CSCI 360
Introduction to Operating Systems

Process Management

Humayun Kabir
Professor, CS, Vancouver Island University, BC, Canada

Outline

- Process
- Thread
- Process Scheduling
    - First-Come First-Served
    - Shortest Job First
    - Shortest Remaining Time Next
    - Round Robin Scheduling
    - Priority Scheduling
    - Multiple Queues Scheduling

Process Abstraction

- A process is an abstraction of a running program.
- Execution of a program starts via GUI mouse clicks or command line entry of its name.
- One program can be several processes.

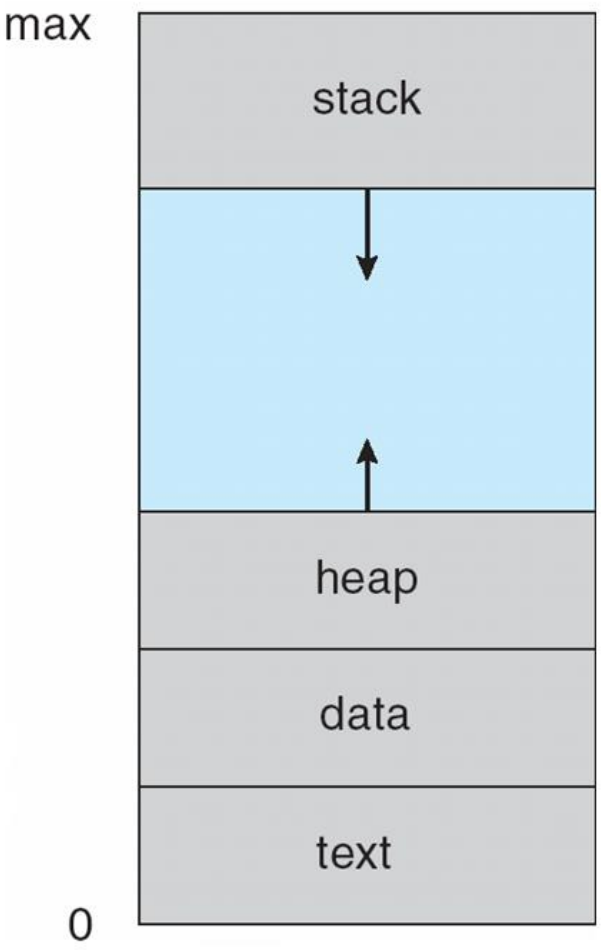
Process Abstraction

- A program becomes a process when the executable code is loaded into memory and starts running.
- Process execution progress in sequential fashion from the beginning to the end of the code.
- A process has more parts other than the code.

Process Abstraction

- A process has following parts.
  - The program code, called **text section**
  - Current activity represented by **program counter** and **processor registers**
  - **Stack** to hold temporary data
    - return addresses, function parameters, and local variables
  - **Data section** to hold global variables
  - **Heap** to hold dynamically allocated variables during run time

Process Abstraction



Process Operations: Creation

Four principal events that cause processes to be created:

- System initialization.
- Execution of a process creation system call by a running process.
- A user request to create a new process.
- Initiation of a batch job.

Process Operations: Termination

Typical conditions which terminate a process:

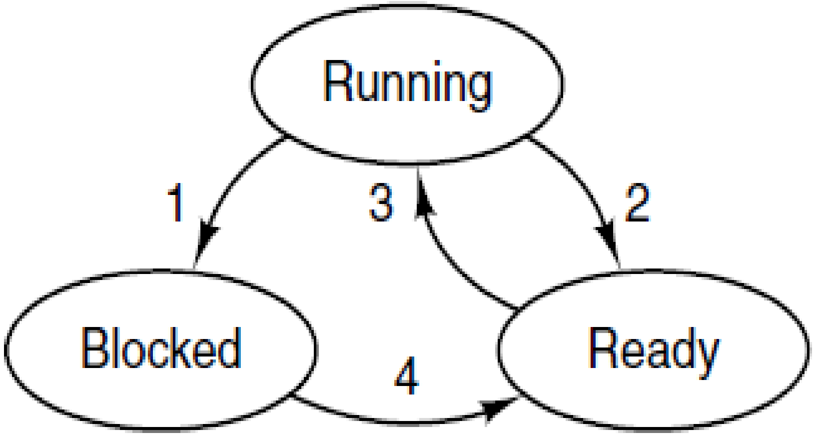
- Normal exit (voluntary).
- Error exit (voluntary).
- Fatal error (involuntary).
- Killed by another process (involuntary).

Process States

Three states a process may be in:

- Running (actually using the CPU at that instant).
- Ready (runnable; temporarily stopped to let another process run).
- Blocked (unable to run until some external event happens).

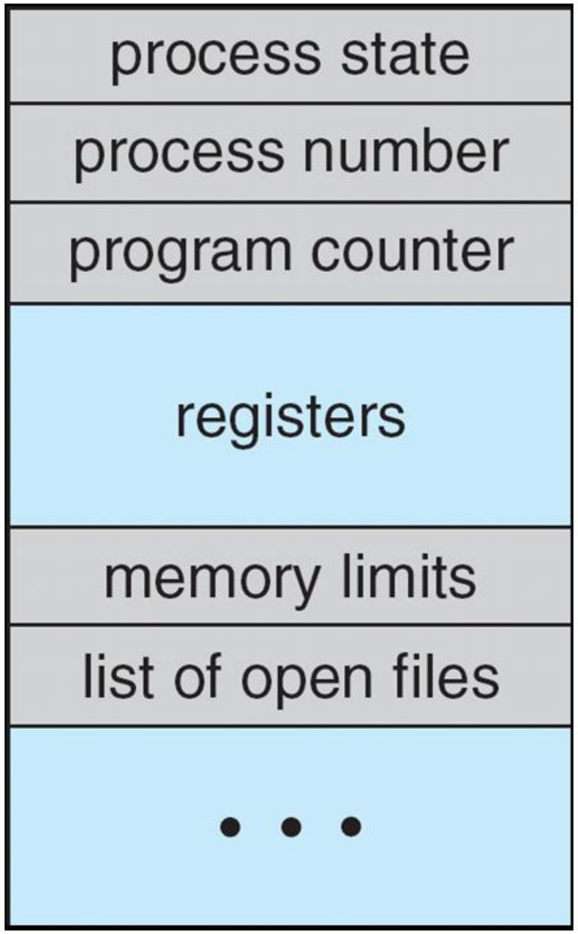
Process States



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

Process Control Block

Each process is represented in the OS by a **process control block**, which holds the information related to the process

