CSMC 412

Operating Systems
Prof. Ashok K Agrawala

Set 5

Processes

- Process Concept
- Process Scheduling
- Operation on Processes
- Cooperating Processes
- Interprocess Communication

Process Concept

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
- Textbook uses the terms *job* and *process* almost interchangeably.
- Process – a program in execution; process execution must progress in sequential fashion.
- A process includes:
  - program counter
  - stack
  - data section

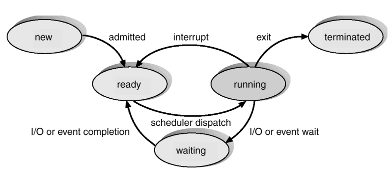
Process State

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

Process State

- Processes switch between different states based on internal and external events
- Each process is in exactly one state at a time
- Typical States of Processes (varies with OS)
  - New: The process is just being created
  - Running: Instructions are being executed
    - » only one process per processor may be running
  - Waiting: The process is waiting for an event to occur
    - » examples: I/O events, signals
  - Ready: The process is waiting to be assigned to a processor
  - Terminated: The process has finished execution

Diagram of Process State



Components of a Process

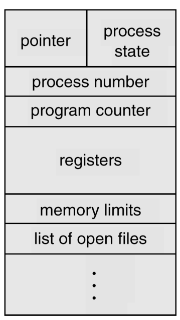
- Memory Segments
  - Program - often called the text segment
  - Data - global variables
  - Stack - contains activation records
- Processor Registers
  - program counter - next instruction to execute
  - general purpose CPU registers
  - processor status word
    - » results of compare operations
  - floating point registers

Process Control Block (PCB)

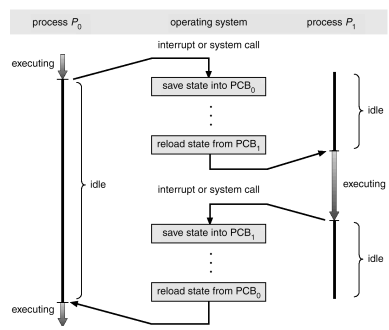
Information associated with each process.

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

Process Control Block (PCB)



CPU Switch From Process to Process



Storing PCBs

- Need to keep track of the different processes in the system
- Collection of PCBs is called a process table
- How to store the process table?
- First Option:

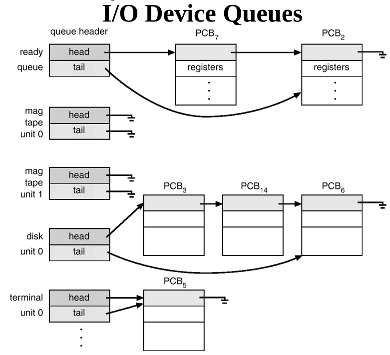
P1	P2	P2	P3	P4	P5
Ready	Waiting	New	Term	Waiting	Ready

- Problems with Option 1:
  - hard to find processes
  - how to fairly select a process

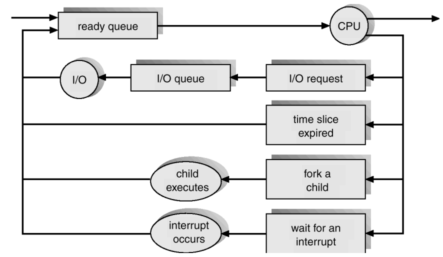
Process Scheduling Queues

- Job queue – set of all processes in the system.
- Ready queue – set of all processes residing in main memory,
  ready and waiting to execute.
- Device queues – set of processes waiting for an I/O device.
- Process migration between the various queues.

Ready Queue And Various I/O Device Queues



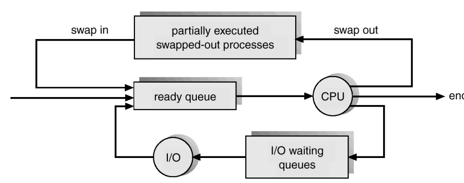
Representation of Process Scheduling



Schedulers

- **Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue.**
- **Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU.**

Addition of Medium Term Scheduling



Schedulers (Cont.)

- **Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast).**
- **Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slow).**
- **The long-term scheduler controls the *degree of multiprogramming.***
- **Processes can be described as either:**
  - **I/O-*bound process* – spends more time doing I/O than computations, many short CPU bursts.**
  - **CPU-*bound process* – spends more time doing computations; few very long CPU bursts.**

Context Switch

- **When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.**
- **Context-switch time is overhead; the system does no useful work while switching.**
- **Time dependent on hardware support.**

Selecting a process to run

- called scheduling
- can simply pick the first item in the queue
  - called round-robin scheduling
  - is round-robin scheduling fair?
- can use more complex schemes
  - we will study these in the future
- use alarm interrupts to switch between processes
  - when time is up, a process is put back on the end of the ready queue
  - frequency of these interrupts is an important parameter
    - » typically 3-10ms on modern systems
    - » need to balance overhead of switching vs. responsiveness

