Recap of the Last Class

- System protection and kernel mode
- System calls and the interrupt interface
- Processes
  - Process concept
  - A process's image in a computer
  - Operations on processes
  - Context switches and the scheduling process

Process and Its Image

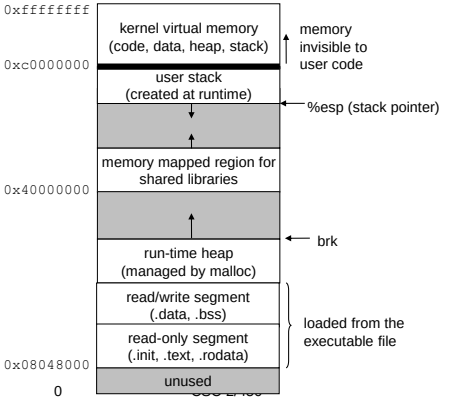
- An operating system executes a variety of programs:
  - A program that browses the Web
  - A program that serves Web requests
- Process – a program in execution.
- A process's state/image in a computer includes:
  - User-mode address space
  - Kernel data structure
  - Registers (including program 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)

Process Management

- *A process* is a program in execution
  - Unit of work – A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task
  - Protection domain
- OS responsibilities for process management:
  - Process creation and deletion
  - Process scheduling, suspension, and resumption
  - Process synchronization, inter-process communication

Private Address Spaces

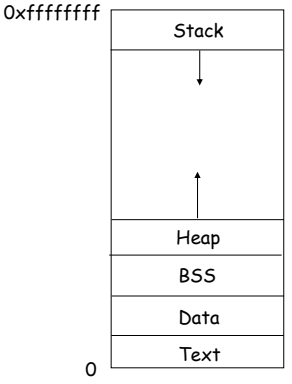
- Each process has its own private address space.



User-mode Address Space

User-mode address space for a process:

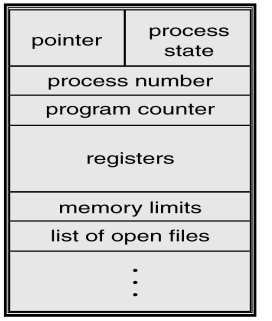
- Text: program code, instructions
- Data: initialized global and static variables (those data whose size is known before the execution)
- BSS (block started by symbol): uninitialized global and static variables
- Heap: dynamic memory (those being malloc-ed)
- Stack: local variables and other stuff for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



Process Control Block (PCB)

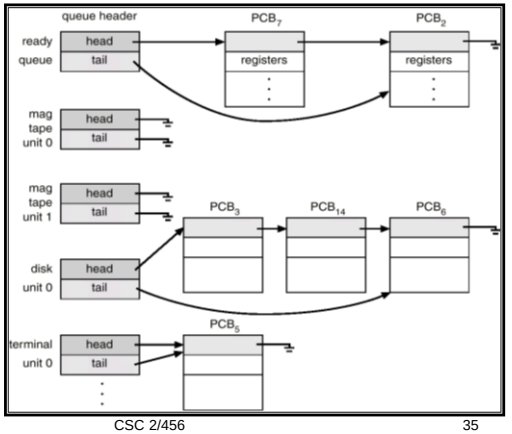
OS data structure (in kernel memory) maintaining information associated with each process.

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



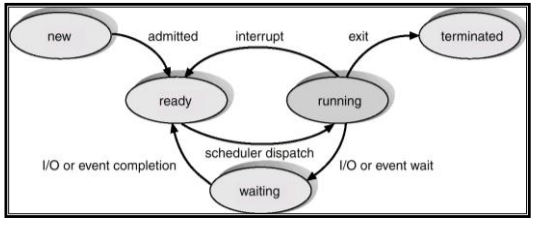
Queues for PCBs

- 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 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

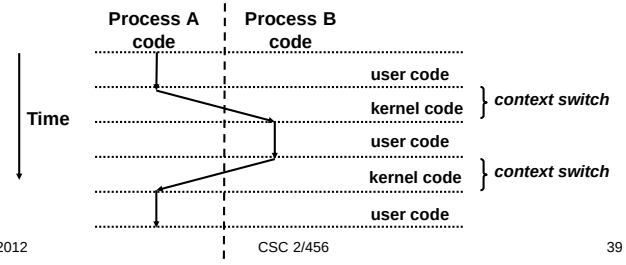
- When a process (parent) creates a new process (child)
  - Execution sequence?
  - Address space sharing?
  - Open files inheritance?
  - … …
- UNIX examples
  - **fork** system call creates new process with a duplicated copy of everything.
  - **exec** system call used after a **fork** to replace the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

Today

- Context switches and the scheduling process
- Inter-process communication
  - Shared memory
- Thread
  - Thread concept
  - Multithreading models
  - Types of threads

Context Switching

- Processes are managed by a shared chunk of OS code called the *kernel*
  - Important: the kernel is not a separate process, but rather runs as part of some user process
- Control flow passes from one process to another via a *context switch.*



Scheduling: Transferring Context Blocks

Coroutines

transfer(other)

save all callee-saves registers on stack, including ra and fp

*current := sp

current := other

sp := *current

pop all callee-saves registers (including ra, but NOT sp!)

return (into different coroutine!)

