Module-2

By Dr. Renuka S. Gound

Module-2

Process Management: Process Concept The Processes, Process States, PCB, Process Scheduling Queues, Schedulers, Context Switch, Operations on Process, Inter-Process Communication Shared-Memory System, Message Passing System

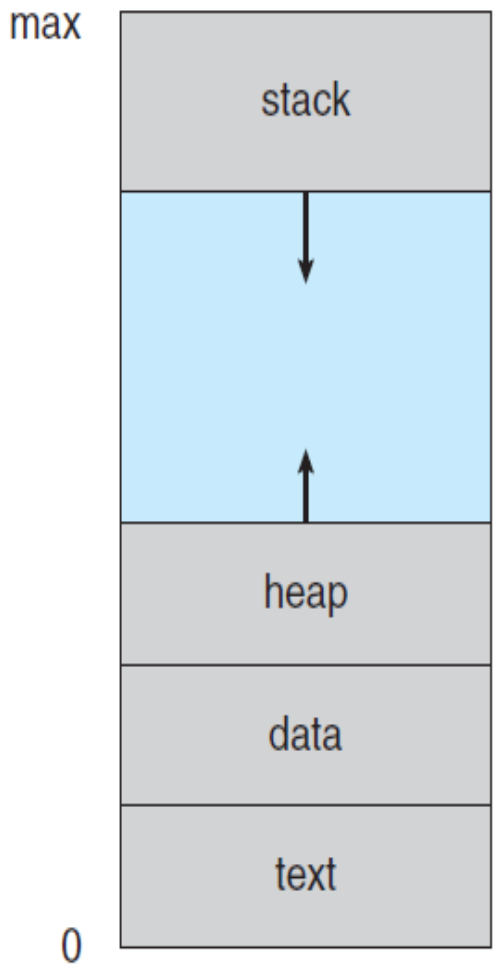
CPU Scheduling: Basics, CPU-I/O Burst Cycle, CPU Scheduler Preemptive Scheduling, Dispatcher, Scheduling Criteria, Scheduling Algorithms FCFS, SJF, Round-Robin, Priority

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 a sequential fashion
- Multiple parts
  - The program code, also called the text section
  - Current activity including program counter, processor registers
  - Stack containing temporary data
    - Function parameters, return addresses, local variables
    - Data section containing global variables
  - Heap containing memory dynamically allocated during run time

