ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG VIỆT - HÀN
Vietnam - Korea University of Information and Communication Technology

CHAPTER 2 - PROCESSES AND THREADS

Processes
Threads
Scheduling
Interprocess communication

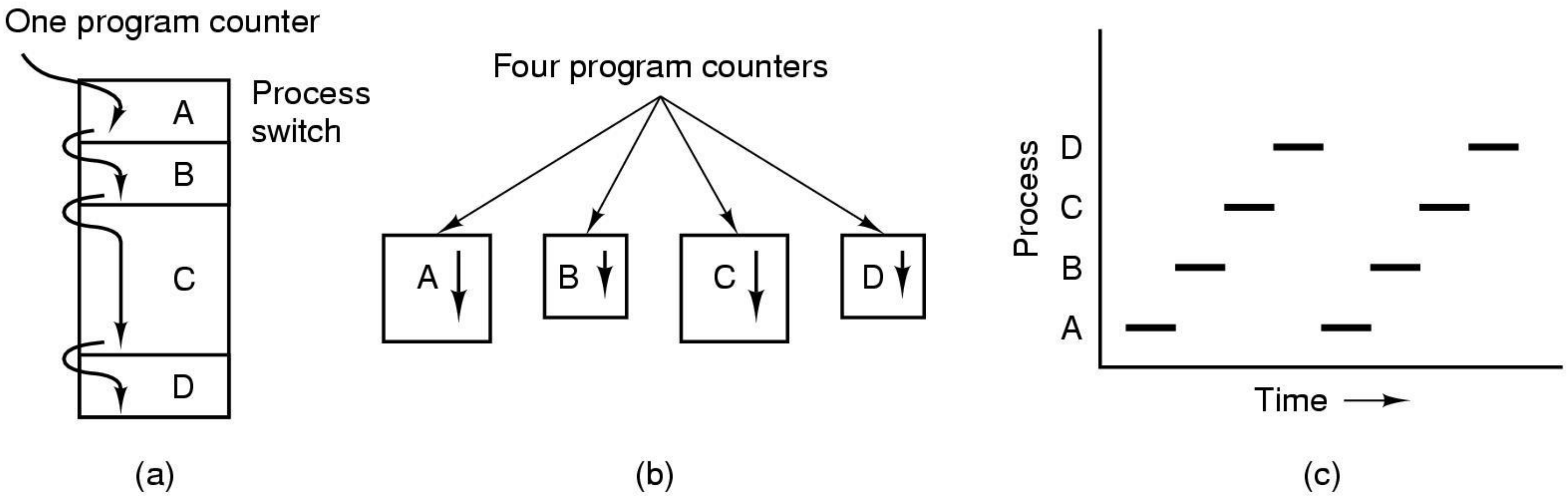
Processes

Users, Programs

- Users have accounts on the system
- Users launch programs
  - Many users may launch the same program
  - One user may launch many instances of the same program
- Then what is a process?

Processes
The Process Model

- Multiprogramming of four programs
- Conceptual model of 4 independent, sequential processes
- Only one program active at any instant



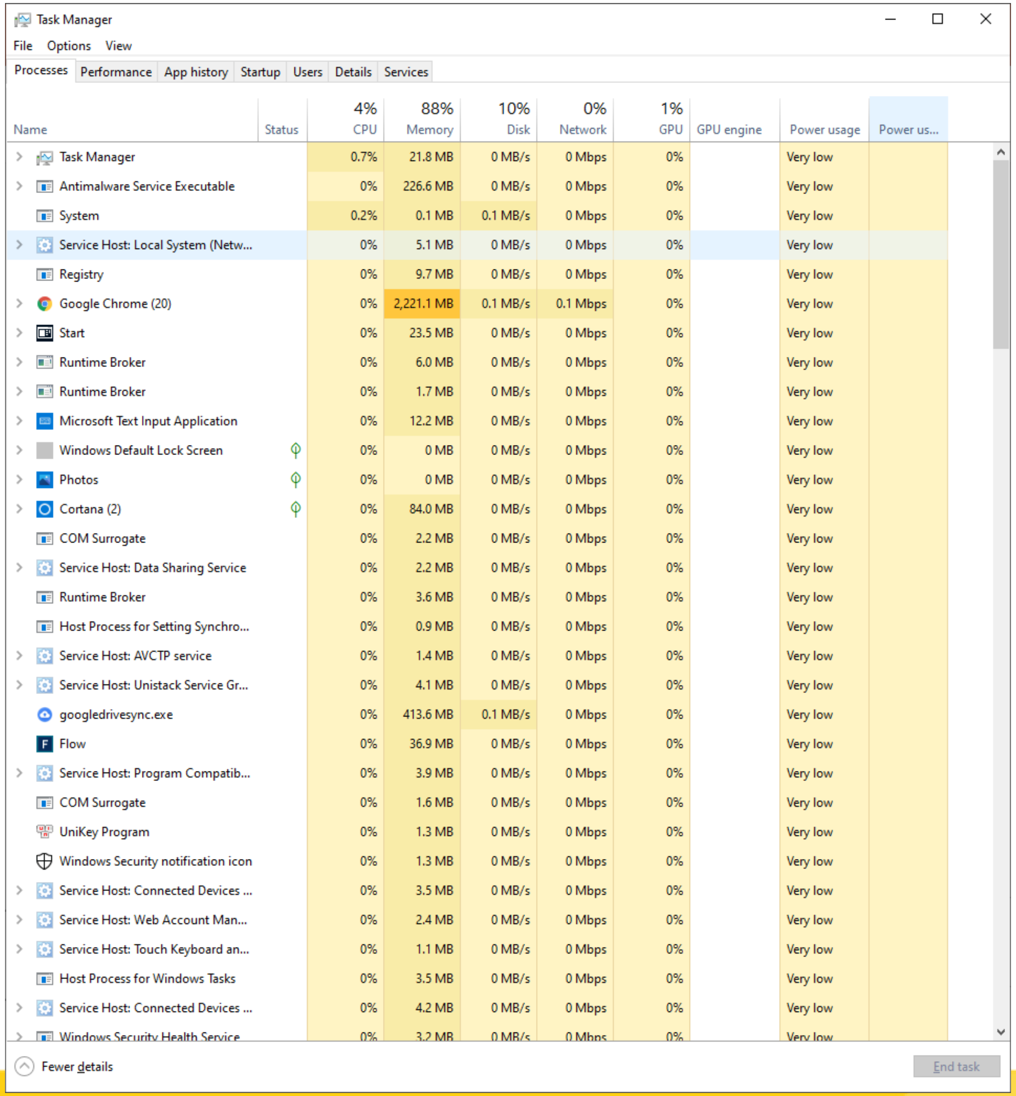
Processes
Process Concept

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
- Process – a program in execution; process execution must progress in sequential fashion
- A process resources includes:
  - Address space (text segment, data segment)
  - CPU (virtual)
    - ✓ Program counter
    - ✓ Registers
    - ✓ Stack
  - Other resource (open files, child processes, etc.)

MacOS example: Activity monitor

Process Name	% CPU	CPU Time	Threads	Idle Wake Ups	PID	User
WindowServer	8.3	6:21:59.24	4	29	187	_windowserver
hidd	4.5	36:01.16	6	0	115	_hidd
kernel_task	3.8	5:34:15.29	250	185	0	root
screencapture	2.6	0.38	2	0	33124	ding
Activity Monitor	2.6	1:59:58.29	6	2	617	ding
Acrobat	2.0	12:46.87	20	90	31159	ding
Microsoft PowerPoint	1.2	2:06:48.28	13	25	2969	ding
distnoted	1.1	2:04:40.47	11	0	283	ding
sysmond	0.8	1:35:33.77	3	1	274	root
distnoted	0.5	1:00:52.69	8	0	118	_distnote
Google Chrome Helper	0.4	13:55.32	18	8	23466	ding
CLion	0.4	1:45:20.67	43	46	2567	ding
Dropbox	0.3	53:34.30	191	4	13921	ding
com.apple.AmbientDi...	0.3	8:00.60	5	0	220	root
iTerm2	0.3	30:50.54	7	5	2610	ding
Adobe Reader and A...	0.2	18.24	5	2	32864	ding
Google Chrome Helper	0.2	10:05.63	19	2	23697	ding
AdobeCrashDaemon	0.2	22:25.51	1	1	469	ding
launchservicesd	0.2	15:29.24	3	0	98	root
Google Chrome	0.1	4:11:59.24	41	0	13353	ding

Windows OS example: Activity monitor



Task Manager

File Options View

Processes | Performance | App history | Startup | Users | Details | Services

Name	Status	4% CPU	88% Memory	10% Disk	0% Network	1% GPU	GPU engine	Power usage	Power us...
Task Manager		0.7%	21.8 MB	0 MB/s	0 Mbps	0%		Very low	
Antimalware Service Executable		0%	226.6 MB	0 MB/s	0 Mbps	0%		Very low	
System		0.2%	0.1 MB	0.1 MB/s	0 Mbps	0%		Very low	
Service Host: Local System (Netw...		0%	5.1 MB	0 MB/s	0 Mbps	0%		Very low	
Registry		0%	9.7 MB	0 MB/s	0 Mbps	0%		Very low	
Google Chrome (20)		0%	2,221.1 MB	0.1 MB/s	0.1 Mbps	0%		Very low	
Start		0%	23.5 MB	0 MB/s	0 Mbps	0%		Very low	
Runtime Broker		0%	6.0 MB	0 MB/s	0 Mbps	0%		Very low	
Runtime Broker		0%	1.7 MB	0 MB/s	0 Mbps	0%		Very low	
Microsoft Text Input Application		0%	12.2 MB	0 MB/s	0 Mbps	0%		Very low	
Windows Default Lock Screen		0%	0 MB	0 MB/s	0 Mbps	0%		Very low	
Photos		0%	0 MB	0 MB/s	0 Mbps	0%		Very low	
Cortana (2)		0%	84.0 MB	0 MB/s	0 Mbps	0%		Very low	
COM Surrogate		0%	2.2 MB	0 MB/s	0 Mbps	0%		Very low	
Service Host: Data Sharing Service		0%	2.2 MB	0 MB/s	0 Mbps	0%		Very low	
Runtime Broker		0%	3.6 MB	0 MB/s	0 Mbps	0%		Very low	
Host Process for Setting Synchro...		0%	0.9 MB	0 MB/s	0 Mbps	0%		Very low	
Service Host: AVCTP service		0%	1.4 MB	0 MB/s	0 Mbps	0%		Very low	
Service Host: Unistack Service Gr...		0%	4.1 MB	0 MB/s	0 Mbps	0%		Very low	
googledrivesync.exe		0%	413.6 MB	0.1 MB/s	0 Mbps	0%		Very low	
Flow		0%	36.9 MB	0 MB/s	0 Mbps	0%		Very low	
Service Host: Program Compatib...		0%	3.9 MB	0 MB/s	0 Mbps	0%		Very low	
COM Surrogate		0%	1.6 MB	0 MB/s	0 Mbps	0%		Very low	
UniKey Program		0%	1.3 MB	0 MB/s	0 Mbps	0%		Very low	
Windows Security notification icon		0%	1.3 MB	0 MB/s	0 Mbps	0%		Very low	
Service Host: Connected Devices ...		0%	3.5 MB	0 MB/s	0 Mbps	0%		Very low	
Service Host: Web Account Man...		0%	2.4 MB	0 MB/s	0 Mbps	0%		Very low	
Service Host: Touch Keyboard an...		0%	1.1 MB	0 MB/s	0 Mbps	0%		Very low	
Host Process for Windows Tasks		0%	3.5 MB	0 MB/s	0 Mbps	0%		Very low	
Service Host: Connected Devices ...		0%	4.2 MB	0 MB/s	0 Mbps	0%		Very low	
Windows Security Health Service		0%	3.2 MB	0 MB/s	0 Mbps	0%		Very low	

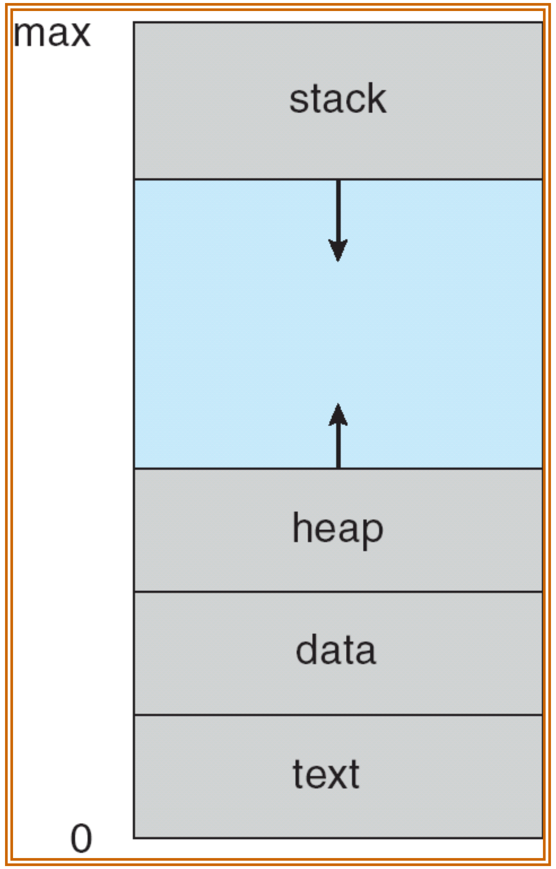
Fewer details

End task

So what is a process?