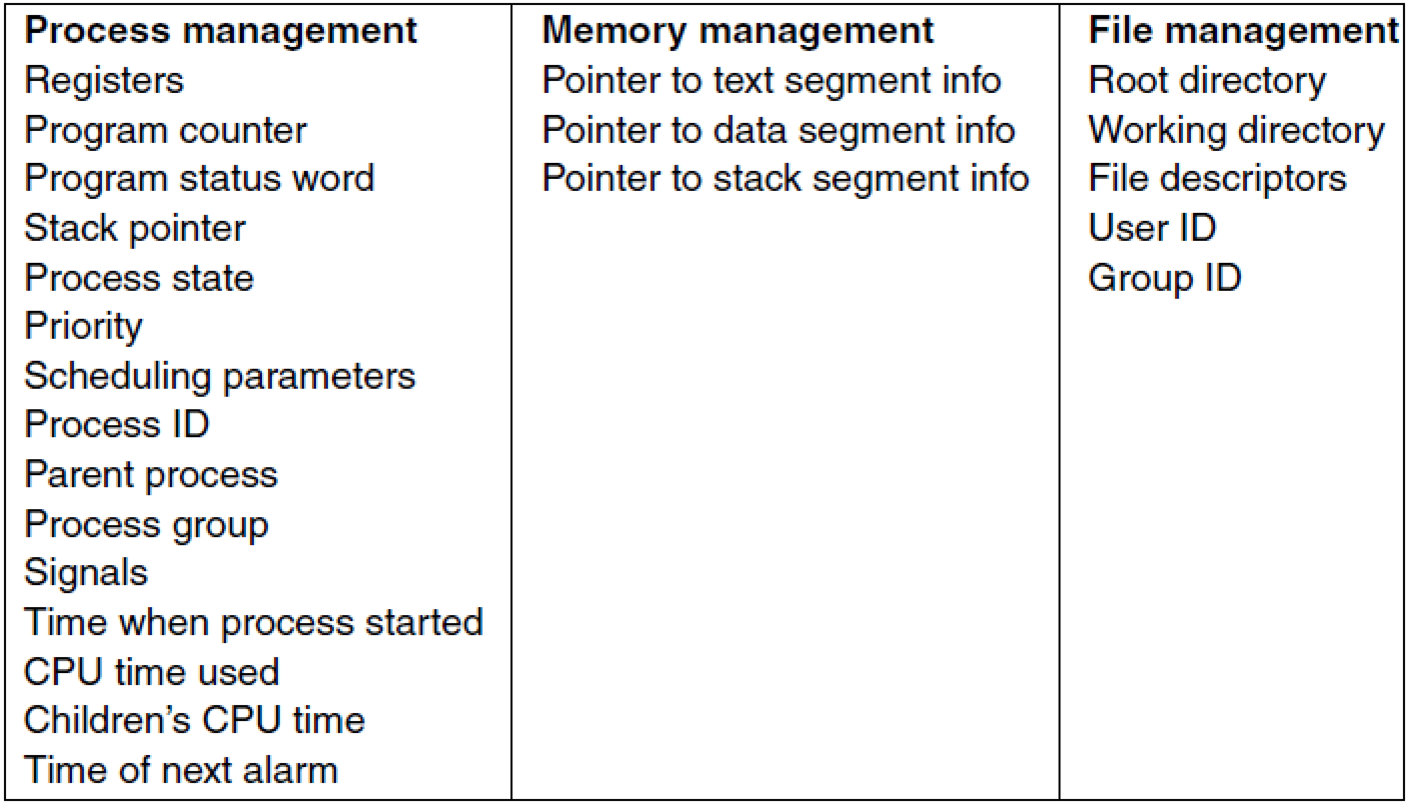


Process Control Block

Information in **process control block**

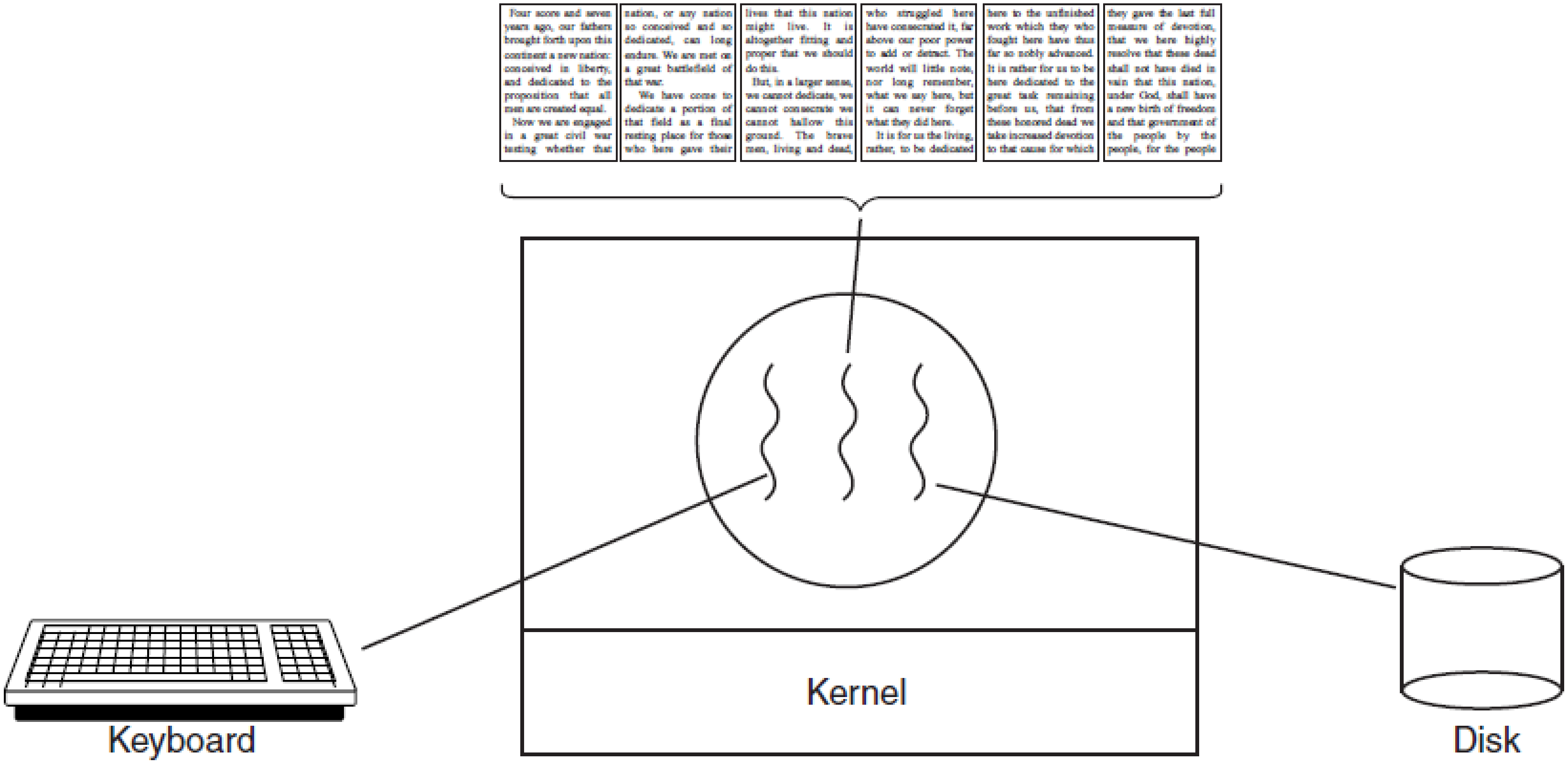
- Process state – ready, running, blocked
- Program counter – location of instruction to execute next
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files

Process Control Block



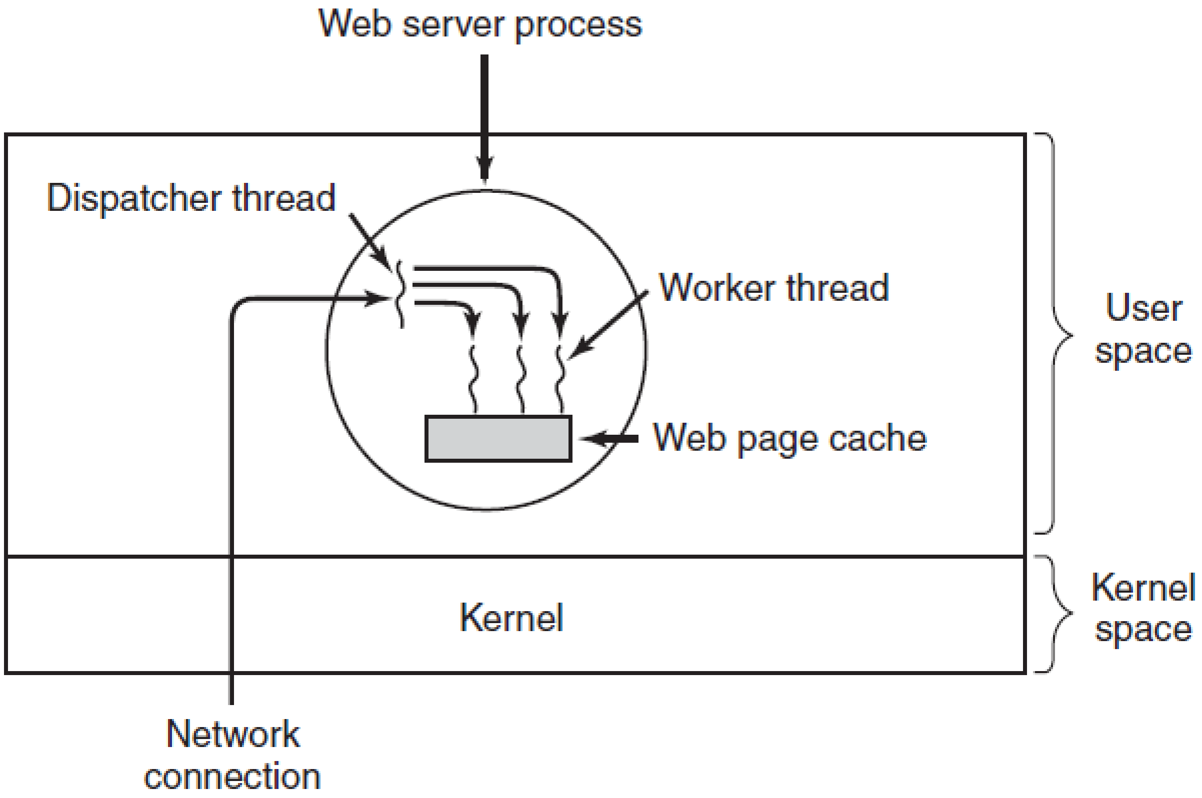
Process management	Memory management	File management
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling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

Thread



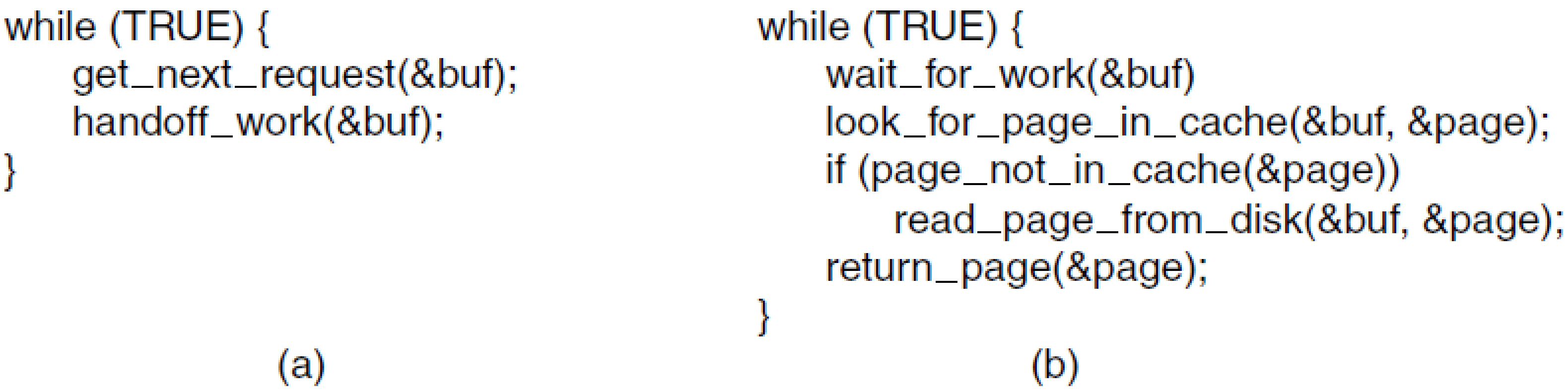
A word processor with three threads.

Thread



A multithreaded Web server.

