buffer is a place in which the producer can produce the information and it can be filled in that place. The consumer will consume from the buffer and empty the buffer. So the buffer is the shared memory. This buffer will reside in a region of memory that is shared by the producer and consumer processes.

The producer can produce one item while the consumer consumes another item. The producer and consumer must be synchronized, so that the consumer does not try to consume the item that has not yet been produced.

There are two types of buffers:

Unbounded buffer and Bounded buffer

The unbounded buffer places no practical limit on the size of the buffer. The consumer may have to wait for new items, but the producer can always produce new items.

The bounded buffer assumes a fixed buffer size. The consumer must wait if the buffer is empty and the producer must wait if the buffer is full.

Message Passing System

https://www.youtube.com/watch?v=LuuSXWkDJOo

https://www.youtube.com/watch?v=S3mS8MR7bUY

https://www.youtube.com/watch?v=fViAyFLOYxU

Operations on Processes

The processes in most systems can execute concurrently, and they may be created and deleted dynamically. Thus these systems must provide a mechanism for process creation and termination.

Mainly we can perform two operations on a process

- Process Creation
- Process Termination

The process may create several new processes via a create process system call, during the course of execution. Creating process is called a **parent** process and the new processes are called the children of that process. Each of these new processes may in turn create other processes forming a tree of processes.

When a process create a new process, there are some possibilities exist in terms of

- Resource Sharing
- Execution
- Address space of child process

When a parent process creates a child process, a parent process can share all of its resources with children. OR Parent process share some of its resources with children. OR Parent process never shares the resources with child process. Thus the child process may use its own resources.

In terms of execution

ure 1.19 Peer-to-peer system

other example of peer-to-peer con
deo calls and to send text message
oice over IP (VoIP). Skype uses a hy
l login server, but it also incorpora
municate.

is a technology that allows ope
perating systems. At first blush, t
lut the virtualization industry is
l importance.

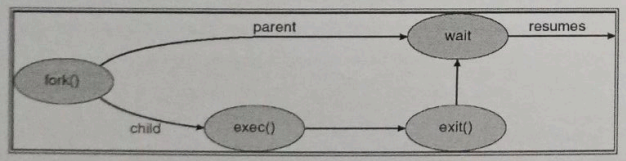
- The parent continues to execute concurrently with the children
- The parent waits until some or all of its children have terminated

Once the children have terminated, the parent process resume its execution.

In terms of **address space**, the possibilities are

- When a parent process create a child process, the child process gets the exact duplicate copy of the parent process
- The child process has a new program loaded into it which is different from parent.

Consider the UNIX operating system. In UNIX, each process is identified by its process identifier which is a unique integer. A new process is created by the fork() system call. The new process consists of a copy of the address space of the original process. This allows the parent process to communicate easily with its child process. **Exec( )** system call is used after a **fork( )** to replace the process' memory space with a new program. The exec( ) system call loads the binary file into memory and starts its execution. Thus the two processes are able to communicate and then go their separate ways. If the parent has nothing to do when the child runs, it can issue a wait( ) system call to move itself off the ready queue until the termination of the child. When the child process completes by invoking exit() system call, the parent process resumes its execution. This is illustrated in the following figure.



Process creation

space of the process creating the shared memory segment. Other processes that wish to communicate using this shared memory segment, must attach it to their address space. Normally the OS tries to prevent one process from accessing another processes' memory. Shared memory requires that two or more processes agree to remove this restriction. They can then exchange information by reading and writing data in the shared areas. The processes are responsible for ensuring that they are not writing to the same location simultaneously.

To illustrate the concept of cooperating processes, consider the producer consumer problem, which is a common paradigm for cooperating processes.

Producer- Consumer Problem(PC)

A producer process produces information that is consumed by a consumer process. For example, a compiler may produce assembly code, which is consumed by an assembler. The assembler in turn may produce object modules, which are consumed by the loader. In client-server paradigm, the server is a producer and client is a consumer. A webserver produces(provides) HTML files and images, which are consumed(that is,read) by the client web browser requesting the resource.

What is the actual problem in PC problem? We need to make the producer and consumer concurrently. They have to work concurrently so that the consumer consumes only what is produced.