Process Concept

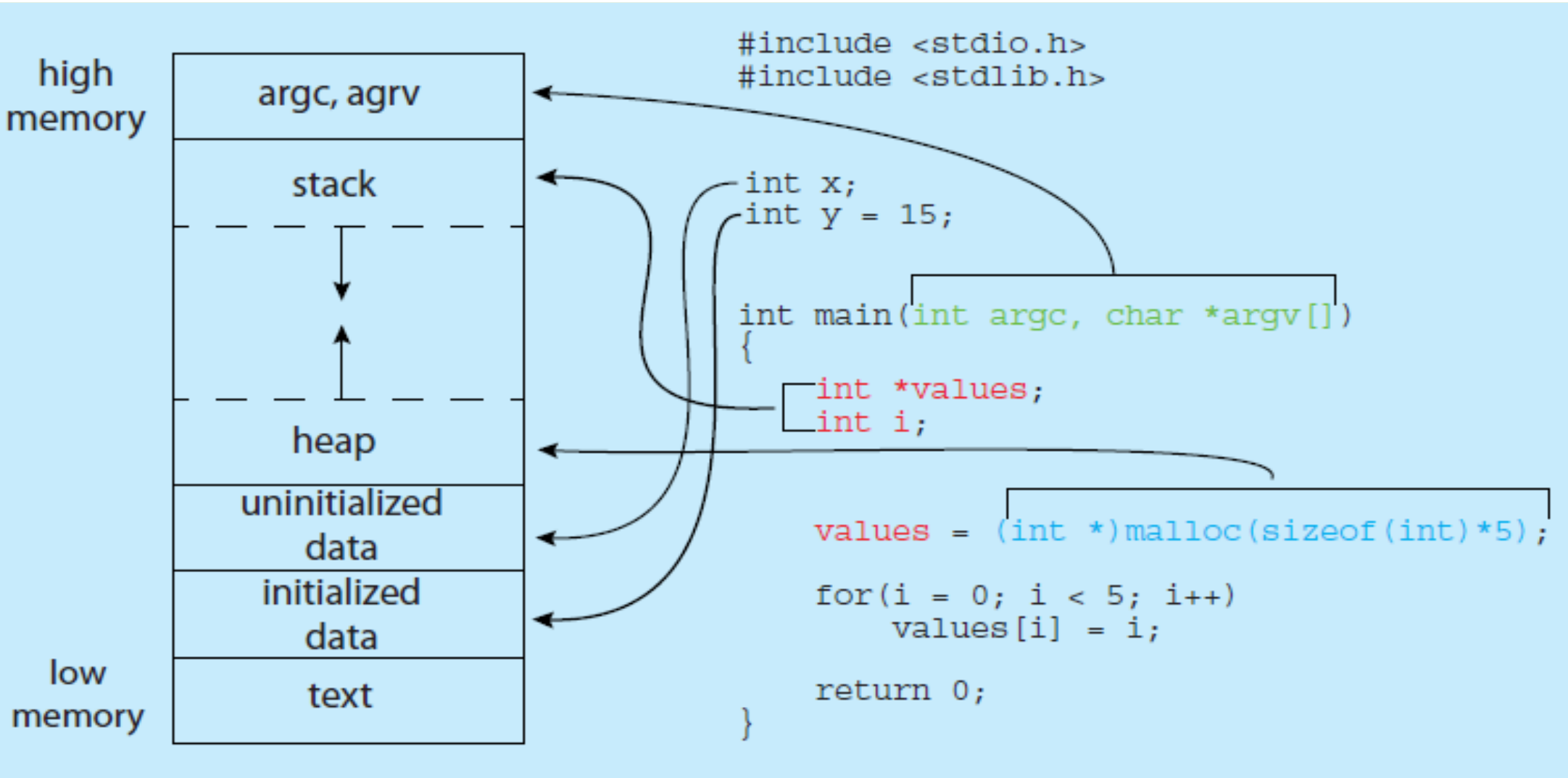
- An operating system executes a variety of programs:
  - Batch system – **jobs or Process**
  - Time-shared systems – **user programs** or **tasks**
- **Process** – a program in execution; process execution must progress in a sequential fashion
- Multiple parts
  - The program code, also called the **text section**
  - Current activity including **program counter**, processor registers
  - **Stack** containing temporary data
    - Function parameters, return addresses, local variables
  - **Data section** containing global variables
  - **Heap** containing memory dynamically allocated during run time



Process Concept

- Program is a ***passive*** entity stored on disk (**executable file**), process is ***active***
  - Program becomes process when executable file loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc
- One program can be several processes
  - Consider multiple users executing the same program