- A process is a program in execution
- It is one executing instance of a program
- It is separated from other instances
- It can start (“launch”) other processes
- It can be launched by them

Processes
Process in Memory



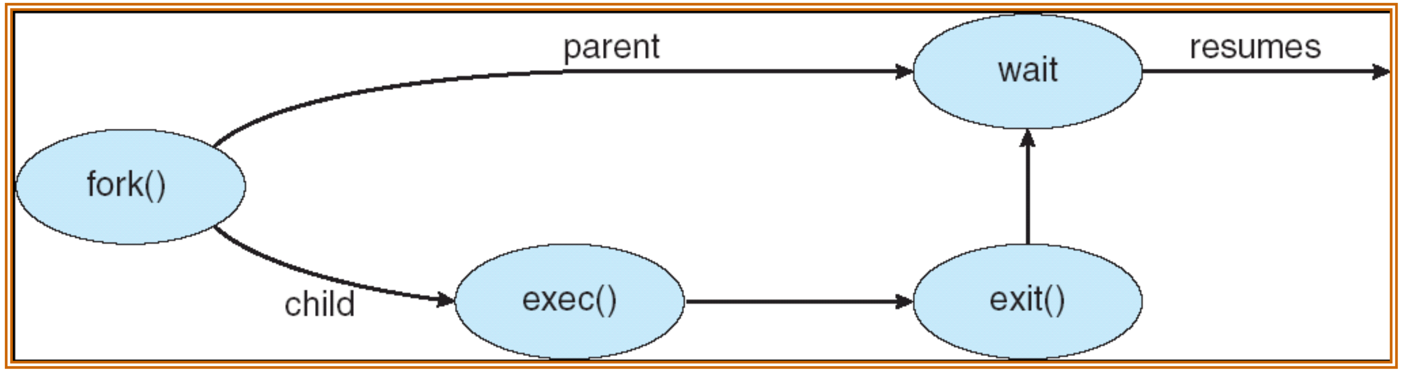
Processes
Process Creation (1)

- Principal events that cause process creation
  - System initialization
  - Execution of a process creation system Call
  - User request to create a new process
  - Initiation of a batch job

Processes
Process Creation (2)

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

Processes
Process Creation (3) : Example



Processes
Process Termination

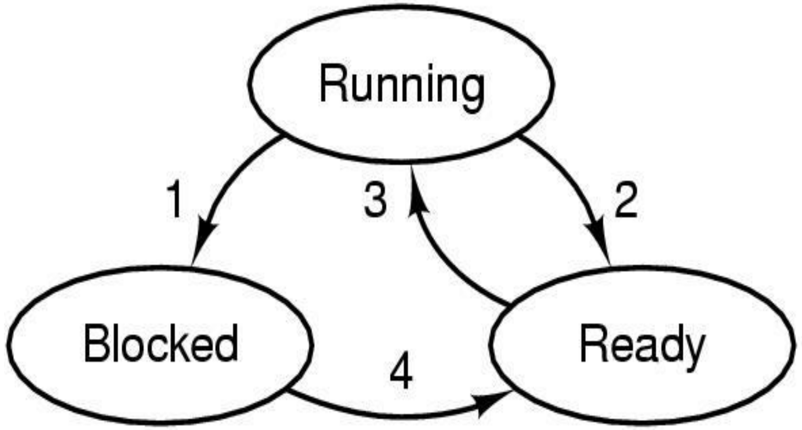
- Conditions which terminate processes
    - Normal exit (voluntary)
    - Fatal error (voluntary)
    - Error exit (involuntary)
    - Killed by another process (involuntary)

Processes
Process Hierarchies

- Parent creates a child process, child processes can create its own process
- Forms a hierarchy
  - UNIX calls this a "process group"
- Windows has no concept of process hierarchy
  - All processes are created equal

Processes
Process States (1)

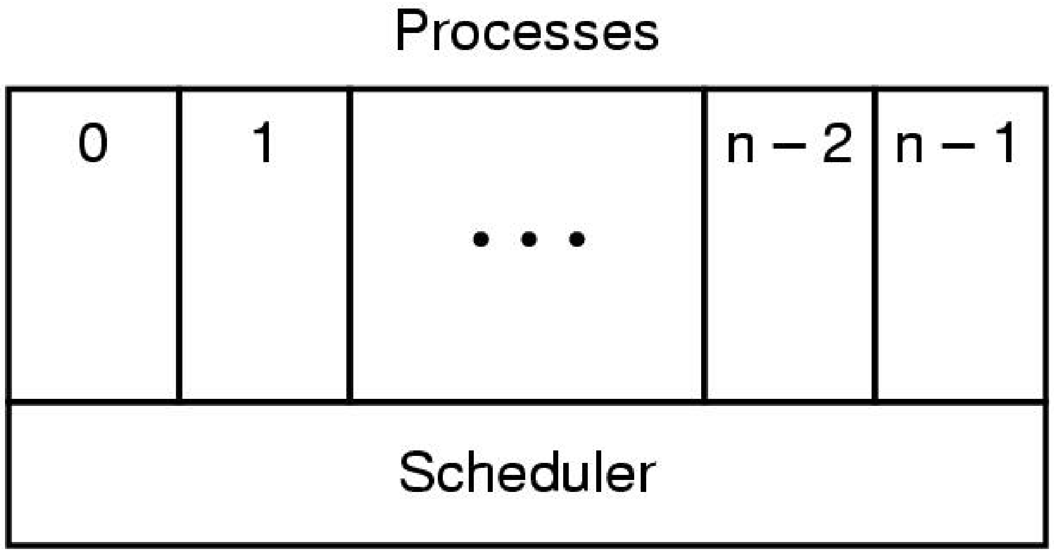
- Possible process states
  - Running: Executing instructions on the CPU
  - Blocked: Waiting for an event, e.g., I/O completion
  - Ready: Waiting to be assigned to the CPU
- Transitions between states shown



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Processes
Process States (2)

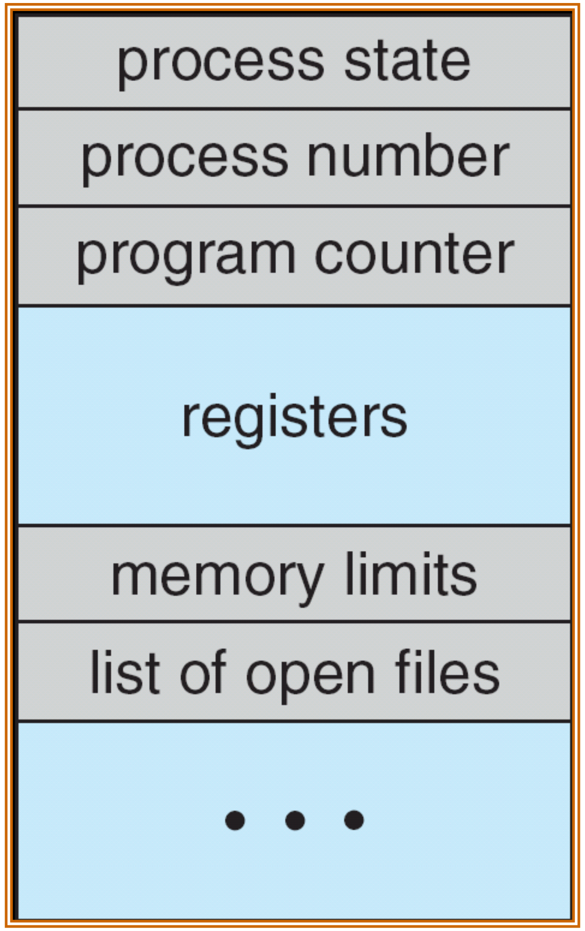
- Lowest layer of process-structured OS
  - handles interrupts, scheduling
- Above that layer are sequential processes



Questions

- What state do you think a process is in most of the time?
- For a uni-processor machine, how many processes can be in running state?
- Benefit of multi-core?

Processes
Process Control Block (PCB) (1)



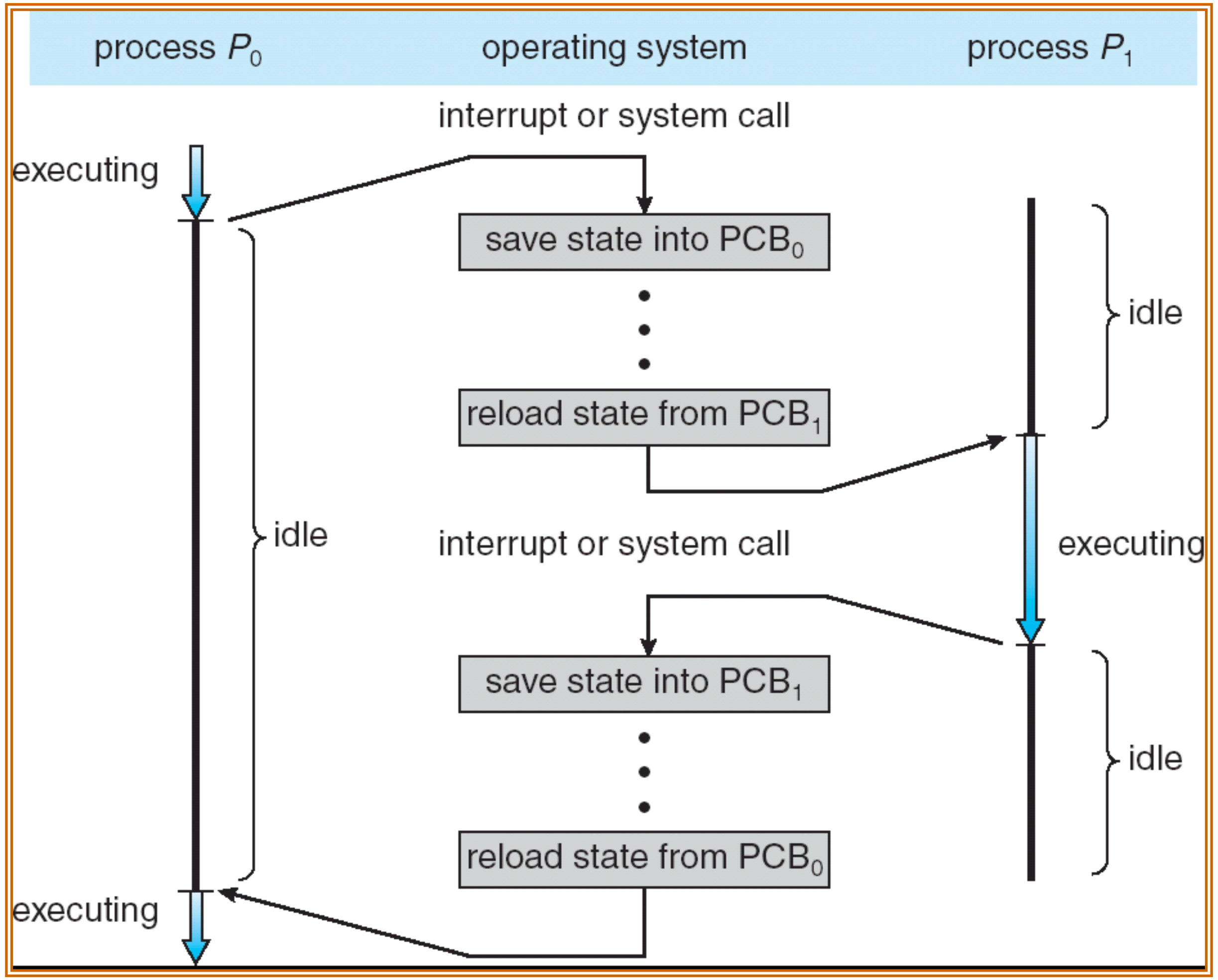
Processes
Process Control Block (PCB) (2)

