COMMUNICATING ACROSS PROCESSES

# OBJECTIVE

Study the features of Communicating Across Processes.

# THEORY

Communication takes place if there is something to share. Ideas, thoughts, feelings, beliefs, culture and just about every other facet that goes to make a human being. Through computers so far we have shared data – data linked to business, science and now as satellite communications make their mark news, views and what have you.

Processes in computer memory are said to be communicating when one process passes data to another or vice-versa. And for these processes in memory to communicate with each other they need to be able to share data effectively. UNIX provides a number of ways to share or communicate data: signals and pipes.

**Signals**

Processes can send each other signals. For example say process A has finished processing some data, it may send signal to process B. Process B will also receive the output of the processed data from process A. Based on this output it has to decide what response to give.

There are many different types of signals, each with some mnemonic name: SIGINT, for example. These names, which are defined in the signal.h header file are symbolic for some positive integer.

Suppose we run a process that takes inordinately long, thus leading us to believe that something is wrong. To terminate it, we either press the DEL or CTRL \ keys. This result in brought back to the shell.

What happens is that the part of the kernel looking after keyboard interrupts realizes that the DEL has been pressed. The kernel then sends a signal called SIGINT to all processes associated with the terminal. And they terminate. For reference consider Program 1.

Compile and run Program 1. Then press the DEL key. The program terminates because the pressing of the DEL key forces the kernel to send a SIGINT signal which is interpreted as “terminate a program”.

**About ‘SIGALRM’**

The alarm( ) function sets up a process alarm clock. Signals are used to tell the process that the clock’s timer has expired. After the call, the process carries on. Whenever we want to place a time limit on some activity we can use the alarm system call. If the process takes longer than the alarm set, it is interrupted by the SIGALRM signal and the process terminates. For reference consider Program 2.

If the above Program 1 is run in the foreground, 5seconds later a message Alarm Call is flashed on screen.

The sleep( ) function which we use so often is written using alarm( ) and one more function pause( ). The pause( ) basically waits for a signal to be sent before it terminates. And the alarm( ) waits for the time period specified to elapse. The moment the time period elapses, a signal, the SIGALRM, is sent. This terminates the pause( ). For reference Program 3.

# Example programs

## Program 1

#include<signal.h>

main( )

{

printf(“Use Ctrl-Z key for exiting\n”);

for(;;);

}

## Program 2

main( )

{

alarm(5);

for(;;);

}

## Program 3

#include<signal.h>

main( )

{

void abc( );

signal(SIGALRM,abc);

alarm(3\*6);

pause( );

}

void abc( )

{

printf(“Time to ring the Sec.Gen. my boy \n”);

}

# Exercise

1. What did you learn after running Program 3? Write few lines.