Thread



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

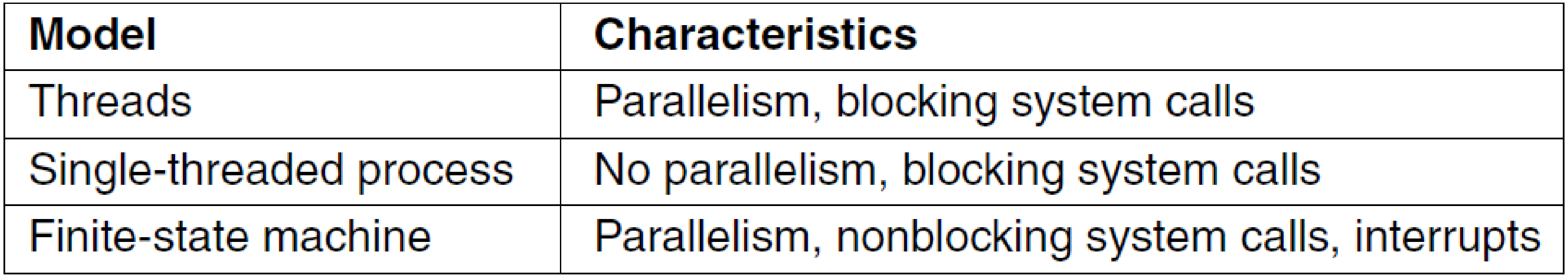
(a)

```
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)

A rough outline of the code for
(a) Dispatcher thread. (b) Worker thread.

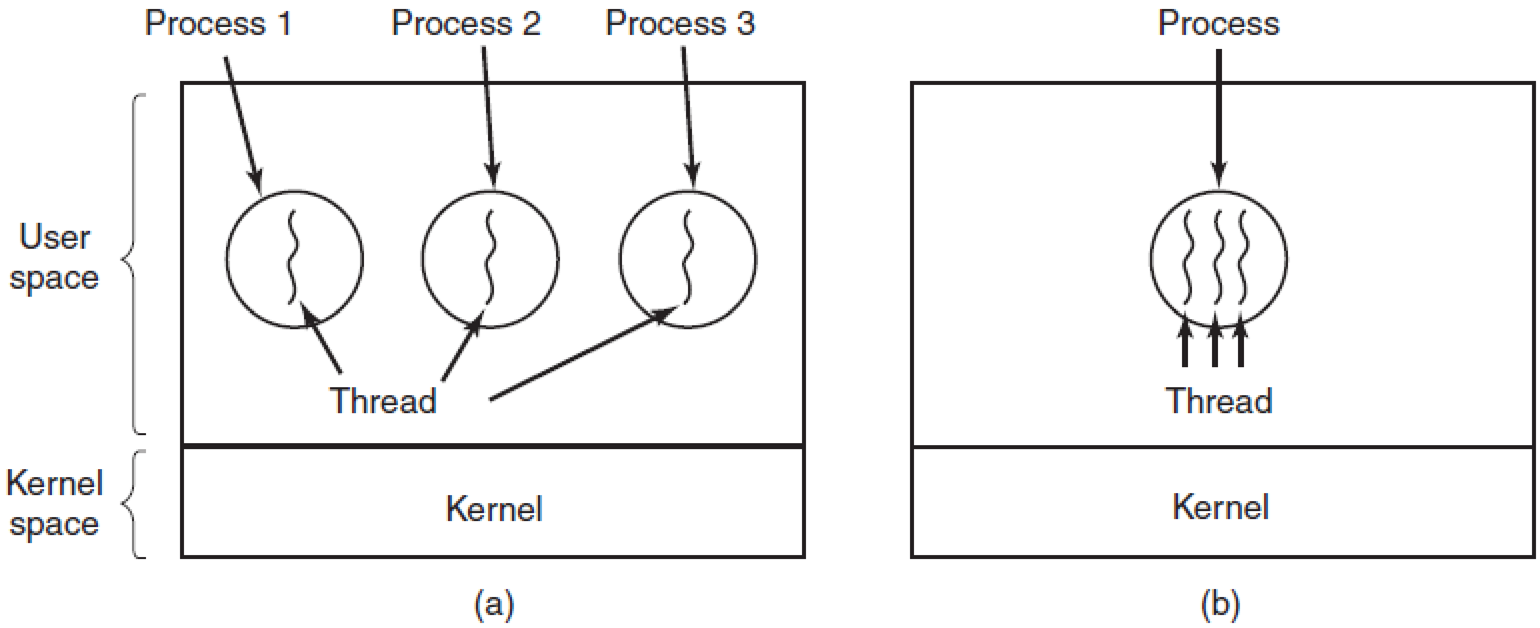
Thread



Model	Characteristics
Threads	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls
Finite-state machine	Parallelism, nonblocking system calls, interrupts

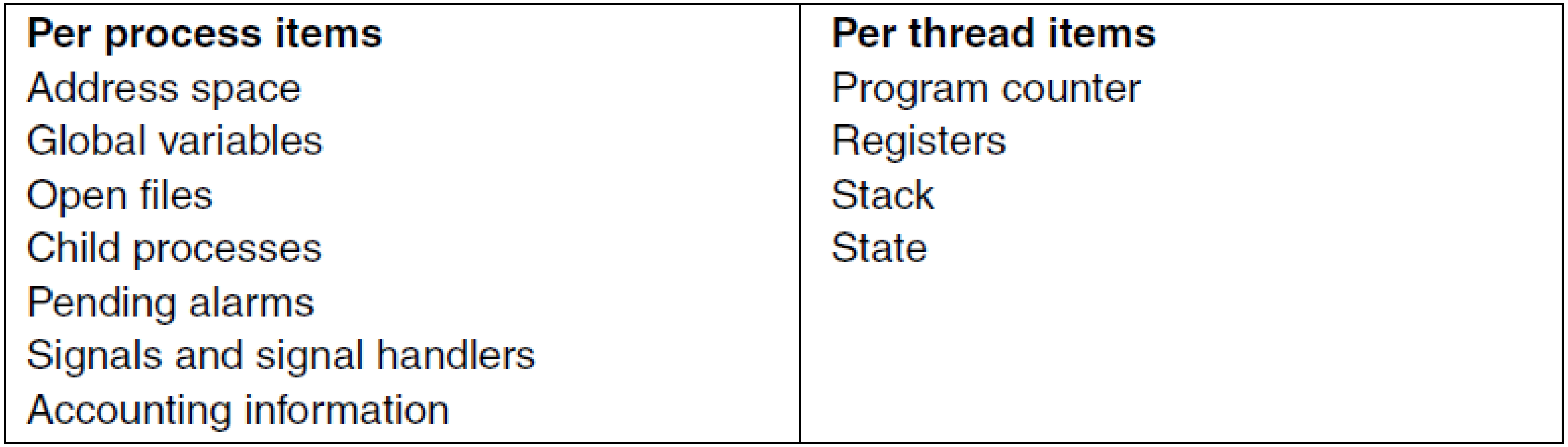
Three ways to construct a server.

Thread



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

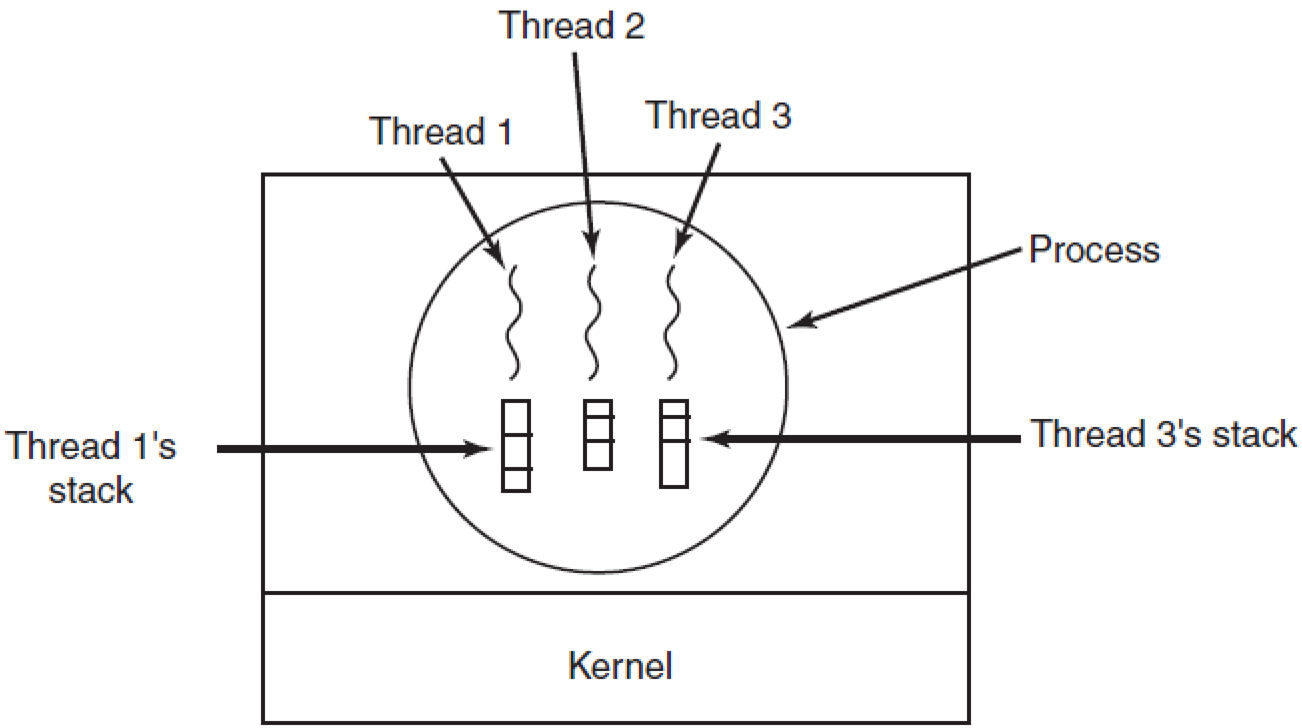
Thread



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	