Uniprocessor Scheduling

- Use Ready List to reschedule voluntarily (cooperative threading)

reschedule:
- t : cb := dequeue(ready_list)
- transfer(t)

yield:
- enqueue(ready_list, current)
- reschedule

sleep_on(q):
enqueue(q, current)
reschedule

Preemption

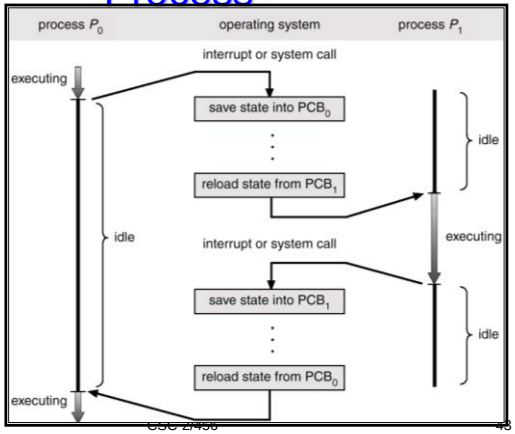
- Use timer interrupts or signals to trigger involuntary yields
- Protect scheduler data structures by disabling/reenabling prior to/after rescheduling

yield:
disable_signals
enqueue(ready_list, current)
reschedule
re-enable_signals

CPU Switch From Process to Process

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

Which one to switch to? - scheduling



Process Termination

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

Interprocess Communication

- Reasons for processes to cooperate
  - Information sharing (e.g., files)
  - Computation speedup
  - Modularity and protection
  - Convenience - multitasking

Mechanisms for Interprocess Communication

- Shared memory
- Message passing
  - Pipes, sockets, remote procedure calls

Shared Memory: POSIX interface

- shm_get – returns the identifier of a shared memory segment
- shmat – attaches the shared memory segment to the address space
- shmdt – detaches the segment located at the specified address
- shmctl – control of shared memory segments, including deletion

- Other possibilities: mmap (file sharing with preserved properties)

Message Passing

- Direct or indirect communication – processes or ports
- Fixed or variable size
- Send by copy or reference
- Automatic or explicit buffering
- Blocking or non-blocking (send or receive)

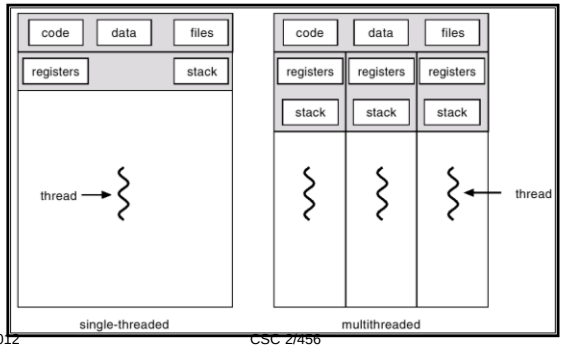
- Examples: client-server sockets, Mach ports, Windows 2000 local procedure call (LPC), remote procedure call (RPC, RMI)

Processes or Threads

- A process or thread is a potentially-active execution context
- Processes/threads can come from
  - Multiple CPUs
  - Kernel-level multiplexing of single physical CPU (kernel-level threads or processes)
  - Language or library-level multiplexing of kernel-level abstraction (user-level threads)
- Threads can run
  - Truly in parallel (on multiple CPUs)
  - Unpredictably interleaved (on a single CPU)
  - Run-until-block (coroutine-style)

Processes and Threads

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



Processes Vs. Threads

- Process
  - Single address space
  - Single thread of control for executing program
  - State information
    - Page tables, swap images, file descriptors, queued I/O requests, saved registers
- Threads
  - Separate notion of execution from the rest of the definition of a process
  - Other parts potentially shared with other threads
  - Program counter, stack of activation records, control block (e.g., saved registers/state info for thread management)
  - Kernel-level (lightweight process) handled by the system scheduler
  - User-level handled in user mode

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

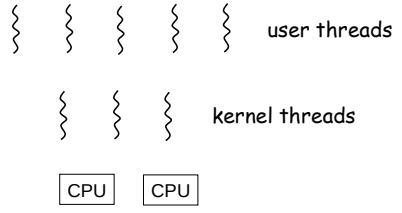
- User threads
  - Thread data structure is in user-mode memory
  - scheduling/switching done at user mode
- Kernel threads
  - Thread data structure is in kernel memory
  - scheduling/switching done by the OS kernel

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 and threads treated in a similar manner (both called tasks)
  - Processes are tasks with exclusive address space
  - Tasks can also share the address space, open files, …

Pthreads

- Each 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.

Issues with the Threading Model

- Thread-local storage – what about globals?
- Stack management
- Interaction with fork and exec system calls
  - Two versions of fork?
- Signal handling – which thread should the signal be delivered to?
  - Synchronous
  - All
  - Assigned thread
  - Unix: could assign a specific thread to handle signals
  - Windows: asynchronous procedure calls, which are thread-specific

Disclaimer

- Parts of the lecture slides contain the original work of Kai Shen, 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).