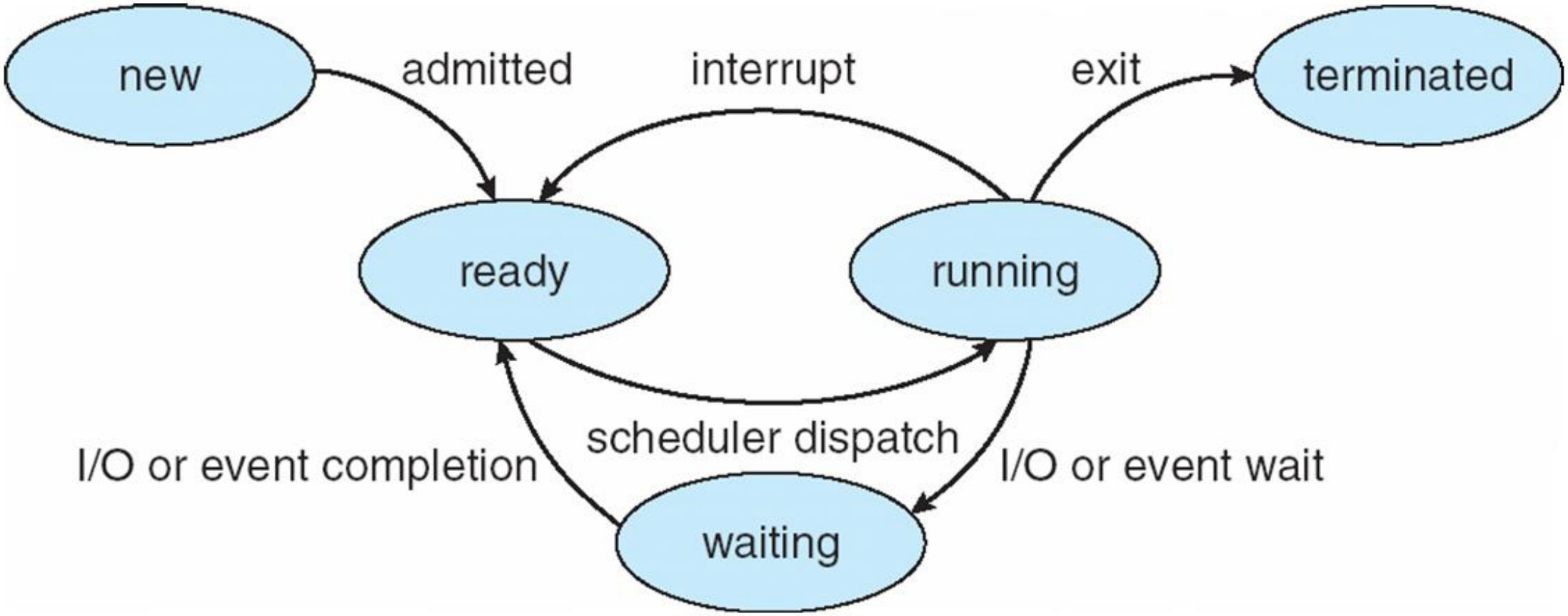
S.N.	Component & Description
1	**Stack** The process Stack contains the temporary data such as method/function parameters, return address and local variables.
2	**Heap** This is dynamically allocated memory to a process during its run time.
3	**Text** This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
4	**Data** This section contains the global and static variables.


high memory
argc, agrv
stack
heap
uninitialized data
initialized data
low memory
text
#include <stdio.h>
#include <stdlib.h>
int x;
int y = 15;
int main(int argc, char *argv[])
{
int *values;
int i;
values = (int *)malloc(sizeof(int)*5);
for(i = 0; i < 5; i++)
values[i] = i;
return 0;
}

Process State

- As a process executes, it changes the **state**
  - **new**: The process is being created
  - **running**: Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.
  - **waiting**: Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.
  - **ready**: The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process.
  - **terminated**: Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.

Diagram of Process State

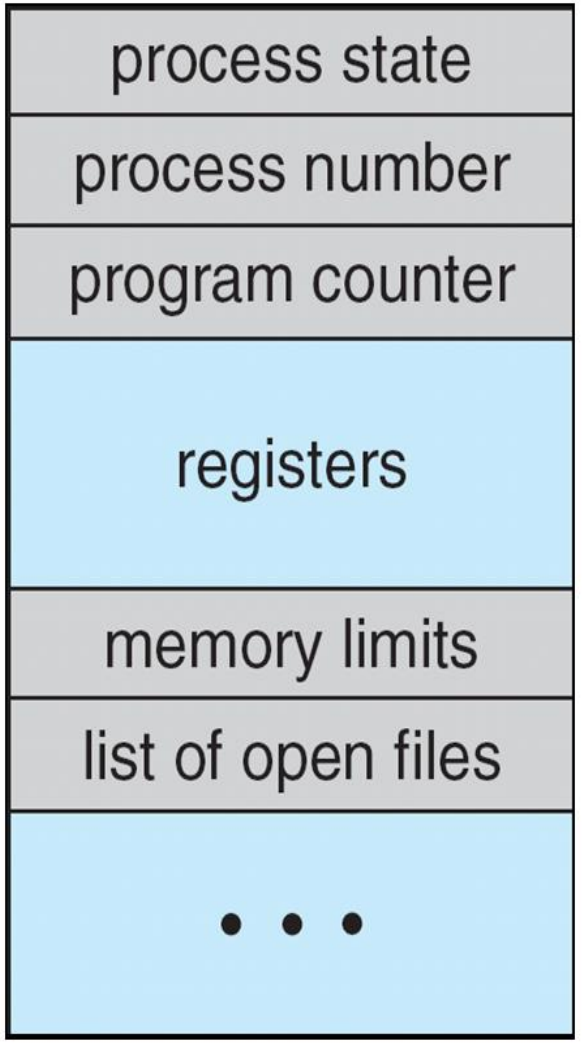


Process Control Block (PCB)

