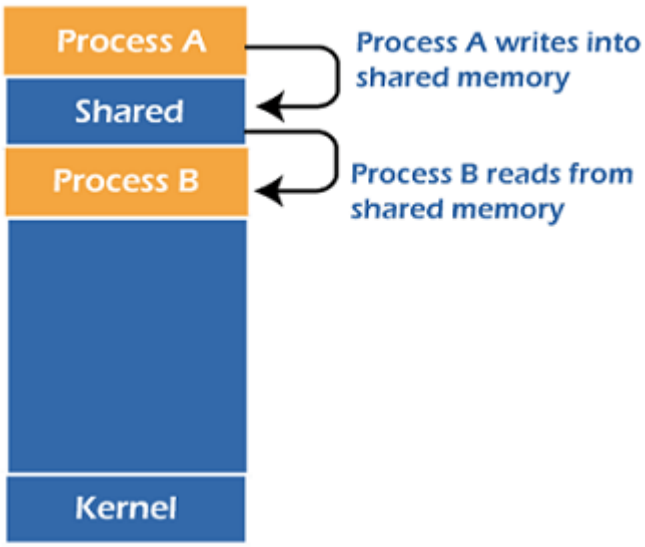
Operating System

Week 5 - Long Descriptive Questions

1. How communication happens in shared memory? 2. Discuss the system call used for reading and writing a pipe.

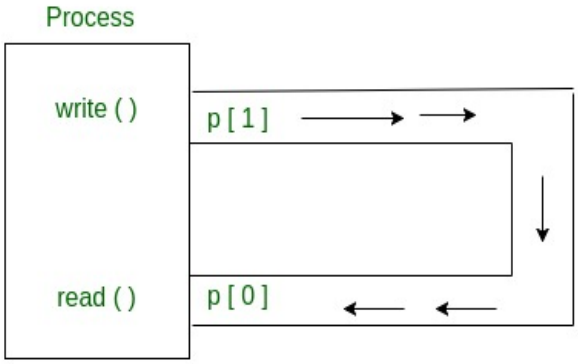
Ans. Shared memory is a memory shared between two or more processes. Each process has its own address space; if any process wants to communicate with some information from its own address space to other processes, then it is only possible with IPC (inter-process communication) techniques.

Shared memory is the fastest inter-process communication mechanism. The operating system maps a memory segment in the address space of several processes to read and write in that memory segment without calling operating system functions.



pipe() System call : Conceptually, a pipe is a connection between two processes, such that the standard output from one process becomes the standard input of the other process. In UNIX Operating System, Pipes are useful for communication between related processes(inter-process communication).

- Pipe is one-way communication only i.e we can use a pipe such that One process write to the pipe, and the other process reads from the pipe. It opens a pipe, which is an area of main memory that is treated as a “virtual file”.
- The pipe can be used by the creating process, as well as all its child processes, for reading and writing. One process can write to this “virtual file” or pipe and another related process can read from it.
- If a process tries to read before something is written to the pipe, the process is suspended until something is written.
- The pipe system call finds the first two available positions in the process’s open file table and allocates them for the read and write ends of the pipe.



2. How a cooperating process is varied from an independent process.

Ans.

Independent process: one that is independent of the rest of the universe.

- Its state is not shared in any way by any other process.
- Deterministic: input state alone determines results.
- Reproducible.
- Can stop and restart with no bad effects (only time varies).

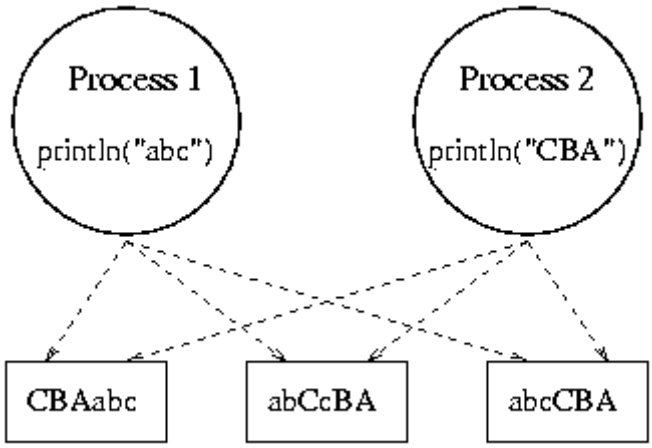
Example: program that sums the integers from 1 to i (input).
There are many different ways in which a collection of independent processes might be executed on a processor:

- Uniprogramming: a single process is run to completion before anything else can be run on the processor.
- Multiprogramming: share one processor among several processes. If no shared state, then order of dispatching is irrelevant.
- Multiprocessing: if multiprogramming works, then it should also be ok to run processes in parallel on separate processors.
    - A given process runs on only one processor at a time.
    - A process may run on different processors at different times (move state, assume processors are identical).
    - Cannot distinguish multiprocessing from multiprogramming on a very fine grain.

How often are processes completely independent of the rest of the universe?

Cooperating processes:

- Machine must model the social structures of the people that use it. People cooperate, so machine must support that cooperation. Cooperation means shared state, e.g. a single file system.
- Cooperating processes are those that share state. (May or may not actually be "cooperating")
- Behavior is nondeterministic: depends on relative execution sequence and cannot be predicted a priori.
- Behavior is irreproducible.
- Example: one process writes "ABC", another writes "CBA".