- **Process State**
  - new, ready, running, waiting, terminated;
- **Program Counter**
  - the address of the next instruction to be executed for this process;
- **CPU Registers**
  - index registers, stack pointers, general purpose registers;
- **CPU Scheduling Information**
  - process priority;

Processes
Process Control Block (PCB) (3)

- Memory Management Information
  - base/limit information, virtual->physical mapping, etc
- Accounting Information
  - time limits, process number; owner
- I/O Status Information
  - list of I/O devices allocated to the process;
- An Address Space
  - memory space visible to one process

Processes
context switch



Processes
Implementation of Processes (1)

- Fields of a process table entry

Process management	Memory management	File management
Registers Program counter Program status word Stack pointer Process state Priority Scheduling parameters Process ID Parent process Process group Signals Time when process started CPU time used Children's CPU time Time of next alarm	Pointer to text segment Pointer to data segment Pointer to stack segment	Root directory Working directory File descriptors User ID Group ID

Processes
Implementation of Processes (2)

Now simultaneously start two instances of this program

- Myval 5
- Myval 6
- What will the outputs be?

```
int myval;
int main(int argc, char *argv[])
{
  myval = atoi(argv[1]);
  while (1)
   printf("myval is %d, loc 0x%lx\n", myval, (long) &myval);
}
```

Processes
Implementation of Processes (3)

Default D

```
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
myval is 5, loc 0x2030
```

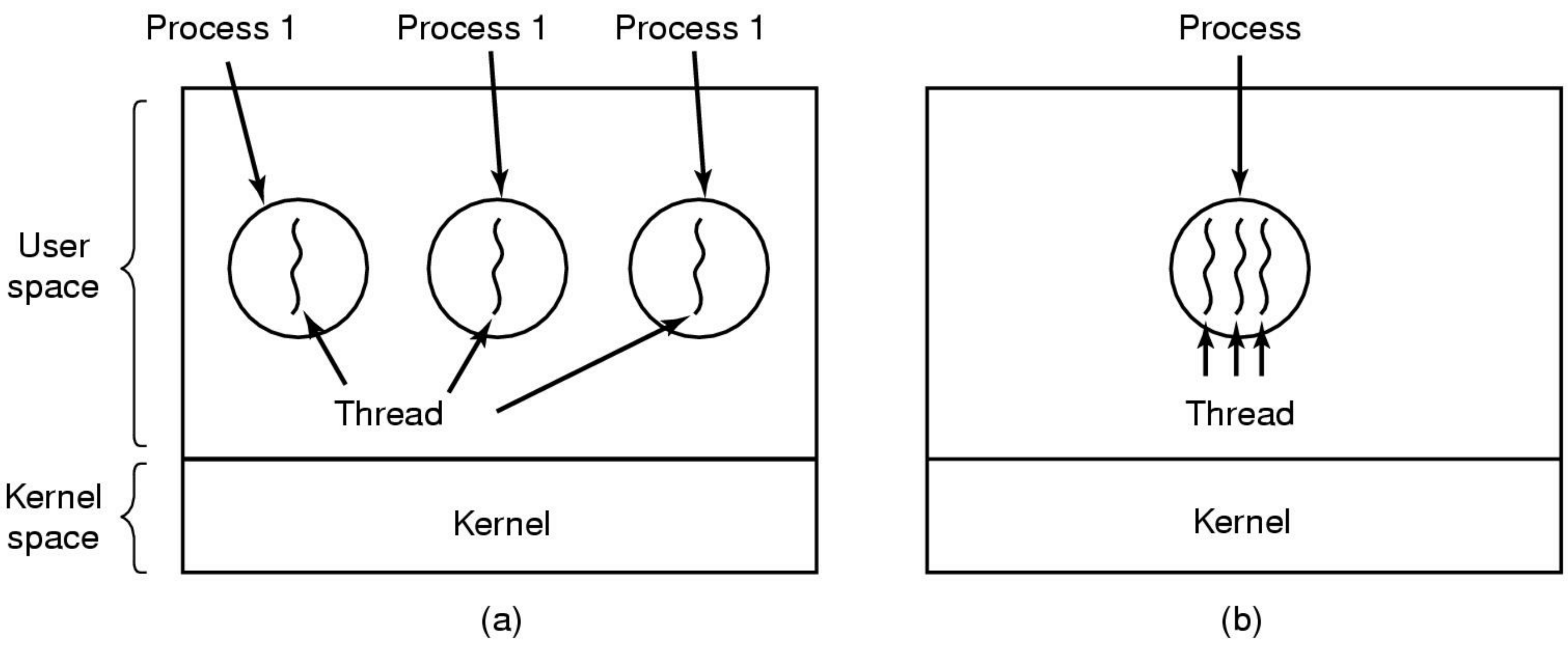
Thank

```
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
myval is 6, loc 0x2030
```

Threads

Threads
The Thread Model

- (a) Three processes each with one thread
- (b) One process with three threads



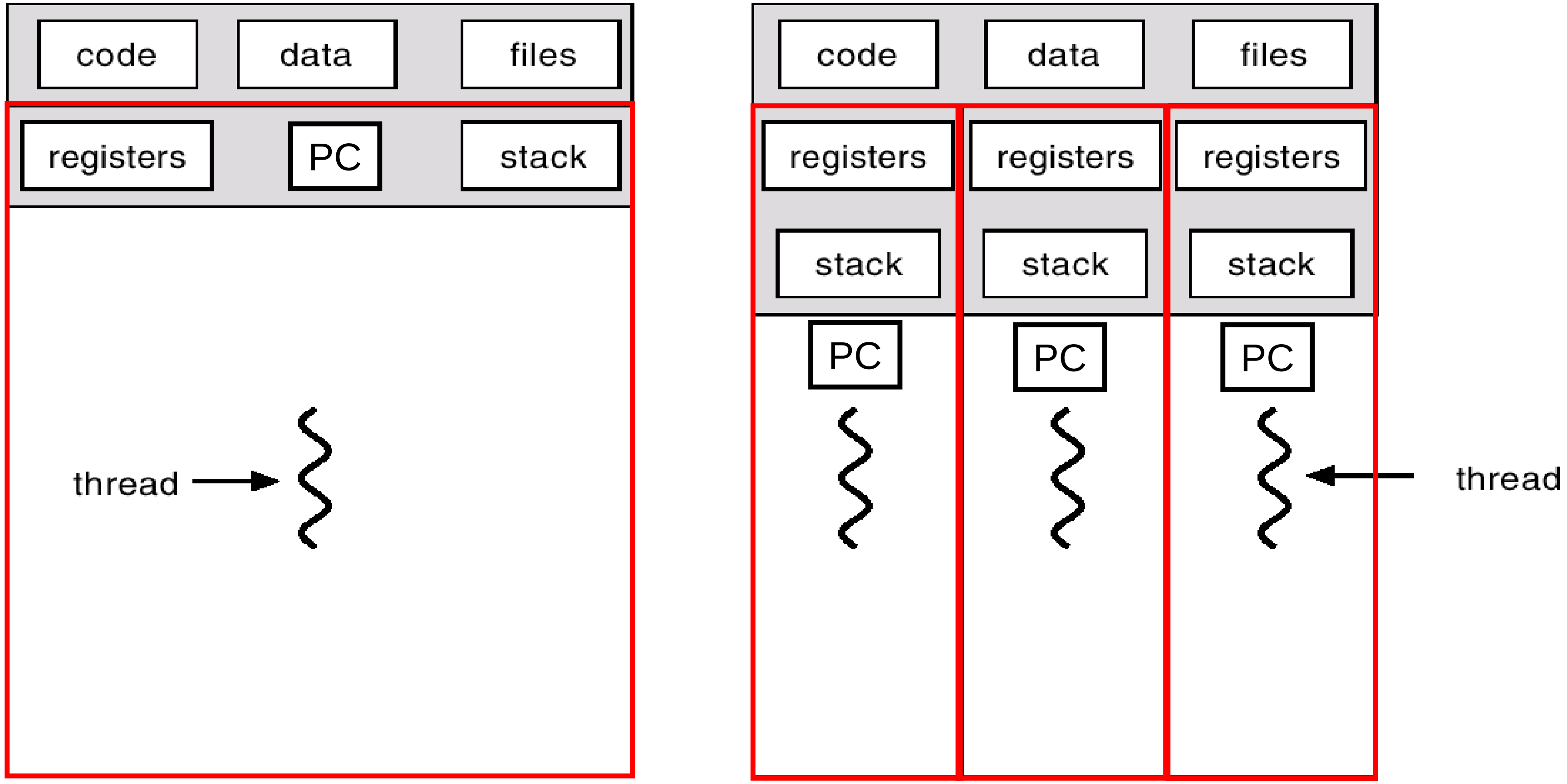
Threads
Process with single thread

- A process:
  - Address space (text section, data section)
  - Single thread of execution
    - ✓ Program counter
    - ✓ Registers
    - ✓ Stack
  - Other resource (open files, child processes, etc.)

Threads
Process with multiple threads

- Multiple threads of execution in the same environment of process
- Address space (text section, data section)
- Multiple threads of execution, each thread has private set:
  - ✓ Program counter
  - ✓ Registers
  - ✓ Stack
- Other resource (open files, child processes, etc.)

Single-threaded and Multithreaded



single-threaded

multithreaded

Threads
Items shared and Items private

- Items shared by all threads in a process
- Items private to each thread

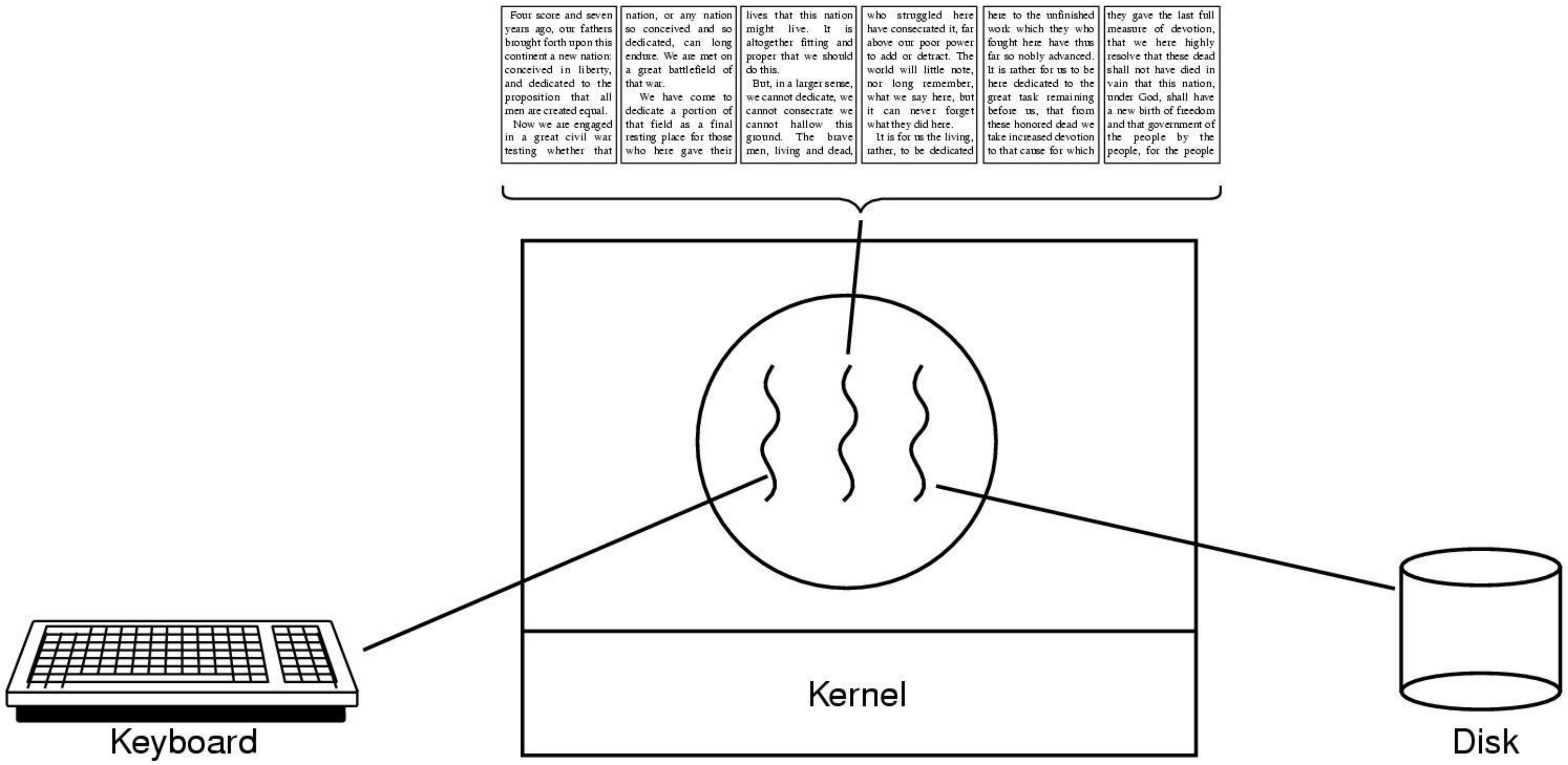
Per process items	Per thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

