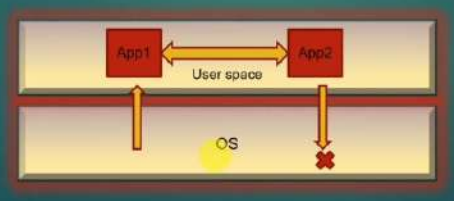
IPC – Techniques -> Signals

Definition :

A system message sent from one process to another, not usually used to transfer data but instead used to remotely command the partnered process.

Eg :

The Cricket Umpire *signaling* the batsman is out. It is an example of Signal, not a data transfer
Signals are usually small msg (1 or 2 byte). Recipient needs to interpret its meaning.



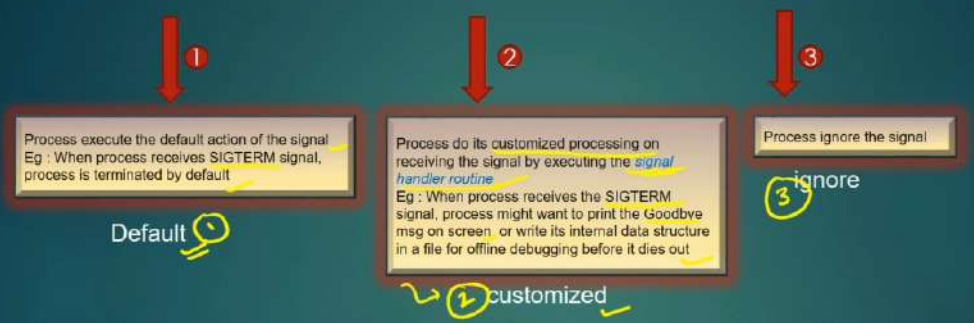
Signals Works from :

OS -> Application
Between Applications

From Application to OS - No

IPC – Techniques -> Signals -> Signal Catching and Signal handler routine

When a process receives a Signal, Either of the three things can happen :



Few points on Signal handler routines :

- A signal handler routine is a special function which is invoked when the process receives a signal.
- We need to register the routine against the type of signal (that is, process needs to invoke the function F when signal S is received)
- Signal handler routine are executed at the highest priority, preempting the normal execution flow of the process

IPC – Techniques -> Sending Signals

Three ways of generating Signals in Linux

1. Raising a signal from OS to a process (ctrl – C etc) SIGINT
2. Sending a signal from process A to itself (using *raise()*)
3. Sending signal from process A to process B (using *kill()*)

IPC – Techniques -> Signal Programming -> Signal Trapping

1. A Process can Trap the signal received from OS and perform user defined function when signal is Received

```
#include <signal.h>

/*Register User defined function ctrlC_signal_handler
which will be invoked When ctrl + C is pressed*/

signal (SIGINT, ctrlC_signal_handler);
```

