

Module 4: Processes

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

Applied Operating System Concepts 4.1 Silberschatz, Galvin, and Gagne 1999

Process Concept

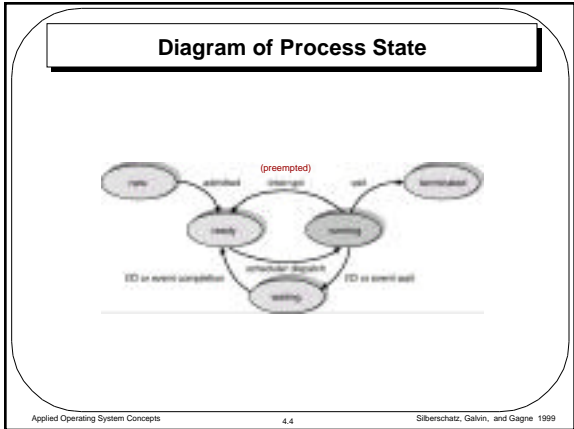
- An operating system executes a variety of programs:
 - Batch system – jobs
 - Time-shared systems – user programs or tasks
- Textbook uses the terms *job* and *process* almost interchangeably.
- Process – a program in execution; process execution must progress in sequential fashion.
- A process includes: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

Applied Operating System Concepts 4.2 Silberschatz, Galvin, and Gagne 1999

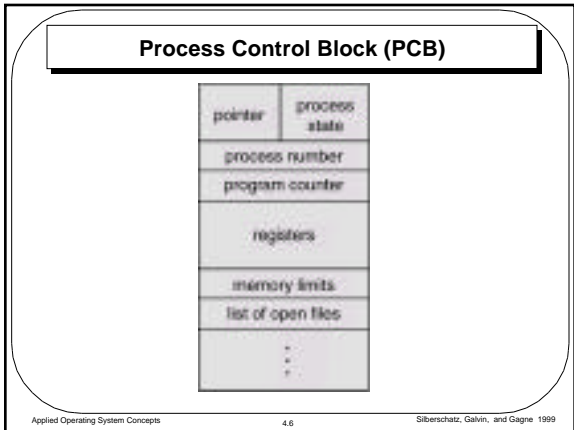
Process State

- 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.

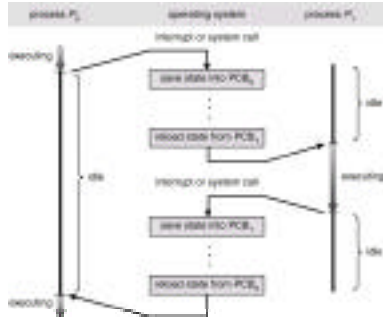
Applied Operating System Concepts 4.3 Silberschatz, Galvin, and Gagne 1999



- ### Process Control Block (PCB)
- Information associated with each process.
- Process state
 - Program counter
 - CPU registers
 - CPU scheduling information (priority, pointers to queues,...) (Chapter 6)
 - Memory-management information (base & limit reg.s, page tables, or segment tables)
 - Accounting information(CPU time, ...)
 - I/O status information(I/O devices allocated, files open, ...)
 - Threads - Chapter 5)
- Applied Operating System Concepts 4.5 Silberschatz, Galvin, and Gagne 1999



CPU Switch From Process to Process



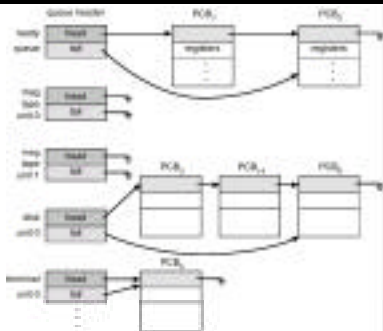
Applied Operating System Concepts 4.7 Silberschatz, Galvin, and Gagne 1999

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.

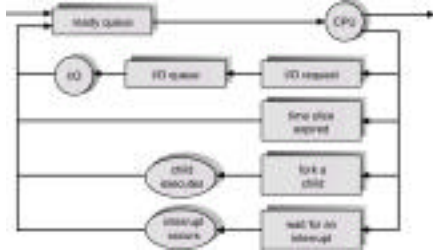
Applied Operating System Concepts 4.8 Silberschatz, Galvin, and Gagne 1999

Ready Queue And Various I/O Device Queues



Applied Operating System Concepts 4.9 Silberschatz, Galvin, and Gagne 1999

Representation of Process Scheduling



Applied Operating System Concepts 4.10 Silberschatz, Galvin, and Gagne 1999

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.