Threads Benefits

- Responsiveness
- Resource Sharing
- Economy
- Utilization of Multiprocessor Architectures

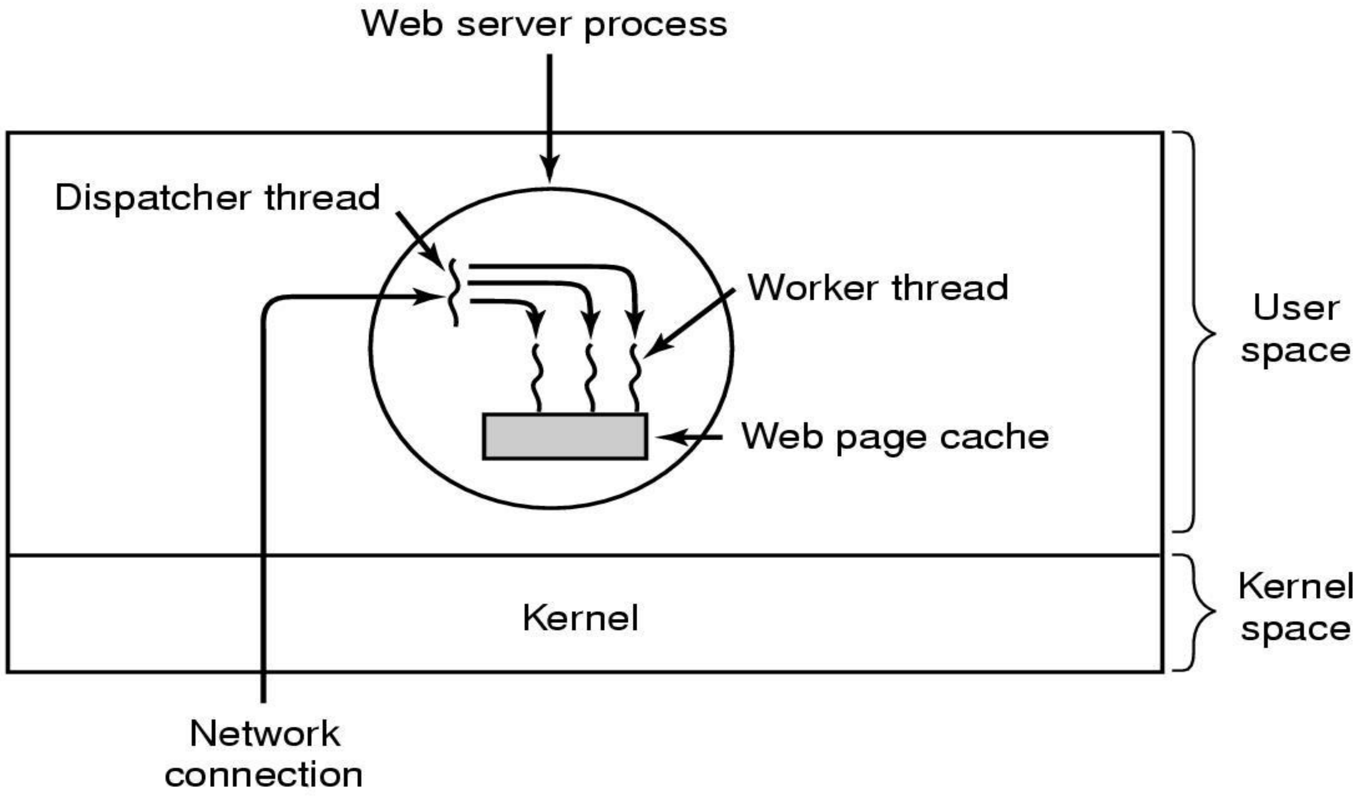
Threads
Thread Usage (1)

- A word processor with three threads



Threads
Thread Usage (2)

- A multithreaded Web server



Threads
Thread Usage (3)

- Rough outline of code for previous slide
  - (a) Dispatcher thread
  - (b) Worker thread

```
while (TRUE) {
  get_next_request(&buf);
  handoff_work(&buf);
}
```

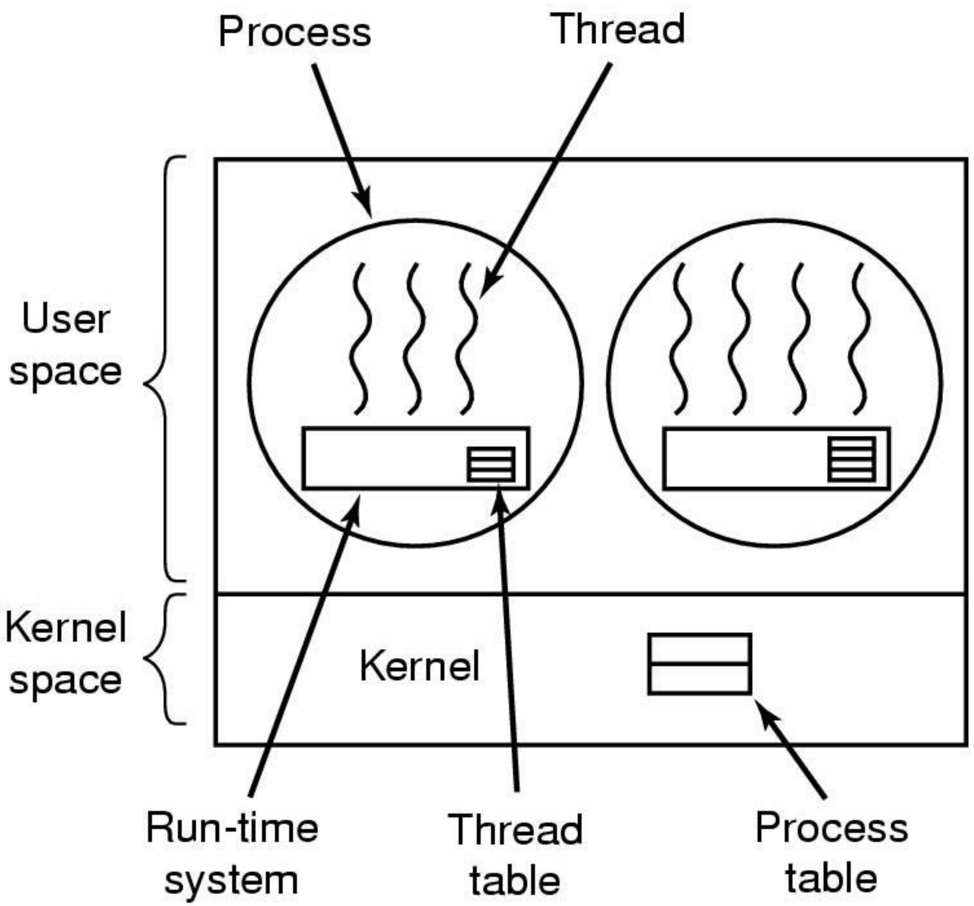
(a)

```
while (TRUE) {
  wait_for_work(&buf)
  look_for_page_in_cache(&buf, &page);
  if (page_not_in_cache(&page)
        read_page_from_disk(&buf, &page);
  return_page(&page);
}
```

(b)

Threads
Implementing Threads in User Space (1)

- A user-level threads package

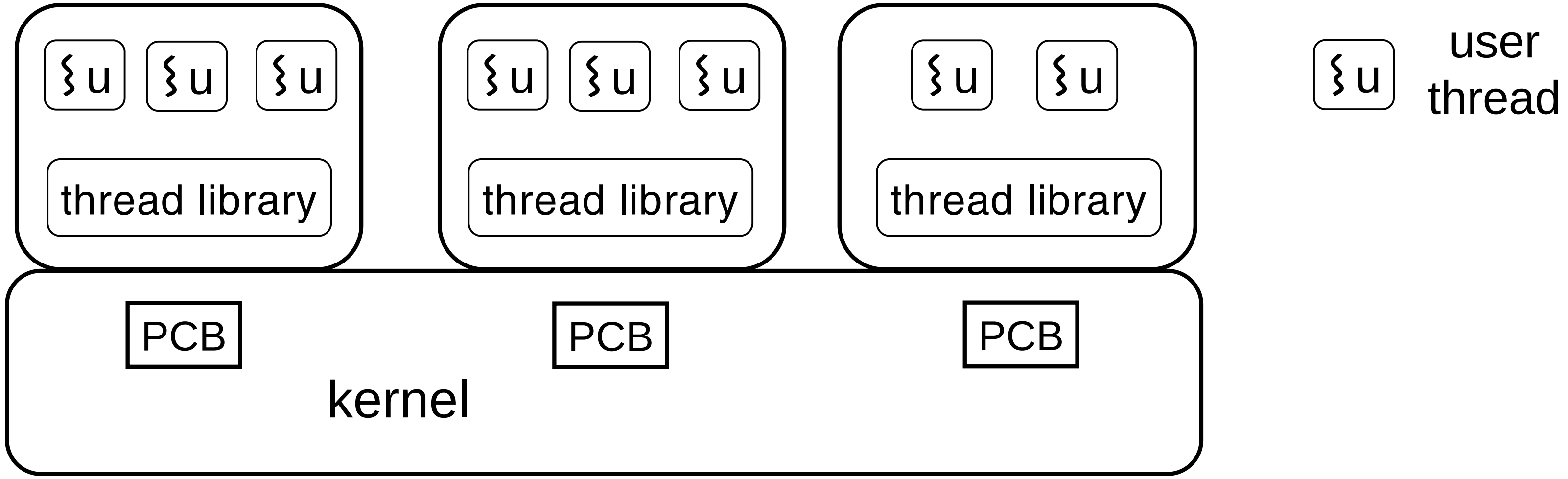


Threads
Implementing Threads in User Space (2)

- Thread library, (run-time system) in user space
  - thread_create
  - thread_exit
  - thread_wait
  - thread_yield (to voluntarily give up the CPU)
- Thread control block (TCB) (Thread Table) stores states of user thread (program counter, registers, stack)
- Kernel does not know the present of user thread