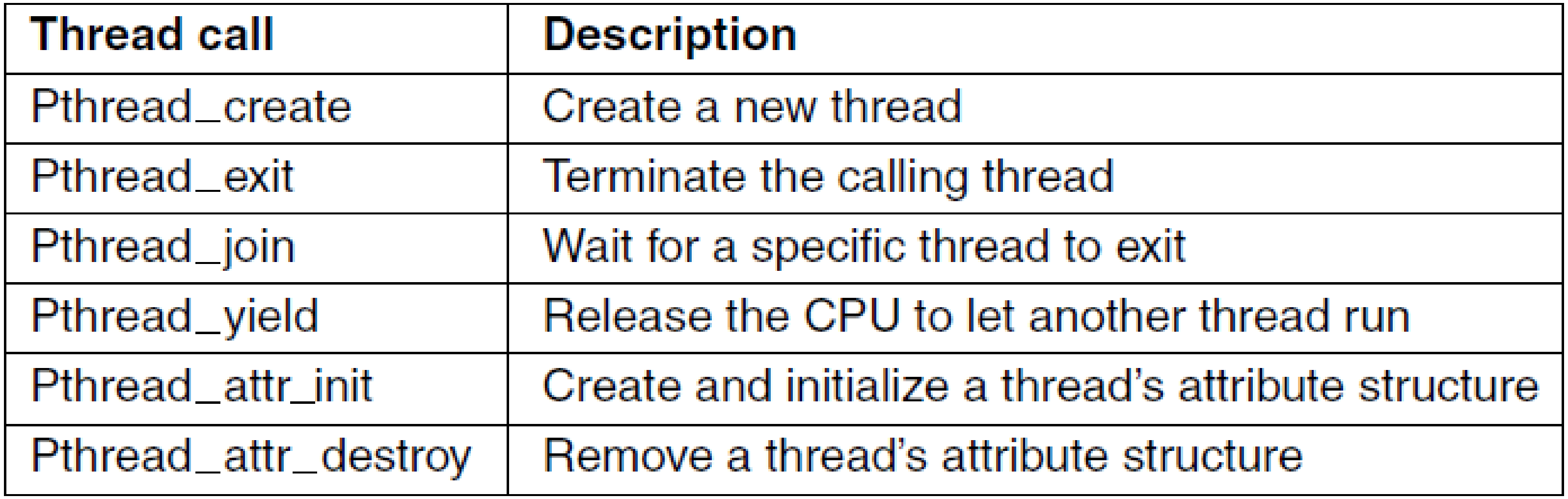
The first column lists some items shared by all threads in a process. The second one lists some items private to each thread.

Thread



Each thread has its own stack.

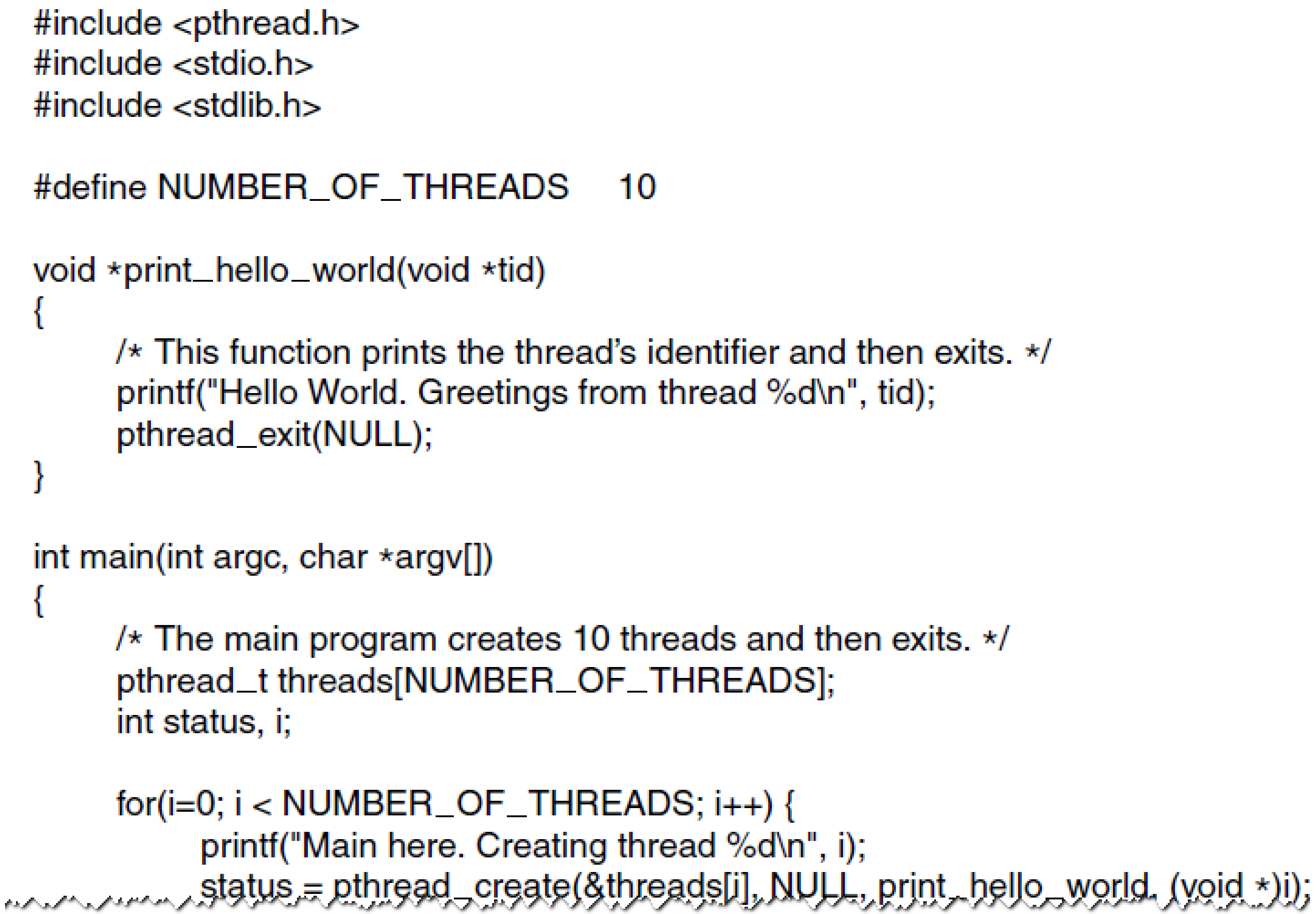
POSIX Thread



Thread call	Description
Pthread_create	Create a new thread
Pthread_exit	Terminate the calling thread
Pthread_join	Wait for a specific thread to exit
Pthread_yield	Release the CPU to let another thread run
Pthread_attr_init	Create and initialize a thread's attribute structure
Pthread_attr_destroy	Remove a thread's attribute structure

Some of the Pthreads function calls.

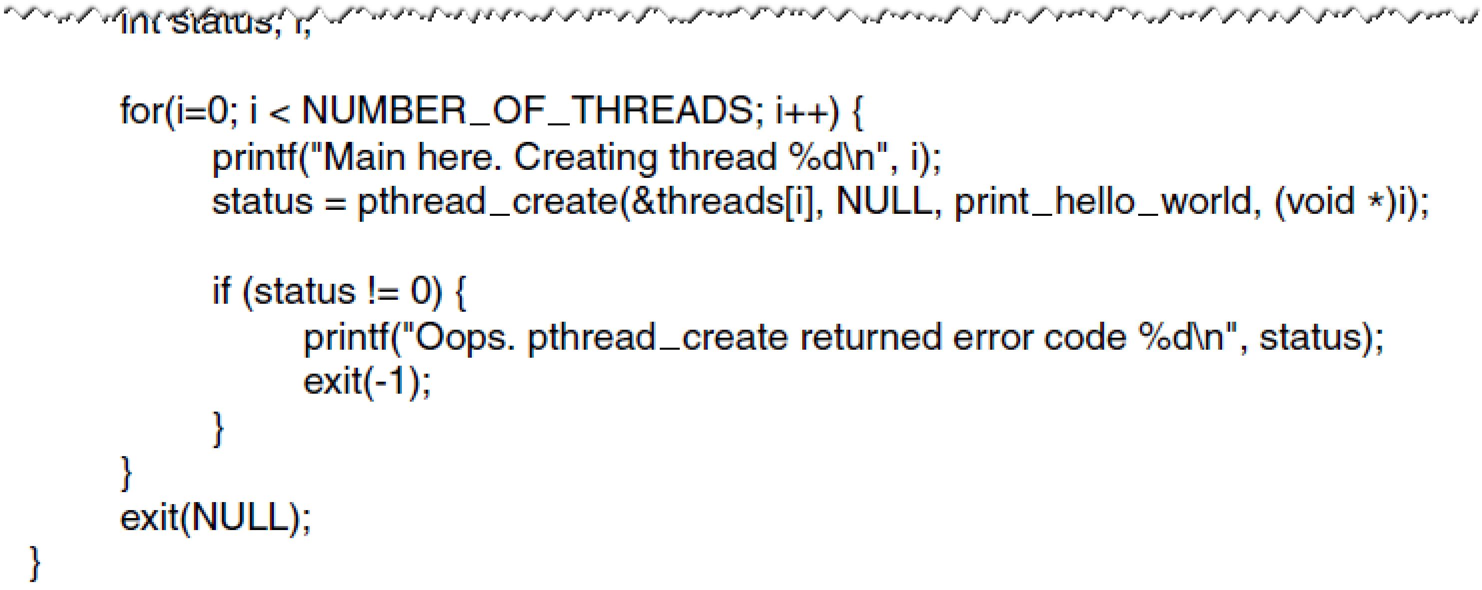
POSIX Thread