Applied Operating System Concepts 4.11 Silberschatz, Galvin, and Gagne 1999

Addition of Medium Term Scheduling



Applied Operating System Concepts 4.12 Silberschatz, Galvin, and Gagne 1999

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.

Applied Operating System Concepts

4.13

Silberschatz, Galvin, and Gagne 1999

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 (frequently the major bottleneck in modern systems)
- Time dependent on hardware support.

Applied Operating System Concepts

4.14

Silberschatz, Galvin, and Gagne 1999

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.

Applied Operating System Concepts

4.15

Silberschatz, Galvin, and Gagne 1999

Process Creation (Cont.)

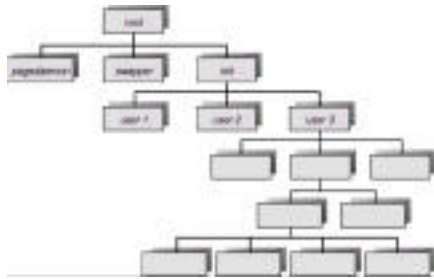
- Address space
 - Child duplicate of parent.
 - Child has a program loaded into it.
- UNIX examples
 - **fork** system call creates new process
 - **execvp** system call used after a **fork** to replace the process' memory space with a new program.

Applied Operating System Concepts

4.16

Silberschatz, Galvin, and Gagne 1999

A Tree of Processes On A Typical UNIX System



Applied Operating System Concepts

4.17

Silberschatz, Galvin, and Gagne 1999

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.

Applied Operating System Concepts

4.18

Silberschatz, Galvin, and Gagne 1999

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

Applied Operating System Concepts

4.19

Silberschatz, Galvin, and Gagne 1999

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.

Applied Operating System Concepts

4.20

Silberschatz, Galvin, and Gagne 1999

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;
```

Applied Operating System Concepts

4.21

Silberschatz, Galvin, and Gagne 1999

Bounded-Buffer (Cont.)

- Consumer process

repeat

while $in = out$ **do** *no-op*;

$nextc := buffer[out]$;

$out := out + 1 \bmod n$;

...

consume the item in *nextc*

...

until *false*;

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

Applied Operating System Concepts

4.22

Silberschatz, Galvin, and Gagne 1999

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

Applied Operating System Concepts

4.23

Silberschatz, Galvin, and Gagne 1999

Threads (Cont.)

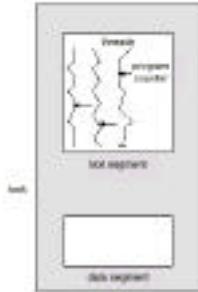
- In a multiple threaded task, while one server thread is blocked and waiting, a second thread in the same task can run.
 - Cooperation of multiple threads in same job confers higher throughput and improved performance.
 - Applications that require sharing a common buffer (i.e., producer-consumer) benefit 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).

Applied Operating System Concepts

4.24

Silberschatz, Galvin, and Gagne 1999

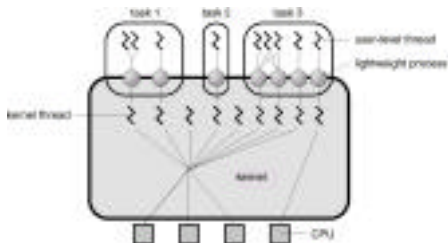
Multiple Threads within a Task



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 need 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)

Applied Operating System Concepts

4.28

Silberschatz, Galvin, and Gagne 1999

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?

Applied Operating System Concepts

4.29

Silberschatz, Galvin, and Gagne 1999

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
 - Links 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.

Applied Operating System Concepts

4.30

Silberschatz, Galvin, and Gagne 1999

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

Applied Operating System Concepts

4.31

Silberschatz, Galvin, and Gagne 1999

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.

Applied Operating System Concepts

4.32

Silberschatz, Galvin, and Gagne 1999

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.

Applied Operating System Concepts

4.33

Silberschatz, Galvin, and Gagne 1999

Exception Conditions – Error Recovery

- Process terminates
- Lost messages
- Scrambled Messages
