

Module 4: Processes

Super important and useful

- 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.

→ Gets all resources except CPU *init*

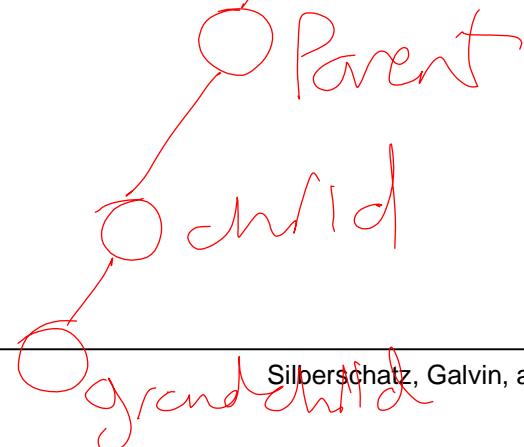
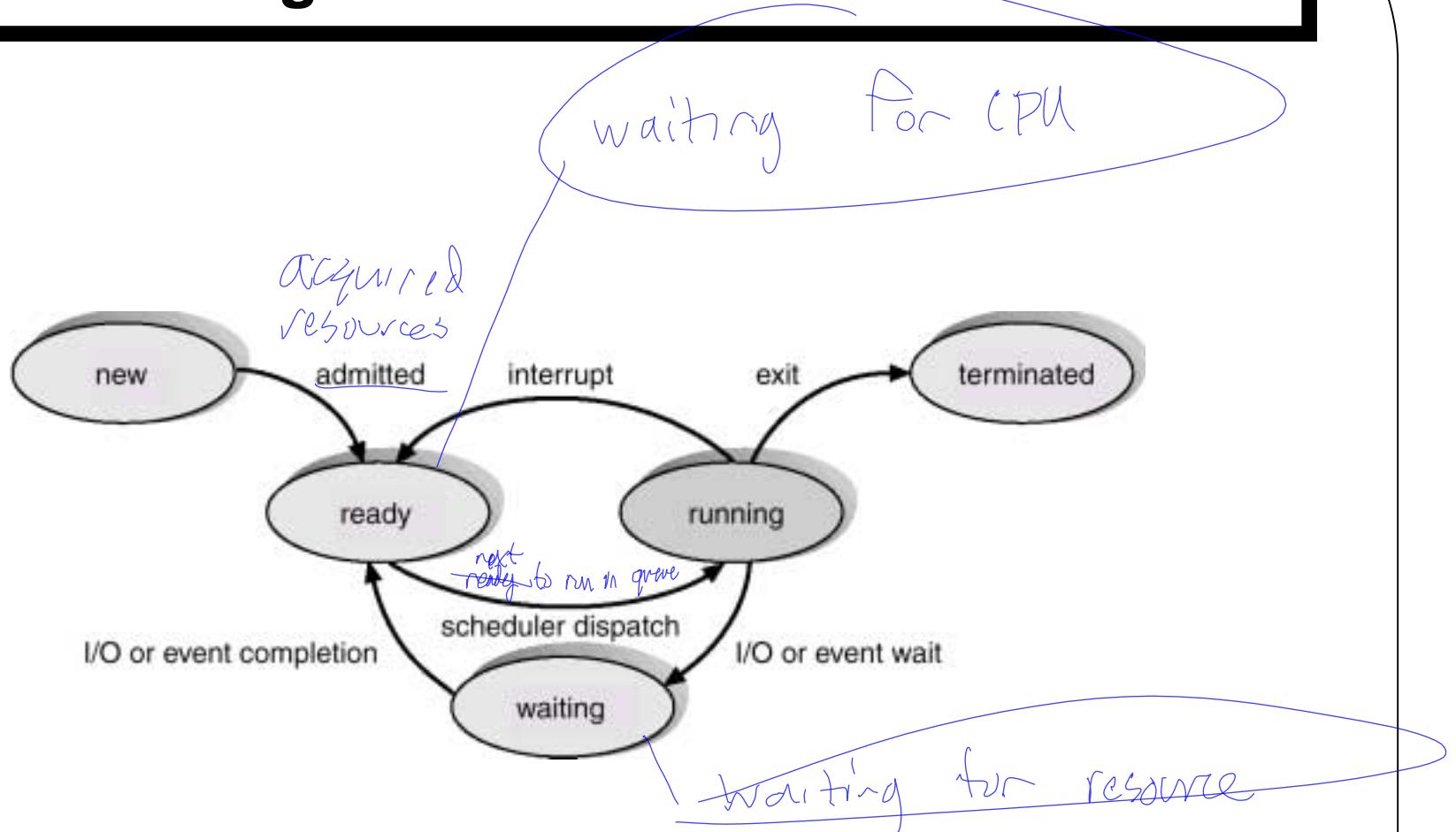


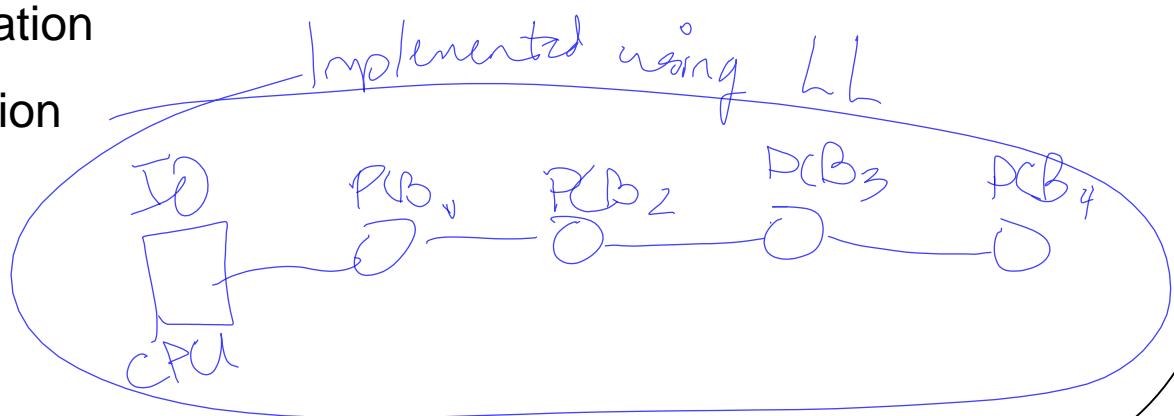
Diagram of Process State



Process Control Block (PCB)

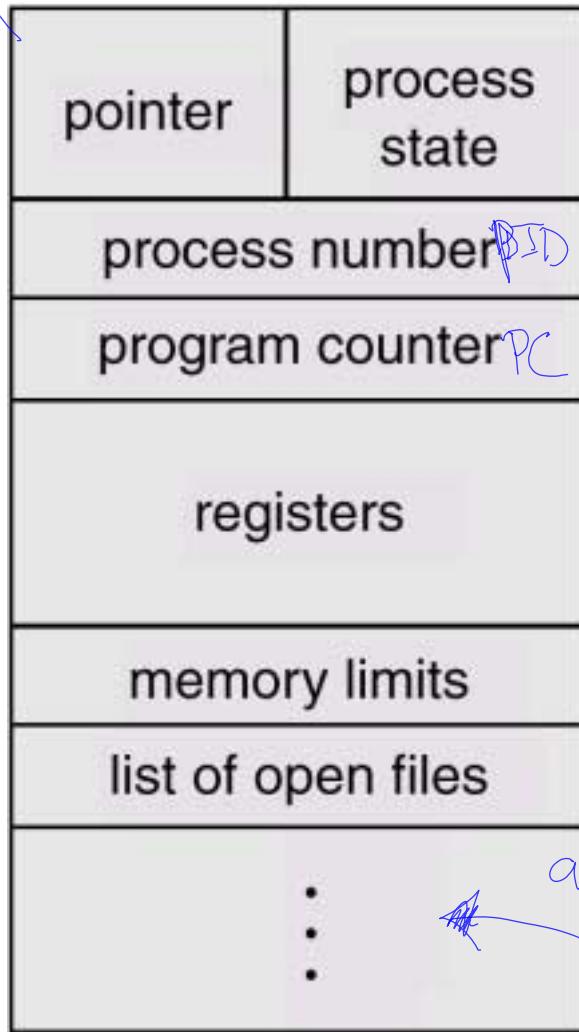
Information associated with each process.

