



Part 2: Processes and Threads

- **Process Concept**
- Process Scheduling
- Operation on Processes
- Interprocess Communication
- Threads

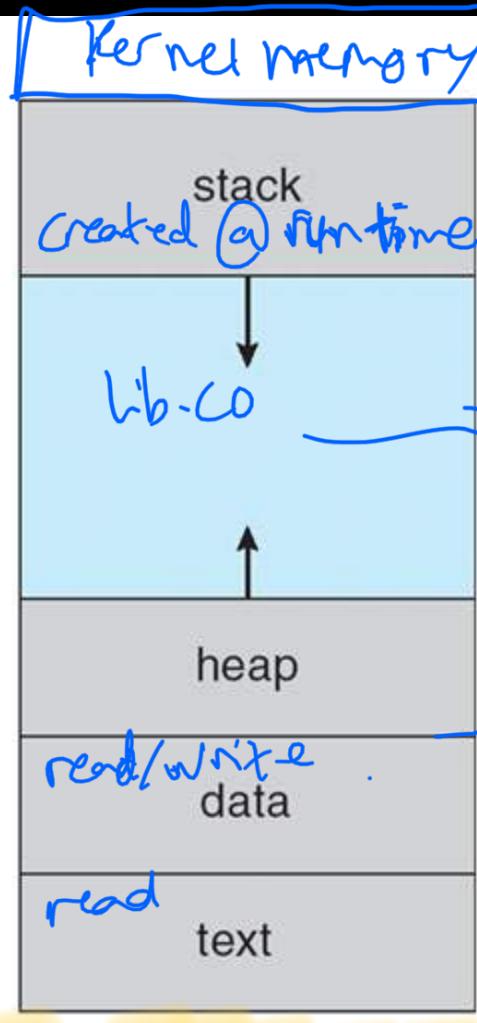
Process Concept

- **Process:** A program in execution; process execution must progress in a sequential fashion
- An operating system executes a variety of processes
 - Batch system – jobs
 - Time-sharing system – user programs & commands
- Textbook uses the terms *job* and *process* almost interchangeably (**Job = Process**)

Process in Memory

Recall base
and limit
registers
from memory
protection?

Base + Limit



(parameters and local variables
in a function)

Statically allocated?

memo mapping segment

(all dynamically allocated variables,
e.g., `a=new int[10]`)

BSS Segment
uninitialized static variable
filled with 0 except char *vs
macloc in C Programming
Static Variables by programmer

(code section, program)

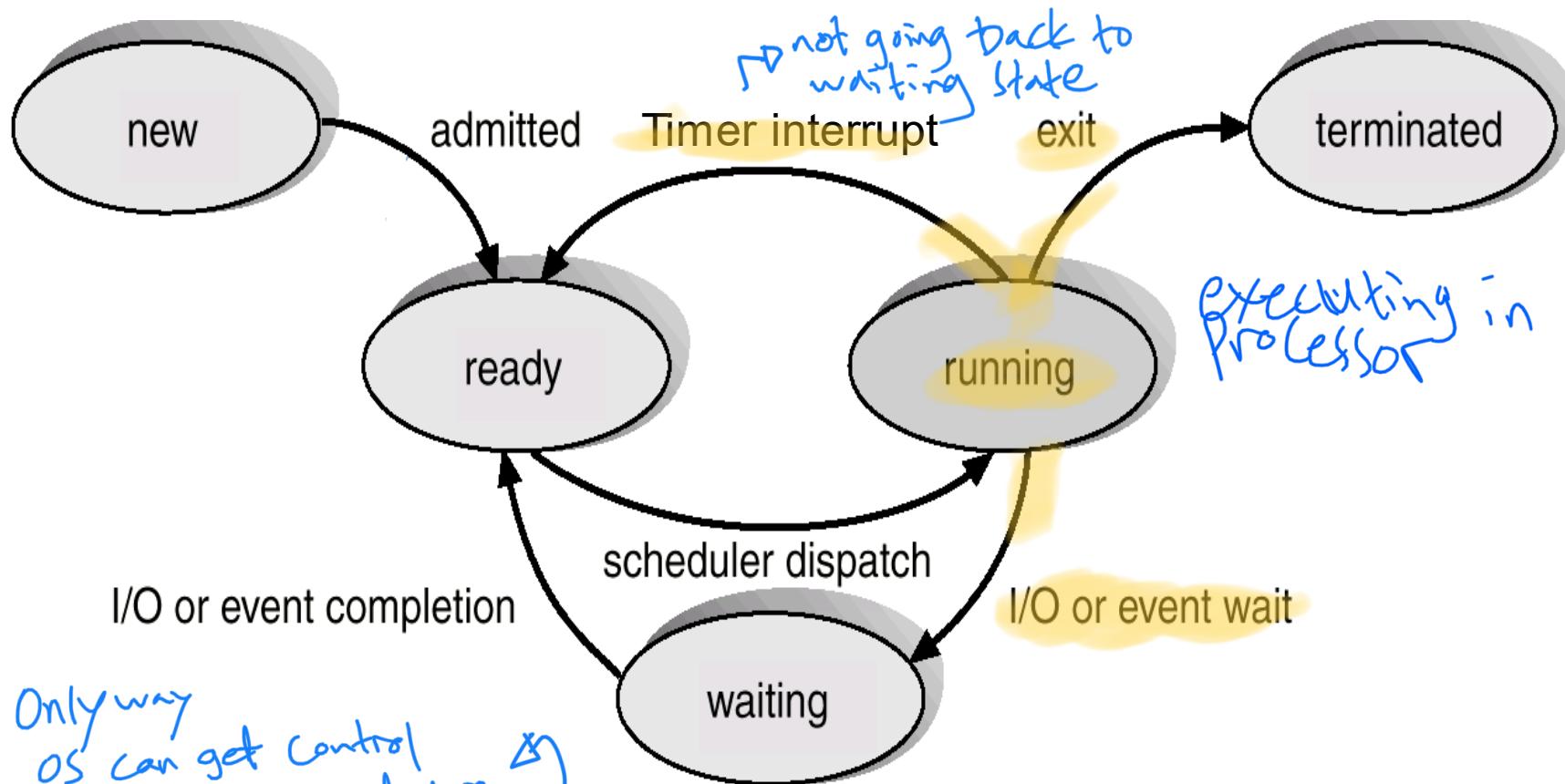


Process State

As a process executes, it changes *state*

1. **New:** The process is being created
2. **Running:** Instructions are being executed
3. **Waiting:** The process is waiting for I/O or event
4. **Ready:** The process is ready to run, but is waiting to be assigned to the CPU
5. **Terminated:** The process has completed

Diagram of Process State Transitions



Timer interrupt is used in multiprogramming systems to switch between ready processes

Process Control Block (PCB)