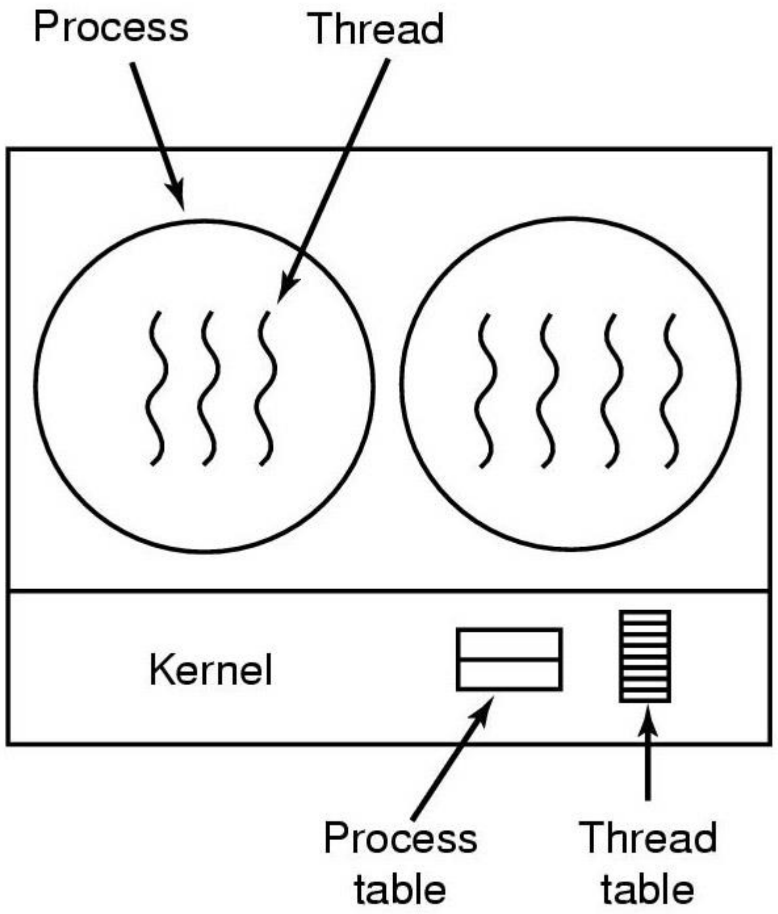
Threads
Implementing Threads in User Space (3)

- Traditional OS provide only one "kernel thread" presented by PCB for each process.
  - Blocking problem: If one user thread is blocked → the kernel thread is blocked → all other threads in process are blocked.



Threads
Implementing Threads in the Kernel (1)

- A threads package managed by the kernel

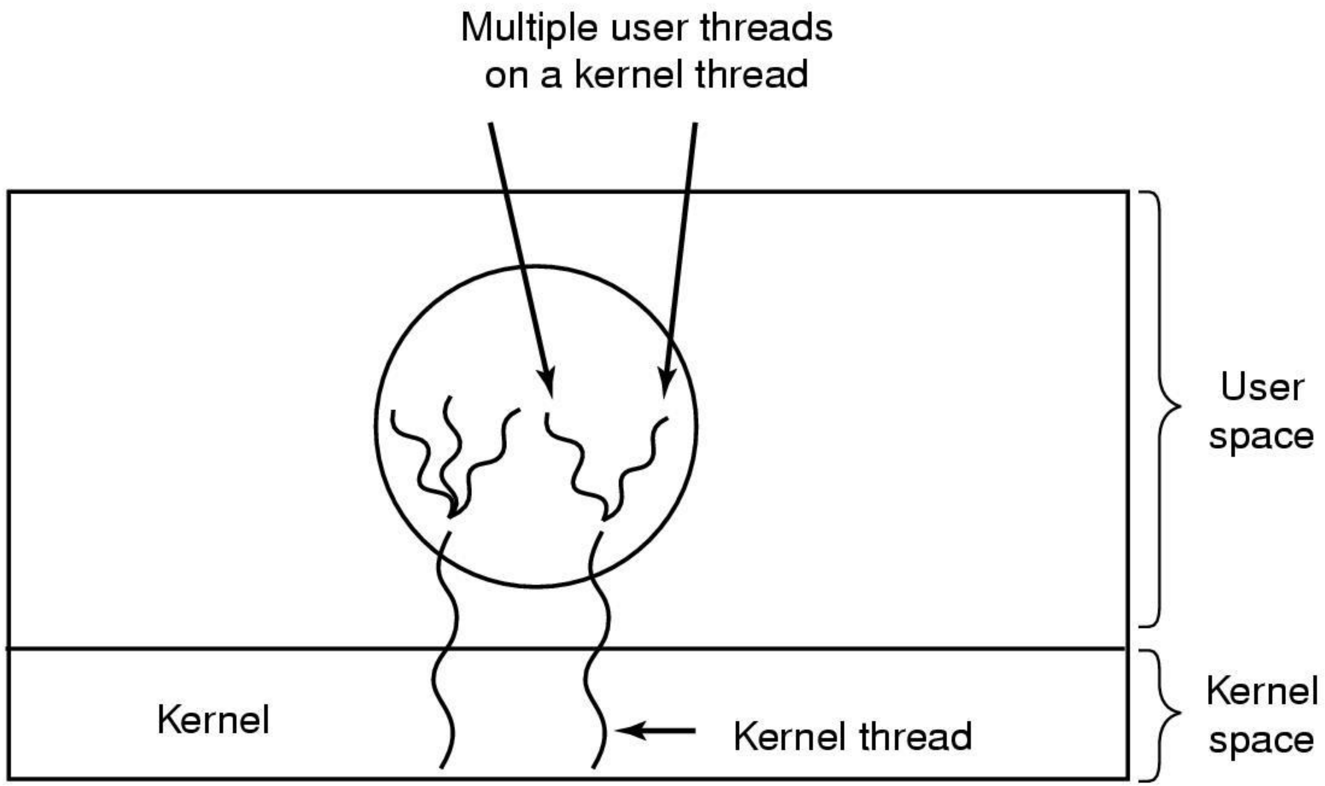


Threads
Implementing Threads in the Kernel (2)

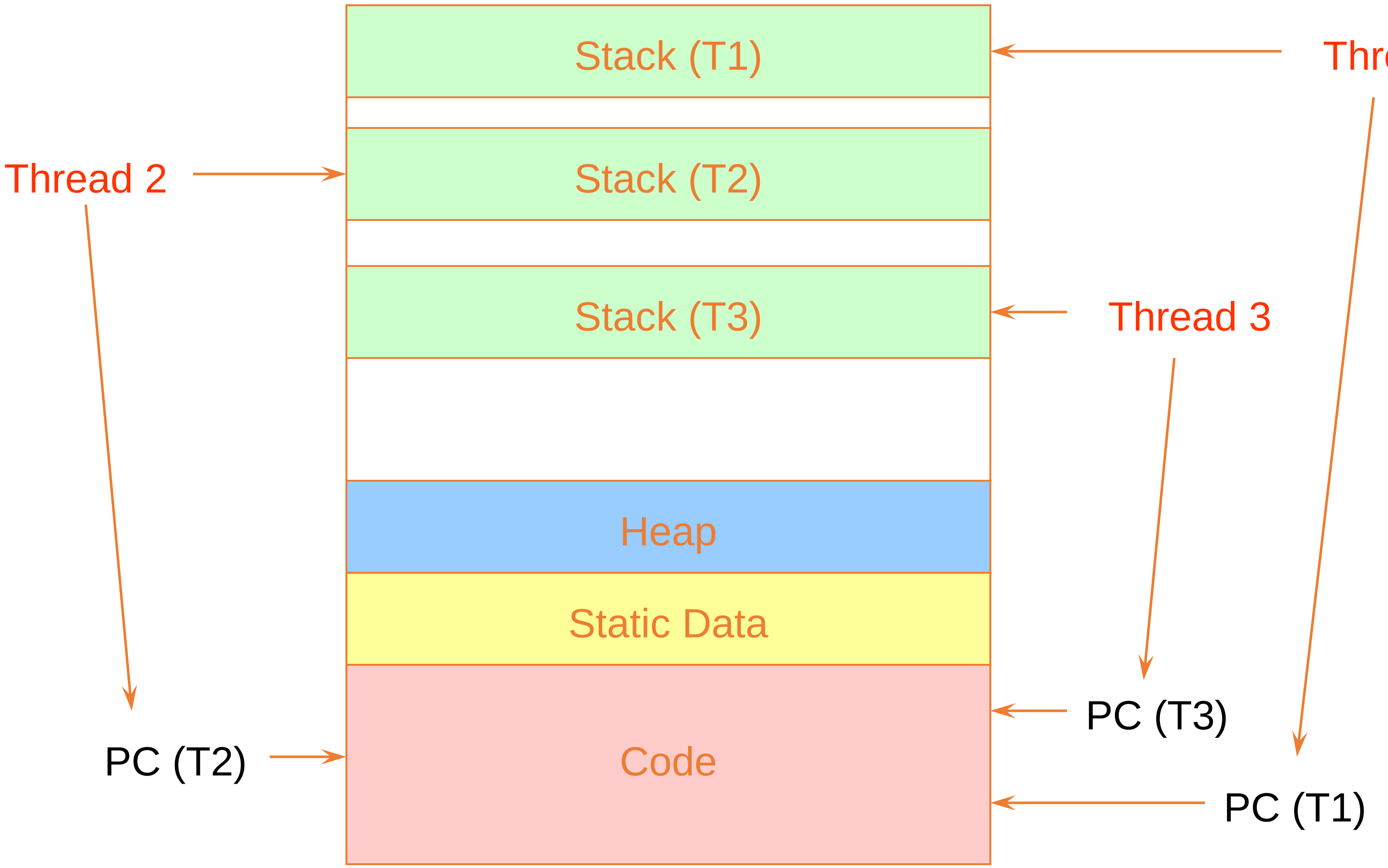
- Multithreading is directly supported by OS:
  - Kernel manages processes and threads
  - CPU scheduling for thread is performed in kernel
- Advantage of multithreading in kernel
  - Is good for multiprocessor architecture
  - If one thread is blocked does not cause the other thread to be blocked.
- Disadvantage of Multithreading in kernel
  - Creation and management of thread is slower

Threads
Hybrid Implementations

➢Multiplexing user-level threads onto kernel-level threads



Threads in a Process



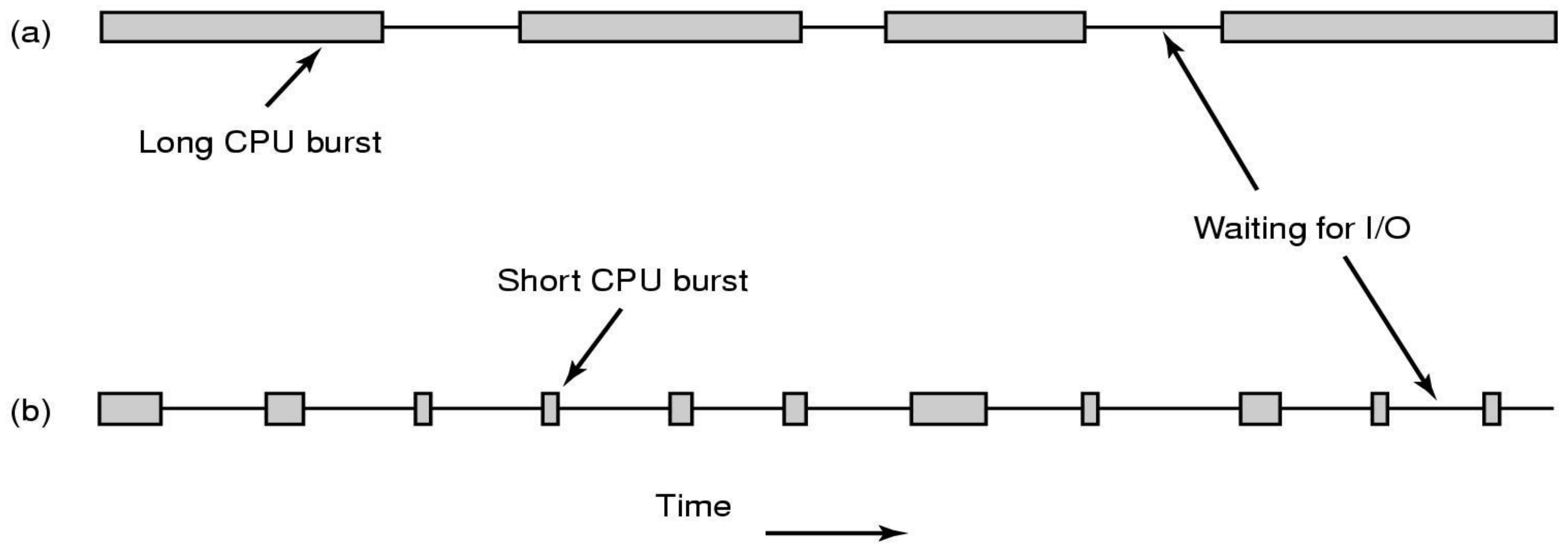
Scheduling

Scheduling
Introduction to Scheduling (1)