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



Process Control Block (PCB)

Is a queue
like a
linked list



each process

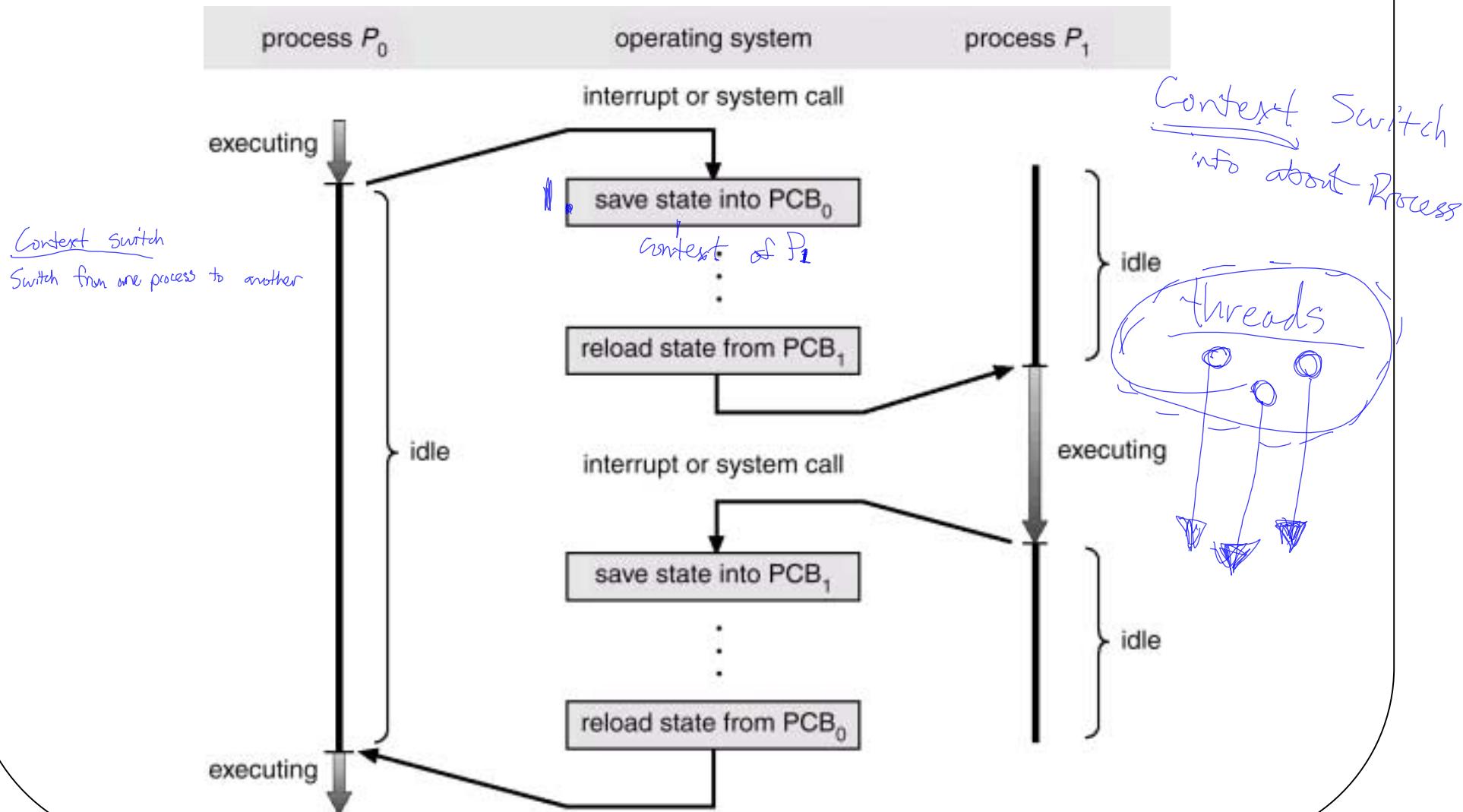
has one

Sometimes used

as a reference

→ SW interrupt

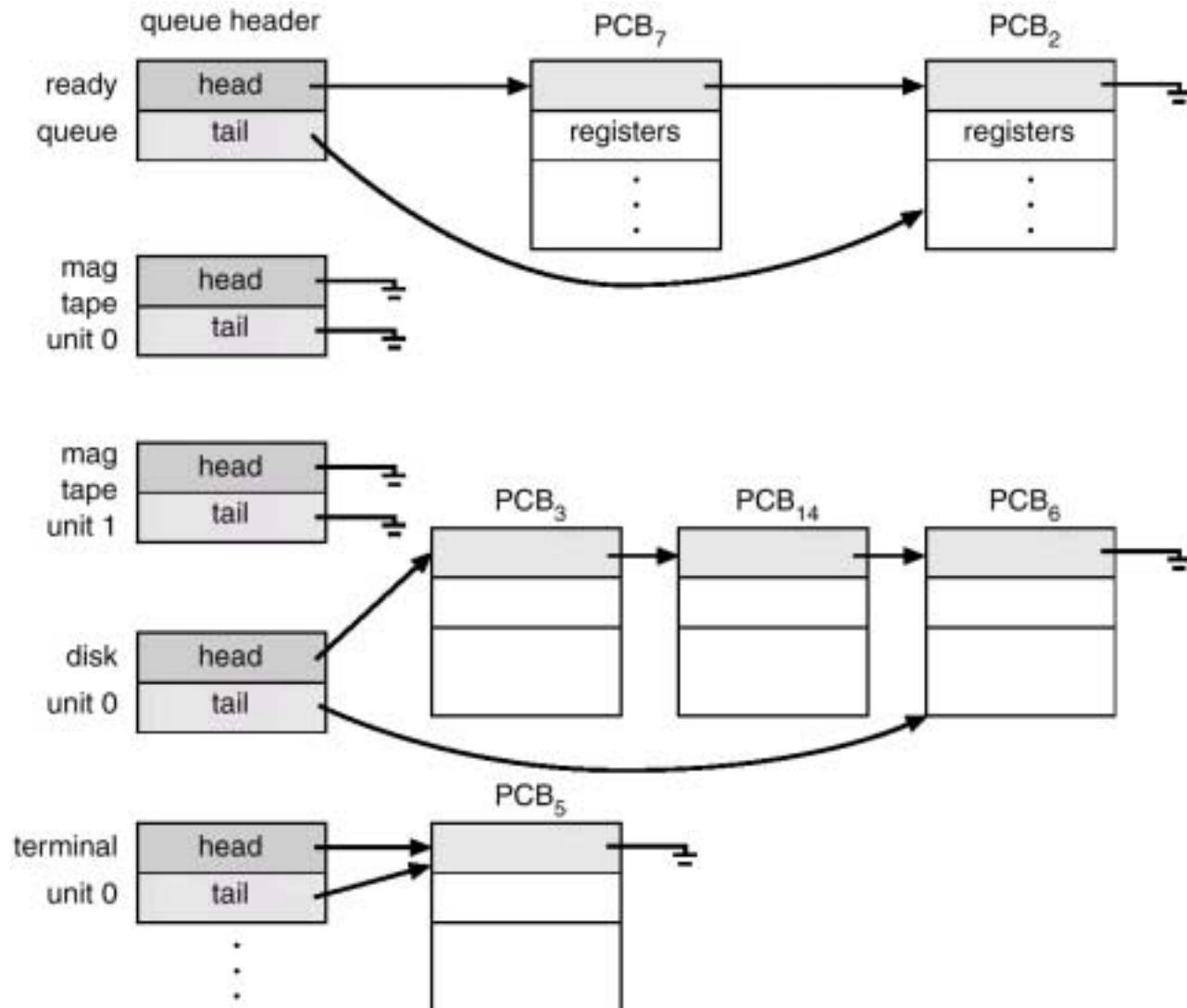
CPU Switch From Process to Process



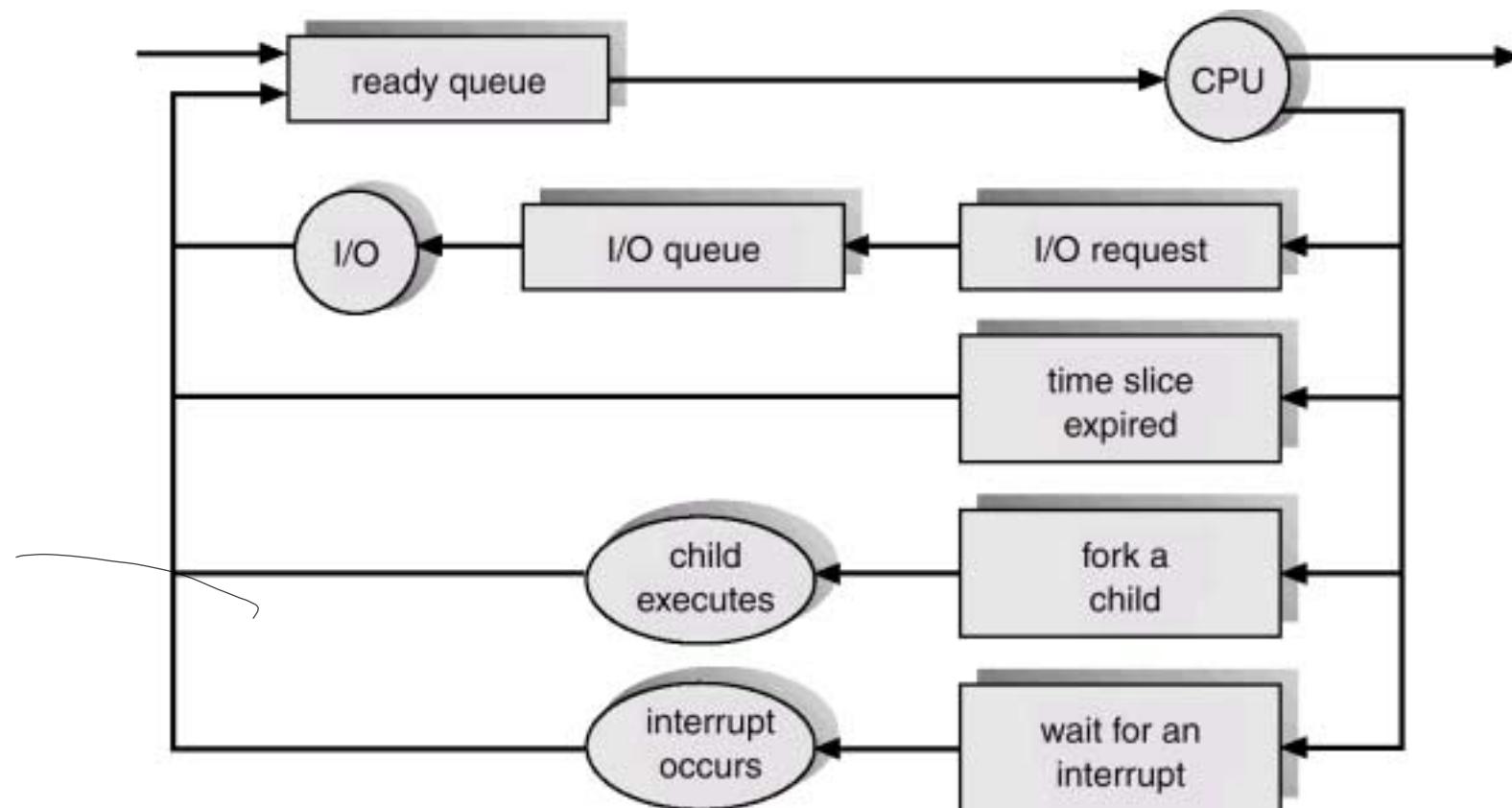
Process Scheduling Queues

- 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



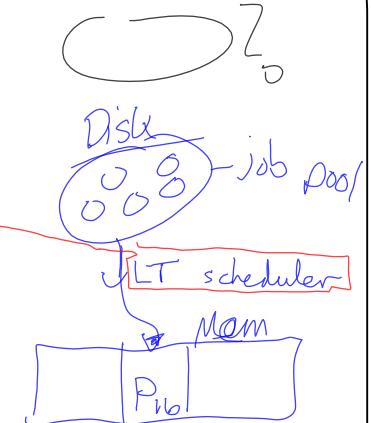
Representation of Process Scheduling