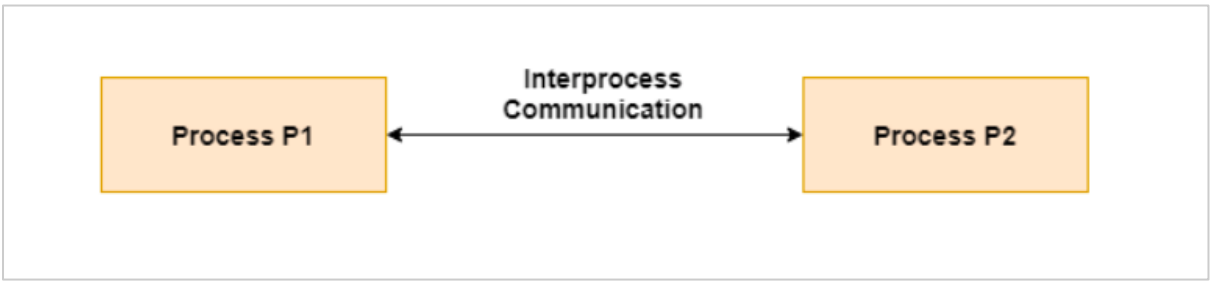
When discussing concurrent processes, multiprogramming is as dangerous as multiprocessing unless you have tight control over the multiprogramming. Also bear in mind that smart I/O devices are as bad as cooperating processes (they share the memory).
Why permit processes to cooperate?

- Want to share resources:
  - One computer, many users.
  - One database of checking account records, many tellers.
- Want to do things faster:
  - Read next block while processing current one.
  - Divide job into sub-jobs, execute in parallel.
- Want to construct systems in modular fashion. (e.g. tbl | eqn | troff)

3. Explain the IPC Model – Pipes with an example.

Ans.
Interprocess communication is the mechanism provided by the operating system that allows processes to communicate with each other. This communication could involve a process letting another process know that some event has occurred or the transferring of data from one process to another.

A diagram that illustrates interprocess communication is as follows −



Synchronization in Interprocess Communication

Synchronization is a necessary part of interprocess communication. It is either provided by the interprocess control mechanism or handled by the communicating processes. Some of the methods to provide synchronization are as follows −

Semaphore: A semaphore is a variable that controls the access to a common resource by multiple processes. The two types of semaphores are binary semaphores and counting semaphores.
Mutual Exclusion: Mutual exclusion requires that only one process thread can enter the critical section at a time. This is useful for synchronization and also prevents race conditions.
Barrier : A barrier does not allow individual processes to proceed until all the processes reach it. Many parallel languages and collective routines impose barriers.
Spinlock: This is a type of lock. The processes trying to acquire this lock wait in a loop while checking if the lock is available or not. This is known as busy waiting because the process is not doing any useful operation even though it is active.

Example program - Program to write and read two messages using pipe.

```
#include<stdio.h>
#include<unistd.h>

int main() {
  int pipefds[2];
  int returnstatus;
  char writemessages[2][20]={"Hi", "Hello"};
  char readmessage[20];
  returnstatus = pipe(pipefds);

  if (returnstatus == -1) {
    printf("Unable to create pipe\n");
    return 1;
  }
   printf("Writing to pipe - Message 1 is %s\n", writemessages[0]);
  write(pipefds[1], writemessages[0], sizeof(writemessages[0]));
  read(pipefds[0], readmessage, sizeof(readmessage));
  printf("Reading from pipe – Message 1 is %s\n", readmessage);
  printf("Writing to pipe - Message 2 is %s\n", writemessages[0]);
  write(pipefds[1], writemessages[1], sizeof(writemessages[0]));
  read(pipefds[0], readmessage, sizeof(readmessage));
  printf("Reading from pipe – Message 2 is %s\n", readmessage);
  return 0;
}
```

4. Elaborately discuss the various IPC models for Message passing.

ANS.

It is a Mechanism for processes to communicate and to synchronize their actions. In 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).

- If processes P and Q wish to communicate, they need to:
  Establish a communication link between them
  Exchange messages via send/receive

- Implementation of communication link
  Physical:
    Shared memory
    Hardware bus
    Network
  Logical:
    (i) Direct or indirect
    (ii) Synchronous or asynchronous
    (iii) Automatic or explicit buffering

(i) Direct Communication /Indirect communication:

(a) Direct Communication links are implemented when the processes use a specific process identifier for the communication, but it is hard to identify the sender ahead of time.

- 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.

(b) Indirect Communication

Messages are directed and received from mailboxes (also referred to as ports)
- Each mailbox has a unique id
- Processes can communicate only if they share a mailbox
- Properties of communication link
- Link established only if processes share a common mailbox
- A link may be associated with many processes
- Each pair of processes may share several communication links

● Link may be unidirectional or bi-directional

Operations

- ● create a new mailbox (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

(ii) 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

(iii) Blocking:

Queue of messages attached to the link.

● Implemented in one of three ways

1. Zero capacity – no messages are queued on a link. Sender must wait for receiver (rendezvous)
2. Bounded capacity – finite length of n messages Sender must wait if link full
3. Unbounded capacity – infinite length Sender never waits