```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NUMBER_OF_THREADS     10

void *print_hello_world(void *tid)
{
        /* This function prints the thread's identifier and then exits. */
        printf("Hello World. Greetings from thread %d\n", tid);
        pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
        /* The main program creates 10 threads and then exits. */
        pthread_t threads[NUMBER_OF_THREADS];
        int status, i;

        for(i=0; i < NUMBER_OF_THREADS; i++) {
                printf("Main here. Creating thread %d\n", i);
                status = pthread_create(&threads[i], NULL, print_hello_world, (void *)i);
```

An example program using threads.

POSIX Thread



```
    int status, i;

    for(i=0; i < NUMBER_OF_THREADS; i++) {
        printf("Main here. Creating thread %d\n", i);
        status = pthread_create(&threads[i], NULL, print_hello_world, (void *)i);

        if (status != 0) {
            printf("Oops. pthread_create returned error code %d\n", status);
            exit(-1);
        }
    }
    exit(NULL);
}
```

An example program using threads.

User Threads and Kernel Threads

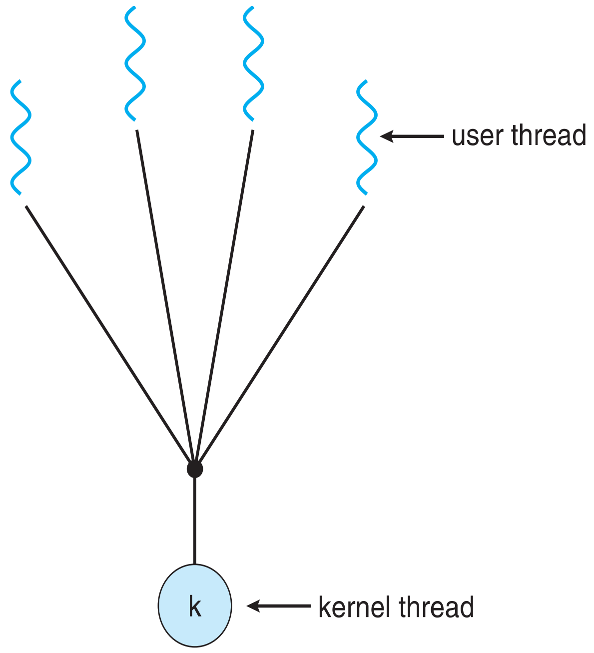
- **User threads** - management done by user-level threads library
- Three primary thread libraries:
  - POSIX **Pthreads**
  - Windows threads
  - Java threads
- **Kernel threads** - Supported by the Kernel
- Examples – virtually all general purpose operating systems, including:
  - Windows
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

Multithreading Models

- Many-to-One
- One-to-One
- Many-to-Many

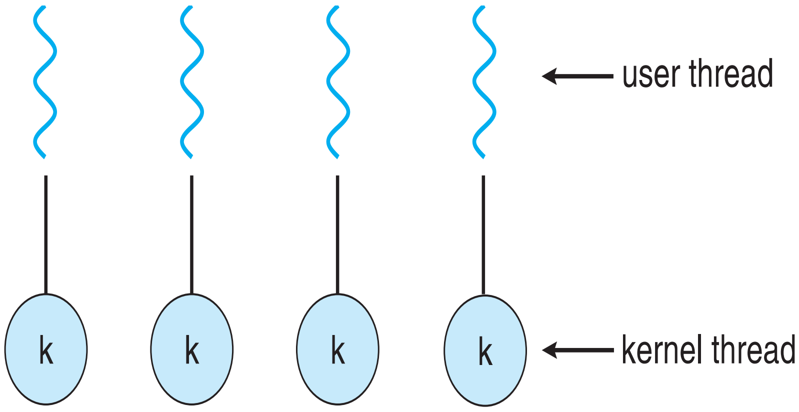
Many-to-One

- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time
- Few systems currently use this model
- Examples:
  - **Solaris Green Threads**
  - **GNU Portable Threads**



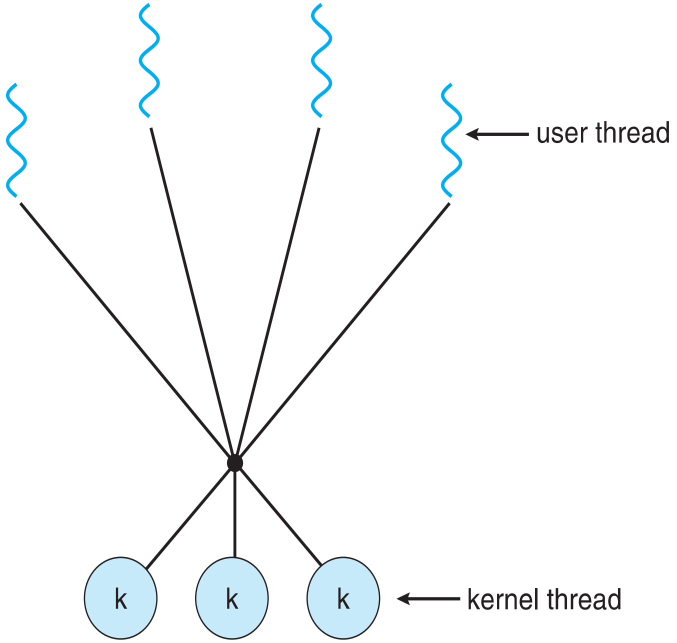
One-to-One

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead
- Examples
  - Windows
  - Linux
  - Solaris 9 and later



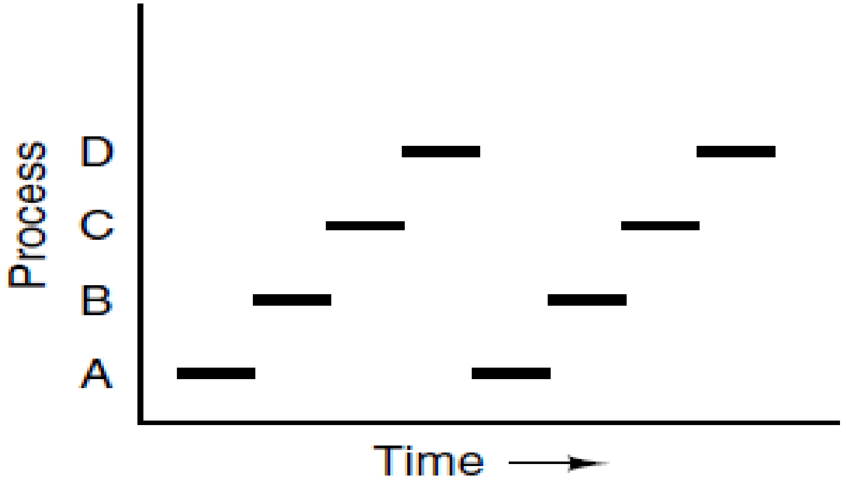
Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Solaris prior to version 9
- Windows with the *ThreadFiber* package



Process Scheduling

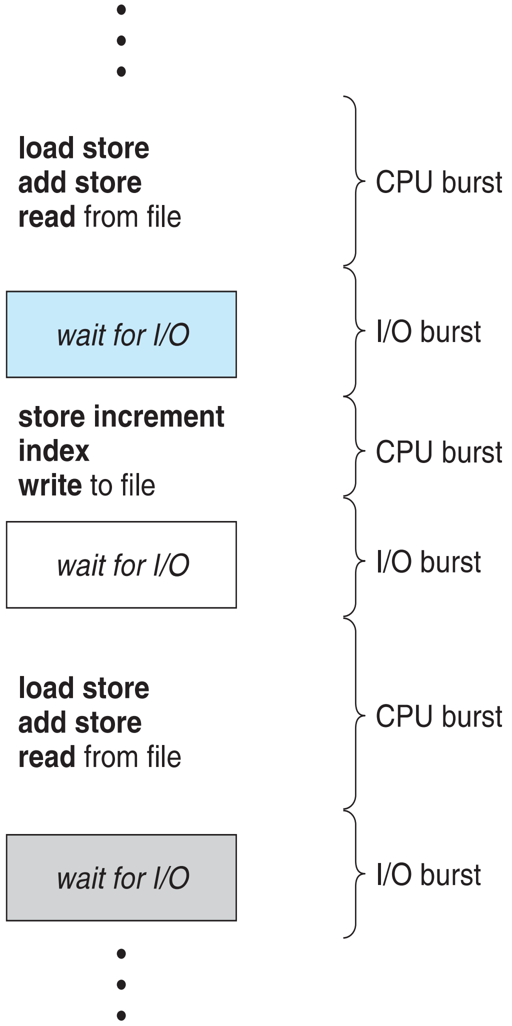
- Modern OS allows multiple processes even on a single CPU.
- CPUs are time-shared among the processes.
- A process scheduler shares the CPUs among the processes in a seamless way.
- Maximum CPU utilization obtained with multiprocessing



(c)

Process Scheduling

- Process execution consists of a **cycle** of CPU execution and I/O wait
- **CPU burst** followed by **I/O burst**
- CPU burst distribution is of main concern



load store
add store
read from file

CPU burst

wait for I/O

I/O burst

store increment
index
write to file

CPU burst

wait for I/O