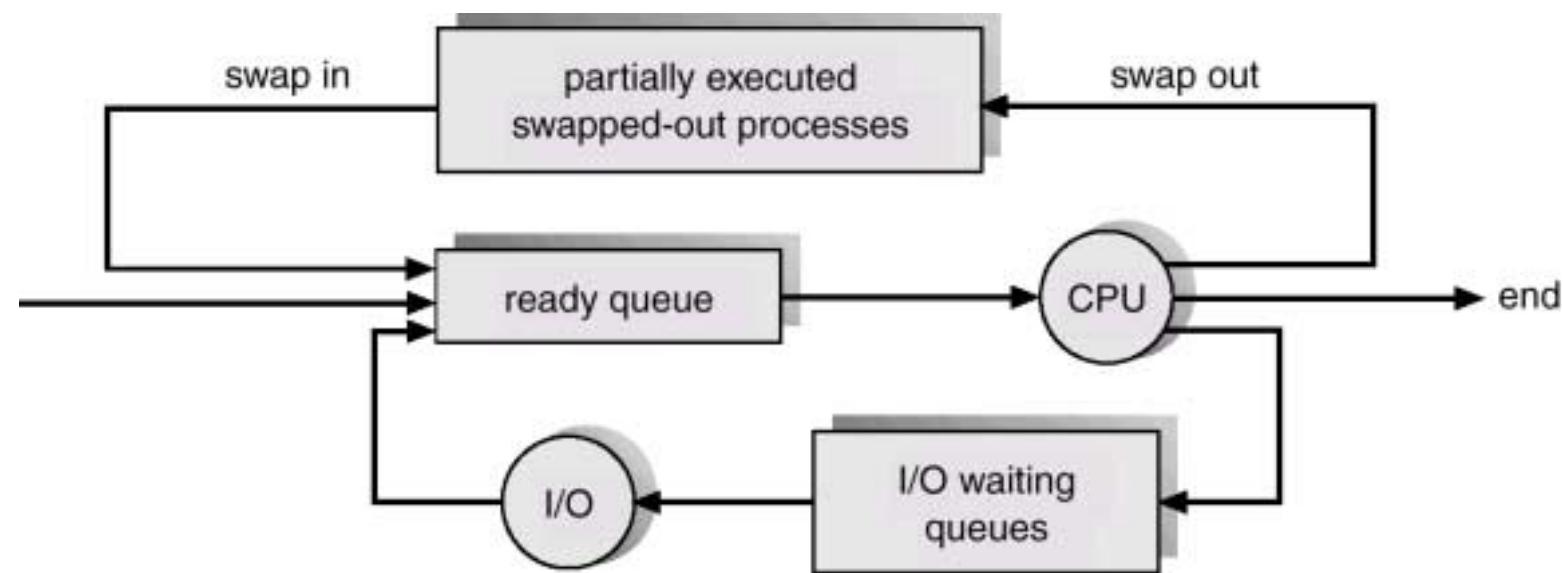
Schedulers

- 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.

Medium Term sch - Swaps in/out partially
executed processes



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.

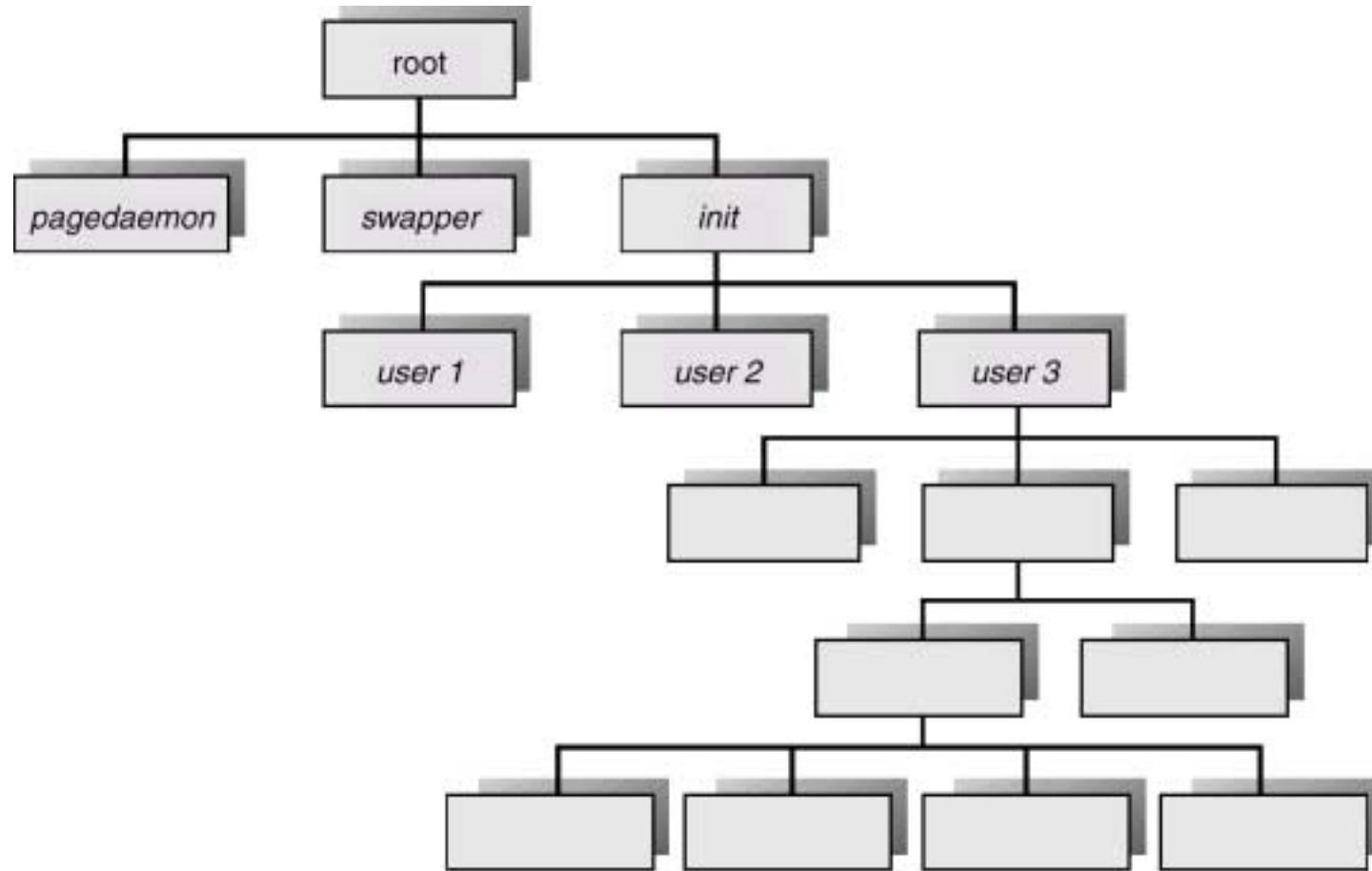
Process Creation

- 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.
- wait (All?)*
- How does process know to continue?*
- OS sends SIGCHLD signal.*

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.