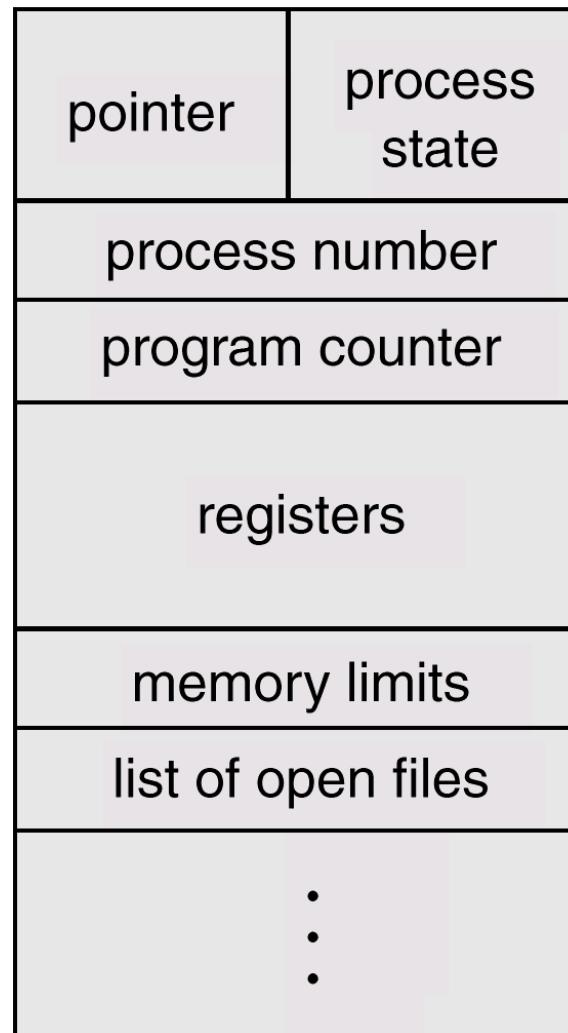
↳ data structure with info of process stored in kernel part of OS memory (access only by OS)

Keeps information associated with each process

1. Pointer to other PCBs (PCBs are maintained in a queue/list structure)
2. Process state
3. Process number (process id)
4. Program counter (pointer to the next instruction)
5. CPU registers
6. Process priority (used in process scheduling)
7. Memory management information (e.g., base and limit register values)
8. Information regarding files (e.g., list of open files)

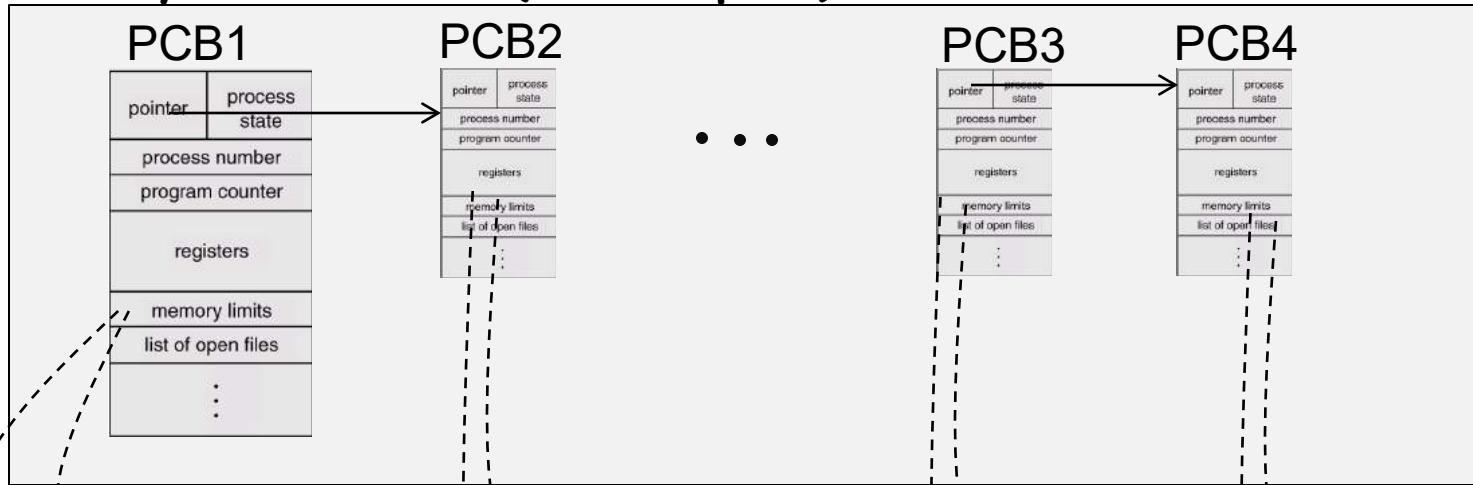
mainly like
linked list

Process Control Block (PCB)



PCB and Processes in Main Memory

Memory area for OS (kernel space)



Base+ Limit

Base

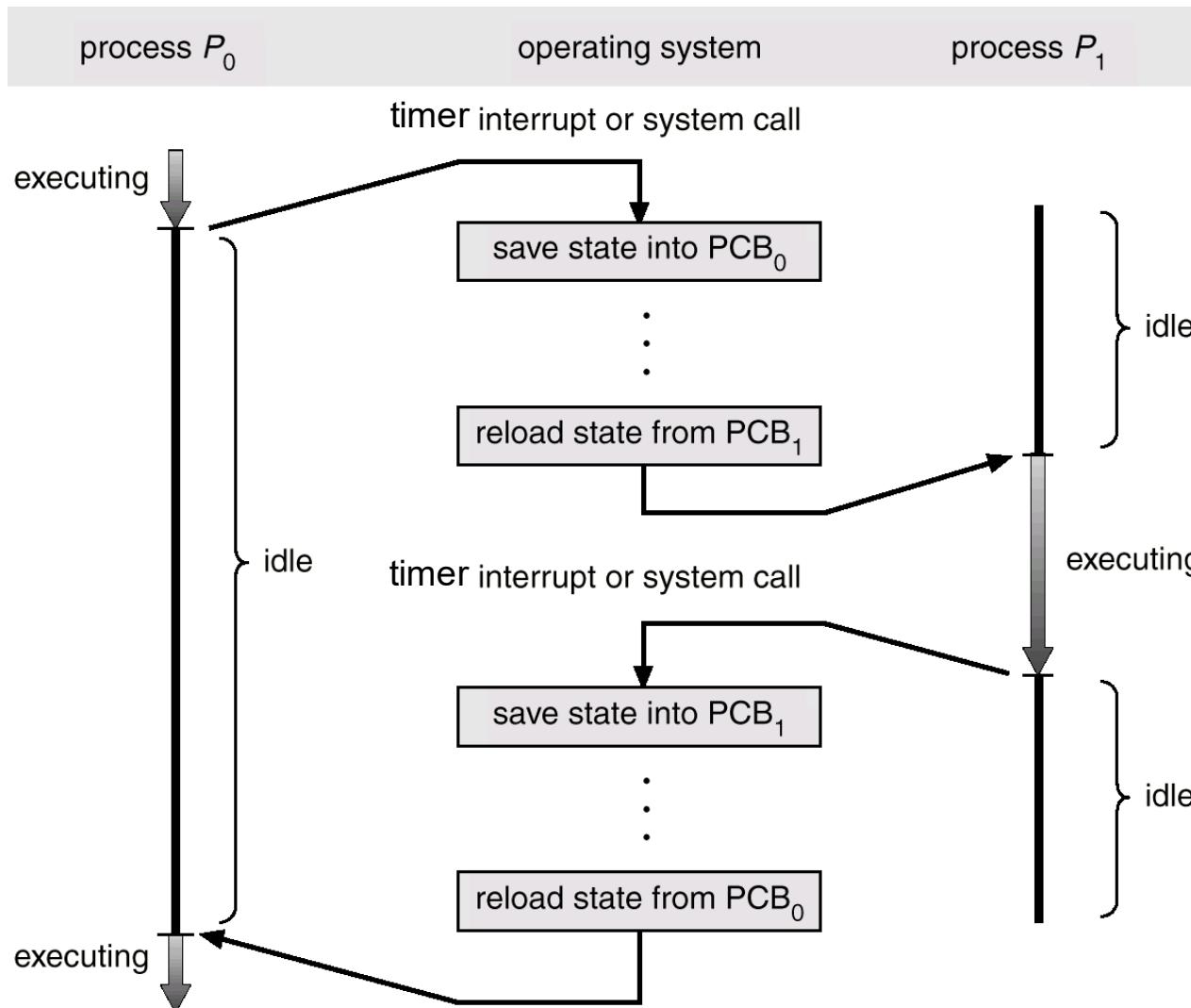
Memory regions for user processes
(user space)