I/O burst

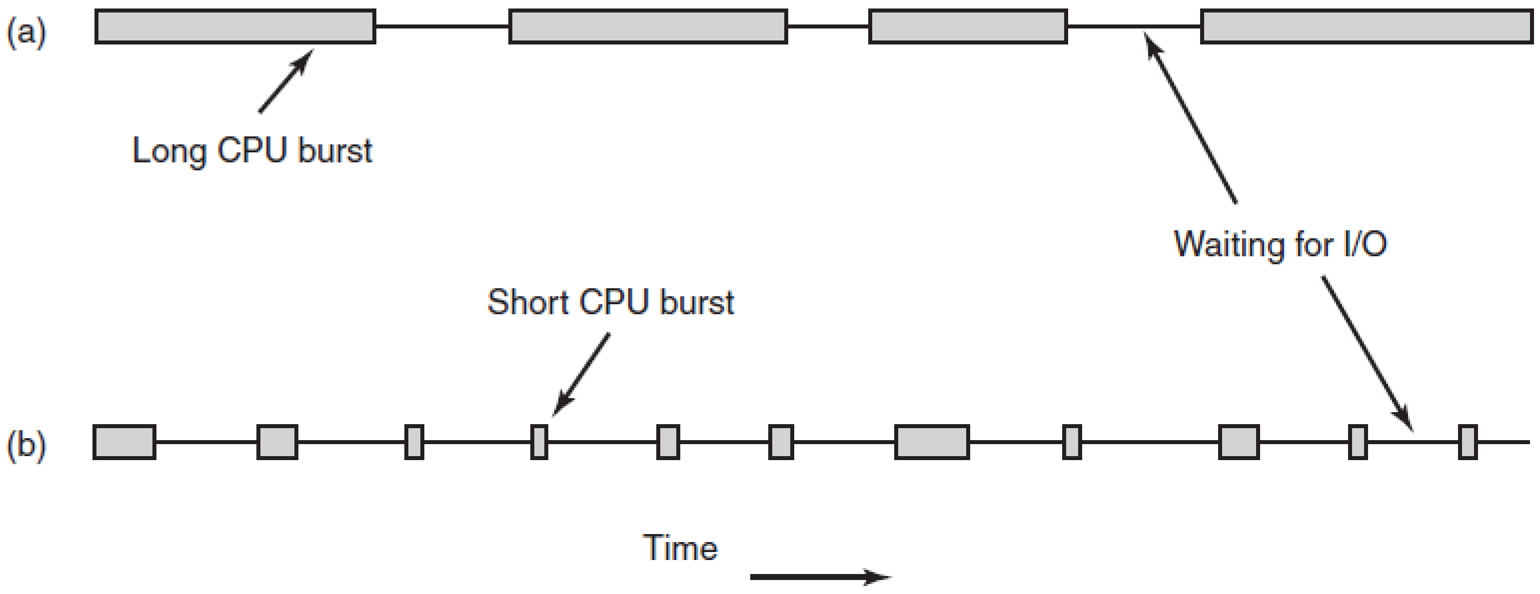
load store
add store
read from file

CPU burst

wait for I/O

I/O burst

Process Scheduling

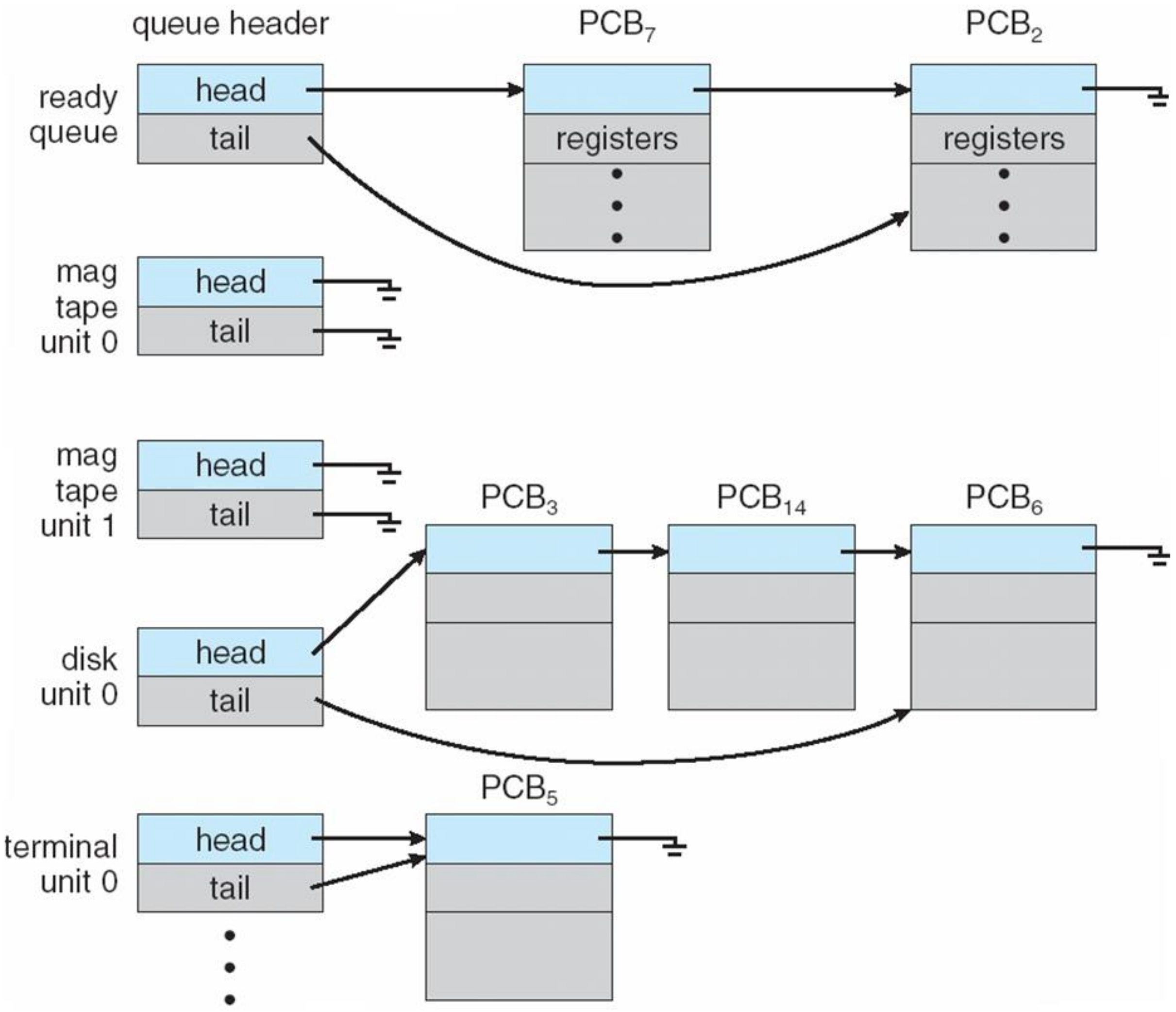


Bursts of CPU usage alternate with periods of waiting for I/O. (a) A CPU-bound process. (b) An I/O-bound process.

Process Scheduling

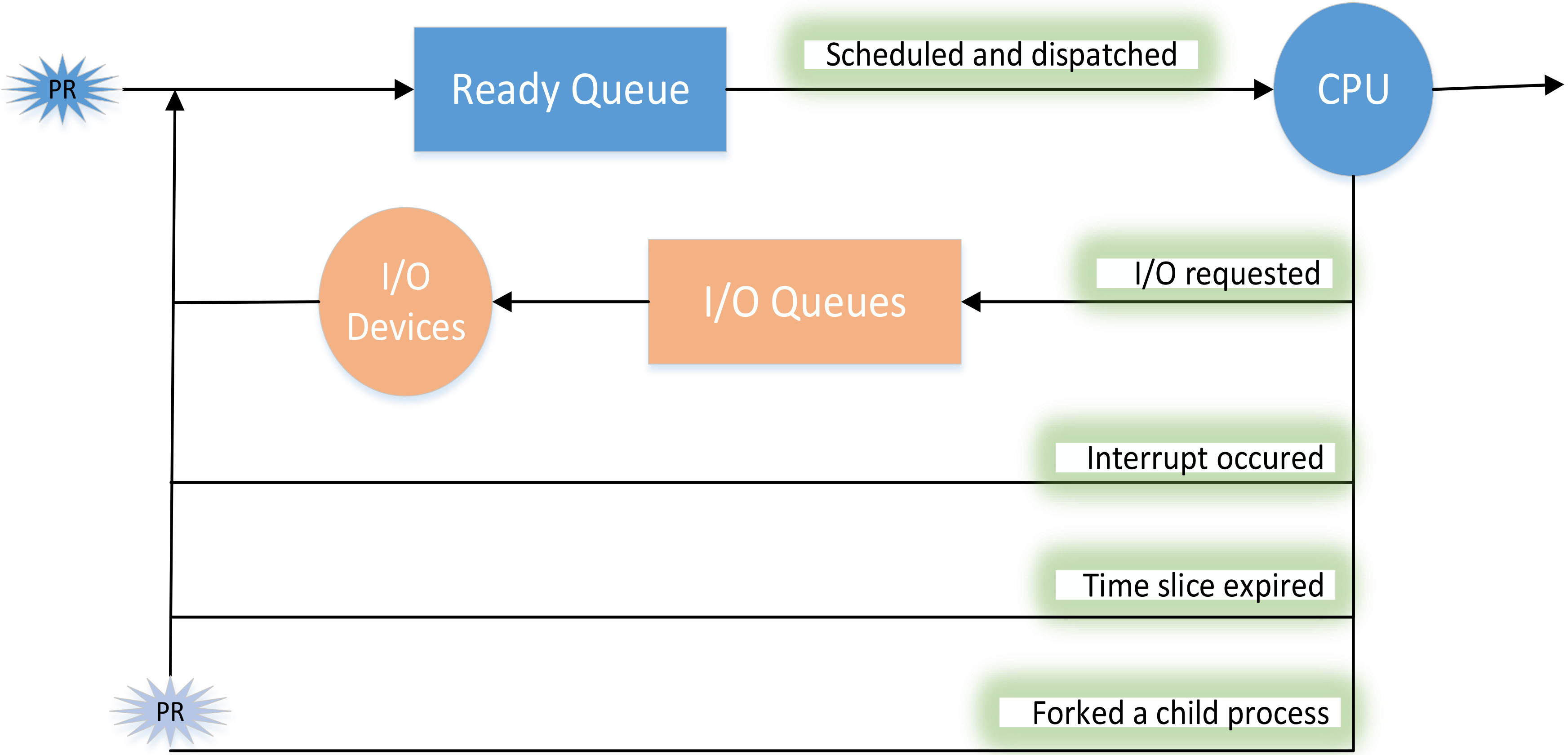
- **Process scheduler** maintains **scheduling queues** of processes
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Device queues** or **I/O queues** – set of processes waiting for an I/O device
- **Process scheduler** selects among available processes for next execution on CPU
- Processes migrate among the various queues

Process Scheduling



Process Scheduling

- **Queueing diagram** represents queues, resources, flows



Process Scheduling

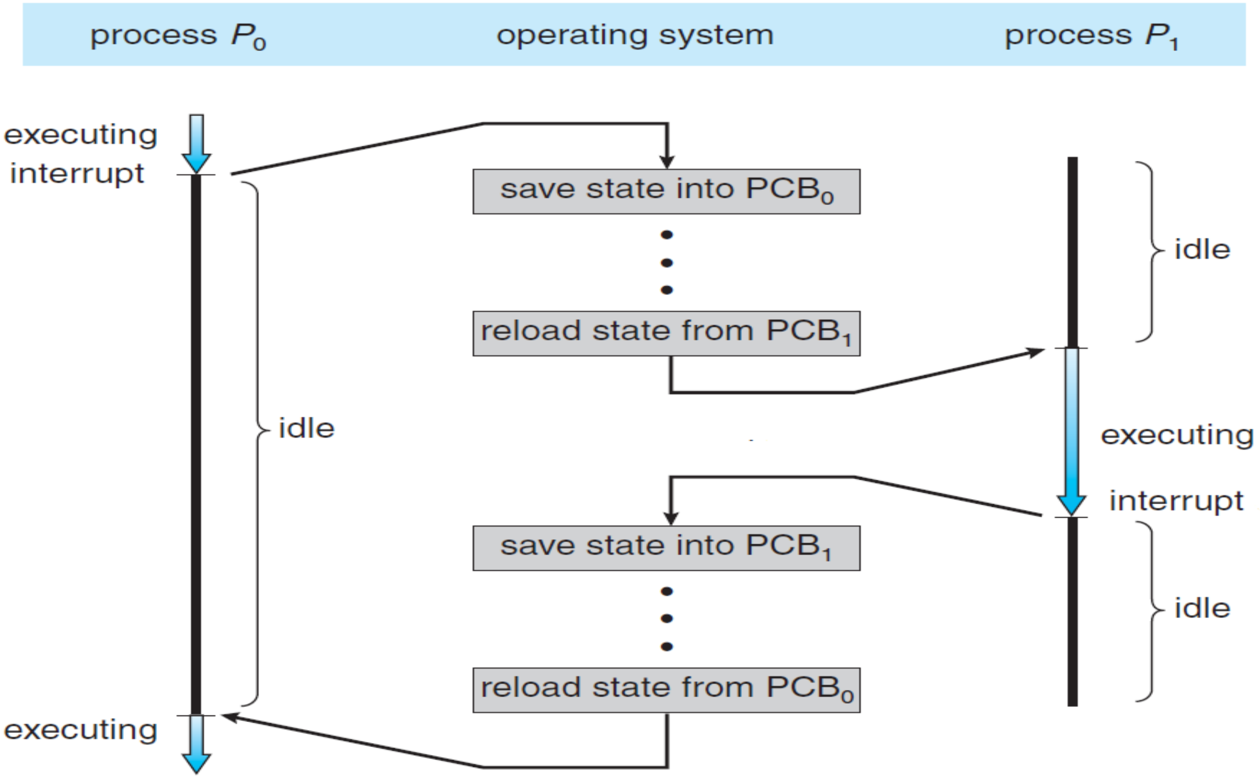
- **Scheduler** selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is **nonpreemptive**
- All other scheduling is **preemptive**
  - Upon expiration of the time slice of a process
  - When interrupt occurs

Process Scheduling

- **Dispatcher** module gives the control of the CPU to the process selected by the scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running

Process Scheduling: 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 via a **context switch**
- **Context** of a process represented in the PCB



Process Scheduling: Context Switch

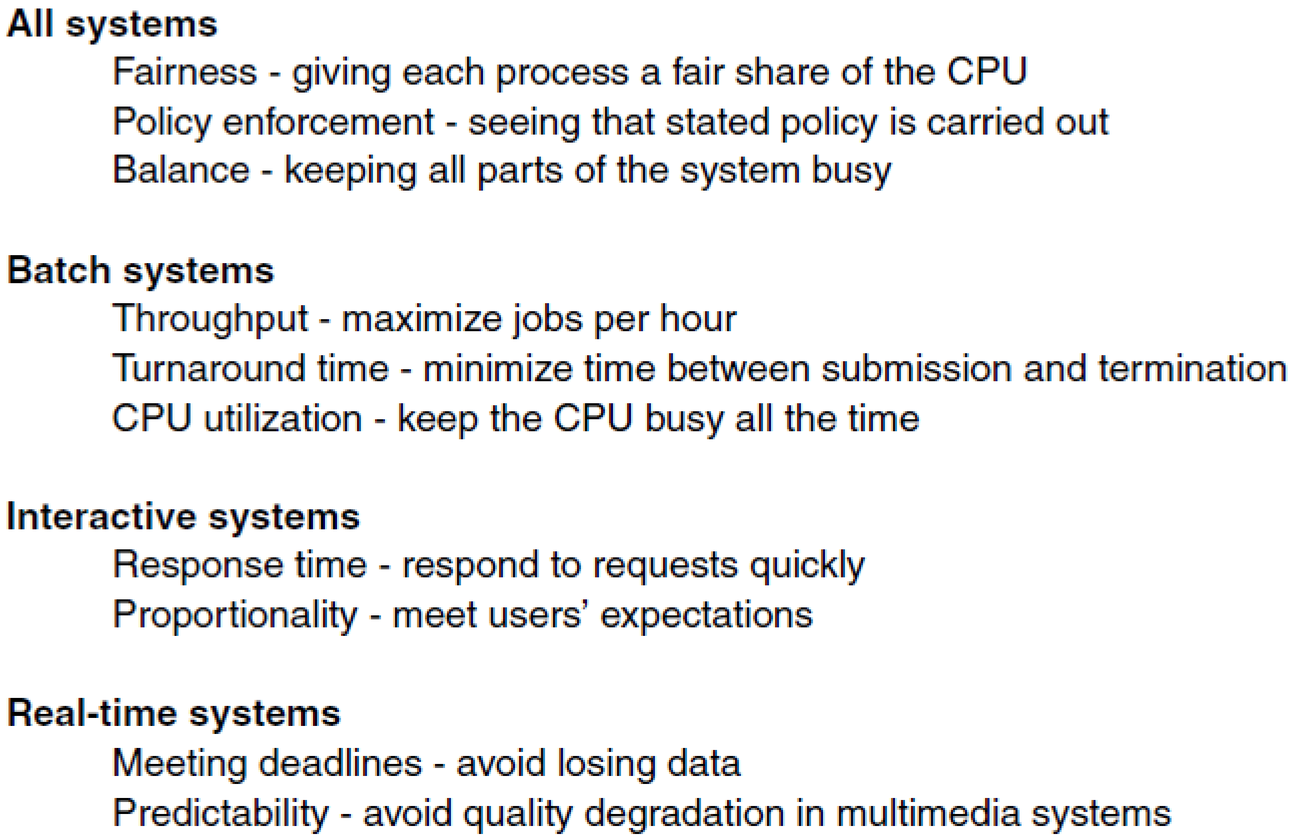
- **Context-switch** time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB ➔ the longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU ➔ multiple contexts loaded at once

Process Scheduling

Categories of Algorithms

- Batch.
- Interactive.
- Real time.

Process 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

Process Scheduling: Batch Systems

- First-Come First-Served
- Shortest Job First
- Shortest Remaining Time Next

Process Scheduling: Interactive Systems

- Round-Robin Scheduling
- Priority Scheduling
- Multiple Queues
- Shortest Process Next
- Guaranteed Scheduling
- Lottery Scheduling
- Fair-Share Scheduling

Process Scheduling: FCFS

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:

- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$
- Turnaround time for $P_1 = 24$; $P_2 = 27$; $P_3 = 30$
- Average turnaround time: $(24 + 27+30)/3 = 27$

Process Scheduling: FCFS

Suppose that the processes arrive in the order:

$$P_2\ ,\ P_3\ ,\ P_1$$

- The Gantt chart for the schedule is:

- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3
- Turnaround time for $P_1 = 30$; $P_2 = 3$; $P_3 = 6$
- Average turnaround time: (30 + 3 + 6)/3 = 13
- Much better than previous case
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

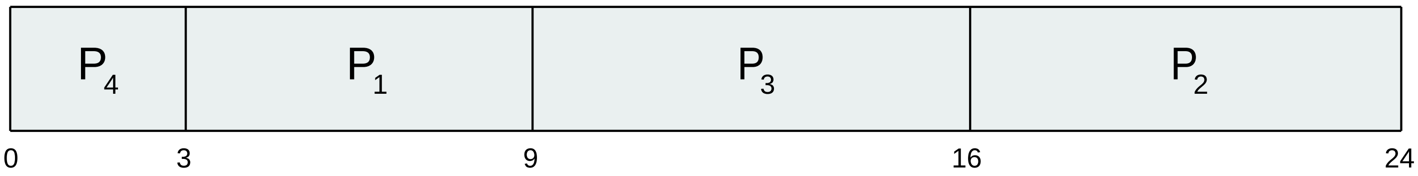