Registration

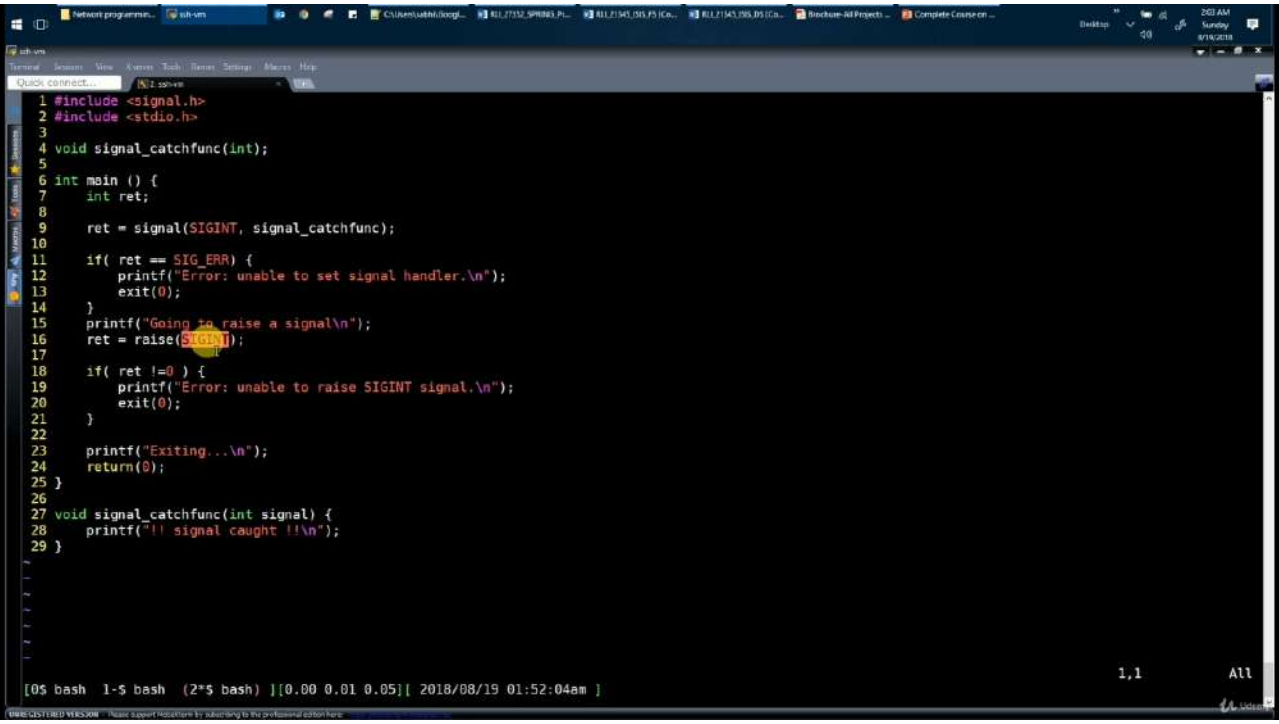
```
static void
ctrlC_signal_handler(int sig){
    printf("Ctrl-C pressed\n");
    printf("Bye Bye\n");
    exit(0);
}
```

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

static void
ctrlc_signal_handler(int sig){
    printf("Ctrl-C pressed\n");
    printf("Bye Bye\n");
    exit(0);
}

static void
abort_signal_handler(int sig){
    printf("process is aborted\n");
    printf("Bye Bye\n");
    exit(0);
}

int
main(int argc, char **argv){

    signal(SIGINT, ctrlc_signal_handler);
    signal(SIGABRT, abort_signal_handler);
    char ch;
    printf("Abort process (y/n) ?\n");
    scanf("%c", &ch);
    if(ch == 'y')
        abort();
    return 0;
}
```

2. A Process can send a signal to itself using *raise()*

```
int
raise (int signo);
```

- The signal denoted by *signo*, when raised using raise() is delivered to process itself


#include <signal.h>
#include <stdio.h>
void signal_catchfunc(int);
int main () {
int ret;
ret = signal(SIGINT, signal_catchfunc);
if(ret == SIG_ERR) {
printf("Error: unable to set signal handler.\n");
exit(0);
}
printf("Going to raise a signal\n");
ret = raise(SIGINT);
if(ret !=0) {
printf("Error: unable to raise SIGINT signal.\n");
exit(0);
}
printf("Exiting...\n");
return(0);
}
void signal_catchfunc(int signal) {
printf("!! signal caught !!\n");
}
1,1 All
[0$ bash 1-$ bash (2*$ bash)][0.00 0.01 0.05][2018/08/19 01:52:04am]

IPC – Techniques -> Signal Programming -> kill()

3. A Process can send a signal to another process using *kill()*

```
int
kill (int process_id, int signo);
```

- The sending process needs to know the recipient process id to send a signal

```c
#include <stdio.h>
#include <signal.h>

int
main(int argc, char **argv){

    kill(5939, SIGUSR1);
    scanf("\n");
    return 0;
}
```

Kill_sender.c

```c
#include <stdio.h>
#include <signal.h>

static void
signal_handler(int sig){

    printf("Signal %d recieved\n", sig);
}

int
main(int argc, char **argv){

    signal(SIGUSR1, signal_handler);
    scanf("\n");
    return 0;
}
```

Kill_recv.c

Path : IPC/IPC/signals/kill_sender.c & kill_recv.c