Part 2: Processes and Threads

- Process Concept
- **Process Scheduling**
- Operation on Processes
- Interprocess Communication
- Threads

Context Switch Between Processes



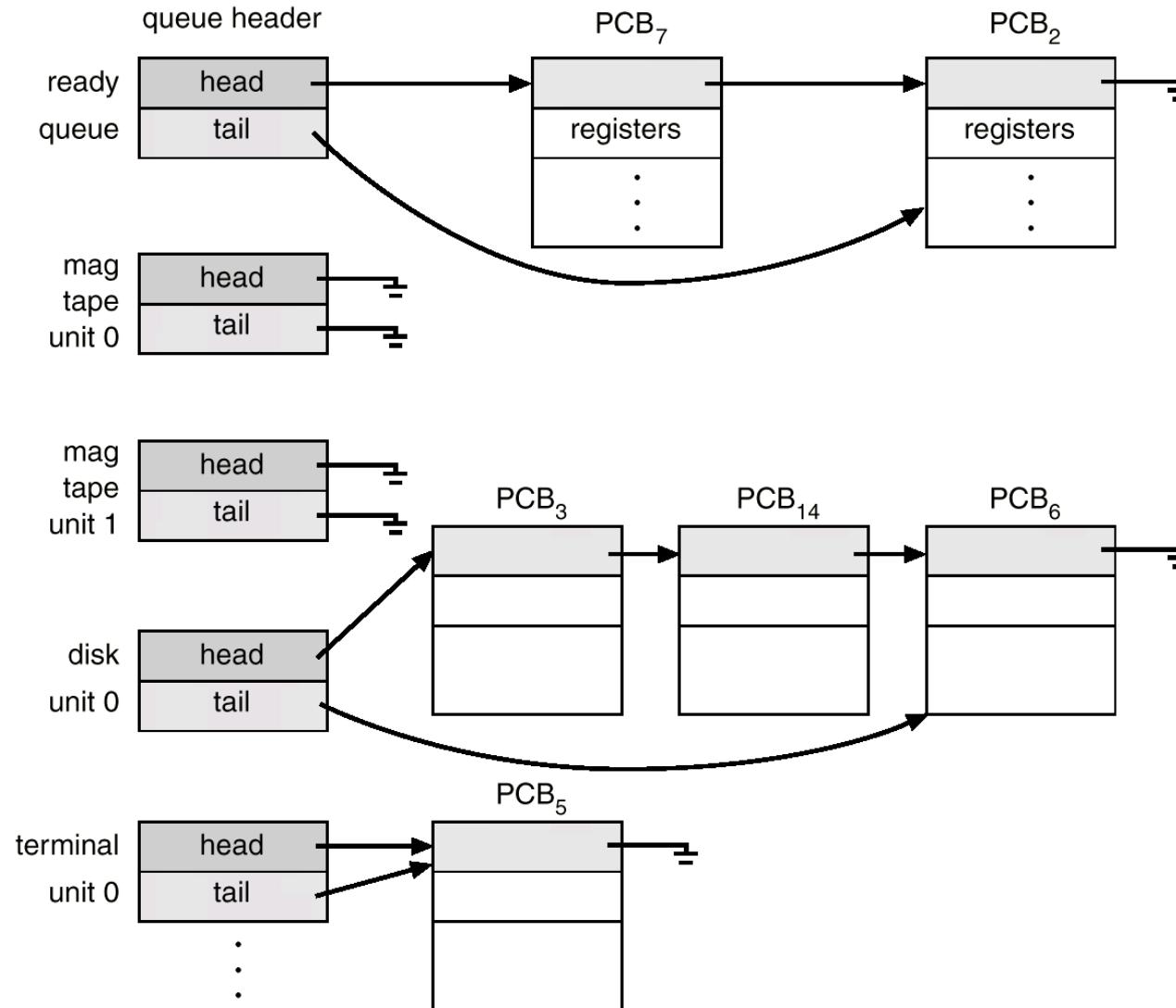
Context Switch

- When CPU switches to another process, the system must save the context (i.e., information) of the old process and load the saved context for the new process
- **Context switch time is overhead:** The system does **NO** useful work while switching between processes

Process Scheduling Queues

- **Job queue:** Set of all processes *with the same state* in the system
 - Ready queue (processes in ready state), device queue (processes in waiting state and waiting for a specific I/O device), etc.
- All the queues are stored in main memory (kernel memory space)
- Process migrates between queues when its state changes

Ready Queue And Various I/O Device Queues



Process Schedulers

- We need multiple schedulers for different purposes
 1. **Long-term scheduler (or job scheduler):** Selects processes from disk and loads them into main memory for execution
(the rest will be stored in HW)
 2. **Short-term scheduler (or CPU scheduler):** Selects from among the processes that are ready to execute, and allocates the CPU to one of them
(decide which one runs next)
 3. **Medium-term scheduler:**
 - When the system load is heavy, swaps out a partially executed process from memory to hard disk
 - When the system load is light, such processes can be swapped back into main memory

Process Schedulers (Cont.)

- Short-term scheduler is invoked frequently (e.g., 100 milliseconds) in a multiprogrammed system for responsiveness and efficiency purposes
- Long-term scheduler is invoked infrequently (e.g., seconds or minutes)
- The degree of multiprogramming is initially controlled by the long-term scheduler, and thereafter by the medium-term scheduler

