**2. IPC: Interrupts and Signals**

Name : Shreyas Vijaykumar Sakri.

PRN : 2019BTEIT00075

Write an application or program that communicates between to process opened in two terminal using kill() and signal().

**Objectives:**

1. To learn about IPC through signal.

2. To know the process management of Unix/Linux OS

3. Use of system call to write effective application programs.

**Theory:**

* In Unix and Unix-like operating systems, kill is a command used to send a signal to a process.
* By default, the message sent is the termination signal, which requests that the process exit. But kill is something of a misnomer; the signal sent may have nothing to do with process killing. The kill command is a wrapper around the kill() system call, which sends signals to processes or process groups on the system, referenced by their numeric process IDs (PIDs) or process group IDs (PGIDs). kill is always provided as a standalone utility as defined by the POSIX standard.
* However, most shells have built-in kill commands that may slightly differ from it.

**Data Variables:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Variable/Function** | **Datatype** | **Use** |
| 1 | SIGINT\_handler | void | Used for handling interrupt signal. |
| 2 | SIGQUIT\_handler | void | Used for dumped core handling. |
| 3 | pid | pid\_t | Process ID. |
| 4 | My Key | key\_t | Shared memory key. |
| 5 | ShmID | int | ID of shared memory. |
| 6 | shmPTR | pid\_t\* | Pointer |
| 7 | i | int | Iterating for loop |

**Program:**

#include <stdio.h>

#include <sys/types.h>

#include <signal.h>

#include <sys/ipc.h>

#include <sys/shm.h>

void SIGINT\_handler(int);

void SIGQUIT\_handler(int);

int ShmID;

pid\_t \*ShmPTR;

void main(void)

{

int i;

pid\_t pid = getpid();

key\_t MyKey;

if (signal(SIGINT, SIGINT\_handler) == SIG\_ERR) {

printf("SIGINT install error\n");

exit(1);

}

if (signal(SIGQUIT, SIGQUIT\_handler) == SIG\_ERR) {

printf("SIGQUIT install error\n");

exit(2);

}

MyKey = ftok(".", 's');

ShmID = shmget(MyKey, sizeof(pid\_t), IPC\_CREAT | 0666);

ShmPTR = (pid\_t \*) shmat(ShmID, NULL, 0);

\*ShmPTR = pid;

for (i = 0; ; i++) {

printf("From process %d: %d\n", pid, i);

sleep(1);

}

}

void SIGINT\_handler(int sig)

{

signal(sig, SIG\_IGN);

printf("From SIGINT: just got a %d (SIGINT ^C) signal\n",

sig); signal(sig, SIGINT\_handler);

}

void SIGQUIT\_handler(int sig)

{

signal(sig, SIG\_IGN);

printf("From SIGQUIT: just got a %d (SIGQUIT ^\\) signal"

" and is about to quit\n", sig);

shmdt(ShmPTR);

shmctl(ShmID, IPC\_RMID, NULL);

exit(3);

}

**Conclusion:**

1. Processes opened in two terminals can also be handled using signal handlers

2.kill() function calls. Shared memory can be used as a mode of IPC.

**References:**

[1] http://www.csl.mtu.edu/cs4411.ck/www/NOTES/signal/kill.html