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a cycle of CPU execution and I/O wait
- CPU burst distribution

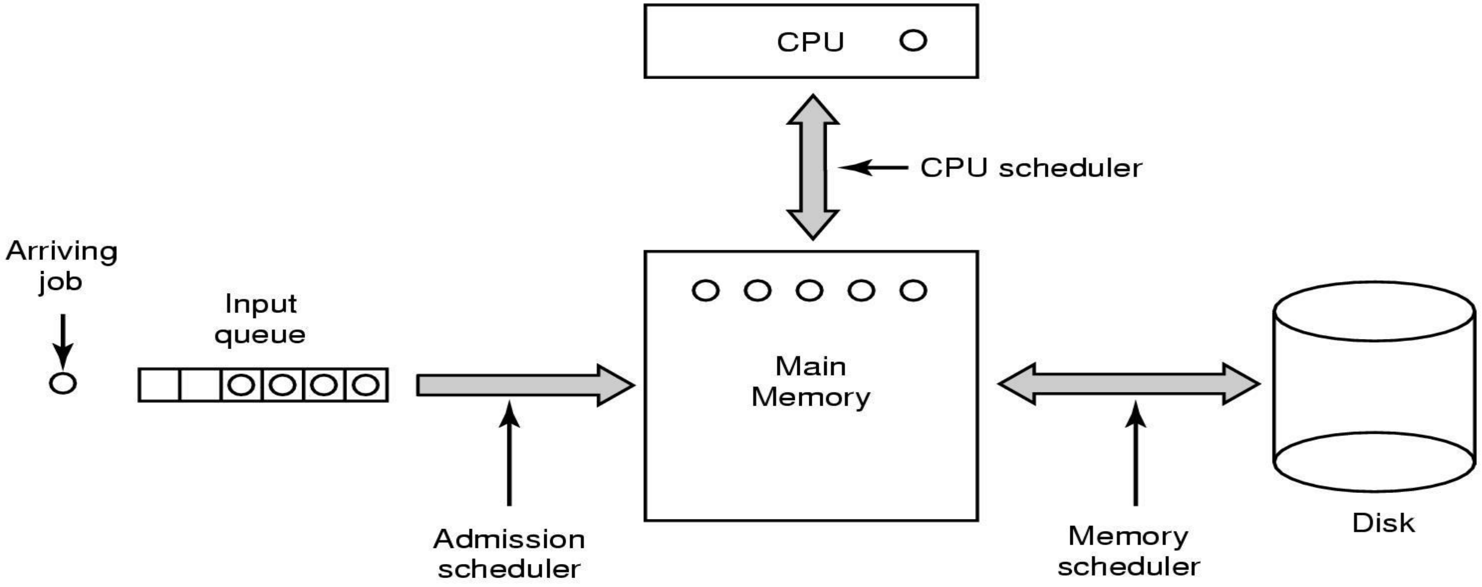
Scheduling
Introduction to Scheduling (2)

- Bursts of CPU usage alternate with periods of I/O wait
  - a CPU-bound process
  - an I/O bound process



Scheduling
Introduction to Scheduling (3)

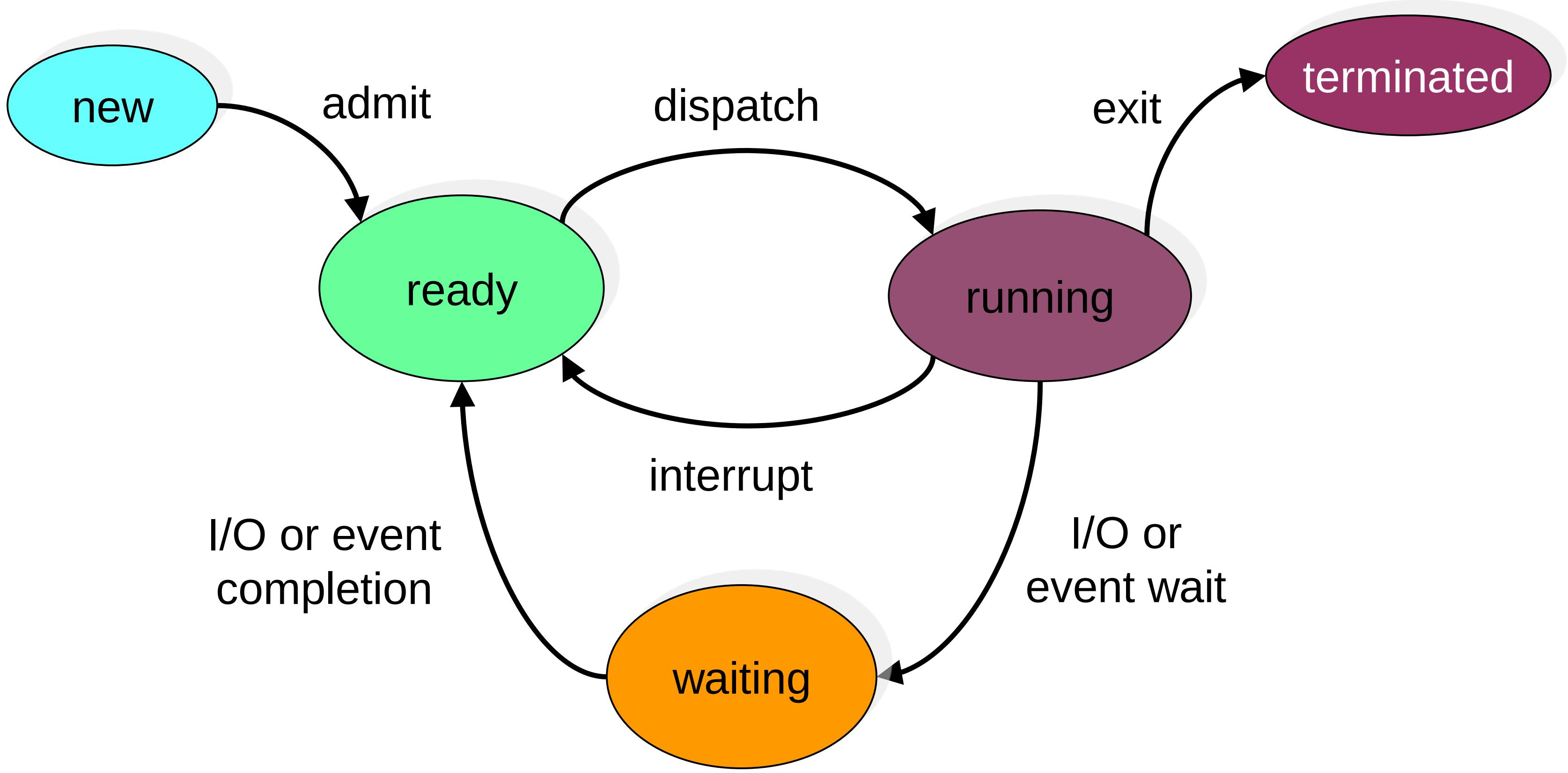
- Three level scheduling



Scheduling
Introduction to Scheduling (4)

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- CPU scheduling decisions may take place when a process:
  - 1. Switches from running to waiting state
  - 2. Switches from running to ready state
  - 3. Switches from waiting or new process is created to ready
  - 4. Terminates
- *Nonpreemptive* scheduling algorithm picks process and let it run until it blocks or until it voluntarily releases the CPU
- *Preemptive* scheduling algorithm picks process and let it run for a maximum of fix time

Scheduling
Introduction to Scheduling (5)



Scheduling
Introduction to Scheduling (6)

➢Scheduling Criteria

- CPU utilization – keep the CPU as busy as possible
- Throughput – the number of processes that complete their execution per time unit
- Turnaround time – amount of time to execute a particular process
- Waiting time – amount of time a process has been waiting in the ready queue
- Response time – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Scheduling
Introduction to Scheduling (7)

➢Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

Scheduling
Introduction to Scheduling (8)

➢Scheduling Algorithm Goals

All systems
- Fairness - giving each process a fair share of the CPU
- Policy enforcement - seeing that stated policy is carried out
- Balance - keeping all parts of the system busy

Batch systems
- Throughput - maximize jobs per hour
- Turnaround time - minimize time between submission and termination
- CPU utilization - keep the CPU busy all the time

Interactive systems
- Response time - respond to requests quickly
- Proportionality - meet users' expectations

Real-time systems
- Meeting deadlines - avoid losing data
- Predictability - avoid quality degradation in multimedia systems

Scheduling
Scheduling in Batch Systems (1)

➢ First-Come, First-Served (FCFS) Scheduling

- Suppose that the processes arrive in the order: P1 , P2 , P3

Process	Burst Time
P_1	24
P_2	3
P_3	3

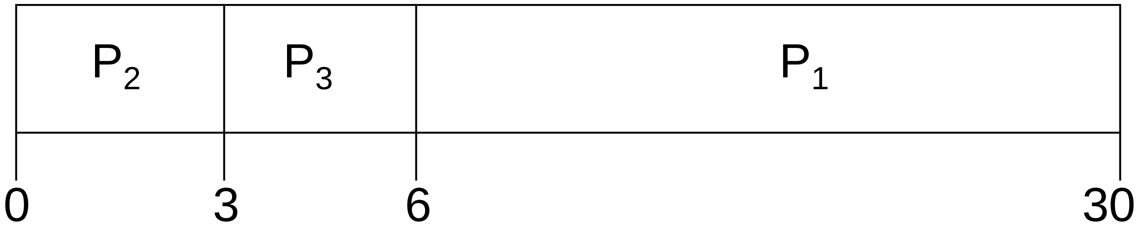
- The Gantt Chart for the schedule is:

- Waiting time for P1 = 0; P2 = 24; P3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

Scheduling
Scheduling in Batch Systems (2)

➢FCFS Scheduling (Cont.)

- Suppose that the processes arrive in the order P2, P3, P1
- The Gantt chart for the schedule is:



- Waiting time for P1 = 6; P2 = 0; P3 = 3
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case
- Convoy effect short process behind long process

Scheduling
Scheduling in Batch Systems (3)

➢Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time
- Two schemes:
  - ✓Nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst
  - ✓Preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF)
- SJF is optimal – gives minimum average waiting time for a given set of processes

Scheduling
Scheduling in Batch Systems (4)

- An example of shortest job first scheduling

Scheduling
Scheduling in Interactive Systems (1)

➢Round Robin Scheduling

- List of runnable processes (a)
- List of runnable processes after B uses up its quantum (b)

(a) (b)

Scheduling
Scheduling in Interactive Systems (2)

➢Round Robin (RR)

- Each process gets a small unit of CPU time (time *quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets 1/n of the CPU time in chunks of at most q time units at once. No process waits more than (n-1)q time units.
- Performance
  - ✓q large $\Rightarrow$ FIFO
  - ✓q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

Scheduling
Scheduling in Interactive Systems (3)

➢ Example of RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 37 57 77 97 117 121 134 154 162

