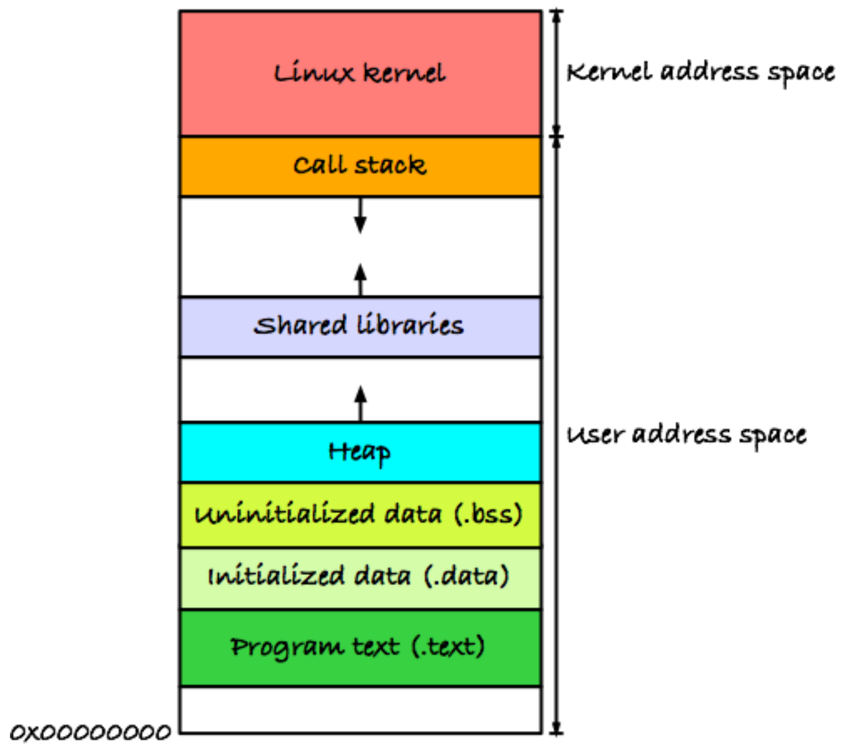
- The operating system's abstraction of the physical memory in the system
- Provides each process with the illusion that the process has exclusive use of the memory and a much larger memory space than that available in the system

Virtual Memory (contd.)

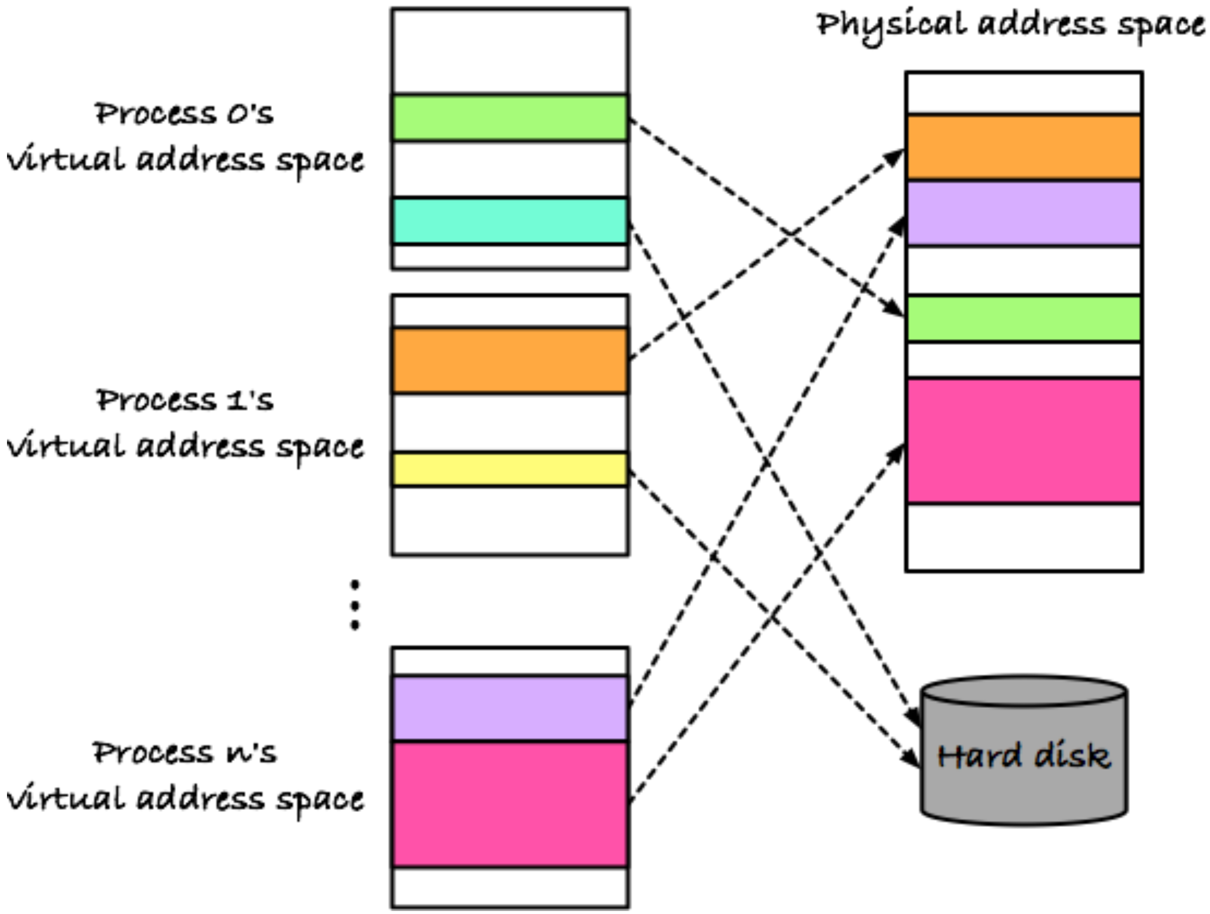
- Logical (virtual) address
  - An address generated by the CPU
  - Logical address space - the set of all logical (virtual) addresses generated by a program
- Physical address
  - An address seen by the physical memory
  - Physical address space - the set of all physical addresses corresponding to the logical addresses
- The virtual memory in the operating system is in charge of the run-time mapping from virtual to physical addresses
  - Exploits a hardware device called the memory-management unit (MMU) for fast translation between virtual and physical addresses