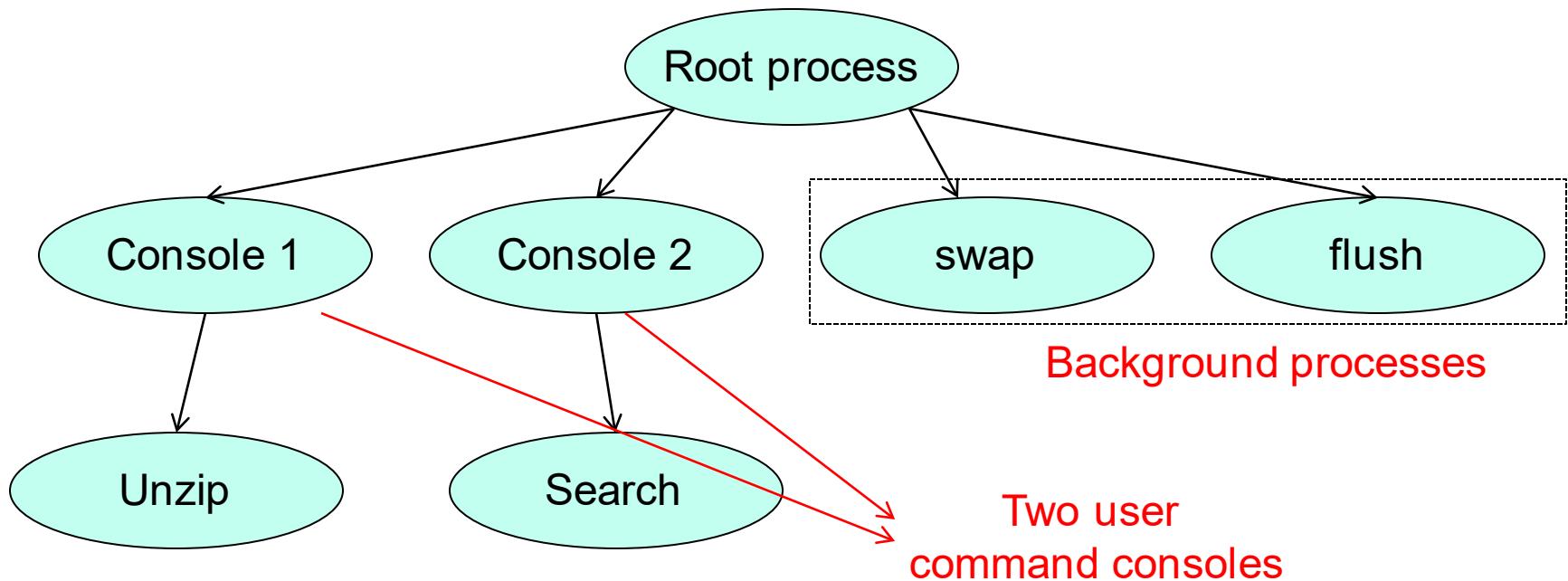


Part 2: Processes and Threads

- Process Concept
- Process Scheduling
- **Operation on Processes**
- Interprocess Communication
- Threads

Process Creation

- Parent process creates children processes, which, in turn create other processes, forming a tree of processes (**fork**)





Process Creation (Cont.)

- Two possible execution orders
 1. Parent and children execute concurrently (and independently)
 2. Parent waits until all children terminate (**wait(),join()**)
- Examples
 - Many web browsers nowadays fork a new process when we access a new page in a “tab”
 - OS may create background processes for monitoring and maintenance tasks



Process Termination

- Two possible ways to terminate a process
 - **Exit:** Process executes last statement and asks the OS to delete it
 - A child may output return data to its parent
 - Process resources are de-allocated by the OS
 - **Abort:** Parent may terminate execution of children processes at any time
 - Child has exceeded allocated resources
 - Task assigned to child is no longer required
 - Parent is exiting



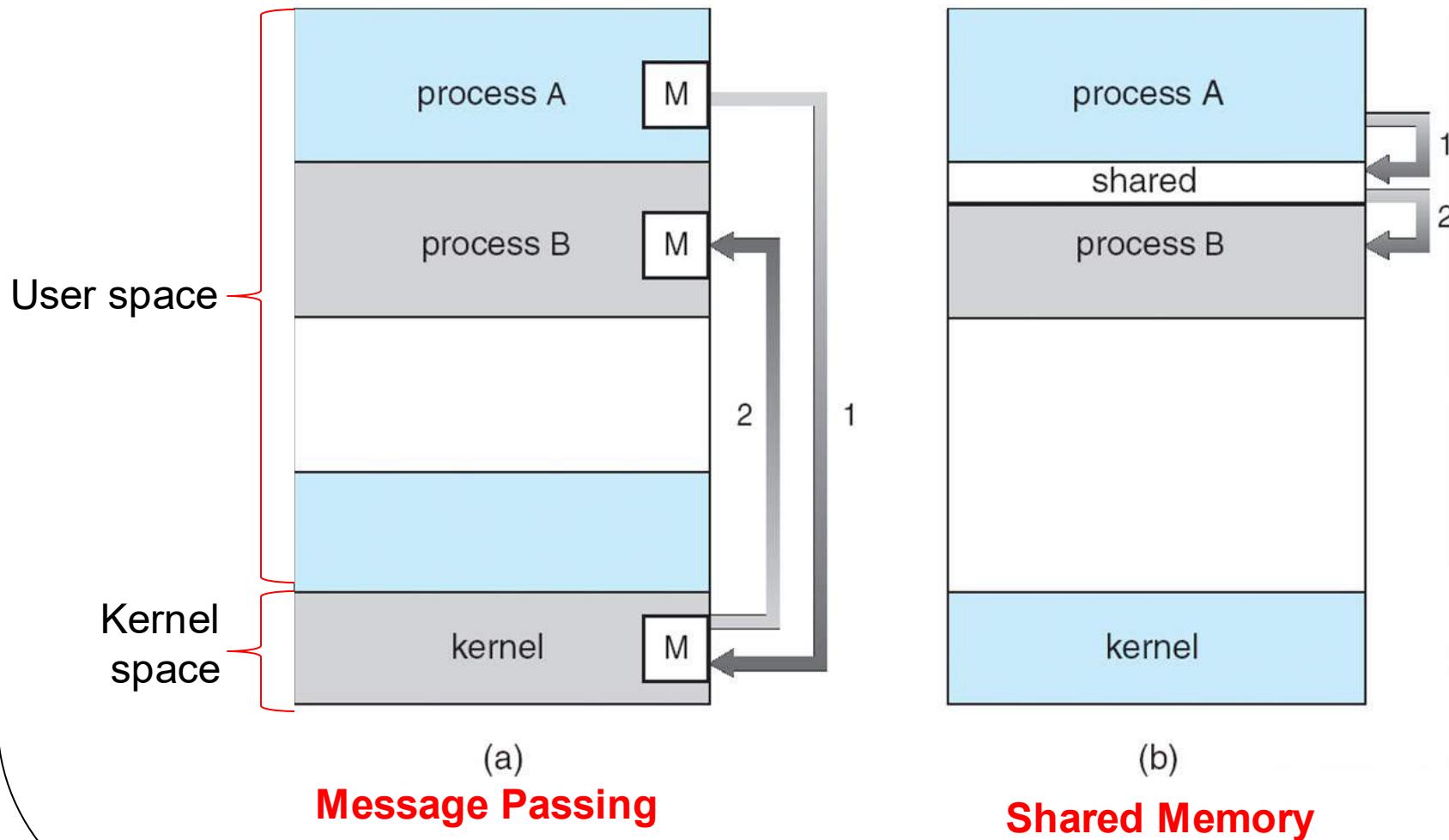
Part 2: Processes and Threads

- Process Concept
- Process Scheduling
- Operation on Processes
- **Interprocess Communication**
- Threads

Cooperating Processes

- An *independent process* cannot affect or be affected by the execution of other processes
- A *cooperating process* on the other hand can affect or be affected by the execution of other processes
 - Such processes have to communicate with each other to share data
 - Two models of Inter-Process Communication (IPC)
 - **Message passing**
 - **Shared memory**

IPC Models



Differentiate between shared memory and message passing model in OS.

Operating System Hardware Software & Coding

Shared memory system is the fundamental model of inter process communication. In a shared memory system, in the address space region the cooperating communicate with each other by establishing the shared memory region.

Shared memory concept works on fastest inter process communication.

If the process wants to initiate the communication and it has some data to share, then establish the shared memory region in its address space.

After that, another process wants to communicate and tries to read the shared data, and must attach itself to the initiating process's shared address space.

Message Passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space.

For example – Chat program on the World Wide Web.

Message passing provides two operations which are as follows –

- Send message
- Receive message

Messages sent by a process can be either fixed or variable size.

For fixed size messages the system level implementation is straight forward. It makes the task of programming more difficult.

The variable sized messages require a more system level implementation but the programming task becomes simpler.

If process P1 and P2 want to communicate they need to send a message to and receive a message from each other that means here a communication link exists between them.