Information associated with each process

(also called **task control block**)

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- 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 Scheduling

- process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.
- Process scheduling is an essential part of Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

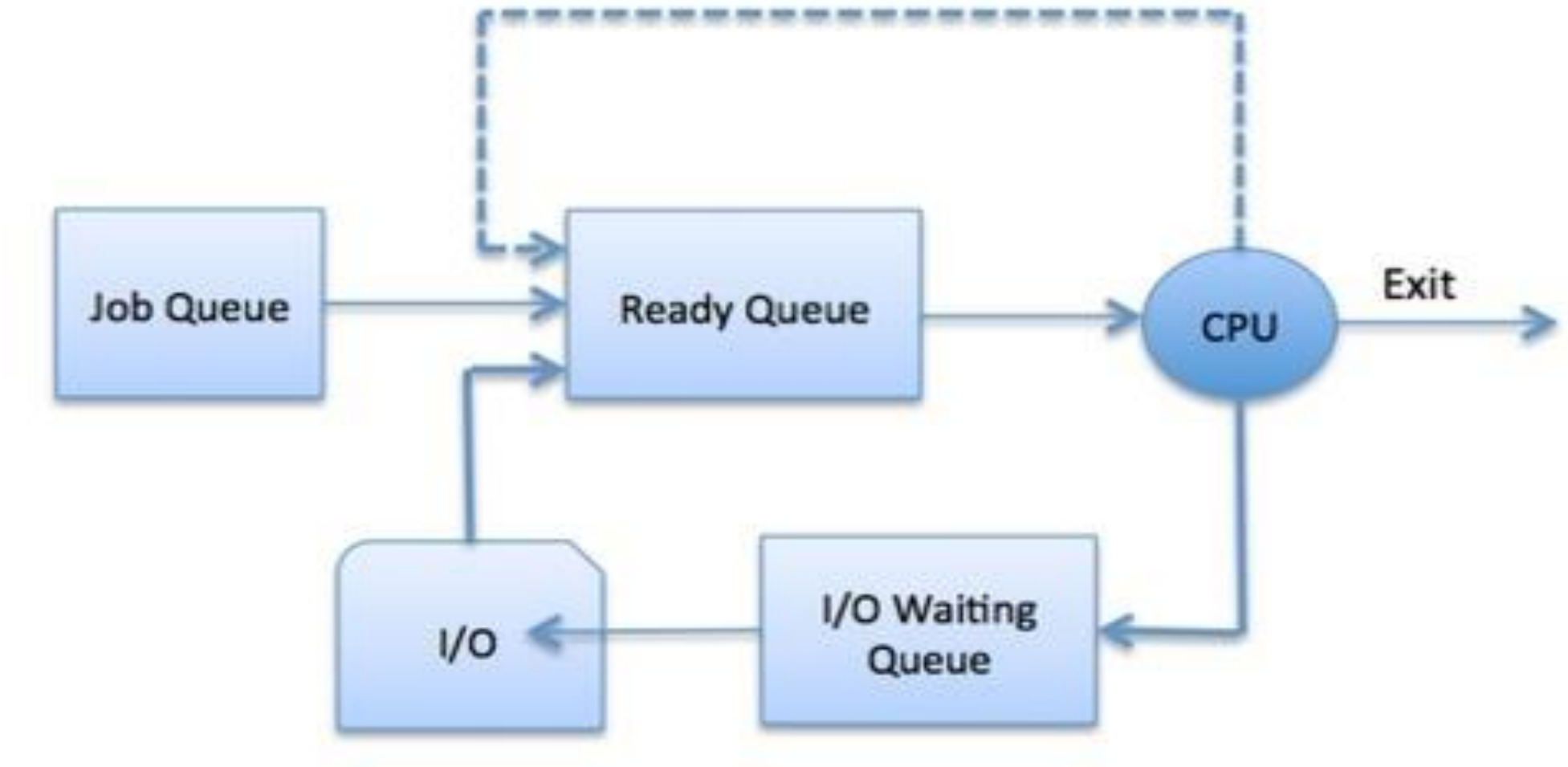
Categories of Scheduling

- There are two categories of scheduling:
- **Non-preemptive:** Here the resource can’t be taken from a process until the process completes execution. The switching of resources occurs when the running process terminates and moves to a waiting state.
- **Preemptive:** Here the OS allocates the resources to a process for a fixed amount of time. During resource allocation, the process switches from running state to ready state or from waiting state to ready state. This switching occurs as the CPU may give priority to other processes and replace the process with higher priority with the running process.