Process Scheduling: SJF

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

Process Scheduling: SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart


P_4 P_1 P_3 P_2

0 3 9 16 24

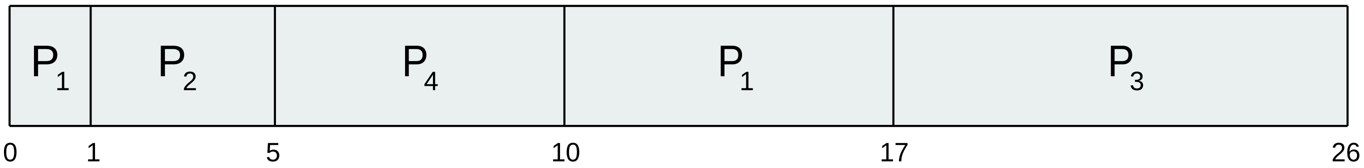
- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7
- Average turnaround time = (9 + 24 +16 + 3) / 4 = 13

Process Scheduling: SRTF

- Now we add the concepts of varying arrival times and preemption to the analysis

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive* SJF Gantt Chart



- Average waiting time = [(0-0)+(1-1)+(17-2)+(5-3)]/4 = 17/4 = 4.25 msec
- Average turnaround time = [(17–0)+(5-1)+(26-2)+(10-3)]/4 = 52/4 = 13 msec

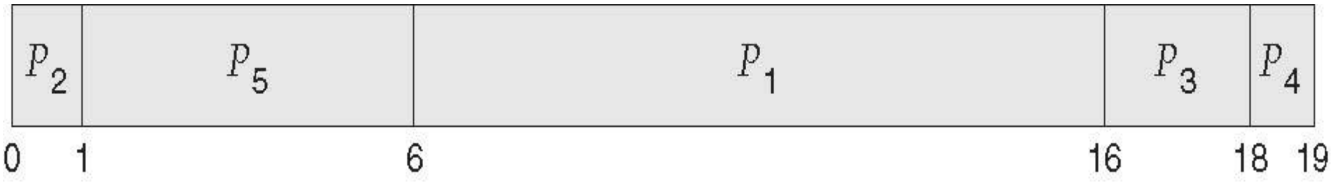
Process Scheduling: Priority

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

Process Scheduling: Priority

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Priority scheduling Gantt Chart



- Average waiting time (6+0+16+18+1)/5 = 41/5 = 8.2 msec
- Average turnaround time (16+1+18+19+6)/5 = 60/5 = 12 msec

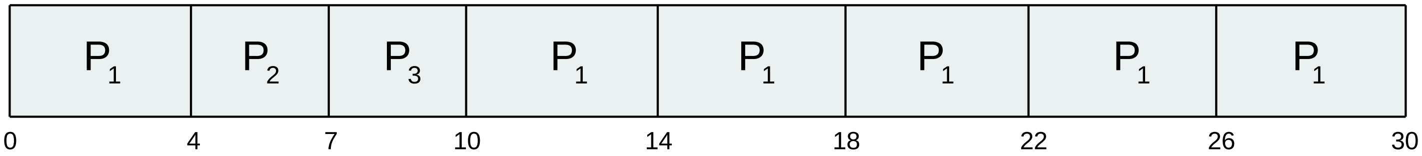
Process Scheduling: RR

- Each process gets a small unit of CPU time (**time quantum** *q*), 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.
- Timer interrupts every quantum to schedule next process
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high

Process Scheduling: RR

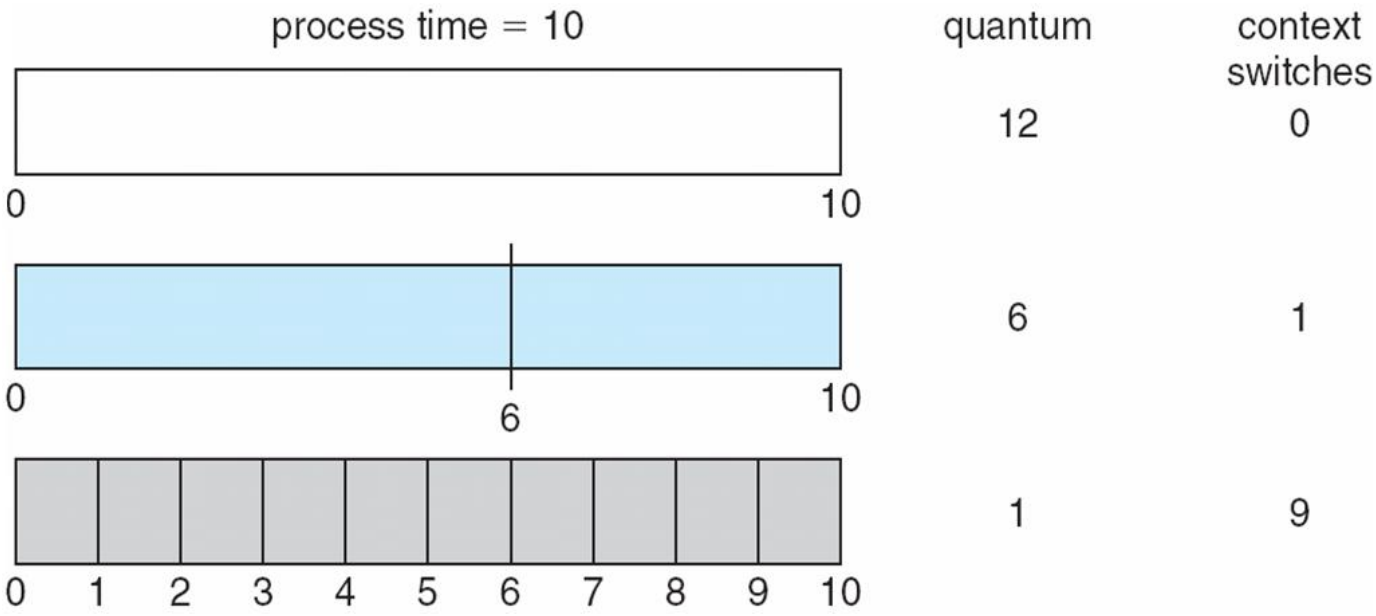
Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart for 4 msec time quanta is:

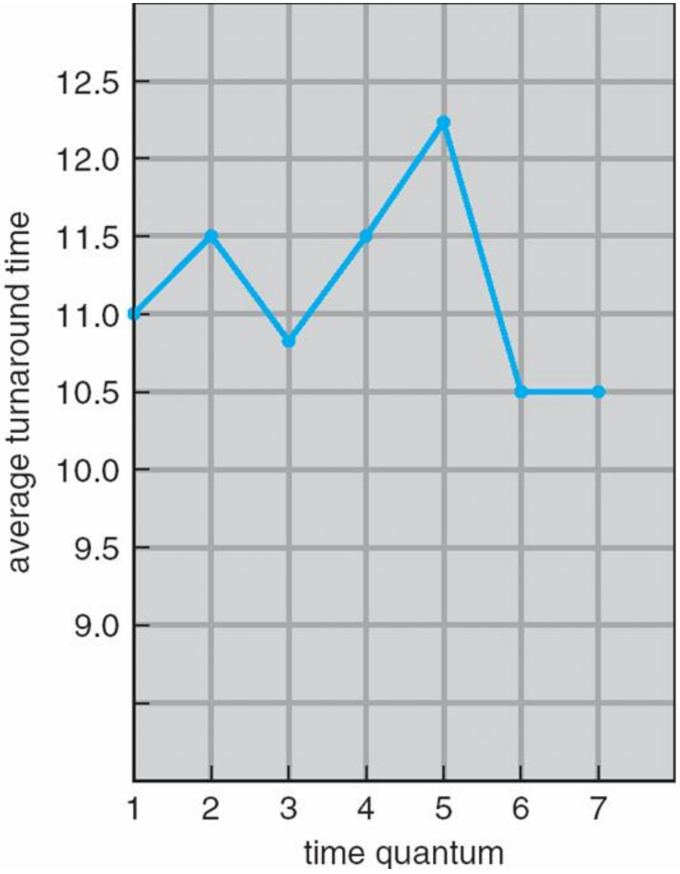


- Average waiting time (0+4+7) = 11/3 = 3.67 msec
- Average turnaround time (30+7+10) = 47/3 = 15.67 msec
- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

Process Scheduling: RR



Process Scheduling: RR



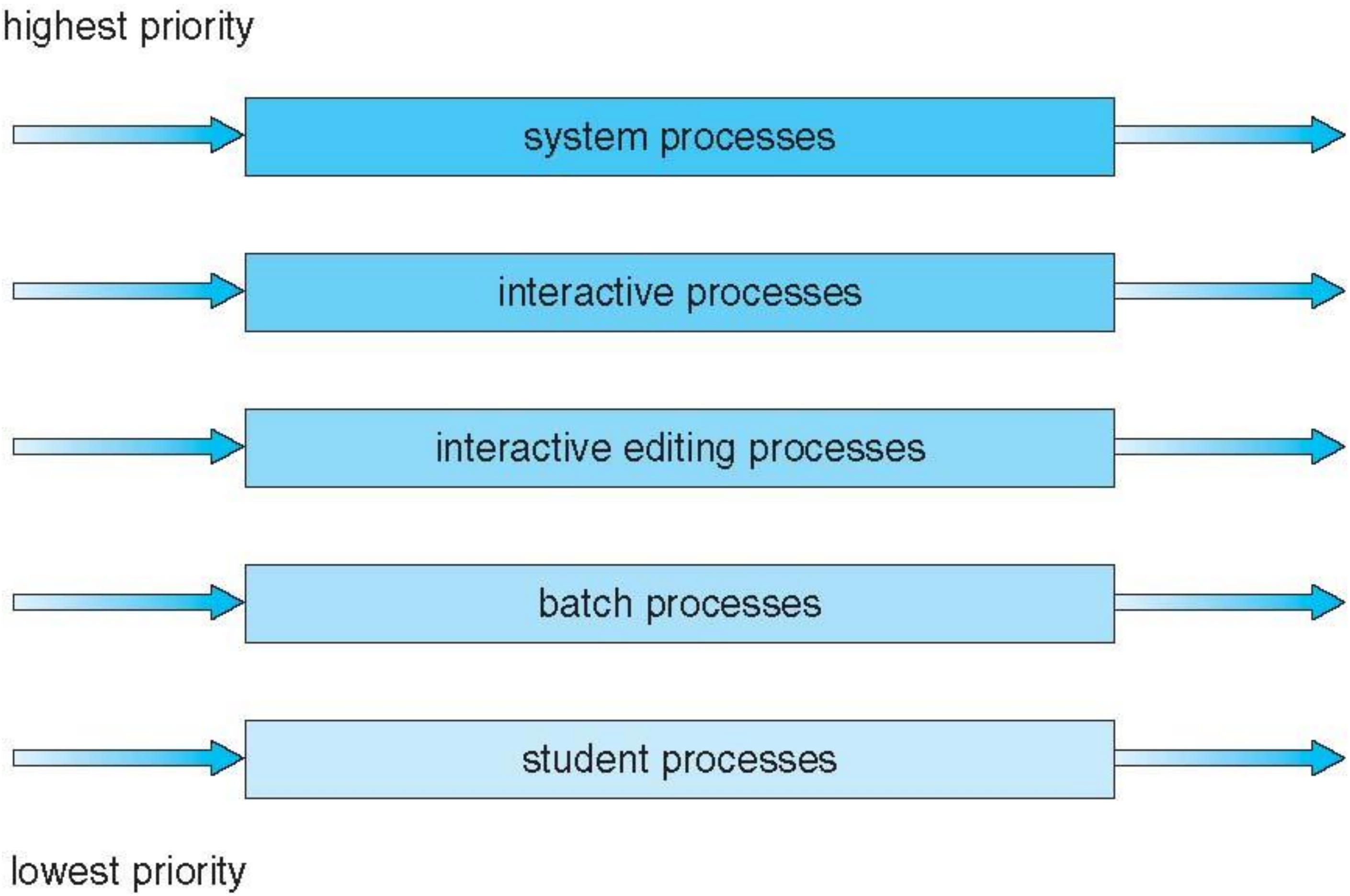
process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts
should be shorter than q

Process Scheduling: Multiple Queue

- Ready queue is partitioned into separate queues, eg:
  - **foreground** (interactive)
  - **background** (batch)
- Process permanently in a given queue
- Each queue has its own scheduling algorithm:
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues:
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
  - 20% to background in FCFS

Process Scheduling: Multiple Queue



highest priority

system processes

interactive processes

interactive editing processes

batch processes

student processes

lowest priority

Process Scheduling: SPN

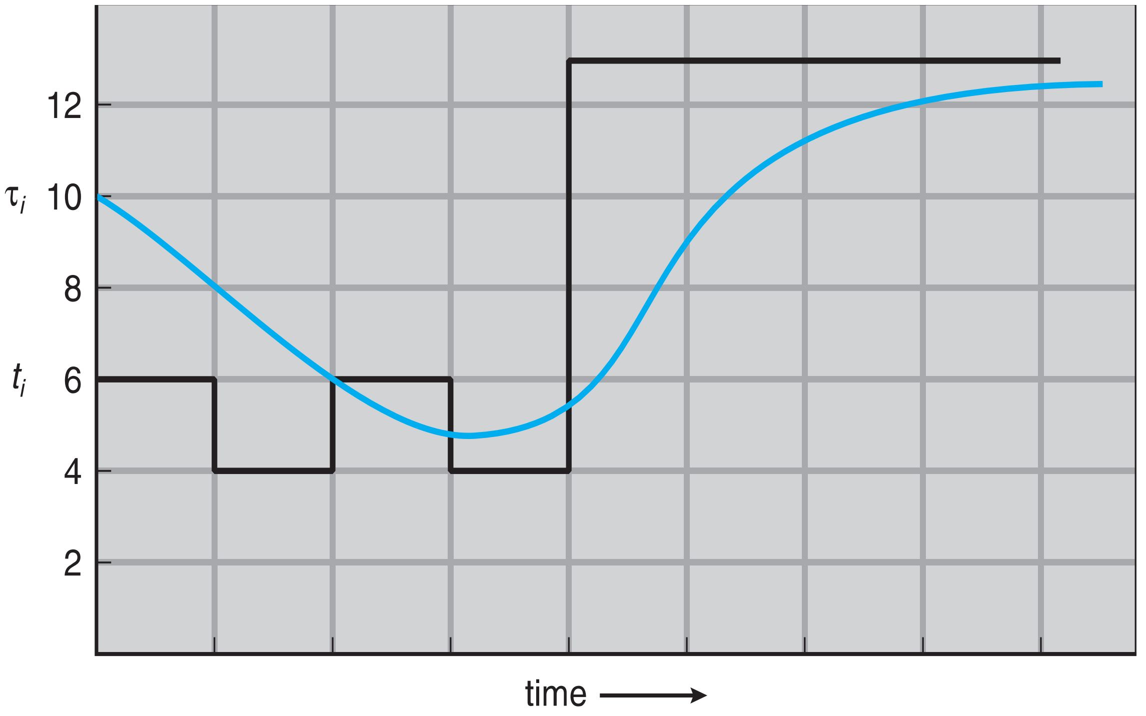
- Predict the length of a **CPU burst**– Then pick the process with shortest predicted next CPU burst

- Can be done by using the length of previous CPU bursts, using exponential averaging

  1. $t_n =$ actual length of n^{th} CPU burst
  2. $\tau_{n+1} =$ predicted value for the next CPU burst
