**IPC**

Inter-Process-Communication (or IPC for short) are mechanisms provided by the kernel to allow processes to communicate with each other.

**Types of Processes**

* Independent Process
* Cooperating Process

**Independent Process**

A process that can not affect or be affected by other process executing in the system.

**Cooperating Process**

A process that can affect or be affected by other process executing in the system.

**Why use IPC**

* Information Sharing
* Computation speedup
* Modularity
* Convenience

**Models of IPC**

Basically, there are two models of IPC:

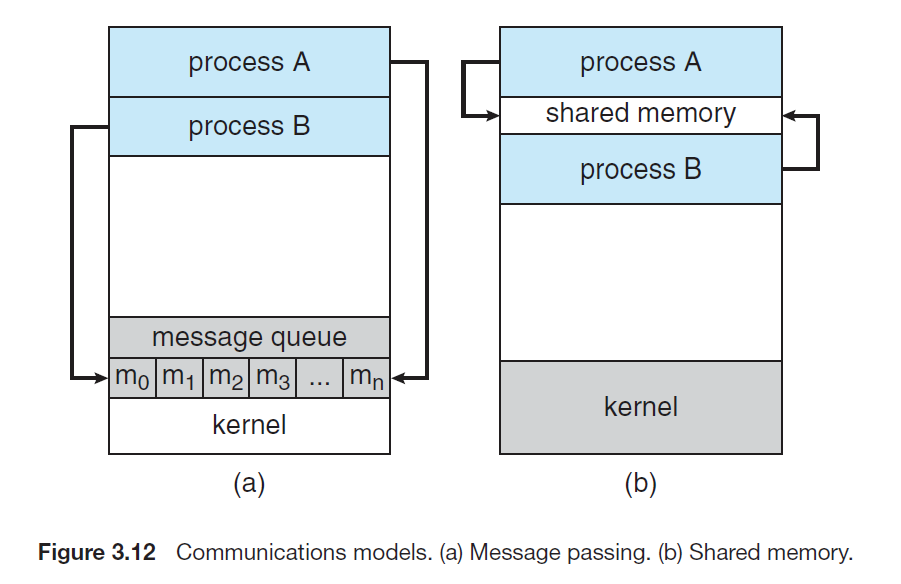
* Shared Memory
* Message Passing

**Shared Memory**

In the shared-memory model, a region of memory that is shared by cooperating processes is established. Processes can then exchange information by reading and writing data to the shared region.

**Message Passing**

In the message-passing model, communication takes place by means of messages exchanged between the cooperating processes.



Shared memory can be faster than message passing, since message-passing systems are typically implemented using system calls and thus require the more time-consuming task of kernel intervention. In shared-memory systems, system calls are required only to establish shared memory regions. Once shared memory is established, all accesses are treated as routine memory accesses, and no assistance from the kernel is required.

Recent research on systems with several processing cores indicates that message passing provides better performance than shared memory on such systems. Shared memory suffers from cache coherency issues, which arise because shared data migrate among the several caches. As the number of processing cores on systems increases, it is possible that we will see message passing as the preferred mechanism for IPC.

**Signal Delivery using Kill**

Kill is the delivery mechanism for sending a signal to a process. Unlike the name, a kill () call is used only to send a signal to a process. It does not necessarily mean that a process is going to be killed (although it can do exactly that as well).

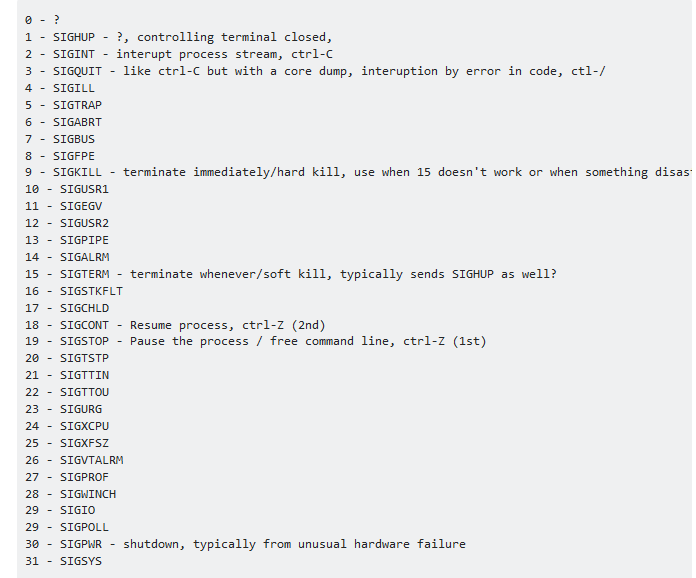
To pass a signal using Kill in a program, we write:

Kill(ProcessID, Signal)

Here ProcessID is the ID of the executing process and Signal is the psossible signal you want top pass to a process. If we want to terminate a process in execution, we write:

Kill(getpid(), SIGTERM)

Other possible signals used in kill are as follows:



**Program**

#include<signal.h>

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<signal.h>

#include<iostream>

using namespace std;

int sigCount = 0;

int main(){

while(1){

cout << "Hello Dears\n";

cout << "Hello Dears " << getpid() << endl;

sleep(1);

sigCount++;

if(sigCount==5)

kill(getpid(),SIGTERM);

}

return 0;

}

**Signal Handling using signal**

Signal delivery is handled by the kill command or the kill () call. The process receiving the signal can behave in a number of ways, which is defined by the signal () call. The syntax of the call is as such:

Signal (int, conditions);

The signal will require the signal.h C library to work. From the syntax above, signal () is the system call, the 1st parameter int is the integer identifier of the respective signal which is sent, and the last parameter is either of the following:

1. SIG\_DFL which will perform the default mechanism provided by the operating system for that particular signal
2. SIG\_IGN which will ignore that particular signal if it is delivered
3. Any function name (for programmer-defined signal handling purposes)

**Program**

#include<signal.h>

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<signal.h>

#include<iostream>

using namespace std;

int sigCount = 0;

void sigHandler(int sigNum){

cout << "Signal received is " << sigNum << endl;

sigCount++;

cout << "Total Signals are " << sigCount << endl;

}

int main(){

signal(SIGINT, SIG\_DFL);

while(1){

cout << "Hello Dears\n";

cout << "Hello Dears " << getpid() << endl;

sleep(1);

}

return 0;

}

When we run the program, we are instructing our program that if in case the SIGINT signal is detected, we will perform the steps provided in the sigHandler function. As long as there is no event, the program will keep on executing the while loop. The event can be delivered by pressing CTRL+C. Try pressing CTRL+C with, and without the signal () call and you will understand the difference yourselves.

**Shared Memory**

[Inter Process Communication](https://www.geeksforgeeks.org/inter-process-communication/) through shared memory is a concept where two or more process can access the common memory. And communication is done via this shared memory where changes made by one process can be viewed by anther process.

A total of four copies of data are required (2 read and 2 write). So, shared memory provides a way by letting two or more processes share a memory segment. With Shared Memory the data is only copied twice – from input file into shared memory and from shared memory to the output file.

**System Calls used in Shared Memory**

**ftok()**: is use to generate a unique key. Its takes two arguments, one for path name and other for id.

**shmget()**: int shmget(key\_t, size\_tsize, intshmflg); upon successful completion, shmget() returns an ID for the shared memory segment.

**shmat()**: Before you can use a shared memory segment, you have to attach yourself  
to it using shmat(). void \*shmat(int shmid, void \*shmaddr, int shmflg);  
shmid is shared memory id. shmaddr specifies specific address to use but we should set  
it to zero and OS will automatically choose the address.

**shmdt()**: When you’re done with the shared memory segment, your program should  
detach itself from it using shmdt(). int shmdt(void \*shmaddr);

**shmctl()**: when you detach from shared memory, it is not destroyed. So, to destroy  
shmctl() is used. shmctl(int shmid, IPC\_RMID, NULL);

**Program to write data in Shared Memory**

#include <iostream>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

using namespace std;

int main()

{

    // ftok to generate unique key

    key\_t key = ftok("shmfile",65);

    // shmget returns an identifier in shmid

    int shmid = shmget(key,1024,0666|IPC\_CREAT);

    // shmat to attach to shared memory

    char \*str = (char\*) shmat(shmid,(void\*)0,0);

    cout<<"Write Data : ";

    cin >> str;

    printf("Data written in memory: %s\n",str);

    //detach from shared memory

    shmdt(str);

    return 0;

}

**Program to Read Data from Shared Memory**

#include <iostream>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

using namespace std;

int main()

{

    // ftok to generate unique key

    key\_t key = ftok("shmfile",65);

    // shmget returns an identifier in shmid

    int shmid = shmget(key,1024,0666|IPC\_CREAT);

    // shmat to attach to shared memory

    char \*str = (char\*) shmat(shmid,(void\*)0,0);

    printf("Data read from memory: %s\n",str);

    //detach from shared memory

    shmdt(str);

    // destroy the shared memory

    shmctl(shmid,IPC\_RMID,NULL);

    return 0;

}