**2. IPC: Interrupts and Signals**

Name : Shreyas Vijaykumar Sakri

PRN : 2019BTEIT00075

Write a application or program that communicates between child and parent processes 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:**

1) Kill:

Name:

kill - send signal to a process

Syntax:

#include <sys/types.h> #include

<signal.h> int kill(pid\_t pid, int sig);

Description:

* The kill() system call can be used to send any signal to any process group or process.
* If pid is positive, then signal sig is sent to pid.
* If pid equals 0, then sig is sent to every process in the process group of the current process.
* If pid equals -1, then sig is sent to every process for which the calling process has permission to send signals, except for process 1 (init), but see below.
* If pid is less than -1, then sig is sent to every process in the process group -pid.
* If sig is 0, then no signal is sent, but error checking is still performed.
* For a process to have permission to send a signal it must either be privileged (under Linux: have the CAP\_KILL capability), or the real or effective user ID of the sending process must equal the real or saved set-user-ID of the target process. In the case of SIGCONT it suffices when the sending and receiving processes belong to the same session.

Return Value:

On success (at least one signal was sent), zero is returned. On error, -1 is returned, and errno is set appropriately.

2) Signal:

Name :

signal - ANSI C signal handling

Syntax:

#include <signal.h>

typedef void (\*sighandler\_t)(int);

sighandler\_t signal(int signum, sighandler\_t handler);

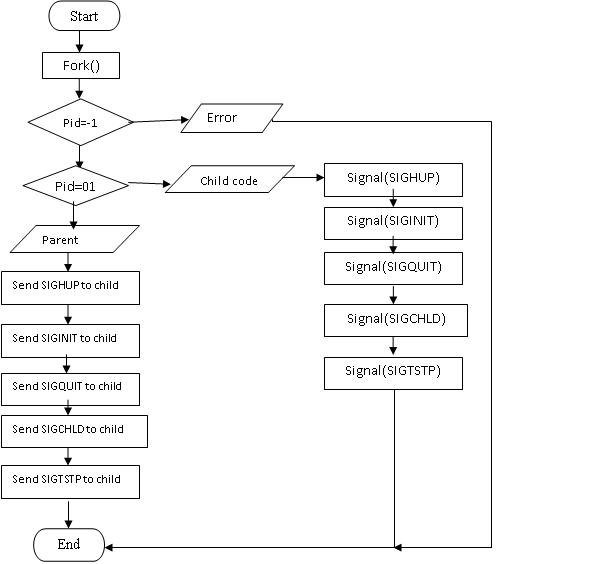
Description:

* The behavior of signal() varies across UNIX versions, and has also varied historically across different versions of Linux. Avoid its use: use sigaction(2) instead. See Portability below. signal() sets the disposition of the signal signum to handler, which is either SIG\_IGN, SIG\_DFL, or the address of a programmer-defined function (a "signal handler"). If the signal signum is delivered to the process, then one of the following happens:
* If the disposition is set to SIG\_IGN, then the signal is ignored.
* If the disposition is set to SIG\_DFL, then the default action associated with the signal occurs.
* If the disposition is set to a function, then first either the disposition is reset to SIG\_DFL, or the signal is blocked (see Portability below), and then handler is called with argument signum.
* If invocation of the handler caused the signal to be blocked, then the signal is unblocked upon return from the handler. The signals SIGKILL and SIGSTOP cannot be caught or ignored.

Return Value:

signal() returns the previous value of the signal handler, or SIG\_ERR on error. In the event of an error, errno is set to indicate the cause.

**Flowchart:**



**Data Dictionary:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | Variable/Function | Datatype | Use |
| 1 | inthandle | Void | Used for handling interrupt signal. |
| 2 | quithandle | void | Used for dumped core handling. |
| 3 | huphandle | void | Used for handling signal hang up. |
| 4 | pid | pid\_t | Used for child ID |
| 5 | ppid | pid\_t | Used for parent ID |

**Program:**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<signal.h>

void inthandle(int sig)

{

signal(SIGINT,inthandle);

printf("SIGINT invoked by daughter\n");

}

void quithandle(int sig)

{

signal(SIGQUIT,quithandle);

printf("SIGQUIT invoked by son\n He killed me\n");

exit(1);

}

void huphandle(int sig)

{

signal(SIGHUP,huphandle);

printf("SIGHUP invoked by child\n");

}

int main()

{

pid\_t ppid,pid;

ppid=getpid();

if((pid=vfork())<0)

{

printf("Fork Failed!!!!\n");

exit(1);

}

else if(pid==0){

printf("In Child!!!!\n");

signal(SIGINT,inthandle);

signal(SIGHUP,huphandle);

signal(SIGQUIT, quithandle);

printf("Looping\n");

for(;;);

}

else

{

printf("In Parent!!!!\n");

printf("kill SIGHUP\n");

kill(pid,SIGHUP);

sleep(2);

printf("kill SIGINT\n");

kill(pid,SIGINT);

sleep(2);

printf("kill SIGQUIT\n");

kill(pid,SIGQUIT);

sleep(2);

//exit(0);

}

}

**Output:**



**Conclusion:**

1. Various signal interrupts can be used in the form form signal handler

2. kill() can be used to evoke these signal to abort processes with different interrupts.

**References:**

[1]www.tutorialspoint.com/unix\_system\_calls/