Address Space of a Process

- A process has its own private address space (virtual address space)
  - A process cannot affect the state of another process directly
  - Memory protection
- Kernel address space vs. user address space



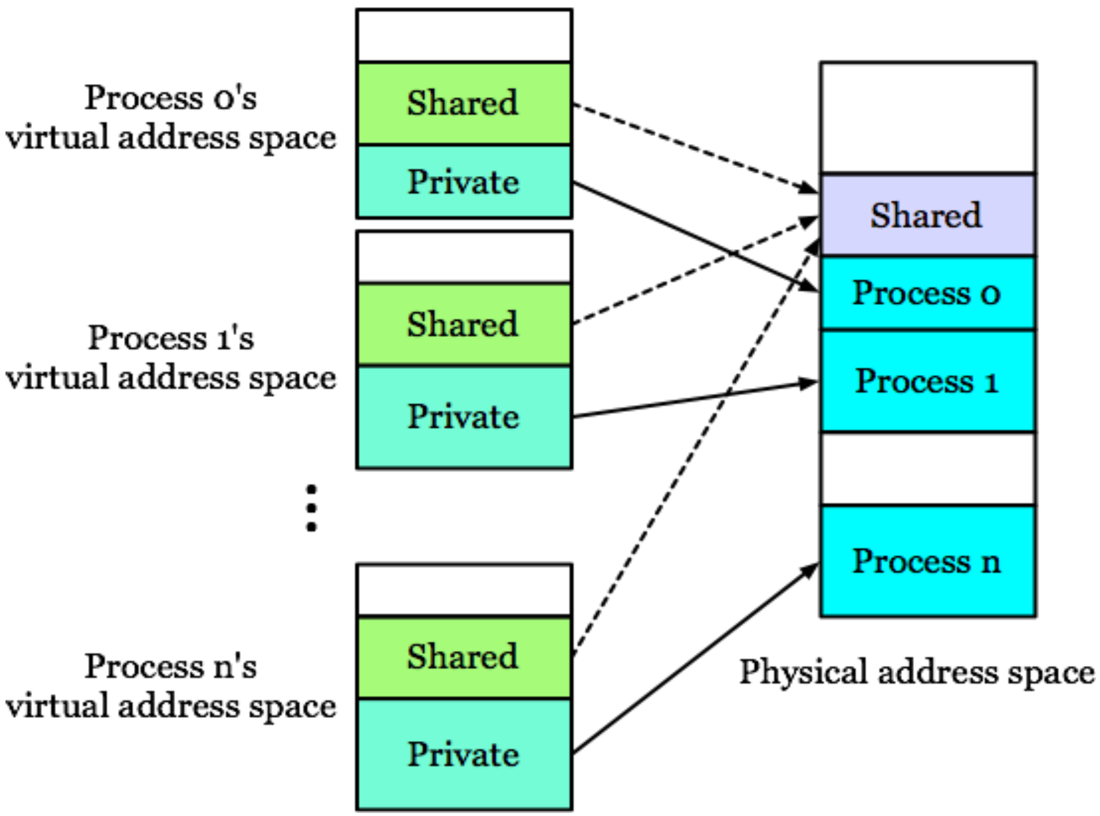
Address Space of a Process (contd.)