A Tree of Processes On A Typical UNIX System



Process Termination

- Process executes last statement and asks the operating system to decide it (**exit**).
 - Output data from child to parent (via wait). *wait will release PID*
 - 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.

1. child dead & parent alive : Pid taken
S. Kill parent

2. child alive & parent dead : init chd +

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

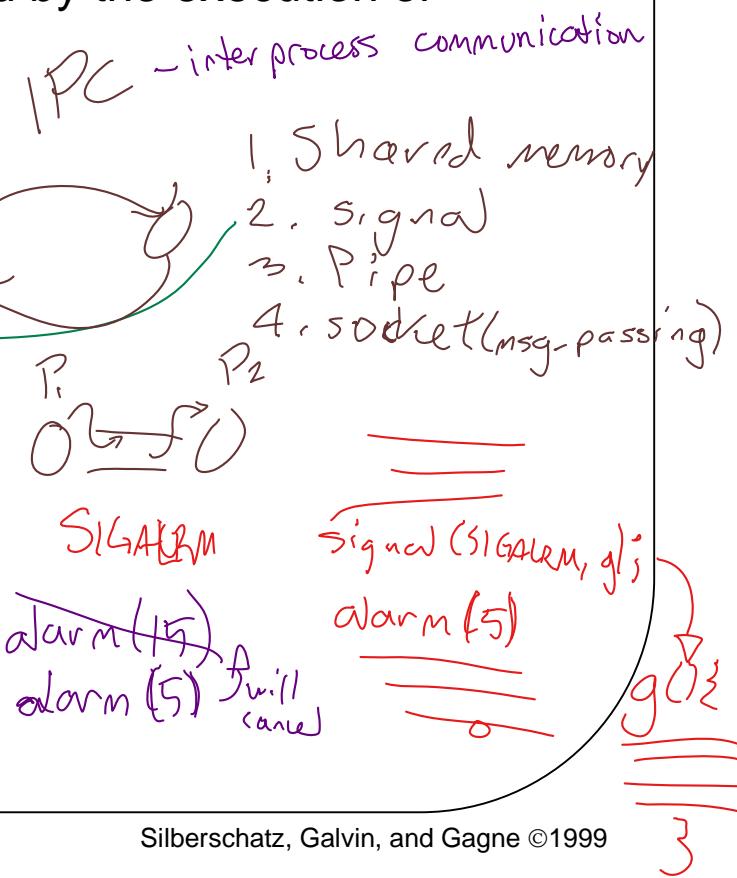
SIGINT: Ctrl+C
SIGALRM
SIGPIPE Broken pipe
SIGCHLD: child finished

\$gcc -o f f.c

./f

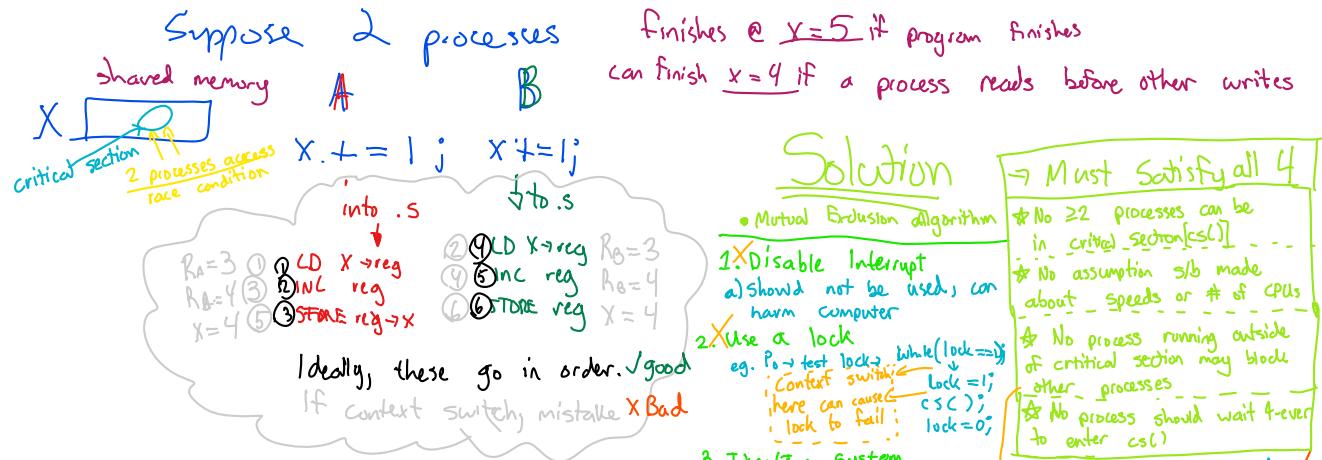


Signal
1. Use keyboard
Ctrl+C = SIGINT
Shift+~ = SIGABRT
2. CMD \$KILL -9 5322 pid
3. Syscall kill(pid, SIGSTOP)



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.



Solution

Mutual Exclusion algorithm

1. Disable Interrupt

a) Should not be used; can harm computer

2. Use a lock

e.g. P₀ → test lock₂, while(lock == 0)
Context switch
here can cause lock to fail
lock = 1;
CSC;
lock = 0;

3. Token/Turn System

```
P0 while(true){  
  while(turn != c)  
    CSC;  
  turn++;  
  non-CSL;  
  process  
  non-CSL;  
}  
when(){  
  wh(turn != c)  
  CSC;  
  turn = 0;  
  non-CSL;  
}
```

other = 1;
flag[0] = t;
wh(flag[0] == t);
flag[0] = f;

other = 0;
flag[1] = t;
wh(flag[1] == t);
flag[1] = f;

other = 1;
flag[2] = t;
wh(flag[2] == t);
flag[2] = f;

other = 0;
flag[3] = t;
wh(flag[3] == t);
flag[3] = f;

other = 1;
flag[4] = t;
wh(flag[4] == t);
flag[4] = f;

other = 0;
flag[5] = t;
wh(flag[5] == t);
flag[5] = f;

other = 1;
flag[6] = t;
wh(flag[6] == t);
flag[6] = f;

other = 0;
flag[7] = t;
wh(flag[7] == t);
flag[7] = f;

other = 1;
flag[8] = t;
wh(flag[8] == t);
flag[8] = f;

other = 0;
flag[9] = t;
wh(flag[9] == t);
flag[9] = f;

other = 1;
flag[10] = t;
wh(flag[10] == t);
flag[10] = f;

other = 0;
flag[11] = t;
wh(flag[11] == t);
flag[11] = f;

other = 1;
flag[12] = t;
wh(flag[12] == t);
flag[12] = f;

other = 0;
flag[13] = t;
wh(flag[13] == t);
flag[13] = f;

other = 1;
flag[14] = t;
wh(flag[14] == t);
flag[14] = f;

other = 0;
flag[15] = t;
wh(flag[15] == t);
flag[15] = f;

other = 1;
flag[16] = t;
wh(flag[16] == t);
flag[16] = f;

other = 0;
flag[17] = t;
wh(flag[17] == t);
flag[17] = f;

other = 1;
flag[18] = t;
wh(flag[18] == t);
flag[18] = f;

other = 0;
flag[19] = t;
wh(flag[19] == t);
flag[19] = f;

other = 1;
flag[20] = t;
wh(flag[20] == t);
flag[20] = f;

other = 0;
flag[21] = t;
wh(flag[21] == t);
flag[21] = f;

other = 1;
flag[22] = t;
wh(flag[22] == t);
flag[22] = f;

other = 0;
flag[23] = t;
wh(flag[23] == t);
flag[23] = f;