Process Scheduling Queues

- The OS maintains all Process Control Blocks (PCBs) in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.
- **Job queue** − This queue keeps all the processes in the system.
- **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** − The processes which are blocked due to the unavailability of an I/O device constitute this queue.

Process Scheduling Queues



Schedulers

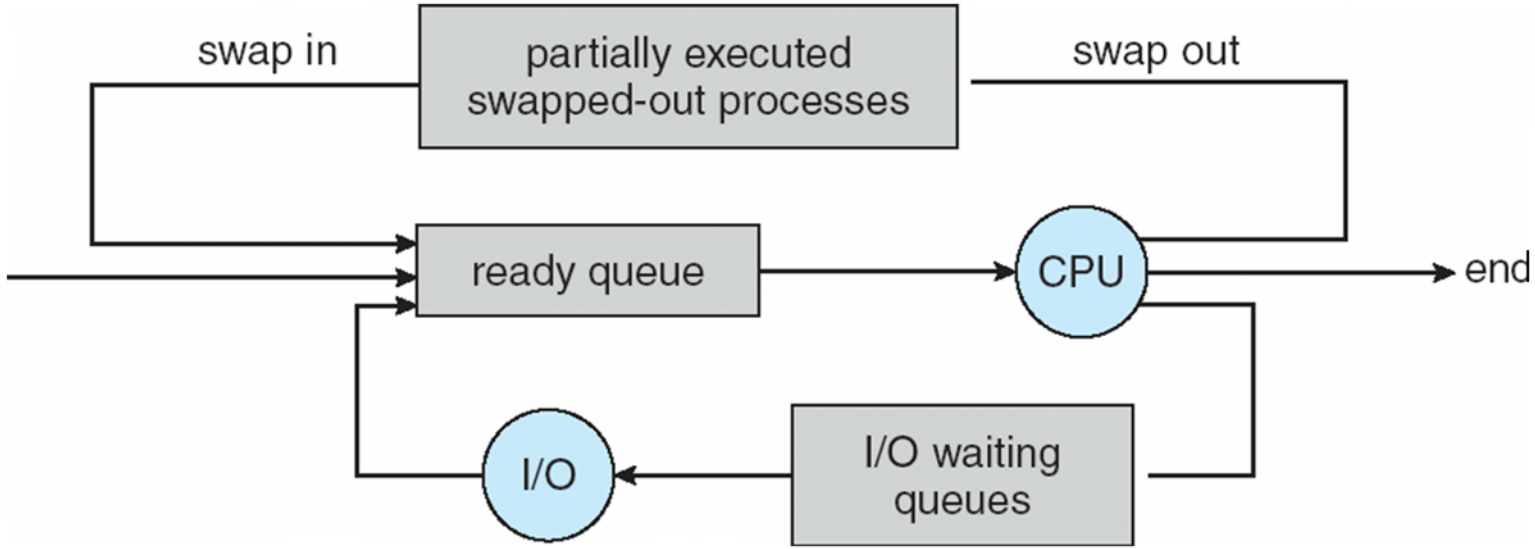
- Schedulers are special system software that handle process scheduling in various ways.
- Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −
- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Schedulers

- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds) ⇒ (must be fast)
- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked 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