- Typically, higher average turnaround than SJF, but better response

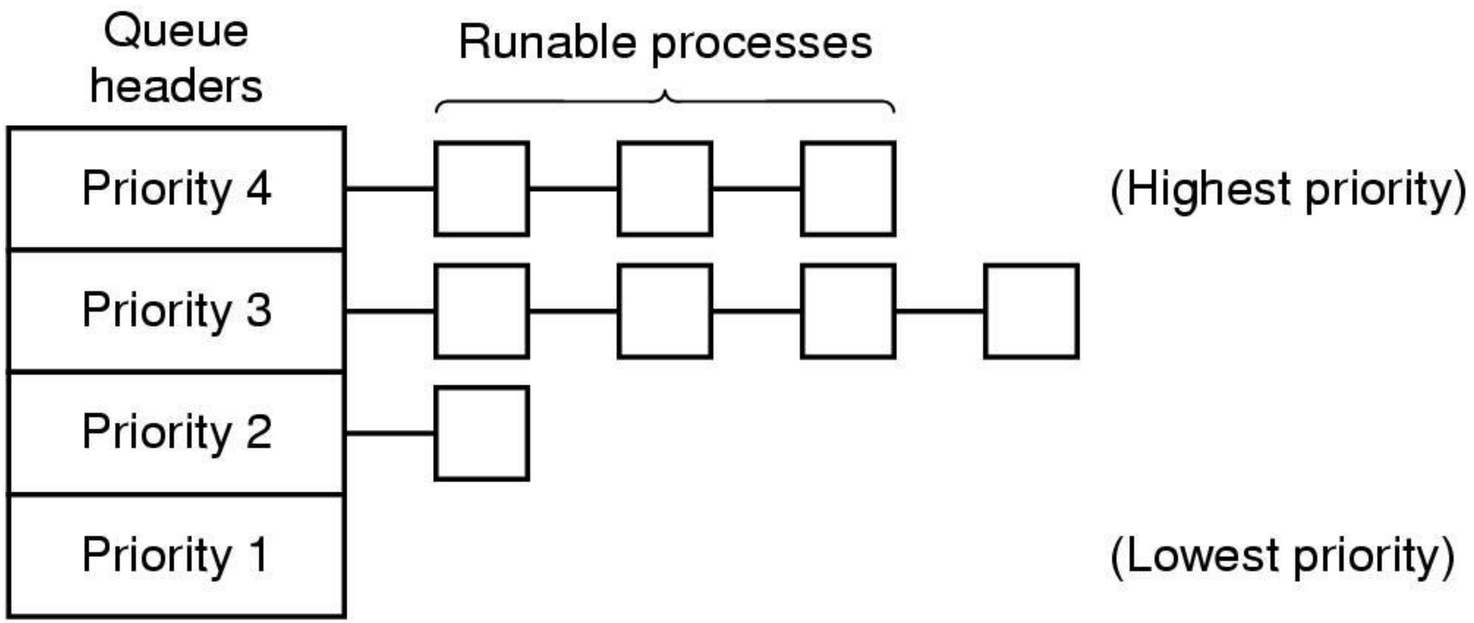
Scheduling
Scheduling in Interactive Systems (4)

➢ Priority Scheduling:

- A priority number (integer) is associated with each process.
  - ✓ The CPU is allocated to the process with the highest priority
  - ✓ Preemptive
  - ✓ Nonpreemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem ≡ Starvation – low priority processes may never execute
- Solution ≡ Aging – as time progresses increase the priority of the process

Scheduling
Scheduling in Interactive Systems (5)

➢A scheduling algorithm with four priority classes



Scheduling
Scheduling in Real-Time Systems (1)

- *Hard real-time systems* – required to complete a critical task within a guaranteed amount of time
- *Soft real-time computing* – requires that critical processes receive priority over less fortunate ones

Scheduling
Scheduling in Real-Time Systems (2)

➢ Schedulable real-time system

- Given
  - ✓ m periodic events
  - ✓ event i occurs within period P_i and requires C_i seconds
- Then the load can only be handled if

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

Scheduling
Policy versus Mechanism

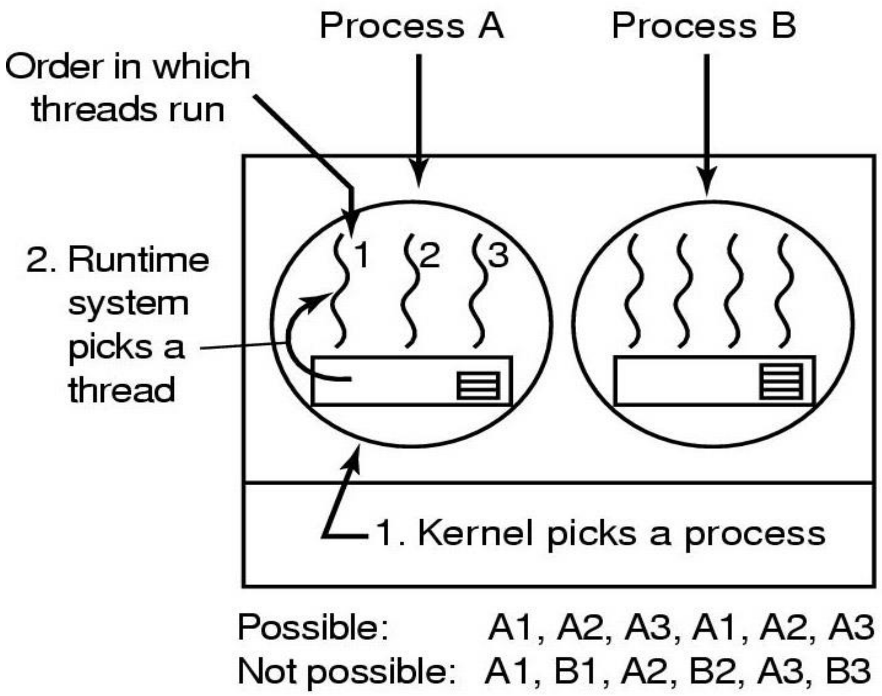
- Separate what is *allowed* to be done with *how* it is done
  - a process knows which of its children threads are important and need priority
- Scheduling algorithm parameterized
  - mechanism in the kernel
- Parameters filled in by user processes
  - policy set by user process

Scheduling
Thread Scheduling (1)

- Local Scheduling – How the threads library decides which thread to put onto an available
- Global Scheduling – How the kernel decides which kernel thread to run next

Scheduling
Thread Scheduling (2)

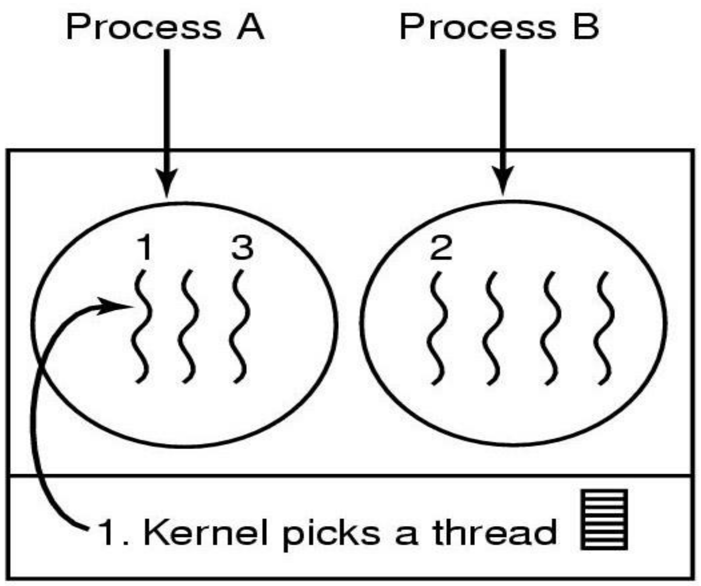
➢ Possible scheduling of user-level threads
- 50-msec process quantum
- threads run 5 msec/CPU burst



Scheduling
Thread Scheduling (3)

➢ Possible scheduling of kernel-level threads

- 50-msec process quantum
- threads run 5 msec/CPU burst



Possible: A1, A2, A3, A1, A2, A3
Also possible: A1, B1, A2, B2, A3, B3

2.4 Interprocess Communication

Cooperating Processes

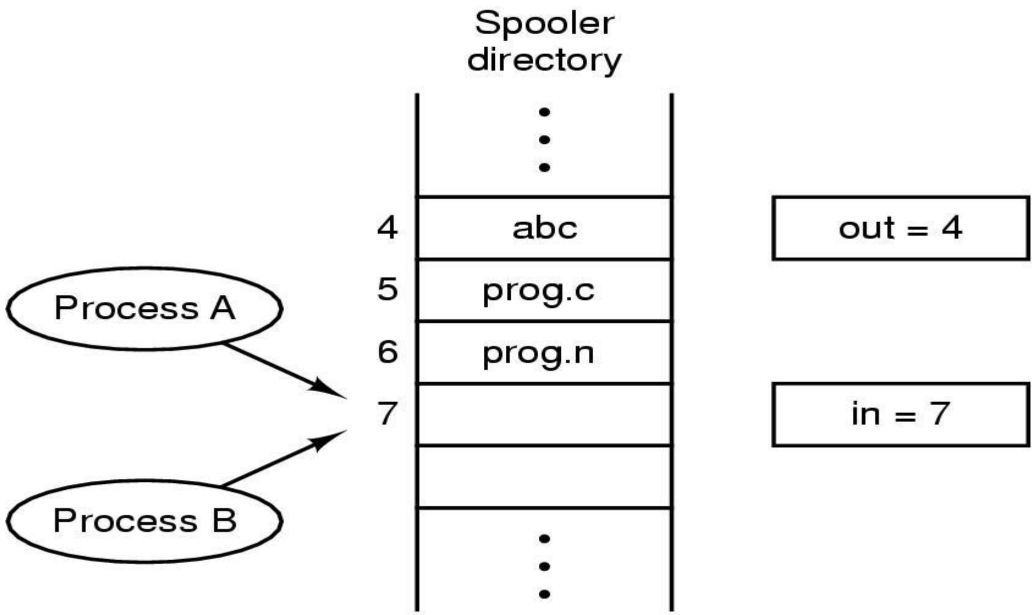
- **Independent** process cannot affect or be affected by the execution of another process
- **Cooperating** process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

Problem of shared data

- Concurrent access to shared data may result in data inconsistency
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes
- Need of mechanism for processes to *communicate* and to *synchronize* their actions

Race Conditions

- Two processes want to access shared memory at same time and the final result depends who runs precisely, are called *race condition*
- *Mutual exclusion* is the way to prohibit more than one process from accessing to shared data at the same time

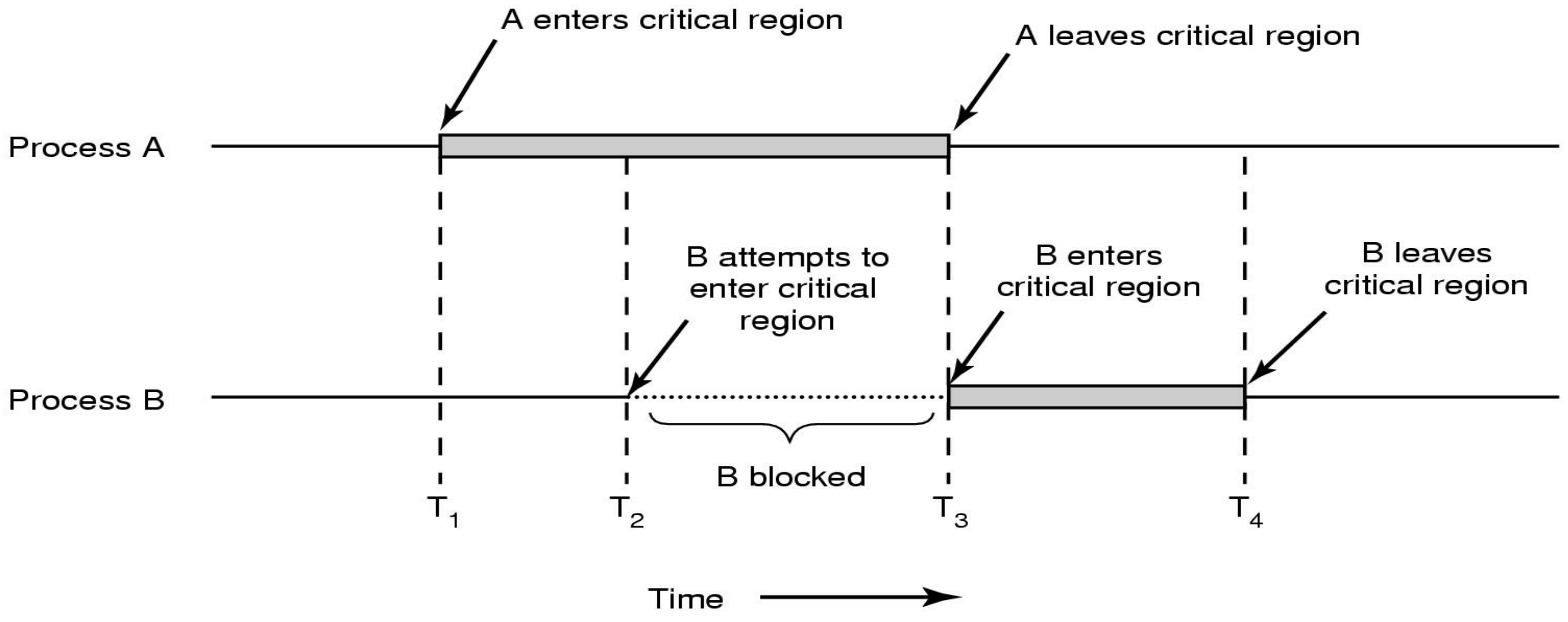


Critical Regions (1)