Process Priority

- Use multiple run queues, one for each priority
- Who decides priority
  - dispatcher - that mixes policy and mechanism too much
  - when the process is created, assign it a priority
  - have a second level scheduler (often called medium term scheduler) to manage priorities
    - » mechanism is to move processes between different queues
- Will discuss scheduling more in a future lecture

Process Creation

- Parent process creates children processes, which, in turn create other processes, forming a tree of processes.
- Resource sharing
  - Parent and children share all resources.
  - Children share subset of parent's resources.
  - Parent and child share no resources.
- Execution
  - Parent and children execute concurrently.
  - Parent waits until children terminate.

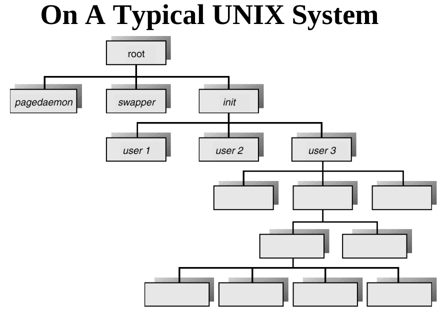
Process Creation (Cont.)

- Address space
  - Child duplicate of parent.
  - Child has a program loaded into it.
- UNIX examples
  - fork system call creates new process
  - execve system call used after a fork to replace the process' memory space with a new program.

Process Creation

- Who creates processes?
  - answer: other processes
  - operations is called fork (or spawn)
  - what about the first process?
- Have a tree of processes
  - » parent-child relationship between processes
- what resources does the child get?
  - » new resources from the OS
  - » a copy of the parent resources
  - » a subset of the parent resources
- What program does the child run?
  - » a copy of the parent (UNIX fork)
    - a process may change its program (execve call in UNIX)
  - » a new program specified at creation (VMS spawn)

A Tree of Processes On A Typical UNIX System



Forking a New Process

- create a PCB for the new process
  - copy most entries from the parent
  - clear accounting fields
  - buffered pending I/O
  - allocate a pid (process id for the new process)
- allocate memory for it
  - could require copying all of the parents segments
  - however, text segment usually doesn't change so that could be shared
  - might be able to use memory mapping hardware to help
    - will talk more about this in the memory management part of the class
- add it to the ready queue

Process Termination

- Process executes last statement and asks the operating system to decide it (exit).
  - Output data from child to parent (via wait).
  - Process' resources are deallocated by operating system.
- Parent may terminate execution of children processes (abort).
  - Child has exceeded allocated resources.
  - Task assigned to child is no longer required.
  - Parent is exiting.
    - Operating system does not allow child to continue if its parent terminates.
    - Cascading termination.

Process Termination

- Process can terminate self
  - via the exit system call
- One process can terminate another process
  - use the kill system call
  - can any process kill any other process?
    - No, that would be bad.
    - Normally an ancestor can terminate a descendant
- OS kernel can terminate a process
  - exceeds resource limits
  - tries to perform an illegal operation
- What if a parent terminates before the child
  - called an orphan process
  - in UNIX becomes child of the root process
  - in VMS - causes all descendants to be killed

Termination (cont.) - UNIX example

- Kernel
  - frees memory used by the process
  - moved process control block to the terminated queue
- Terminated process
  - signals parent of its death (SIGCHILD)
  - is called a zombie in UNIX
  - remains around waiting to be reclaimed
- parent process
  - wait system call retrieves info about the dead process
    - exit status
    - accounting information
  - signal handler is generally called the reaper
    - since its job is to collect the dead processes

Cooperating Processes

- *Independent* process cannot affect or be affected by the execution of another process.
- *Cooperating* process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

Producer-Consumer Problem

- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process.
  - *unbounded-buffer* places no practical limit on the size of the buffer.
  - *bounded-buffer* assumes that there is a fixed buffer size.

Bounded-Buffer – Shared-Memory Solution

- **Shared data**

  var *n*;
  type *item* = … ;
  var *buffer*. array [0..*n*–1] of *item*;
  in, out: **0..*n*–1;**

- **Producer process**

  repeat
  …
  produce an item in *nextp*
  …
  while ***in*+1** mod ***n = out*** do ***no-op;***
  buffer **[*in*] :=*nextp*;**
  ***in :=in*+1** mod ***n*;**
  until ***false*;**

Bounded-Buffer (Cont.)