Medium Term Scheduling

- Medium-term scheduler can be added if degree of multiple programming needs to decrease
- Remove process from memory, store on disk, bring back in from disk to continue execution: swapping



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

Operations on Process

1. Process Creation:

- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- Generally, process identified and managed via a process identifier (pid)

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

Operations on Process

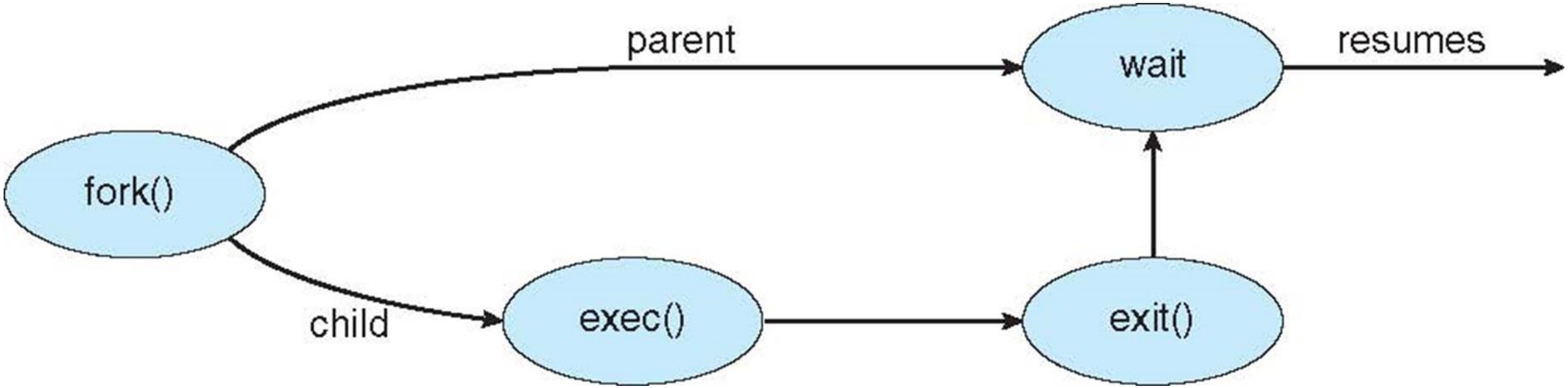
Process Creation(contd…):

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



Operations on Process

- **Process Termination**

- Process executes last statement and asks the operating system to delete it (**exit**)
  - Output data from child to parent (via **wait**)
  - Process' resources are deallocated by operating system

- Parent may terminate execution of children processes (**abort**)
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating systems do not allow child to continue if its parent terminates
      - All children terminated - **cascading termination**

Interprocess Communication

- A process is **independent** if it does not share data with any other processes executing in the system.
- A process is **cooperating** if it can affect or be affected by the other processes executing in the system.

There are several reasons for providing an environment that allows process cooperation:

Information sharing. To provide an environment to allow concurrent access to the same information.