Differences

The major differences between shared memory and message passing model -

Shared Memory

It is one of the region for data communication

It is used for communication between single processor and multiprocessor systems where the processes that are to be communicated present on the same machine and they are sharing common address space.

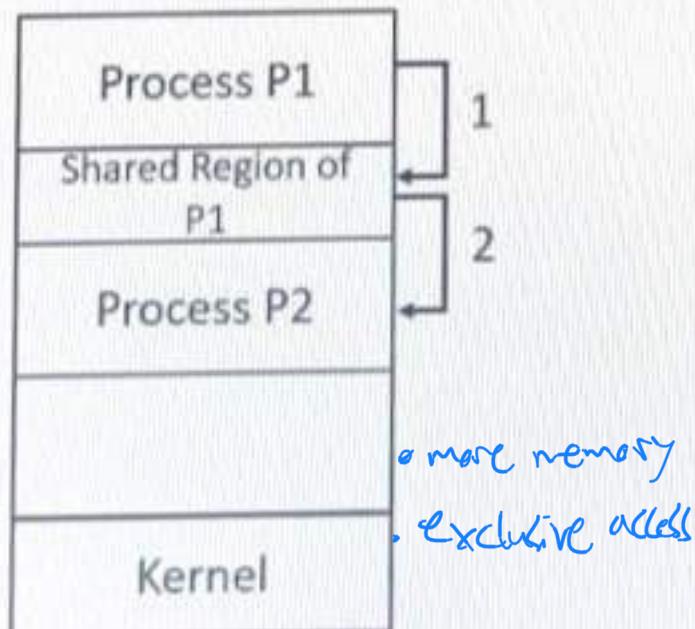
The shared memory code that has to be read or write the data that should be written explicitly by the application programmer.

It is going to provide a maximum speed of computations because the communication is done with the help of shared memory so system calls are used to establish the shared memory.

In shared memory make sure that the processes are not writing to the same location simultaneously.

It follows a faster communication strategy when compared to message passing technique.

Given below is the structure of shared memory system -



Shared Memory System

Message Passing

Mainly the message passing is used for communication.

It is used in distributed environments where the communicating processes are present on remote machines which are connected with the help of a network.

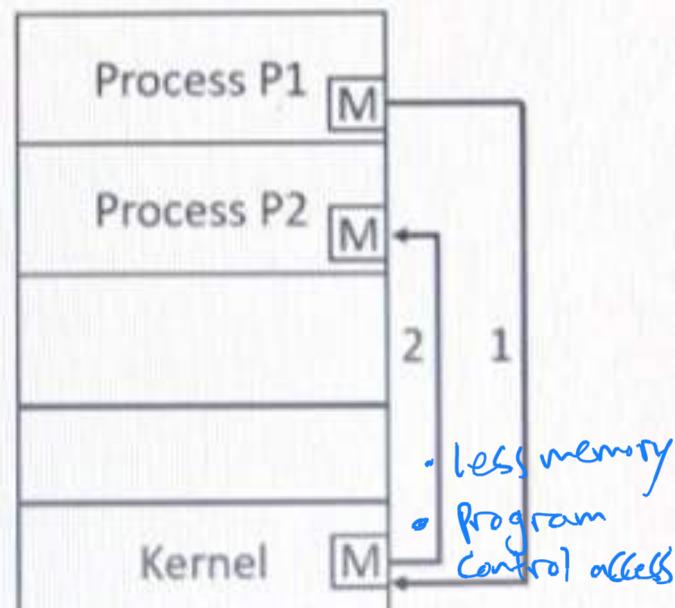
Here no code is required because the message passing facility provides a mechanism for communication and synchronization of actions that are performed by the communicating processes.

Message passing is a time consuming process because it is implemented through kernel (system calls).

Message passing is useful for sharing small amounts of data so that conflicts need not occur.

In message passing the communication is slower when compared to shared memory technique.

Given below is the structure of message passing system -



Message Passing System

IPC – Message Passing

- Processes communicate and synchronize their actions **without resorting to shared variables**
- Two operations (system calls) are required
 - **send(message)** – message size fixed or variable
 - **receive(message)**
- If two processes wish to communicate, they need to
 - Establish a *communication link* between them
 - Exchange messages via send/receive

Direct vs. Indirect Message Passing

- **Direct:** Processes must name each other explicitly
 - *send (P,message):* Send a message to process P
 - *receive(Q,message):* Receive a message from process Q
- **Indirect:** Messages are sent to or received from mailboxes (also referred to as ports)
 - Mailbox is an object into which messages are placed and removed (like a queue)
 - Primitives are:
 - *send(A,message):* Send a message to mailbox A
 - *receive(A,message):* Receive a message from mailbox A

Producer-Consumer Process Paradigm

- Classical paradigm for cooperating processes
 - *Producer* process produces information that is consumed by a *consumer* process
 - Shared buffer used for storing information
- Two models for shared buffer
 1. *unbounded-buffer* places no practical limit on the size of the buffer
 2. *bounded-buffer* assumes that there is a fixed buffer size

Producer-Consumer: Bounded Buffer

//declare a *mailbox* with capacity of B messages (buffer size B)

void producer(void)

```
{  
    message m;  
    while (1) {  
        //pre-processing...  
        while(mailbox is full) wait;  
        send(mailbox, m);  
    }  
}
```



Does this example use direct
or indirect communication

void consumer(void)

```
{  
    message m;  
    while (1) {  
        while(mailbox is empty) wait;  
        receive(mailbox, m);  
        //post-processing...  
    }  
}
```



Part 2: Processes and Threads

- Process Concept
- Process Scheduling
- Operation on Processes
- Interprocess Communication
- **Threads**

Threads

- This part is for **self-learning**
- **About Labs**
 - Nachos uses term “thread”, but means “process”
 - All concepts and mechanisms that we learnt about processes are also applicable to Nachos threads
 - Thread control block = Process control block
 - Thread state = Process state
 - System calls: fork, exit, etc.