other = 1;
flag[24] = t;
wh(flag[24] == t);
flag[24] = f;

other = 0;
flag[25] = t;
wh(flag[25] == t);
flag[25] = f;

other = 1;
flag[26] = t;
wh(flag[26] == t);
flag[26] = f;

other = 0;
flag[27] = t;
wh(flag[27] == t);
flag[27] = f;

other = 1;
flag[28] = t;
wh(flag[28] == t);
flag[28] = f;

other = 0;
flag[29] = t;
wh(flag[29] == t);
flag[29] = f;

other = 1;
flag[30] = t;
wh(flag[30] == t);
flag[30] = f;

other = 0;
flag[31] = t;
wh(flag[31] == t);
flag[31] = f;

other = 1;
flag[32] = t;
wh(flag[32] == t);
flag[32] = f;

other = 0;
flag[33] = t;
wh(flag[33] == t);
flag[33] = f;

other = 1;
flag[34] = t;
wh(flag[34] == t);
flag[34] = f;

other = 0;
flag[35] = t;
wh(flag[35] == t);
flag[35] = f;

other = 1;
flag[36] = t;
wh(flag[36] == t);
flag[36] = f;

other = 0;
flag[37] = t;
wh(flag[37] == t);
flag[37] = f;

other = 1;
flag[38] = t;
wh(flag[38] == t);
flag[38] = f;

other = 0;
flag[39] = t;
wh(flag[39] == t);
flag[39] = f;

other = 1;
flag[40] = t;
wh(flag[40] == t);
flag[40] = f;

other = 0;
flag[41] = t;
wh(flag[41] == t);
flag[41] = f;

other = 1;
flag[42] = t;
wh(flag[42] == t);
flag[42] = f;

other = 0;
flag[43] = t;
wh(flag[43] == t);
flag[43] = f;

other = 1;
flag[44] = t;
wh(flag[44] == t);
flag[44] = f;

other = 0;
flag[45] = t;
wh(flag[45] == t);
flag[45] = f;

other = 1;
flag[46] = t;
wh(flag[46] == t);
flag[46] = f;

other = 0;
flag[47] = t;
wh(flag[47] == t);
flag[47] = f;

other = 1;
flag[48] = t;
wh(flag[48] == t);
flag[48] = f;

other = 0;
flag[49] = t;
wh(flag[49] == t);
flag[49] = f;

other = 1;
flag[50] = t;
wh(flag[50] == t);
flag[50] = f;

other = 0;
flag[51] = t;
wh(flag[51] == t);
flag[51] = f;

other = 1;
flag[52] = t;
wh(flag[52] == t);
flag[52] = f;

other = 0;
flag[53] = t;
wh(flag[53] == t);
flag[53] = f;

other = 1;
flag[54] = t;
wh(flag[54] == t);
flag[54] = f;

other = 0;
flag[55] = t;
wh(flag[55] == t);
flag[55] = f;

other = 1;
flag[56] = t;
wh(flag[56] == t);
flag[56] = f;

other = 0;
flag[57] = t;
wh(flag[57] == t);
flag[57] = f;

other = 1;
flag[58] = t;
wh(flag[58] == t);
flag[58] = f;

other = 0;
flag[59] = t;
wh(flag[59] == t);
flag[59] = f;

other = 1;
flag[60] = t;
wh(flag[60] == t);
flag[60] = f;

other = 0;
flag[61] = t;
wh(flag[61] == t);
flag[61] = f;

other = 1;
flag[62] = t;
wh(flag[62] == t);
flag[62] = f;

other = 0;
flag[63] = t;
wh(flag[63] == t);
flag[63] = f;

other = 1;
flag[64] = t;
wh(flag[64] == t);
flag[64] = f;

other = 0;
flag[65] = t;
wh(flag[65] == t);
flag[65] = f;

other = 1;
flag[66] = t;
wh(flag[66] == t);
flag[66] = f;

other = 0;
flag[67] = t;
wh(flag[67] == t);
flag[67] = f;

other = 1;
flag[68] = t;
wh(flag[68] == t);
flag[68] = f;

other = 0;
flag[69] = t;
wh(flag[69] == t);
flag[69] = f;

other = 1;
flag[70] = t;
wh(flag[70] == t);
flag[70] = f;

other = 0;
flag[71] = t;
wh(flag[71] == t);
flag[71] = f;

other = 1;
flag[72] = t;
wh(flag[72] == t);
flag[72] = f;

other = 0;
flag[73] = t;
wh(flag[73] == t);
flag[73] = f;

other = 1;
flag[74] = t;
wh(flag[74] == t);
flag[74] = f;

other = 0;
flag[75] = t;
wh(flag[75] == t);
flag[75] = f;

other = 1;
flag[76] = t;
wh(flag[76] == t);
flag[76] = f;

other = 0;
flag[77] = t;
wh(flag[77] == t);
flag[77] = f;

other = 1;
flag[78] = t;
wh(flag[78] == t);
flag[78] = f;

other = 0;
flag[79] = t;
wh(flag[79] == t);
flag[79] = f;

other = 1;
flag[80] = t;
wh(flag[80] == t);
flag[80] = f;

other = 0;
flag[81] = t;
wh(flag[81] == t);
flag[81] = f;

other = 1;
flag[82] = t;
wh(flag[82] == t);
flag[82] = f;

other = 0;
flag[83] = t;
wh(flag[83] == t);
flag[83] = f;

other = 1;
flag[84] = t;
wh(flag[84] == t);
flag[84] = f;

other = 0;
flag[85] = t;
wh(flag[85] == t);
flag[85] = f;

other = 1;
flag[86] = t;
wh(flag[86] == t);
flag[86] = f;

other = 0;
flag[87] = t;
wh(flag[87] == t);
flag[87] = f;

other = 1;
flag[88] = t;
wh(flag[88] == t);
flag[88] = f;

other = 0;
flag[89] = t;
wh(flag[89] == t);
flag[89] = f;

other = 1;
flag[90] = t;
wh(flag[90] == t);
flag[90] = f;

other = 0;
flag[91] = t;
wh(flag[91] == t);
flag[91] = f;

other = 1;
flag[92] = t;
wh(flag[92] == t);
flag[92] = f;

other = 0;
flag[93] = t;
wh(flag[93] == t);
flag[93] = f;

other = 1;
flag[94] = t;
wh(flag[94] == t);
flag[94] = f;

other = 0;
flag[95] = t;
wh(flag[95] == t);
flag[95] = f;

other = 1;
flag[96] = t;
wh(flag[96] == t);
flag[96] = f;

other = 0;
flag[97] = t;
wh(flag[97] == t);
flag[97] = f;

other = 1;
flag[98] = t;
wh(flag[98] == t);
flag[98] = f;

other = 0;
flag[99] = t;
wh(flag[99] == t);
flag[99] = f;

other = 1;
flag[100] = t;
wh(flag[100] == t);
flag[100] = f;

other = 0;
flag[101] = t;
wh(flag[101] == t);
flag[101] = f;

other = 1;
flag[102] = t;
wh(flag[102] == t);
flag[102] = f;

other = 0;
flag[103] = t;
wh(flag[103] == t);
flag[103] = f;

other = 1;
flag[104] = t;
wh(flag[104] == t);
flag[104] = f;

other = 0;
flag[105] = t;
wh(flag[105] == t);
flag[105] = f;

other = 1;
flag[106] = t;
wh(flag[106] == t);
flag[106] = f;

other = 0;
flag[107] = t;
wh(flag[107] == t);
flag[107] = f;

other = 1;
flag[108] = t;
wh(flag[108] == t);
flag[108] = f;

other = 0;
flag[109] = t;
wh(flag[109] == t);
flag[109] = f;

other = 1;
flag[110] = t;
wh(flag[110] == t);
flag[110] = f;

other = 0;
flag[111] = t;
wh(flag[111] == t);
flag[111] = f;

other = 1;
flag[112] = t;
wh(flag[112] == t);
flag[112] = f;

other = 0;
flag[113] = t;
wh(flag[113] == t);
flag[113] = f;

other = 1;
flag[114] = t;
wh(flag[114] == t);
flag[114] = f;

other = 0;
flag[115] = t;
wh(flag[115] == t);
flag[115] = f;

other = 1;
flag[116] = t;
wh(flag[116] == t);
flag[116] = f;

other = 0;
flag[117] = t;
wh(flag[117] == t);
flag[117] = f;

other = 1;
flag[118] = t;
wh(flag[118] == t);
flag[118] = f;

other = 0;
flag[119] = t;
wh(flag[119] == t);
flag[119] = f;

other = 1;
flag[120] = t;
wh(flag[120] == t);
flag[120] = f;

other = 0;
flag[121] = t;
wh(flag[121] == t);
flag[121] = f;