Communication Between Processes

- Cooperation and coordination between processes are accomplished by writing and reading to a location in the shared address space
- Another way to achieve them is using an interprocess communication (IPC) mechanism
  - The IPC is a way of exchanging data between processes without sharing any portion of their virtual address space
  - Expensive

Communication Between Processes (contd.)



Concurrency

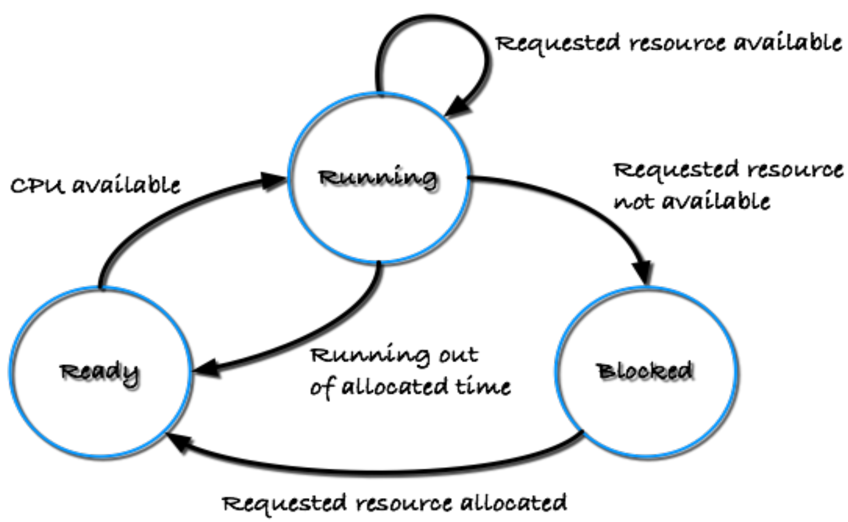
- A computer system is typically a multiprogramming system
  - Kernel processes execute system code
  - User processes execute user code
  - All these processes may execute concurrently on the same system
- Concurrency
  - Instructions of one process are interleaved with those of another process
  - Implemented by context switches

Context Switch

- The CPU switches back and forth from process to process to execute the instructions from different processes
  - The CPU scheduler in the operating system is in charge of it
- A process context is the information that must be saved before a context switch occurs to allow the continuation of the process later
- The operating system transfers control from the current process (say, p) to another process (say, q) after saving the context of p and restoring the context of q
- Then, control is passed to the location of q 's code where it left off due to a previous context switch

Process State Transitions

- When a program runs, the corresponding process changes state
  - Running: using the CPU
  - Ready: no CPU available
  - Blocked: waiting for some event (e.g., I/O) to occur



Preemptive vs. Cooperative

- Preemptive multitasking
  - Permits preemption of tasks
  - All processes will get some amount of CPU time at any given time
  - More reliably guarantee each process a regular slice of operating time
  - Nearly all modern operating systems support preemptive multitasking
- Cooperative multitasking
  - Tasks must be explicitly programmed to yield when they do not need system resources (e.g., CPU)
  - Rarely used in these days