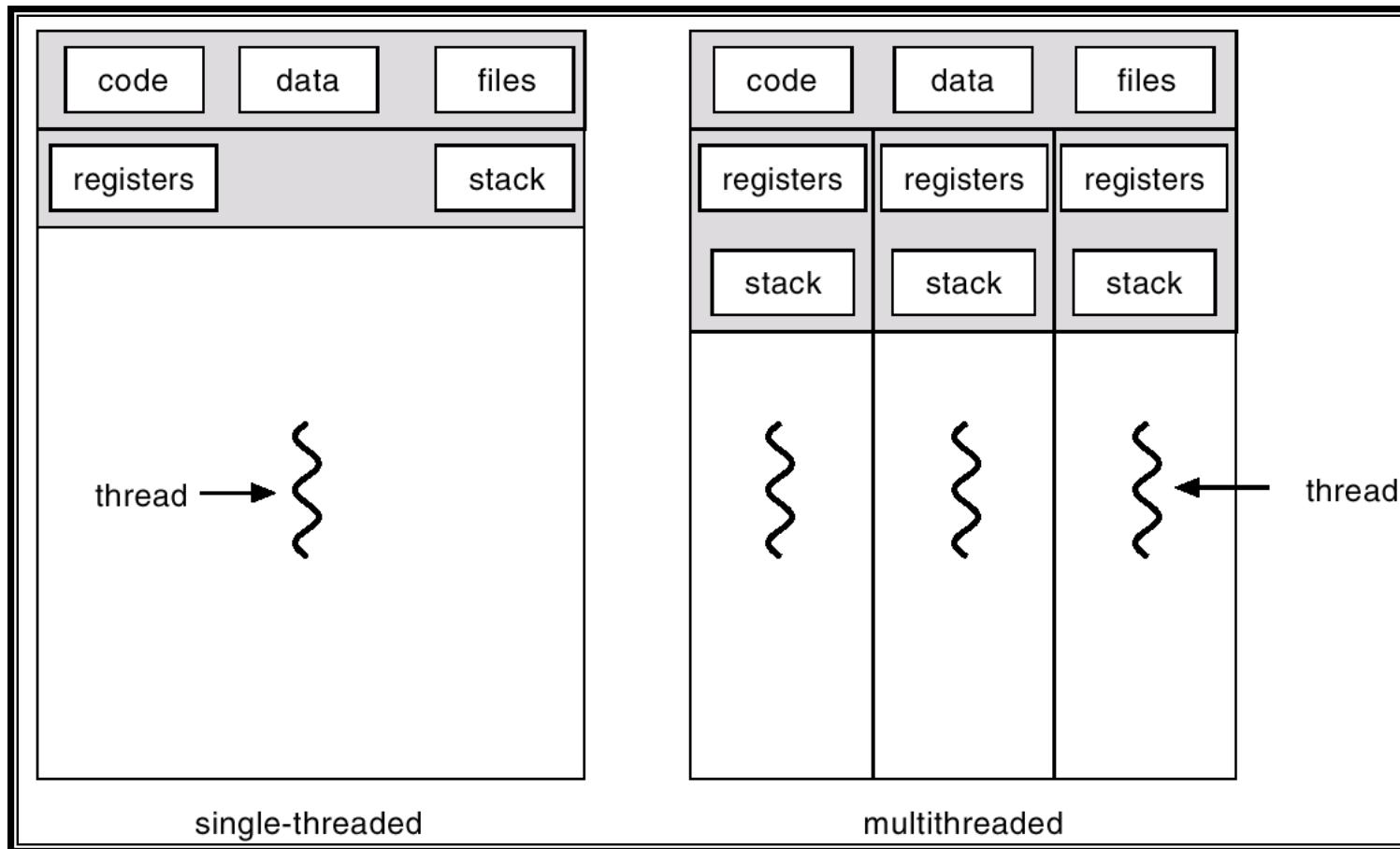
Threads

- Overview
- Single vs Multithreading Process
- Benefits of threads
- Types of threads
- Multithreading models

Overview

- A thread (or lightweight process) is a basic unit of CPU utilization; it consists of:
 - Thread id
 - Program counter
 - Register set
 - Stack space
 - A thread shares with its peer threads in the same process:
 - Code and data sections
 - Operating system resources (open files, etc.)
 - A traditional or heavyweight process is an executing program with a single thread of control
- ↳ Efficient information sharing

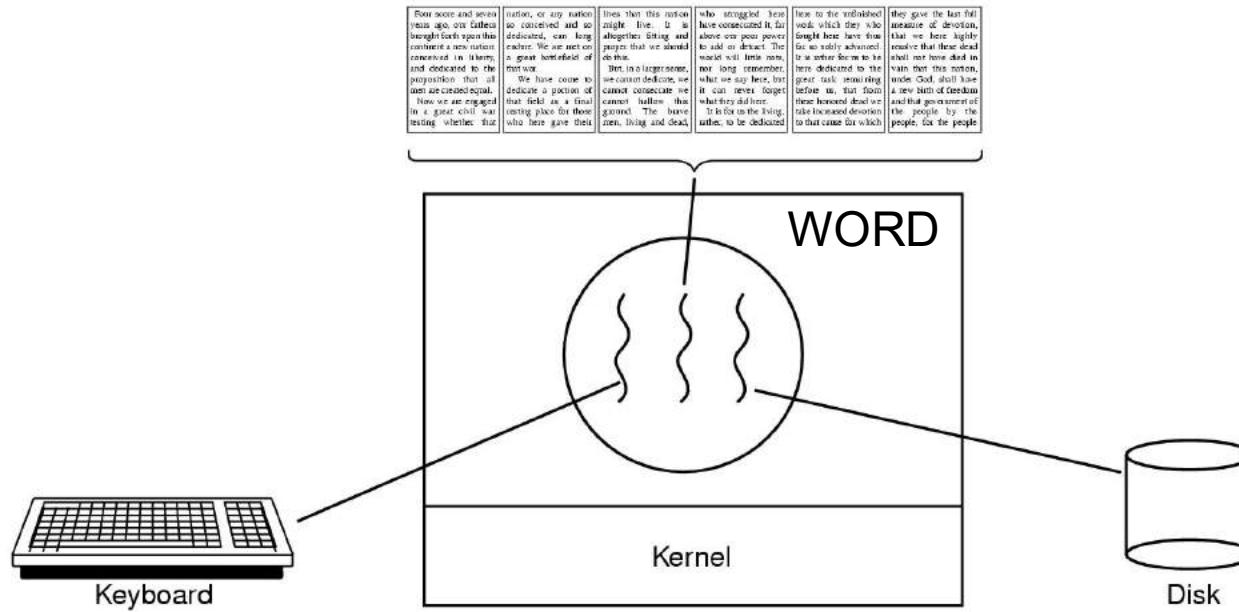
Single vs Multithreaded Processes



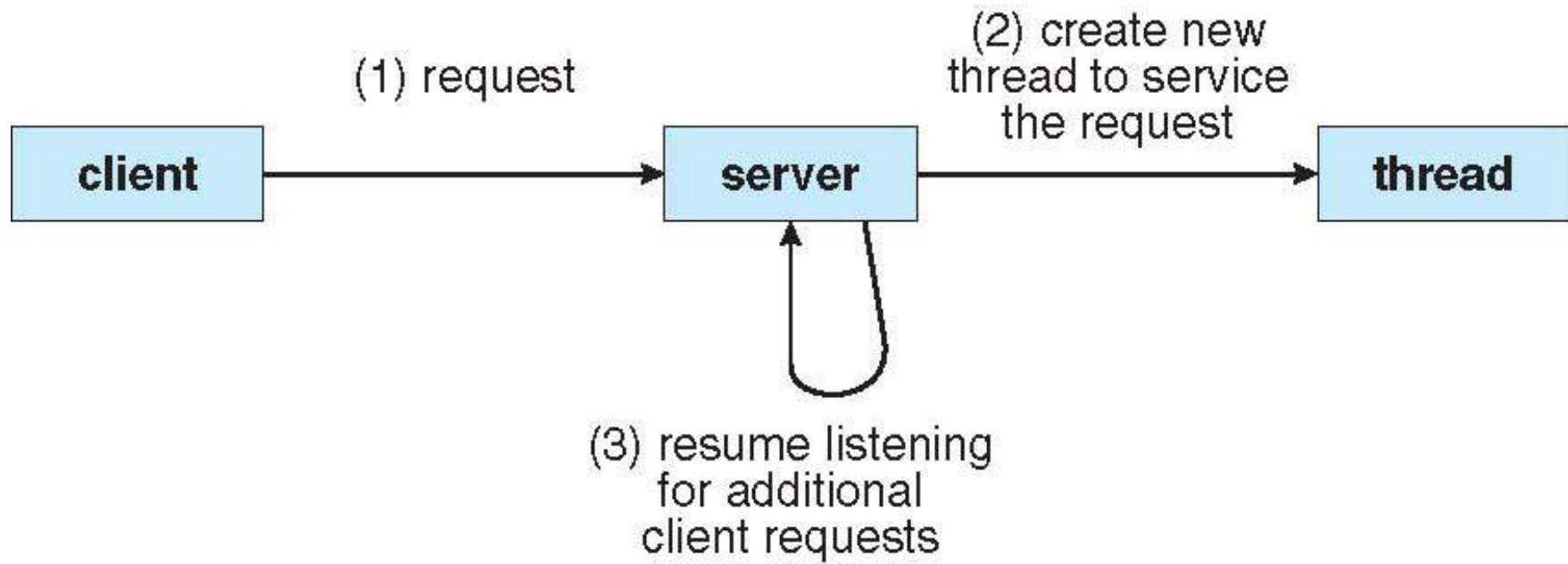
difference is amount of Sharing

Advantage of Multithreaded Processes

- In a multithreaded process, while one thread is blocked and waiting, a second thread can run
 - Cooperation of multiple threads in the same process results in higher throughput