  3. $\alpha, 0 \leq \alpha \leq 1$
  4. Define : $\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n.$

- Commonly, α set to ½

Process Scheduling: SPN



CPU burst (t_i)		6	4	6	4	13	13	13	. . .
"guess" (τ_i)	10	8	6	6	5	9	11	12	. . .

Process Scheduling: SPN

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = \alpha t_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get:

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\alpha t_{n-1} + \ldots$$
$$+ (1 - \alpha)^j \alpha t_{n-j} + \ldots$$
$$+ (1 - \alpha)^{n+1} \tau_0$$

- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

Process Scheduling: GS

- In **Guaranteed Scheduling**, if *n* processes are running, each one is entitled to get 1/*n* of the CPU cycles.
- Keeps track of how much CPU cycles each process has had since its creation.
- Computes the ratio of actual CPU time consumed to CPU time entitled to.
- Runs the process with the lowest ratio until its ratio has moved above that of its closest competitor.

Process Scheduling: LS

- **Lottery Scheduling** gives processes lottery tickets for CPU time.
- Whenever a scheduling decision has to be made, a lottery ticket is chosen at random, and the process holding that ticket gets the CPU.
- Scheduler might hold a lottery 50 times a second, with each winner getting 20 msec of CPU time as a prize.
- More important processes can be given extra tickets, to increase their odds of winning.
- A process holding a fraction *f* of the tickets will get about a fraction *f* of the CPU share.

Process Scheduling: FSS

- **Fair-Share Scheduling** takes into account which user owns a process before scheduling it.
- Each user is allocated some fraction of the CPU.
- Scheduler picks processes in such a way as to enforce the share.
- If two users have each been promised 50% of the CPU, they will each get that, no matter how many processes they have in existence.

Process Scheduling: FSS

- User 1 has four processes, *A*, *B*, *C*, and *D*, and user 2 has only one process, *E*.
- If round-robin scheduling is used, a possible scheduling sequence is this:
  - **A B C D E | A B C D E | A B C D E** …
- If user 1 is entitled to as much CPU time as user 2, FSS scheduling sequence is this:
  - **A E | B E | C E | D E | A E | B E | C E | D E** …
- If user 1 is entitled to twice as much CPU time as user 2, FSS scheduling sequence is this:
  - **A B E | C D E | A B E | C D E** …

Summary

- Process
- Process States
- Process Control Block
- Thread
- Process Scheduling
- Context Switch
- First-Come First-Served
- Shortest Job First
- Shortest Remaining Time Next
- Round Robin Scheduling
- Priority Scheduling
- Multiple Queues Scheduling

Next

Process Management

  - Inter Process Communications (IPC)
  - Process Synchronization