- **Consumer process**

  repeat
  while ***in = out*** do ***no-op*;**
  ***nextc* := *buffer* [*out*];**
  ***out* := *out*+1** mod ***n*;**
  …
  consume the item in nextc
  …
  until ***false*;**

- **Solution is correct, but can only fill up n–1 buffer.**

Threads

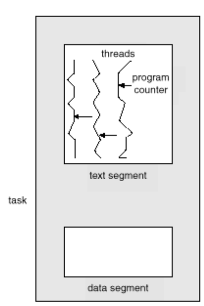
- **A *thread* (or *lightweight process*) is a basic unit of CPU utilization; it consists of:**
  - **program counter**
  - **register set**
  - **stack space**
- **A thread shares with its peer threads its:**
  - **code section**
  - **data section**
  - **operating-system resources**

  collectively know as a *task*.
- **A traditional or *heavyweight* process is equal to a task with one thread**

Threads (Cont.)

- **In a multiple threaded task, while one server thread is blocked and waiting, a second thread in the same task can run.**
  - **Cooperation of multiple threads in same job confers higher throughput and improved performance.**
  - **Applications that require sharing a common buffer (i.e., producer-consumer) benefit from thread utilization.**
- **Threads provide a mechanism that allows sequential processes to make blocking system calls while also achieving parallelism.**
- **Kernel-supported threads (Mach and OS/2).**
- **User-level threads; supported above the kernel, via a set of library calls at the user level (Project Andrew from CMU).**
- **Hybrid approach implements both user-level and kernel-supported threads (Solaris 2).**

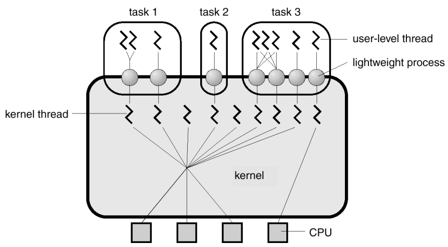
Multiple Threads within a Task



Threads Support in Solaris 2

- **Solaris 2 is a version of UNIX with support for threads at the kernel and user levels, symmetric multiprocessing, and real-time scheduling.**
- **LWP – intermediate level between user-level threads and kernel-level threads.**
- **Resource needs of thread types:**
  - **Kernel thread: small data structure and a stack; thread switching does not require changing memory access information – relatively fast.**
  - **LWP: PCB with register data, accounting and memory information,; switching between LWPs is relatively slow.**
  - **User-level thread: only ned stack and program counter; no kernel involvement means fast switching. Kernel only sees the LWPs that support user-level threads.**

Solaris 2 Threads



Interprocess Communication (IPC)

- **Mechanism for processes to communicate and to synchronize their actions.**
- **Message system – processes communicate with each other without resorting to shared variables.**
- **IPC facility provides two operations:**
  - send(***message***) **– message size fixed or variable**
  - receive(***message***)
- **If *P* and *Q* wish to communicate, they need to:**
  - **establish a *communication link* between them**
  - **exchange messages via send/receive**
- **Implementation of communication link**
  - **physical (e.g., shared memory, hardware bus)**
  - **logical (e.g., logical properties)**

Implementation Questions

- **How are links established?**
- **Can a link be associated with more than two processes?**
- **How many links can there be between every pair of communicating processes?**
- **What is the capacity of a link?**
- **Is the size of a message that the link can accommodate fixed or variable?**
- **Is a link unidirectional or bi-directional?**

Direct Communication

- **Processes must name each other explicitly:**
  - send **(*P, message*) – send a message to process P**
  - receive**(*Q, message*) – receive a message from process Q**
- **Properties of communication link**
  - **Lilnks are established automatically.**
  - **A link is associated with exactly one pair of communicating processes.**
  - **Between each pair there exists exactly one link.**
  - **The link may be unidirectional, but is usually bi-directional.**

Indirect Communication

- **Messages are directed and received from mailboxes (also referred to as ports).**
  - **Each mailbox has a unique id.**
  - **Processes can communicate only if they share a mailbox.**
- **Properties of communication link**
  - **Link established only if processes share a common mailbox**
  - **A link may be associated with many processes.**
  - **Each pair of processes may share several communication links.**
  - **Link may be unidirectional or bi-directional.**
- **Operations**
  - **create a new mailbox**
  - **send and receive messages through mailbox**
  - **destroy a mailbox**

Indirect Communication (Continued)

- **Mailbox sharing**
  - P_1, P_2, **and** P_3 **share mailbox A.**
  - P_1**, sends;** P_2 **and** P_3 **receive.**
  - **Who gets the message?**
- **Solutions**
  - **Allow a link to be associated with at most two processes.**
  - **Allow only one process at a time to execute a receive operation.**
  - **Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.**

Buffering

- Queue of messages attached to the link; implemented in one of three ways.
  1. Zero capacity – 0 messages
     Sender must wait for receiver (rendezvous).
  2. Bounded capacity – finite length of *n* messages
     Sender must wait if link full.
  3. Unbounded capacity – infinite length
     Sender never waits.

Exception Conditions – Error Recovery

- Process terminates
- Lost messages
- Scrambled Messages