Example: Multithreaded Server



Thread Implementation Models

- **Paradox**

- Allow users to implement an arbitrary number of threads, **BUT**
- OS kernel can support a limited number of threads due to resource constraints

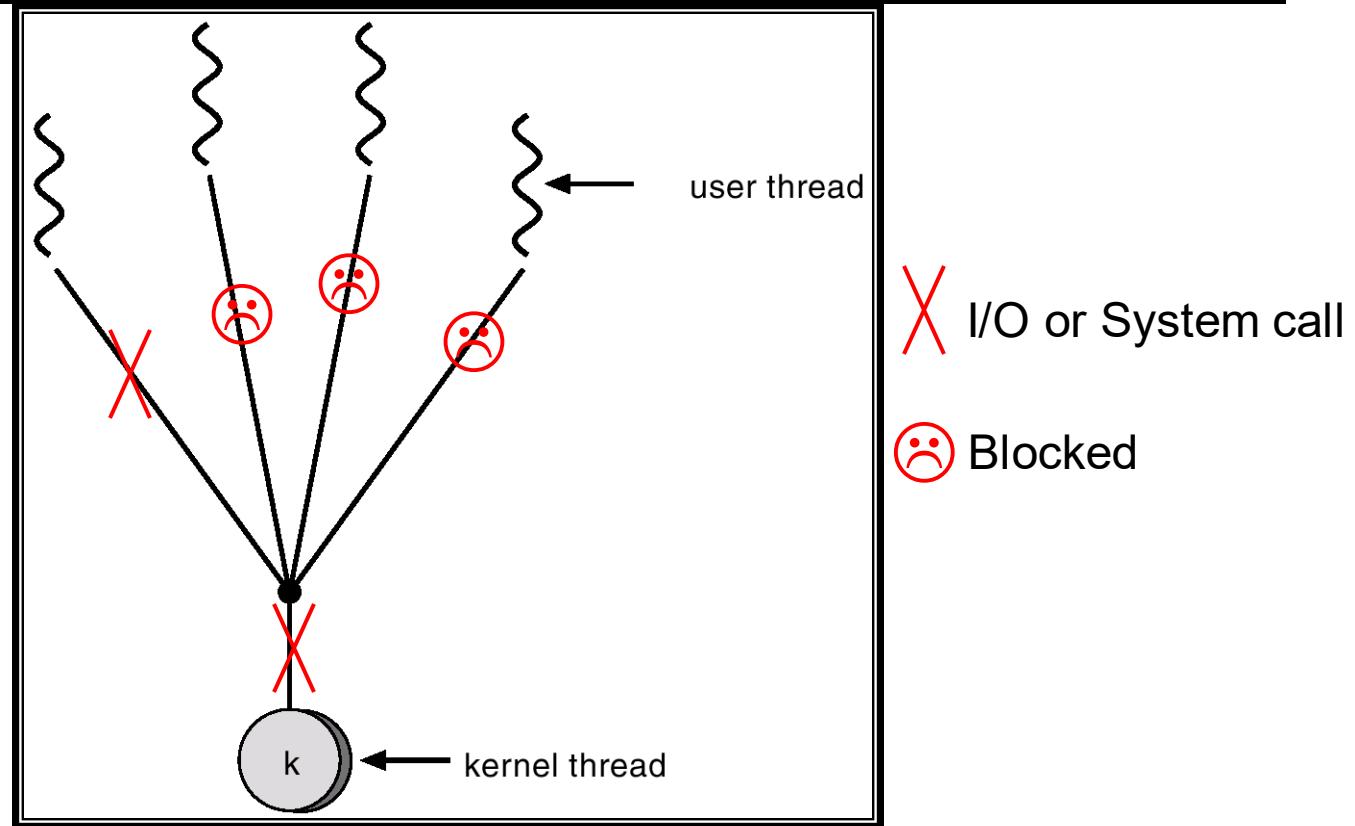
- **Solution:** Two layers of abstraction

- **User threads (logical):** Created in user space
 - Allows users to create as many threads as they want
- **Kernel threads (physical):** Created in kernel space
 - Slower to create and manage than user threads
 - Resources are eventually allocated in kernel threads
- OS maintains the mapping from user to kernel threads

Multithreading Models

- Model that defines the mapping between user threads and kernel threads
 - Many-to-One
 - One-to-One
 - Many-to-Many
- Why is there no One-to-Many mapping?
 - Wastage of OS resources to map a single user thread to many kernel threads

Many-to-One Mapping



Disadvantage: A blocked user thread, will also block other user threads mapped to the same kernel thread