- The Part of the program where the shared memory is accessed is called *Critical Regions* (Critical Section)
- Four conditions to provide mutual exclusion
  - No two processes simultaneously in critical region
  - No assumptions made about speeds or numbers of CPUs
  - No process running outside its critical region may block another process
  - No process must wait forever to enter its critical region

Critical Regions (2)

- Mutual exclusion using critical regions (Example)



Solution: Mutual exclusion with Busy waiting

- Software proposal
  - Lock Variables
  - Strict Alternation
  - Peterson's Solution
- Hardware proposal
  - Disabling Interrupts
  - The TSL Instruction

Mutual exclusion with Busy waiting Software Proposal 1: Lock Variables

int lock = 0

P0

```
NonCS;

while (lock == 1); // wait
lock = 1;

CS;

lock = 0;

NonCS;
```

P1

```
NonCS;

while (lock == 1); // wait
lock = 1;

CS;

lock = 0;

NonCS;
```

Mutual exclusion with Busy waiting Software Proposal 1: Event

int lock = 0

P0

NonCS;

```
while (lock == 1); // wait
lock = 1;
```

CS;

```
lock = 0;
```

NonCS;

P1

NonCS;

```
while (lock == 1); // wait
lock = 1;
```

CS;

```
lock = 0;
```

NonCS;

Mutual exclusion with Busy waiting Software Proposal 2: Strict Alternation

int turn = 1

P0

```
NonCS;

while (turn !=0); // wait

CS;

turn = 1;

NonCS;
```

P1

```
NonCS;

while (turn != 1); // wait

CS;

turn = 0;

NonCS;
```

Mutual exclusion with Busy waiting Software Proposal 2: Strict Alternation

- Only 2 processes
- Responsibility Mutual Exclusion
  - One variable "*turn*“, one process “*turn*” come in CS at the moment.

Mutual exclusion with Busy waiting Software Proposal 3: Peterson's Solution

- int turn;
- int interest[2] = FALSE;

```
Pi                NonCS;

    j = 1 - i;
    interest[i] = TRUE;
    turn = j;
    while (turn==j && interest[j]==TRUE);

                    CS;

    interest[i] = FALSE;

                  NonCS;
```

Mutual exclusion with Busy waiting Software Proposal 3: Peterson's Solution

Pj

NonCS;

```
i = 1 - j;
interest[j] = TRUE;
turn = i;
while (turn==i && interest[i]==TRUE);
```

CS;

```
interest[j] = FALSE;
```

NonCS;

Mutual exclusion with Busy waiting Comment for Software Proposal 3: Peterson's Solution

- Satisfy 3 conditions:
  - Mutual Exclusion
    - ✓ Pi can enter CS when *interes[j] == F*, or *turn == i*
    - ✓ If both want to come back, because turn can only receive value 0 or 1, so one process enter CS
  - Progress
    - ✓ Using 2 variables distinct *interest[i]* ==> opposing cannot lock
  - Bounded Wait: both *interest[i]* and turn change value
- Not extend into N processes

Mutual exclusion with Busy waiting Comment for Busy-Waiting solutions

- Don't need system’s support
- Hard to extend
- Solution 1 is better when *atomicity* is supported

Busy waiting – Hardware Proposal

- Software proposal
  - Lock Variables
  - Strict Alternation
  - Peterson's Solution
- Hardware proposal
  - Disabling Interrupts
  - The TLS Instruction

Busy waiting – Hardware Proposal 1: Disabling Interrupt

- Disable Interrupt: prohibit all interrupts, including spin interrupt
- Enable Interrupt: permit interrupt

NonCS;

$\boxed{\text{Disable Interrupt;}}$

CS;

$\boxed{\text{Enable Interrupt;}}$

NonCS;

Hardware proposal 1: Disable Interrupt

- Not be careful
  - If process is locked in CS?
    - ✓ System Halt
  - Permit process use command privileges
    - ✓ Danger!
- System with N CPUs?
  - Don't ensure Mutual Exclusion

Hardware proposal 1: TSL Instruction

- CPU support primitive Test and Set Lock
  - Return a variable's current value, set variable to true value
  - Cannot divide up to perform (Atomic)

```
TSL (boolean &target)
{
        TSL = target;
        target = TRUE;
}
```

Applied TSL

int lock = 0

```
Pi
        NonCS;

    while (TSL(lock));  // wait

        CS;

    lock = 0;

        NonCS;
```

Comment for hardware solutions in Busy-Waiting

- Necessary hardware mechanism's support
  - Not easy with n-CPUs system
- Easily extend to N processes

Comment for hardware solutions in Busy-Waiting

- Using CPU not effectively
  - Constantly test condition when wait for coming in CS
- Overcome
  - Lock processes that not enough condition to come in CS, concede CPU to other process
    - ✓ Using Scheduler
    - ✓ Wait and See...

Synchronous solution

- Sleep & Wakeup
  - Semaphore
  - Message passing

"Sleep & Wake up" solution

- Give up CPU when not come in CS
- When CS is empty, will be waken up to come in CS
- Need support of OS
  - Because of changing status of process

```
if not Sleep();

CS;

Wakeup(somebody);
```

"Sleep & Wake up" solution: Idea

- OS support 2 primitive:
  - Sleep(): System call receives blocked status
  - WakeUp(P): P process receive ready status
- Application
  - After checking condition, coming in CS or calling Sleep() depend on result of checking
  - Process that using CS before, will wake up processes blocked before

Apply Sleep() and Wakeup()

- int busy;
- int blocked;

```
if (busy) {
            blocked = blocked + 1;
            Sleep();
        }
else busy = 1;
```

CS;

```
busy = 0;
    if(blocked) {        WakeUp(P);
                         blocked = blocked - 1;
                }
```

Problem with Sleep & WakeUp

- Reason:
  - Checking condition and giving up CPU can be broken
  - Lock variable is not protected

Semaphore

- Suggested by Dijkstra, 1965
- Properties: Semaphore s;
  - Unique value
  - Manipulate with 2 primitives:
    - ✓ Down(s)
    - ✓ Up(s)
  - Down and Up primitives excuted cannot divide up

Install Semaphore (Sleep & Wakeup)

- Semaphore: similar to resource
  - Processes "request" semaphore: call Down(s)
    - If Down(s) is not finished: resource is not allocated
      - Blocked, insert to s.L
- Need OS's support
  - Sleep() & Wakeup()

Install Semaphore (Sleep & Wakeup)

```
Down (S)
{
   S.value --;
   if S.value < 0
   {
      Add(P,S.L);
      Sleep();
   }
}
```

```
Up(S)
{
   S.value ++;
   if S.value ≤ 0
   {
      Remove(P,S.L);
      Wakeup(P);
   }
}
```

Using Semaphore

Semaphore s = 1

P_i

Down (s)

CS;

Up(s)

Semaphore s = 0

P_1 :

Job1;

Up(s)

P_2:

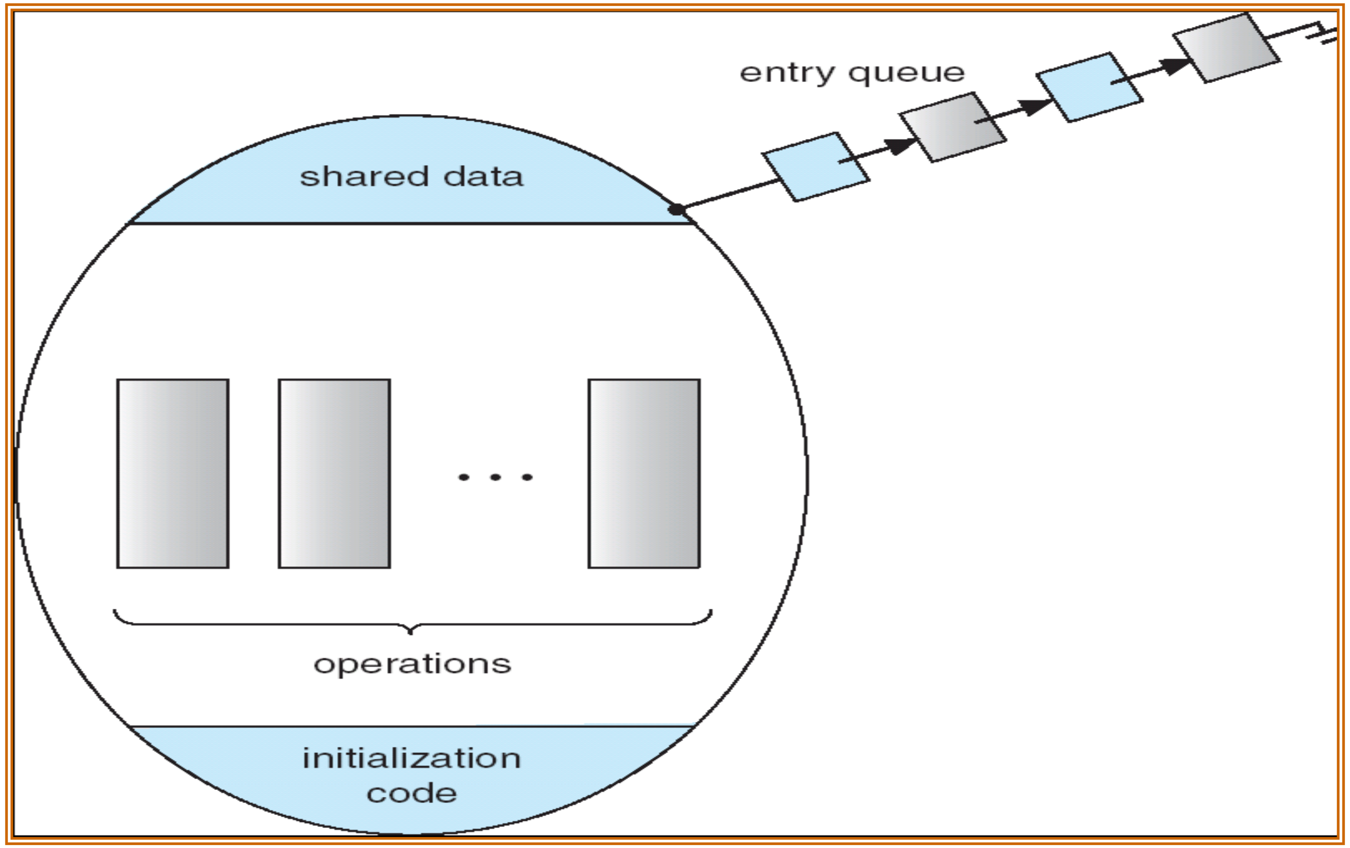
Down (s);

Job2;

Monitor

- Hoare (1974) & Brinch (1975)
- Synchronous mechanism is provided by programming language
  - Support with functions, such as Semaphore
  - Easier for using and detecting than Semaphore
    - ✓ Ensure Mutual Exclusion automatically
    - ✓ Using condition variable to perform Synchronization

Monitor: structure



Monitor: structure

```
monitor monitor-name
{
        shared variable declarations
        procedure body P1 (...) {
                . . .
        }
        procedure body P2 (...) {
                . . .
        }
        procedure body Pn (...) {
                . . .
        }
        {
                initialization code
        }
}
```

Using Monitor

```
Monitor      M
<resource type> RC;
Function   AccessMutual
                  CS; // access RC
```

```
Pi
 M.AccessMutual(); //CS
```

```
Monitor    M
Condition  c;
Function   F1
           Job1;
           Signal(c);
Function F2
           Wait(c);
           Job2;
```

```
P1  :
M.F1();
```

```
P2:
M.F2();
```

Message Passing

- 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

Classical Problems of Synchronization

- Bounded-Buffer Problem (Producer-Consumer Problem)
- Readers and Writers Problem
- Dining-Philosophers Problem

SUMMARY

- Processes
- Threads
- Scheduling
- Interprocess communication