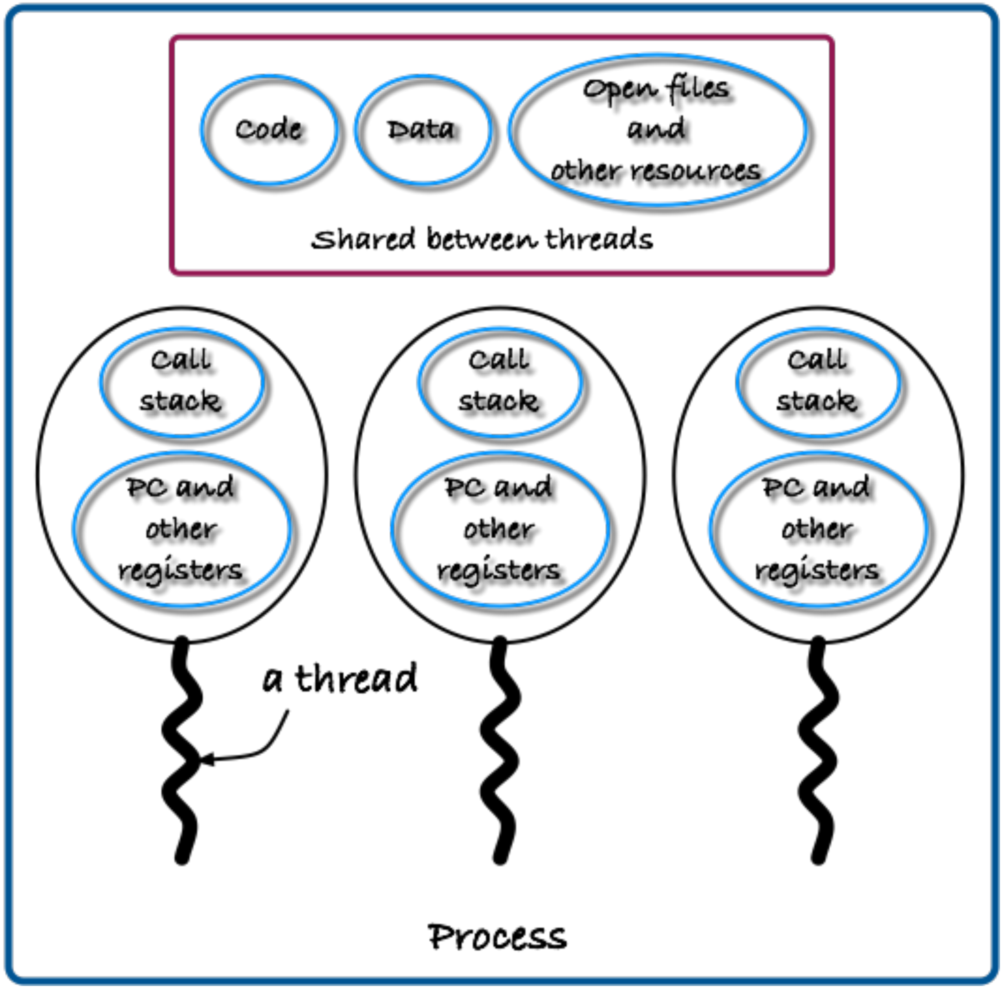
Threads

- Thread of control
  - Independent Fetch/Decode/Execute loop
  - The smallest unit of processing that can be scheduled by an operating system
  - A thread logically consists of:
    - Code
    - Registers
    - Stack
    - Thread-local data
- User-level thread vs. kernel-level thread

Threads (contd.)

- In general, a thread is contained in a process
  - Multiple thread can exist within the same process
  - Share resources with other threads
    - Code
    - Data
    - OS resources: open files, signals, etc.

Multi-threaded Process



Communication Between Threads

- Multiple threads within a process share portions of the virtual address space of the process (e.g., text (code) and data sections) by default

- Cooperation and coordination between threads in the same process is accomplished by reading and writing variables allocated in the shared space
  - Writes to a shared address by one thread is visible to reads of the other threads

Thread Library

- Provides the programmer an API for creating and managing threads
  - A user-level library entirely in user space with no kernel support
  - A kernel-level library supported directly by the operating system
    - Code and data structures for the library exist in kernel space
    - An API function call typically results in a system call

- POSIX Pthreads

User-level Threads vs. Kernel-level Threads

- User-level threads
  - Threading operations occur in user space
  - Threads are managed by a runtime library
- Kernel-level threads
  - Each thread has its own execution context
  - Threads are managed by the operating system