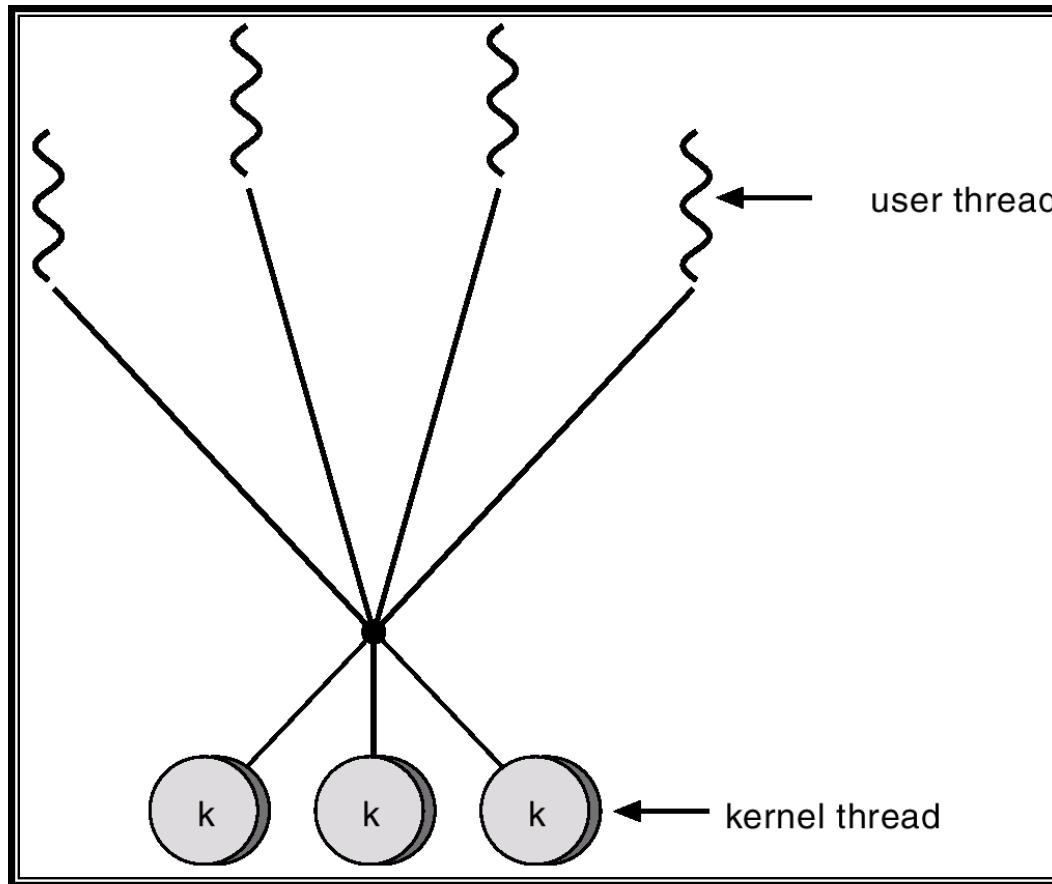
One-to-One Mapping

- Each user-level thread maps to a unique kernel thread
 - Examples: Windows 95/98/NT/2000, OS/2, Linux
- Provides more concurrency than many-to-one model
- **Disadvantages:**
 - Creating a user thread requires creating the corresponding kernel thread
 - May create too many kernel threads which is a burden on the system

Many-to-Many Mapping

- Allows many user level threads to be mapped to many kernel threads (see next slide)
 - Examples: Solaris 2, Windows NT/2000
- Allows the operating system to create a sufficient number of kernel threads
- Does not have the disadvantages of one-to-one and many-to-one models
- **Disadvantage:** Not easy to decide the mapping

Many-to-Many Mapping



Advanced Readings

- Beej's Guide to Unix Interprocess Communication
 - <http://beej.us/guide/bgipc/>
 - Very good introduction on the IPC implementation inside Unix
- Tutorials on how to use thread libraries:
 - POSIX pthreads, <http://randu.org/tutorials/threads/>
 - Windows Process and Thread Functions
[http://msdn.microsoft.com/en-us/library/windows/desktop/ms684847\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms684847(v=vs.85).aspx)

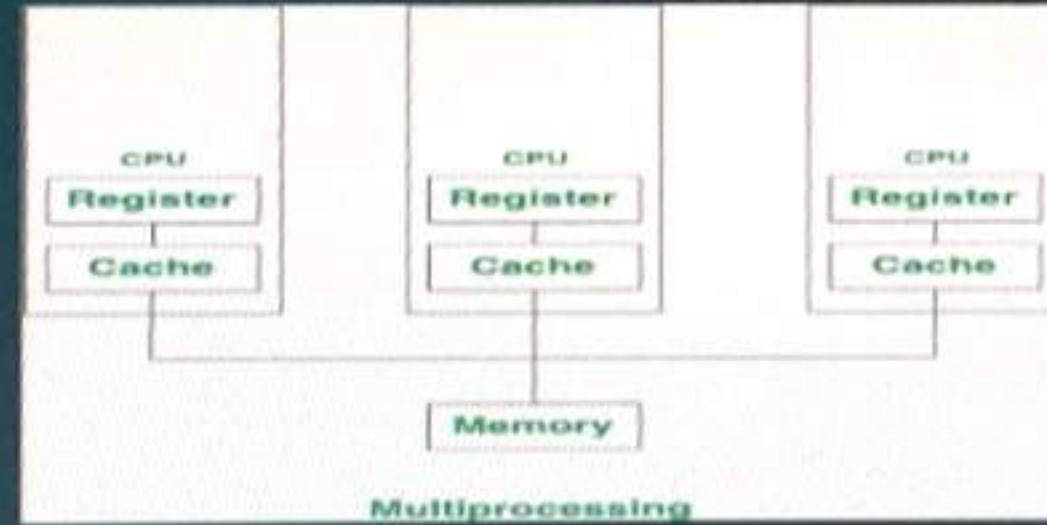
X

Difference between Multiprogramming, Multithreading, Multitasking and Multiprocessing

S.No.	Multiprogramming	Multitasking	Multithreading	Multiprocessing
1.	In multiprogramming, multiple programs execute at a same time on a single device.	In Multitasking, a single resource is used to process multiple tasks.	Multithreading is an extended form of multitasking.	In multiprocessing, multiple processing units are used by a single device.
2.	The process resides in the main memory.	The process resides in the same CPU.	More than one thread processed on a single CPU.	The process switches from one to another CPU as multiple processing units are used.
3.	It uses batch OS. The CPU is utilized completely while execution.	It is time sharing as the task assigned switches regularly.	The tasks are always further divided into sub tasks.	It carries multiple processors to execute the task.
4.	The processing is slower, as a single job resides in the main memory while execution.	Multitasking follows the concept of context switching.	It allows a single process to get multiple code segments.	A large amount of work can be done in a short period of time.

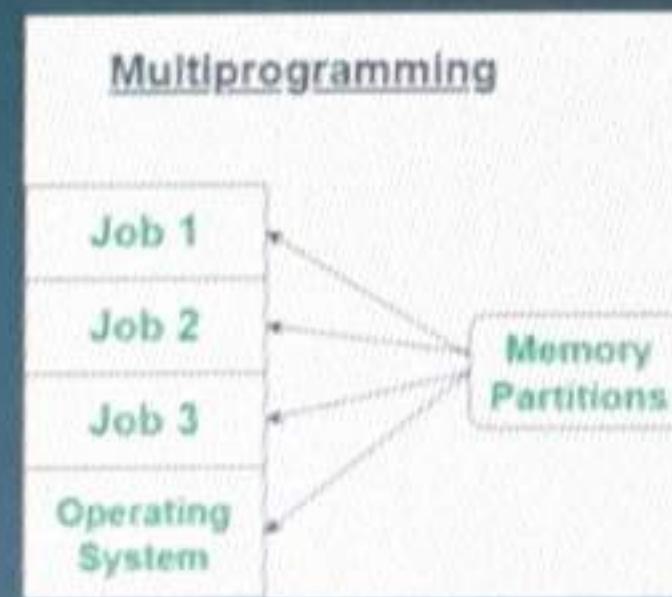
1. Multiprocessing :

Multiprocessing is a system that has two or more than one processors. In this, CPUs are added for increasing computing speed of the system. Because of Multiprocessing, there are many processes that are executed simultaneously. Multiprocessing are further classified into two categories: Symmetric Multiprocessing, Asymmetric Multiprocessing.



2. Multi-programming :

Multi-programming is more than one process running at a time, it increases CPU utilization by organizing jobs (code and data) so that the CPU always has one to execute. The motive is to keep multiple jobs in main memory. If one job gets occupied with Input/output, CPU can be assigned to other job.



Difference between Multiprocessing and Multiprogramming :

S.No.	Multiprocessing	Multiprogramming
1.	The availability of more than one processor per system, that can execute several set of instructions in parallel is known as multiprocessing.	The concurrent application of more than one program in the main memory is known as multiprogramming.
2.	The number of CPU is more than one.	The number of CPU is one.
3.	It takes less time for job processing.	It takes more time to process the jobs.
4.	In this, more than one process can be executed at a time.	In this, one process can be executed at a time.
5.	It is economical.	It is economical.
6.	The number of users is can be one or more than one.	The number of users is one at a time.
7.	Throughput is maximum.	Throughput is less.
8.	Its efficiency is maximum.	Its efficiency is Less.