other = 1;
flag[122] = t;
wh(flag[122] == t);
flag[122] = f;

other = 0;
flag[123] = t;
wh(flag[123] == t);
flag[123] = f;

other = 1;
flag[124] = t;
wh(flag[124] == t);
flag[124] = f;

other = 0;
flag[125] = t;
wh(flag[125] == t);
flag[125] = f;

other = 1;
flag[126] = t;
wh(flag[126] == t);
flag[126] = f;

other = 0;
flag[127] = t;
wh(flag[127] == t);
flag[127] = f;

other = 1;
flag[128] = t;
wh(flag[128] == t);
flag[128] = f;

other = 0;
flag[129] = t;
wh(flag[129] == t);
flag[129] = f;

other = 1;
flag[130] = t;
wh(flag[130] == t);
flag[130] = f;

other = 0;
flag[131] = t;
wh(flag[131] == t);
flag[131] = f;

other = 1;
flag[132] = t;
wh(flag[132] == t);
flag[132] = f;

other = 0;
flag[133] = t;
wh(flag[133] == t);
flag[133] = f;

other = 1;
flag[134] = t;
wh(flag[134] == t);
flag[134] = f;

other = 0;
flag[135] = t;
wh(flag[135] == t);
flag[135] = f;

other = 1;
flag[136] = t;
wh(flag[136] == t);
flag[136] = f;

other = 0;
flag[137] = t;
wh(flag[137] == t);
flag[137] = f;

other = 1;
flag[138] = t;
wh(flag[138] == t);
flag[138] = f;

other = 0;
flag[139] = t;
wh(flag[139] == t);
flag[139] = f;

other = 1;
flag[140] = t;
wh(flag[140] == t);
flag[140] = f;

other = 0;
flag[141] = t;
wh(flag[141] == t);
flag[141] = f;

other = 1;
flag[142] = t;
wh(flag[142] == t);
flag[142] = f;

other = 0;
flag[143]

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
```

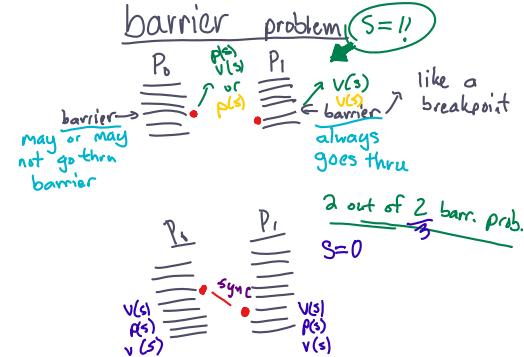
```
...
```

```
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.

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
 - 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.

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.

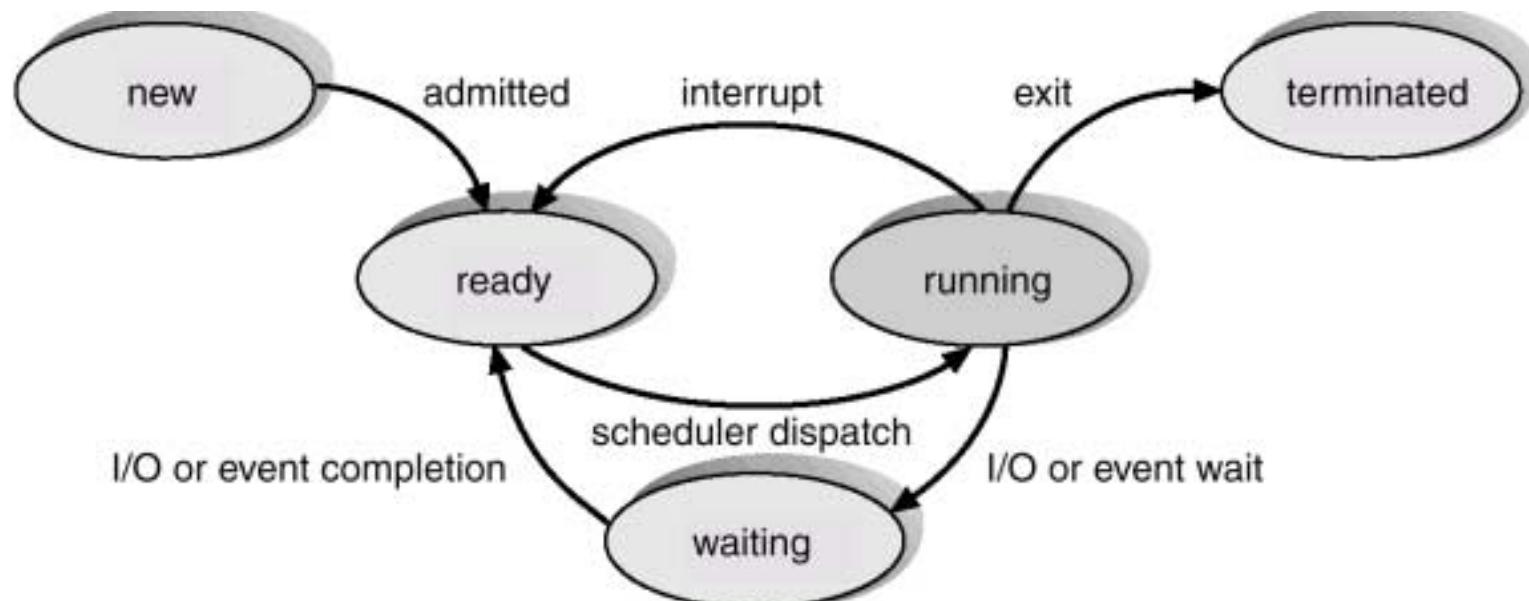
Buffering

- 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.