Linux Schedulers

- Completely Fair Scheduler (CFS)
  - Since kernel 2.6.23
- No distinction between processes and threads in scheduling
- To maintain fairness in providing processor time to processes
- A run-queue for each processor
  - Contains processes whose state is 'ready'
- Nice values
  - A processes' relative weight used in CFS
  - Lower nice value → higher weight → higher priority

Time Slice and Virtual Runtime

- Time slice
  - The time interval for which a process can run without being preempted
  - Proportional to the processes′ weight
- Virtual runtime
  - A measure for the amount of time provided to a given process
  - The smaller a processes′ virtual runtime, the higher its need for the processor

Virtual Runtime

- A processes′ cumulative execution time inversely scaled by its weight
  - The weight is a decay factor for the time for which a process has run

$$\text{virtual runtime}(P_i, t) = \text{Weight}_0/\text{Weight}_{P_i} * \text{physical runtime}(P_i, t)$$

$$\text{Weight}_0 = \text{the weight for the nice value } 0$$

Red-black Tree

- A self-balanced tree
- No path in the tree will ever be more than twice as long as any other
- Operations on the tree occur in O(log n), where n is the number of nodes in the tree

Red-black Tree (contd.)

- CFS maintains a red-black tree ordered by the virtual runtime
  - Run-queue
- Maintained independently for each processor
- The process with lowest virtual runtime is the left-most leaf node (highest: the right-most leaf node)
- The scheduler picks the left-most node to schedule next to maintain fairness
  - The task will be added to the tree with a new virtual runtime after running

CFS Algorithm

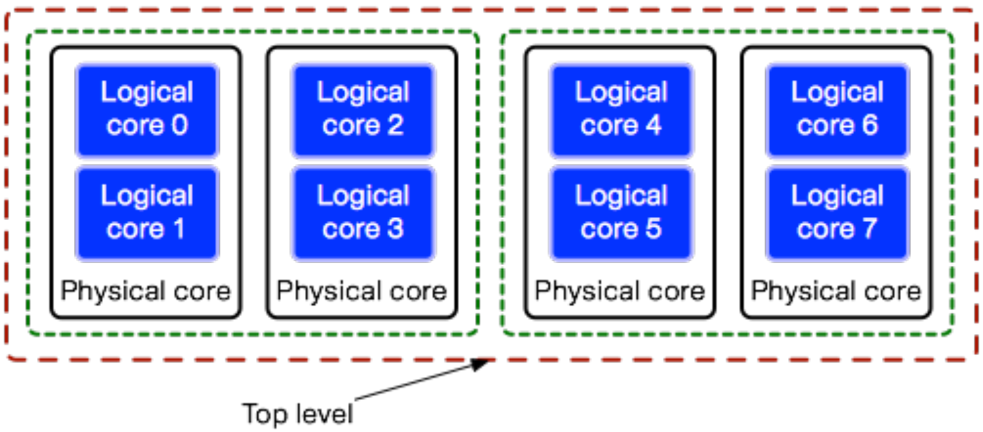
- Performs scheduling on each scheduling tick
- Decrement the time slice of the currently running process P by the tick period
  - When the time slice reaches 0, a flag is set
- Update the virtual runtime of P
- Check the flag
  - If set, preempt P and insert it to the run-queue
  - Schedule the process in the left-most node in the red-black tree

SMP Scheduling

- CFS
  - Scheduling processes for a single processor
- Run-queue load balancing
  - Distribute processes across multiple processors

Scheduling Domains

- A set of processors whose workloads should be kept balanced by the kernel
- Partitioned in one or more groups
- Hierarchically organized
  - Top scheduling domain: the set of all processors in the system



Run-queue Balancing

- Perform load balancing on each rebalancing tick
  - Push migration
- Check hierarchically if a scheduling domain is significantly unbalanced
  - Find the busiest run-queue in the domain
    - By calculating the load of each processor or group
    - Load of a processor: run-queue length
  - Migrate processes from the busiest run-queue to another one

Push vs. Pull

- Push migration
  - A specific process periodically checks the load on each processor and evenly distributes the load by moving (or pushing) processes from overloaded to idle or less-busy processors
- Pull migration
  - Occurs when an idle processor pulls a waiting process from a busy processor
- The Linux scheduler implements both techniques
  - Linux runs its load balancing algorithm every 200 milliseconds (push migration) or whenever the run-queue for a processor is empty (pull migration)

Negative Aspect of Process Migration

- The new processor's cache is cold for a migrated task
  - Needs to pull its data into the cache