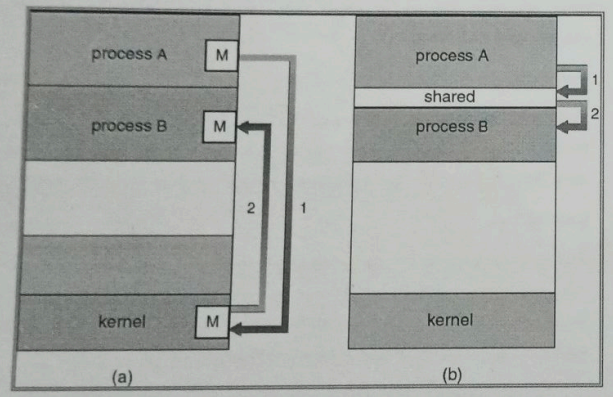
One solution to PC problem is shared memory. To allow producer and consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer and emptied by the consumer. A

In the shared memory
cooperating processes is estab
reading and writing data to the
memory that will be shared by
model, communication take
the cooperating process
IPC Models- Mess

– Message passing

In the shared memory model, a region of memory that is shared by cooperating processes is established. Processes can exchange information by reading and writing data to the shared region. We establish a region of memory that will be shared by the cooperating processes. In message passing model, communication take place by means of messages exchanged between the cooperating processes.

IPC Models- Message passing & Shared Memory



Shared Memory System

IPC using shared memory requires communicating processes to establish a region of shared memory. Typically a shared memory resides in the address

The following c program illustrates the UNIX system call

```
int main()
{
int  pid;
   /* fork a child process */
   pid = fork();
   if (pid < 0) { /* error occurred */
    fprintf(stderr, "Fork Failed");
    exit(-1);
   }
   else if (pid == 0) { /* child process */
    execlp("/bin/ls", "ls", NULL);
   }
   else { /* parent process */
    /* parent will wait for the child to complete */
    wait (NULL);
    printf ("Child Complete");
    exit(0);
   }
}
```

Process Termination

Process termination means killing a process. When all the operation on a process are over, that process should be terminated so that space can be utilized by other processes. A process terminates when it finishes executing its final statement and ask the OS to delete it by using the exit( ) system call. At that point, the process may return a status value to its parent process via the wait( ) system call. All the resources of the process are deallocated by the OS.

A process can cause the termination of another via an appropriate system call. Usually such a system call can be invoked only by the parent of the process

that is to be terminated. Otherwise, users could kill each other's job. A parent may terminate the execution of one of its children for a variety of reasons.

- The child has exceeded its usage of some of its resources that has been allocated.
- The task assigned to the child no longer required
- The parent is exiting, and the OS does not allow a child to continue if its parent terminates.

Interprocess Communication

Processes executing concurrently in the OS may be either **independent** processes or **cooperating** processes. Independent processes – They cannot affect or be affected by the other processes executing in the system. Any process that does not share data with any other process is Independent. Cooperating processes - They can affect or be affected by the other processes executing in the system. Any process that shares data with other process is a cooperating process. Interprocess communication will be required in cooperating process.

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

- Information sharing
- Computation speedup
- Modularity
- Convenience

Information Sharing

When several users are interested in same piece of information, we will have to provide an environment in which the information can be accessed concurrently.

Computational speed up

In order to achieve computational speed up, if we are having a task, we divide that task into several subtasks and all this subtasks will be made to run concurrently. In that way we speed up our system. When the task are divided into several different task, they are assigned as different processes. Since all these processes belongs to single task, those processes will need to communicate each other.

Modularity

Modularity means we want to design the system by dividing into different modules. These modules will later be put together to work to achieve a single goal. These modules will need to communicate for cooperate with each other

Convenience

If we allow the processes to cooperate with each other, then that becomes very convenient for the user. For eg, a user may be using a system and he may be doing several task at the same time. The user may be listening to music, he may be editing a text, he may be printing some document etc.

Cooperating process require an inter process communication (IPC) mechanism that will allow them to exchange data and information. There are two fundamental process of IPC

- Shared Memory