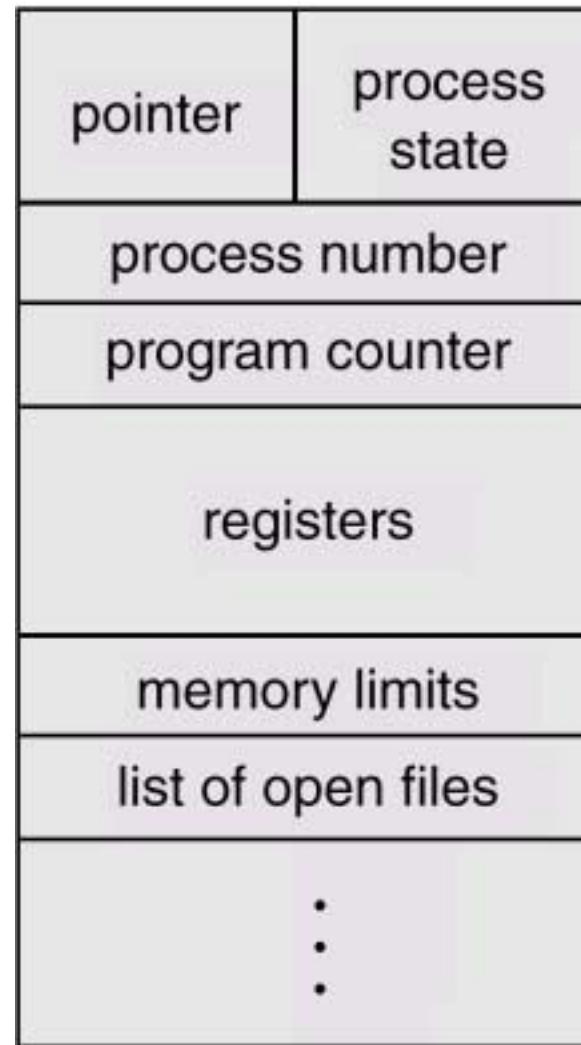
Exception Conditions – Error Recovery

- Process terminates
- Lost messages
- Scrambled Messages

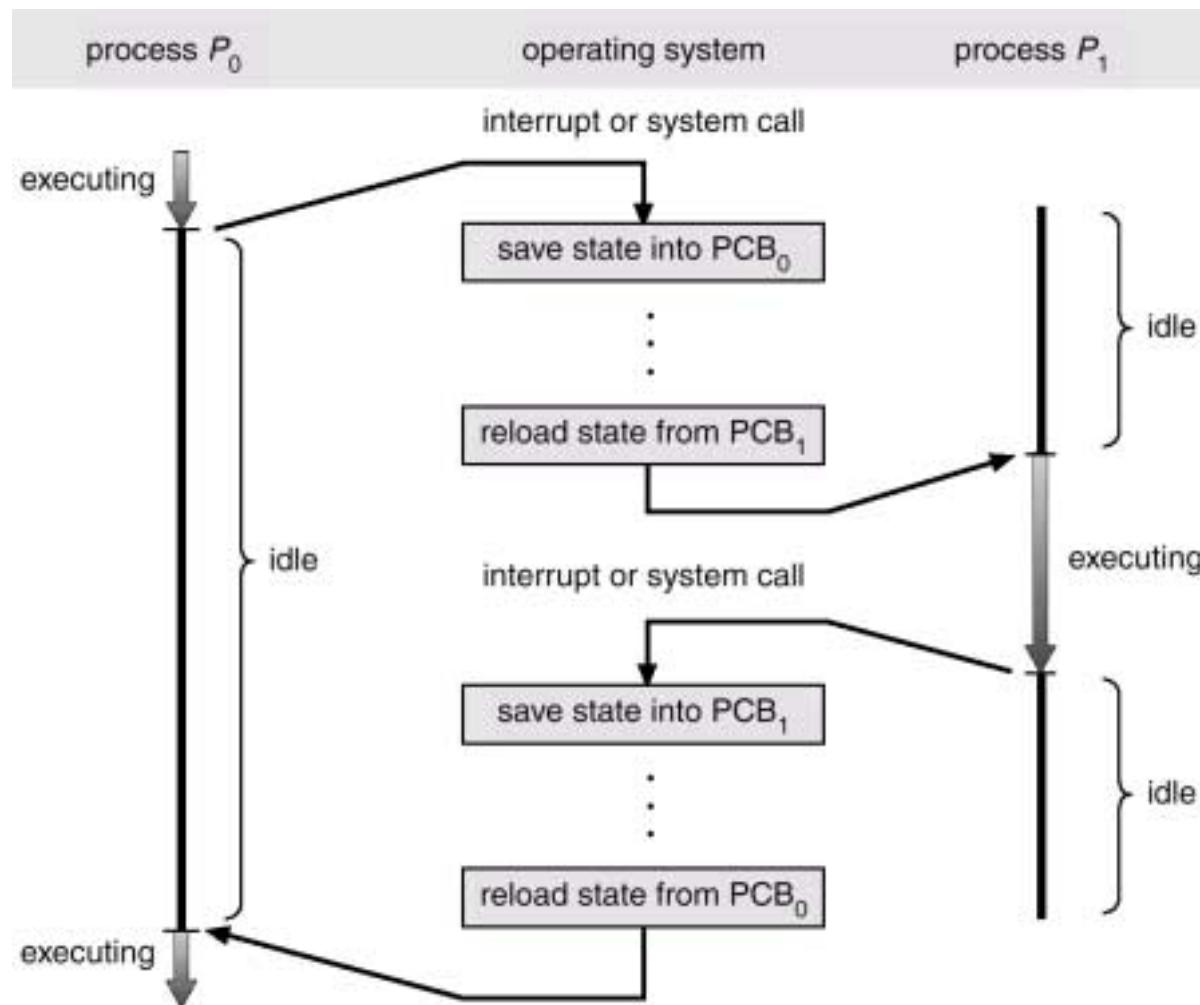
4.01



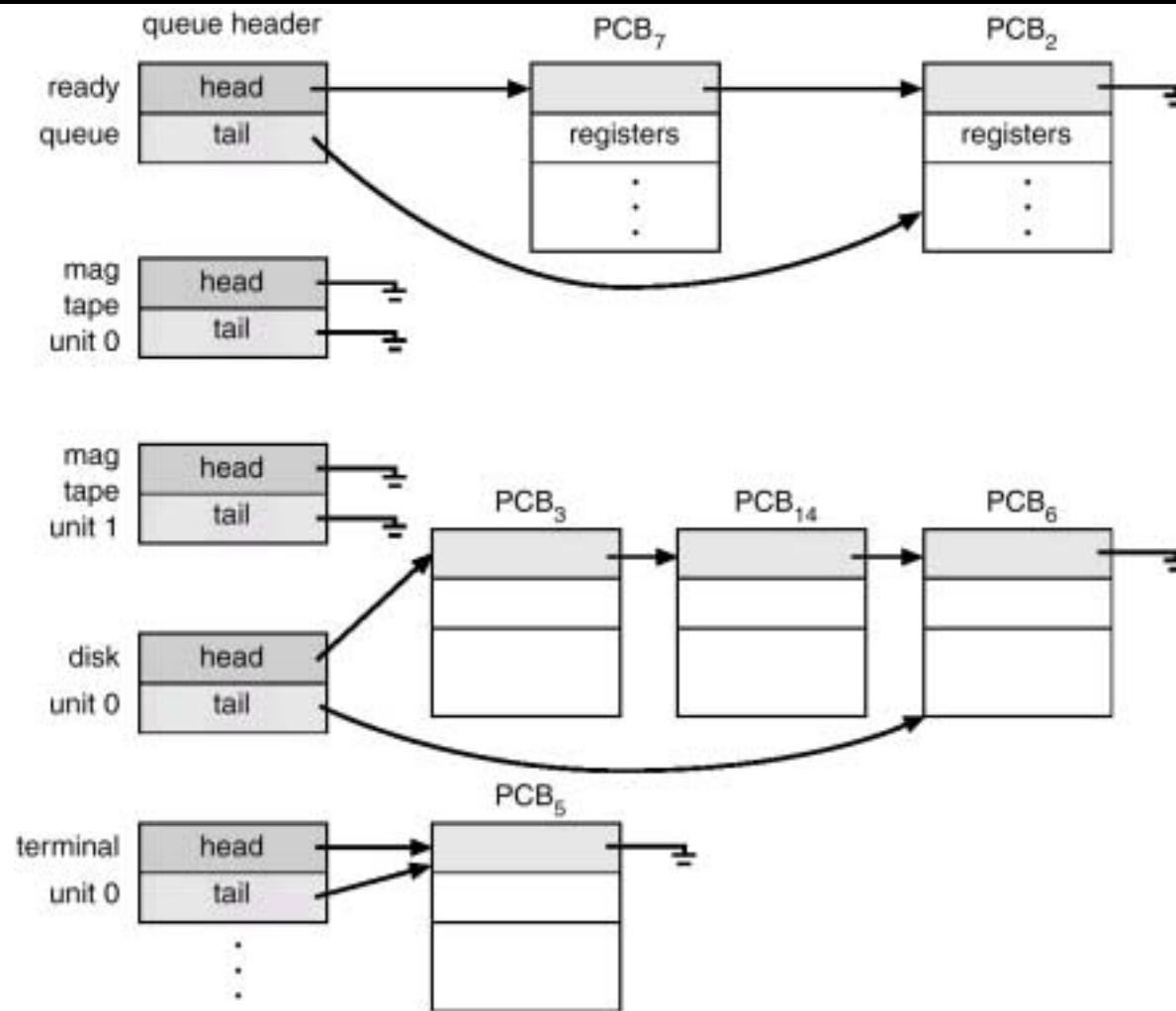