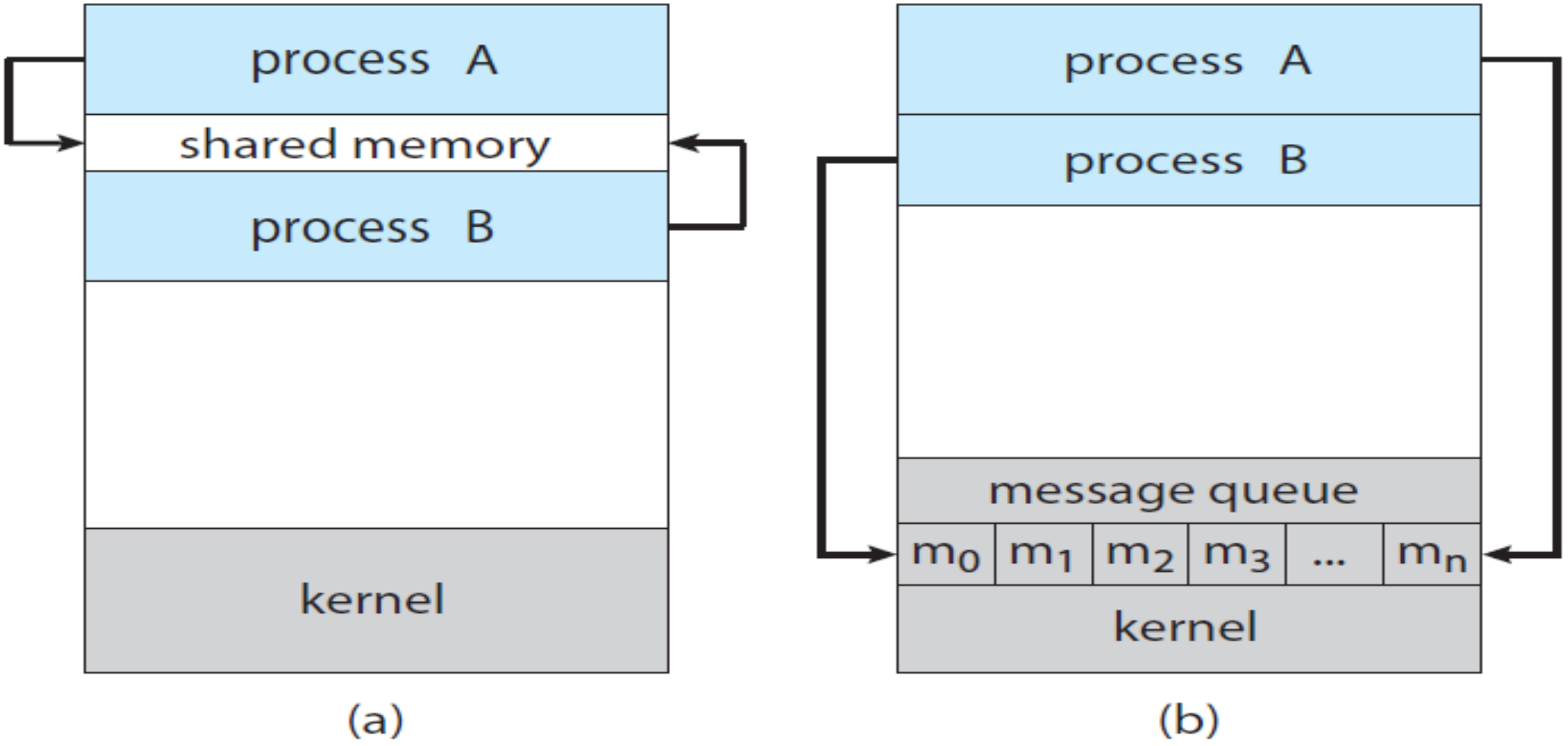
Computation speedup. To execute the task quickly, it must be broken into subtasks, each of which will be executed in parallel with the others. Notice that such a speedup can be achieved only if the computer has multiple processing cores.

Modularity. To construct the system in a modular fashion, dividing the system functions into separate processes or threads.

IPC through Shared Memory

- **Cooperating processes** require **interprocess communication (IPC)** mechanism that will allow them to exchange data— that is, send data to and receive data from each other.
- There are two fundamental models of interprocess communication: **shared memory and message passing**.
- **Shared-memory model:** A region of memory that is shared by the cooperating processes is established. Processes can then exchange information by reading and writing data to the shared region.
- **Message-passing model:** Communication takes place by means of messages exchanged between the cooperating processes.

Communications models. (a) Shared memory. (b) Message passing.



Producer–Consumer problem using Shared Memory

- To illustrate the concept of cooperating processes, let's consider the producer–consumer problem, which is a common paradigm for cooperating processes.
- A producer process produces information that is consumed by a consumer process.
- Buffer is shared memory paradigm
  - **unbounded-buffer** places no practical limit on the size of the buffer
  - **bounded-buffer** assumes that there is a fixed buffer size

```
Shared data
#define BUFFER_SIZE 10
typedef struct {
        . . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

Solution is correct, but can only use BUFFER_SIZE-1 elements

Bounded-Buffer – Producer and Consumer

```
item next_produced;
while (true) {
/* produce an item in next produced */
        while (((in + 1) % BUFFER_SIZE) == out)
                ; /* do nothing */
        buffer[in] = next_produced;
        in = (in + 1) % BUFFER_SIZE;
}
```

