

Processes and Threads

CS 256/456
 Dept. of Computer Science, University
 of Rochester

10/20/2010

CSC 2/456

1

Recap of the Last Class

- System calls
 - Signals and the interrupt interface
- Processes
 - Process concept
 - A process's image in a computer
 - Operations on processes

10/20/2010

CSC 2/456

2

Today

- Context switches and the scheduling process
- System calls
 - Pipes
- Thread
 - Thread concept
 - Multithreading models
 - Types of threads
- Inter-process communication

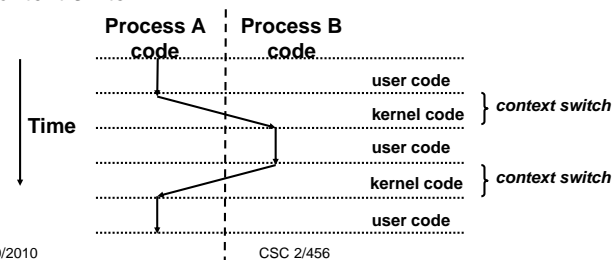
10/20/2010

CSC 2/456

3

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*.



10/20/2010

CSC 2/456

4

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!)

10/20/2010

CSC 2/456

5

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

10/20/2010

CSC 2/456

6

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

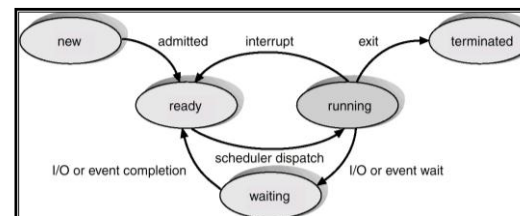
10/20/2010

CSC 2/456

7

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



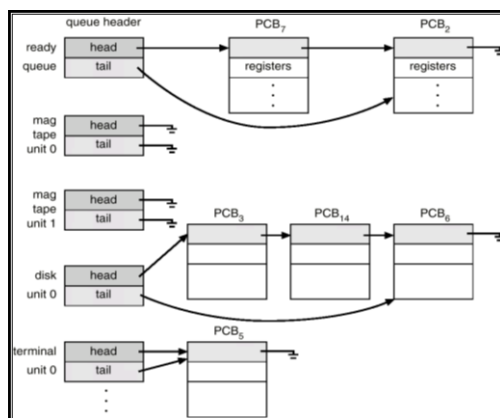
10/20/2010

CSC 2/456

8

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.



10/20/2010

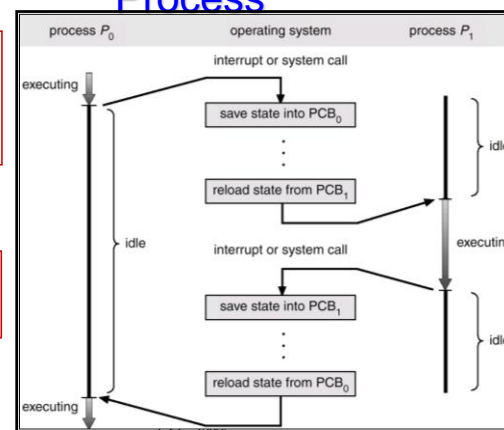
CSC 2/456

9

CPU Switch From Process to Process

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

Which one to switch to? - scheduling



10/20/2010

CSC 2/456

10

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 due to
 - Software exceptions
 - Receiving certain signals

10/20/2010

CSC 2/456

11

Interprocess Communication

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

10/20/2010

CSC 2/456

12

Mechanisms for Interprocess Communication

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

10/20/2010

CSC 2/456

13

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)

10/20/2010

CSC 2/456

14

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)

10/20/2010

CSC 2/456

15

Interprocess Communication: Pipes

- Conduit allowing two processes to communicate
 - Unidirectional or bidirectional
 - Full-duplex or half-duplex two-way communication
 - Is parent-child relationship required?
 - Is communication across a network allowed?

10/20/2010

CSC 2/456

16

Unix Pipes

- A unidirectional data channel that can be used for interprocess communication
- Pipe ceases to exist once closed or when process terminates
- System calls
 - pipe (int fd[])
 - dup2

10/20/2010

CSC 2/456

17

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)

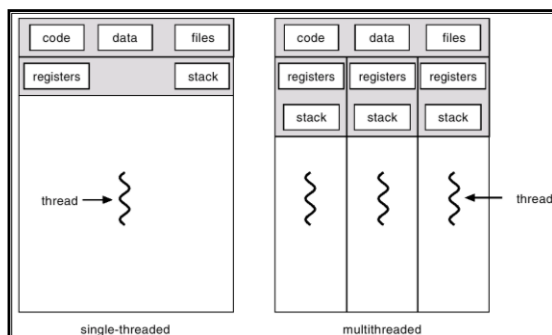
10/20/2010

CSC 2/456

18

Processes and Threads

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



10/20/2010

CSC 2/456

19

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

10/20/2010

CSC 2/456

20

Why Use Threads?

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

10/20/2010

CSC 2/456

21

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).

10/20/2010

CSC 2/456

110