4.02



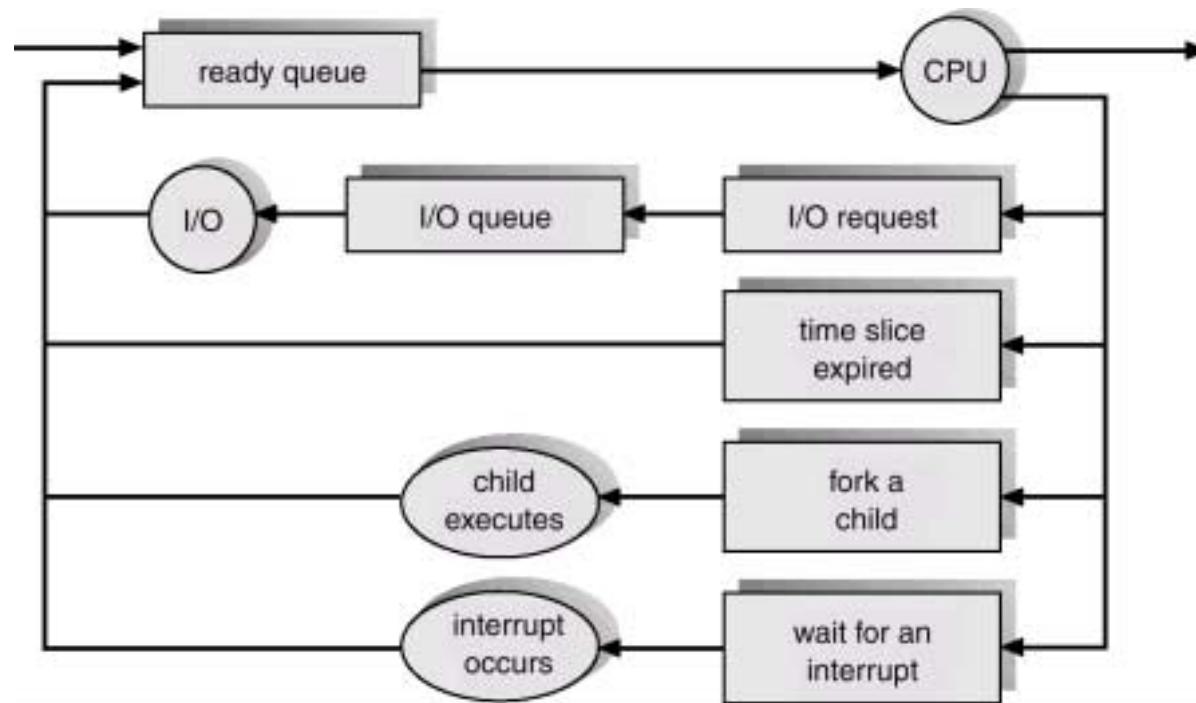
4.03



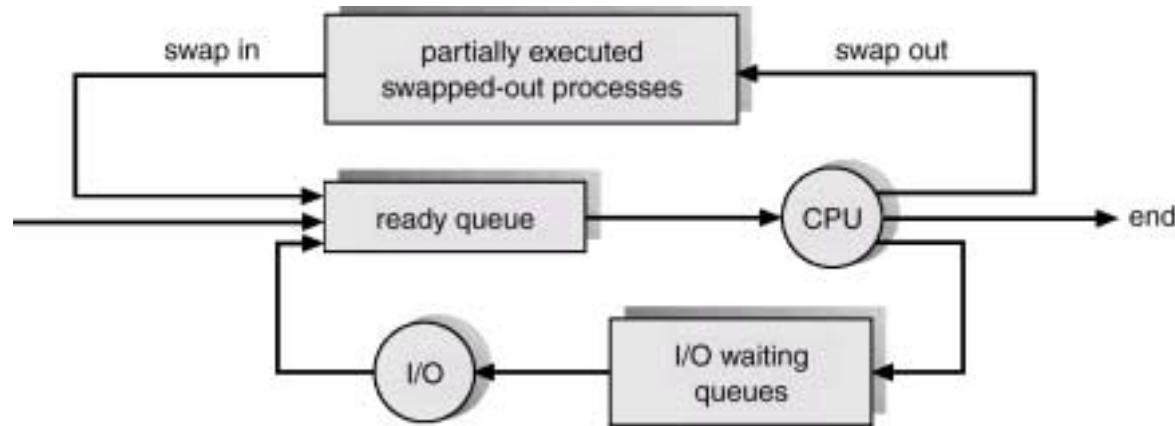
4.04



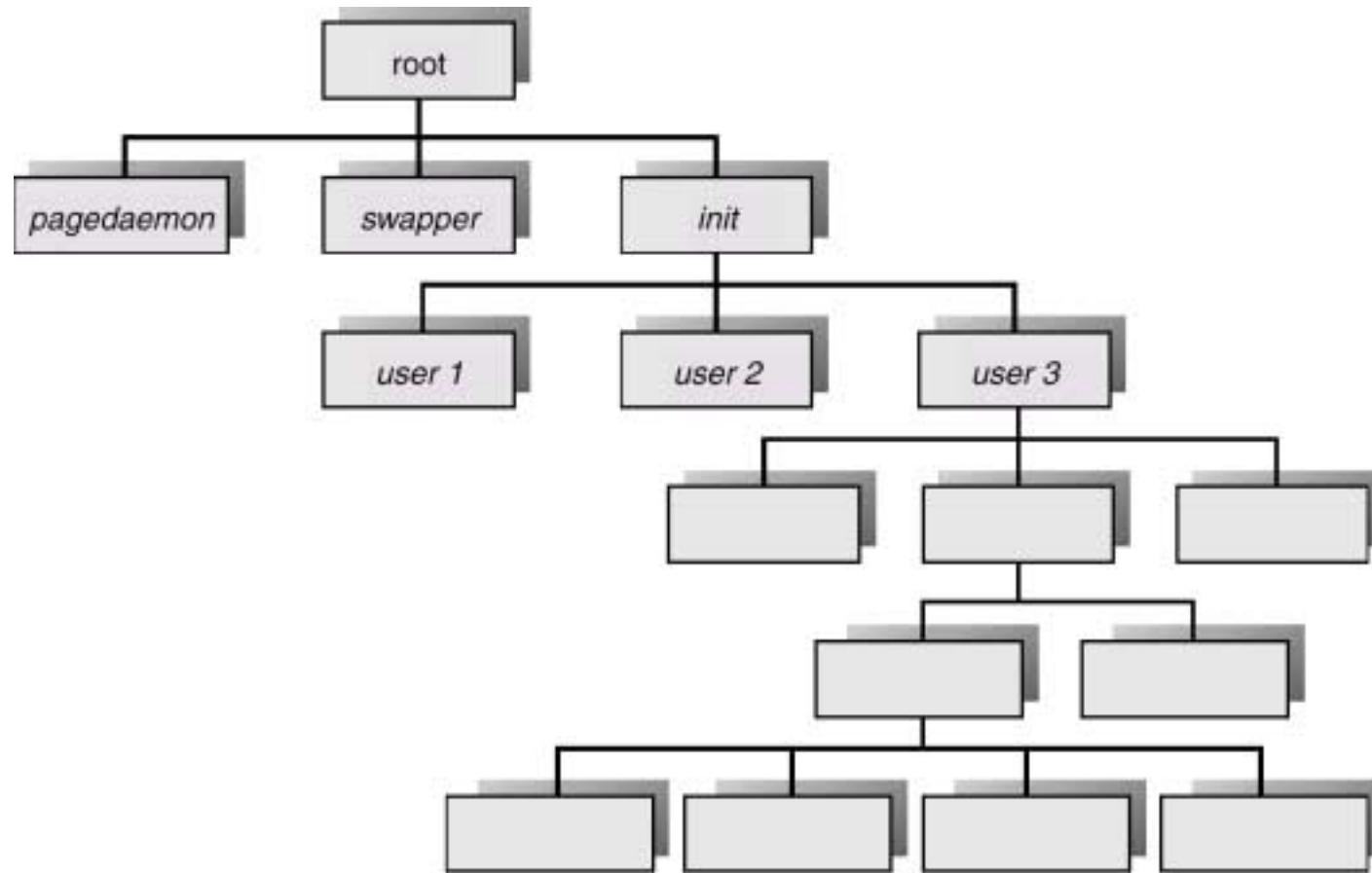
4.05



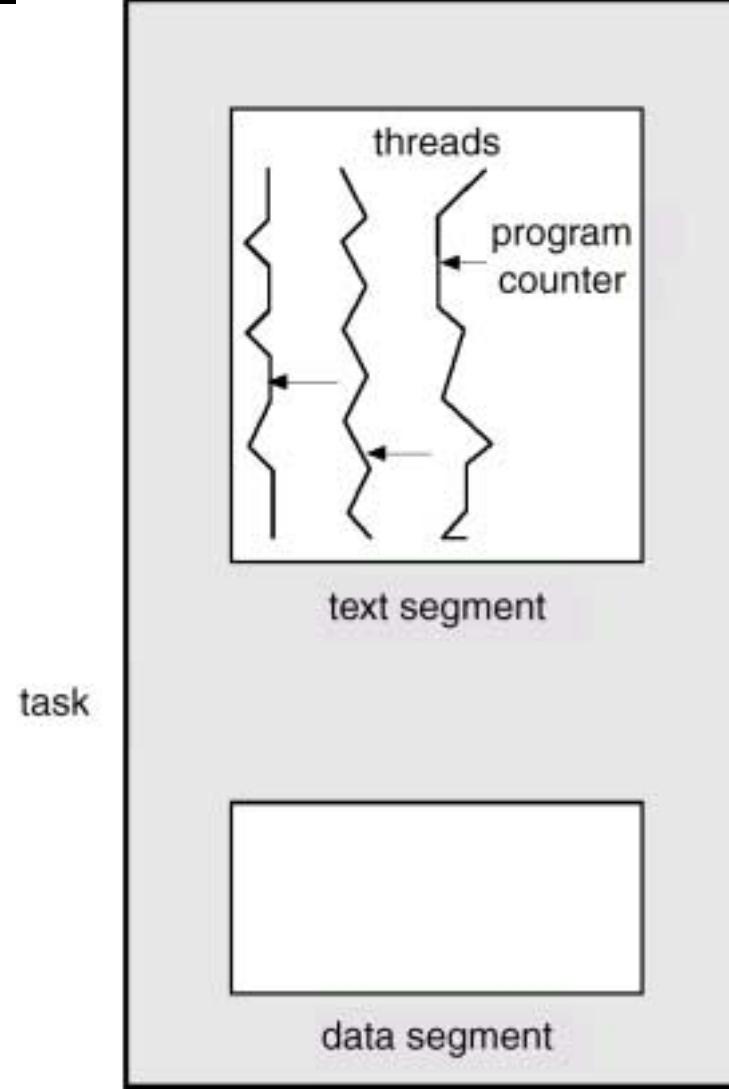
4.06



4.07



4.08



4.09