```
item next_consumed;
while (true) {
        while (in == out)
                ; /* do nothing */
        next_consumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;

        /* consume the item in next consumed */
}
```

Shared Memory

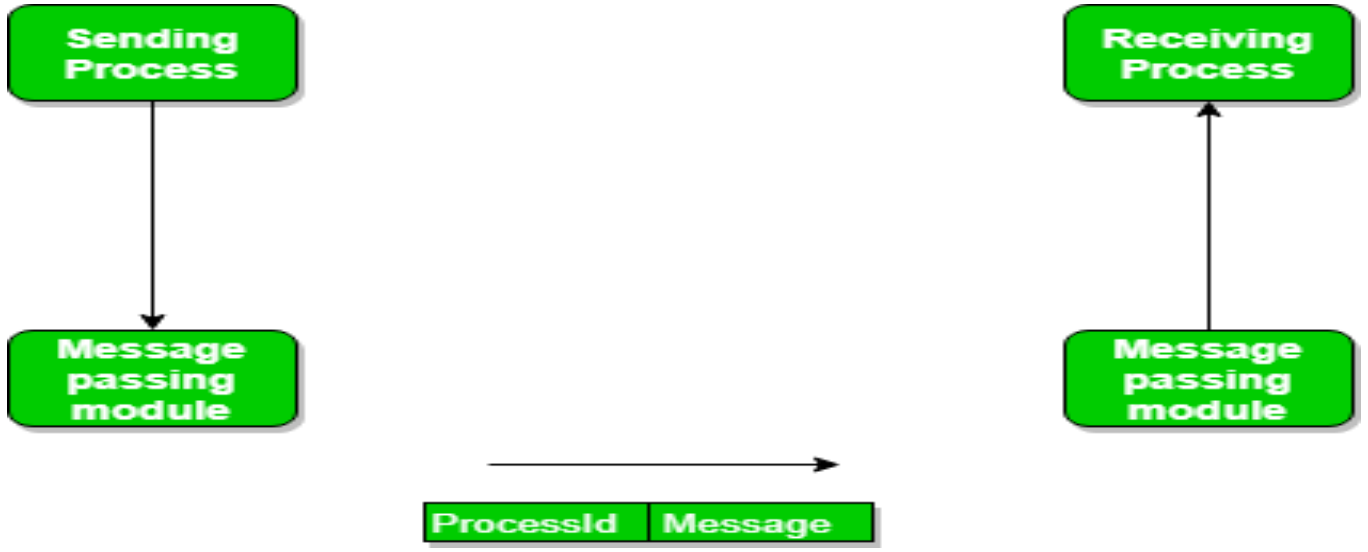
- An area of memory shared among the processes that wish to communicate
- The communication is under the control of the users processes not the operating system.
- Major issues is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory.

Interprocess Communication – Message Passing

- 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)
- receive(message)
- The message size is either fixed or variable

Message Passing Module

- A standard message can have two parts: header and body.
- The header part is used for storing message type, destination id, source id, message length, and control information.
- The control information contains information like what to do if runs out of buffer space, sequence number, priority.
- Generally, message is sent using FIFO style.



Message Passing

If processes P and Q wish to communicate, they need to:

- Establish a communication link between them
- Exchange messages via send/receive

Implementation issues:

- 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 the 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

Indirect Communication

Operations

- create a new mailbox (port)
- send and receive messages through mailbox
- destroy a mailbox

Primitives are defined as:

- send(A, message) – send a message to mailbox A
- receive(A, message) – receive a message from mailbox A

Indirect Communication

Mailbox sharing

- P1, P2, and P3 share mailbox A
- P1, sends; P2 and P3 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.

Synchronization

Message passing may be either blocking or non-blocking

Blocking is considered synchronous

- Blocking send -- the sender is blocked until the message is received
- Blocking receive -- the receiver is blocked until a message is available

Non-blocking is considered asynchronous

- Non-blocking send -- the sender sends the message and continue
- Non-blocking receive -- the receiver receives:
  - A valid message, or
  - Null message

Different combinations possible

- If both send and receive are blocking, we have a rendezvous