

NAME – ASHUTOSH ARDU

REGNO – 20BRS1262

OS LAB 3

1. Using Ctrl – z instead of Ctrl – c or Ctrl - \\ to terminate any actively running program abruptly.

CODE

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
// Using ctrl-z instead of ctrl-c or ctrl-\\
void sigproc()
{
    signal(SIGINT, sigproc);
    printf("you have pressed ctrl-c \n");
}

void ch(){
    signal(SIGQUIT, ch);
    printf("you have pressed ctrl-\\ \n");
}

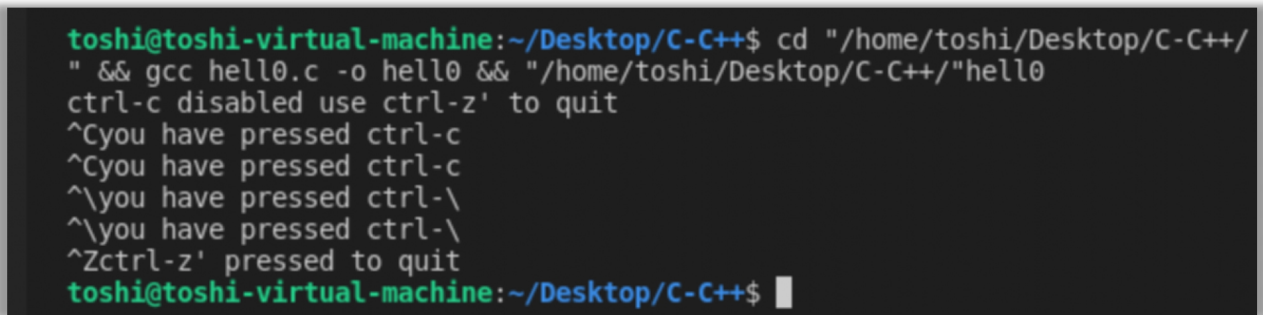
void quit()
{
    signal(SIGTSTP, quit);
    printf("ctrl-z' pressed to quit\n");
    exit(0); /* normal exit status */
}
```

```

void main()
{
    signal(SIGINT, sigproc);
    signal(SIGQUIT, ch);
    signal(SIGTSTP, quit);
    printf("ctrl-c disabled use ctrl-
z' to quit\n");
    for(;;); /* infinite loop */
}

```

OUTPUT



```

toshi@toshi-virtual-machine:~/Desktop/C-C++$ cd "/home/toshi/Desktop/C-C++/"
" && gcc hell0.c -o hell0 && "/home/toshi/Desktop/C-C++/"hell0
ctrl-c disabled use ctrl-z' to quit
^Cyou have pressed ctrl-c
^Cyou have pressed ctrl-c
^\\you have pressed ctrl-\\
^\\you have pressed ctrl-\\
^Zctrl-z' pressed to quit
toshi@toshi-virtual-machine:~/Desktop/C-C++$

```

2. Inter-process communication between the Parent and Child process.

CODE

```

#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <wait.h>
#include <unistd.h>
void sighup();
void sigint();
void hell();
void sigquit();

```

```
void main()
{
    int pid;
    pid=fork();
    /* get child process */

    if (pid < 0){
        perror("fork");
        exit(1);
    }

    if (pid == 0)
    { /* child */
        signal(SIGHUP,sighup);
        sleep(2);
        signal(SIGINT,sigint);
        sleep(2);
        signal(SIGTSTP,hell);
        sleep(2);
        signal(SIGQUIT, sigquit);
        for(;;); /* loop for ever */
    }
    else /* parent */
    { /* pid hold id of child */
        sleep(2);
        printf("\nPARENT: sending SIGHUP\n\n");
        kill(pid,SIGHUP);
        sleep(2); /* pause for 3 secs */
        printf("\nPARENT: sending SIGINT\n\n");
        kill(pid,SIGINT);
        sleep(2);
        printf("\nPARENT: sending SIGTSTP\n\n");
        kill(pid,SIGTSTP);
    }
}
```

```

        sleep(2);/* pause for 3 secs */
        printf("\nPARENT: sending SIGQUIT\n\n");
        kill(pid,SIGQUIT);
        sleep(2);
    }
}

void sighup(){    signal(SIGHUP,sighup);
    printf("CHILD: I have received a SIGHUP\n");
}

void sigint(){    signal(SIGINT,sigint);
    printf("CHILD: I have received a SIGINT\n");
}

void hell(){
    signal(SIGTSTP,hell);
    printf("CHILD: Hello Father, I have recieved SI
GTSTP\n\n");
}

void sigquit(){
    signal(SIGQUIT,sigquit);
    printf("CHILD: My DADDY has reaped me!!!\n");
    exit(0);
}

```

OUTPUT



```

toshi@toshi-virtual-machine:~/Desktop/C-C++$ cd "/home/toshi/Desktop/C-C++/"
" && gcc sandbox.c -o sandbox && "/home/toshi/Desktop/C-C++/"sandbox

PARENT: sending SIGHUP
CHILD: I have received a SIGHUP
PARENT: sending SIGINT
CHILD: I have received a SIGINT
PARENT: sending SIGTSTP
CHILD: Hello Father, I have recieved SIGTSTP

PARENT: sending SIGQUIT
CHILD: My DADDY has reaped me!!!
toshi@toshi-virtual-machine:~/Desktop/C-C++$ █

```

3. Write a chat application between two processes using signals and shared memory. User1 and User2 have created as two processes and shared memory is used to store the process id(s) of two processes. In this, *handler function* is used to print the message received from another process and vice versa.

USER 1 CODE

```

#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>

#define FILLED 0
#define Ready 1
#define NotReady -1

```

```

struct memory {
    char buff[100];
    int status, pid1, pid2;
};

struct memory* shmptr;
void handler(int signum)
{

    if (signum == SIGUSR1) {
        printf("Received User2: ");
        puts(shmptr->buff);
    }
}

int main()
{
    // process id of user1
    int pid = getpid();
    int shmid;
    // key value of shared memory
    int key = 12345;
    // shared memory create
    shmid = shmget(key, sizeof(struct memory),
IPC_CREAT | 0666);
    // attaching the shared memory
    shmptr = (struct memory*)sh
mat(shmid, NULL, 0);
    // store the process id of user1 in shared
memory
    shmptr->pid1 = pid;
    shmptr->status = NotReady;
}

```

```

    // calling the signal function using signal type SIGUSR1
    signal(SIGUSR1, handler);

    while (1) {
        while (shmptr->status != Ready)
            continue;
        sleep(1);

        // taking input from user1

        printf("User1: ");
        fgets(shmptr->buff, 100, stdin);

        shmptr->status = FILLED;

        // sending the message to user2 using kill function

        kill(shmptr->pid2, SIGUSR2);
    }

    shmdt((void*)shmptr);
    shmctl(shmid, IPC_RMID, NULL);
    return 0;
}

```

USER 2 CODE

```

#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>

```

```

#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>

#define FILLED 0
#define Ready 1
#define NotReady -1

struct memory {
    char buff[100];
    int status, pid1, pid2;
};

struct memory* shmptr;

// handler function to print message received from user1

void handler(int signum)
{
    // if signum is SIGUSR2, then user 2 is receiving a message from user1

    if (signum == SIGUSR2) {
        printf("Received From User1: ");
        puts(shmptr->buff);
    }
}

// main function

int main()
{
    // process id of user2

```



```

    int pid = getpid();
    int shmid;
    // key value of shared memory
    int key = 12345;
    // shared memory create
    shmid = shmget(key, sizeof(struct memory), IPC_
CREAT | 0666);
    // attaching the shared memory
    shmptr = (struct memory*)shmat(shmid, NULL, 0);
    // store the process id of user2 in shared memo
ry
    shmptr->pid2 = pid;
    shmptr->status = NotReady;
    signal(SIGUSR2, handler);

    while (1) {
        sleep(1);
        // taking input from user2
        printf("User2: ");
        fgets(shmptr->buff, 100, stdin);
        shmptr->status = Ready;
        // sending the message to user1 using kill
function
        kill(shmptr->pid1, SIGUSR1);
        while (shmptr->status == Ready)
            continue;
    }
    shmdt((void*)shmptr);
    return 0;
}

```

OUTPUT

USER – 1

USER - 2

```
OS Bootloader
toshi@toshi-virtual-machine: ~/Desktop/C-C++
toshi@toshi-virtual-machine:~$ cd "/home/toshi/Desktop/C-C++/" && gcc User1.c -o User1 && "/home/toshi/Desktop/C-C++/"User1
Received User2: Hey
User1: Hey
Received User2: Are you done with Assignment?
User1: Aaah..., Yes..?
Received User2: Oh could you plz send it to me
User1: No sorry man, plagiarism issues
Received User2: Oh i see. Anyways Thanks
User1: k bye
Received User2: bye
User1: ^C
toshi@toshi-virtual-machine:~/Desktop/C-C++$ █

toshi@toshi-virtual-machine:~/Desktop/C-C++
toshi@toshi-virtual-machine:~$ cd "/home/toshi/Desktop/C-C++/" && gcc User2.c -o User2 && "/home/toshi/Desktop/C-C++/"User2
User2: Hey
Received From User1: Hey
User2: Are you done with Assignment?
Received From User1: Aaah..., Yes..?
User2: Oh could you plz send it to me
Received From User1: No sorry man, plagiarism issues
User2: Oh i see. Anyways Thanks
Received From User1: k bye
User2: bye
^C
toshi@toshi-virtual-machine:~/Desktop/C-C++$ █
```