

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Programme | : | **BTech. CSE Core** | Semester | : | **Win 2021-22** |
| Course | : | **Operating Systems** | Code | : | **CSE2005** |
| Faculty | : | **Dr. Shyamala L** | Slot | : | **L25+L26** |
| Name | : | **Hariket Sukesh Kumar Sheth** | Register No. | : | **20BCE1975** |

**Lab 6:**Signal Handling in OS

**1**

|  |  |  |  |
| --- | --- | --- | --- |
| Date: 04-03-2022 | LAB 06 | Signal handling in OS |  |

**LAB 6**

1. Write your own C handlers to handle the following signals
   1. Send a stop signal using Ctrl-Z
   2. Segmentation fault
   3. Divide by zero error

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <signal.h>**

**#include <unistd.h>**

**void sighandler(int *sig\_num*){**

**switch(sig\_num){**

**case 8:{**

**printf("\n\n------------------------------------\n");**

**printf("Caught Division By Zero. Exiting.. :(\n");**

**printf("------------------------------------\n");**

**exit(1);**

**}**

**case 11:{**

**printf("\n\n------------------------------------\n");**

**printf("Caught Segmentation Fault. Exiting.. :(\n");**

**printf("------------------------------------\n");**

**exit(1);**

**}**

**case 20:{**

**printf("\n\n------------------------------------\n");**

**printf("Caught Ctrl+Z. Exiting.. :(\n");**

**printf("------------------------------------\n");**

**exit(1);**

**}**

**}**

**}**

1. Send a stop signal using Ctrl-Z

**Main program**

**int main(){**

**signal(SIGTSTP, sighandler);**

**signal(SIGFPE, sighandler);**

**signal(SIGSEGV, sighandler);**

**int num1 = 1;**

**int num2 = 5/num1;**

**2**

**char \*s = "CSE2005: Operating System";**

***//*\*s = 'A';**

**while(1){**

**printf("This is running..........\n");**

**sleep(1);**

**}**

**return 0;**

**}**

1. Segmentation Fault

**int main(){**

**signal(SIGTSTP, sighandler);**

**signal(SIGFPE, sighandler);**

**signal(SIGSEGV, sighandler);**

**int num1 = 1;**

**int num2 = 5/num1;**

**char \*s = "CSE2005: Operating System";**

**\*s = 'A';**

**while(1){**

**printf("This is running..........\n");**

**sleep(1);**

**}**

**return 0;**

**}**

1. Divide by Zero Error

**int main(){**

**signal(SIGTSTP, sighandler);**

**signal(SIGFPE, sighandler);**

**signal(SIGSEGV, sighandler);**

**int num1 = 0;**

**int num2 = 5/num1;**

**char \*s = "CSE2005: Operating System";**

***//*\*s = 'A';**

**while(1){**

**printf("This is running..........\n");**

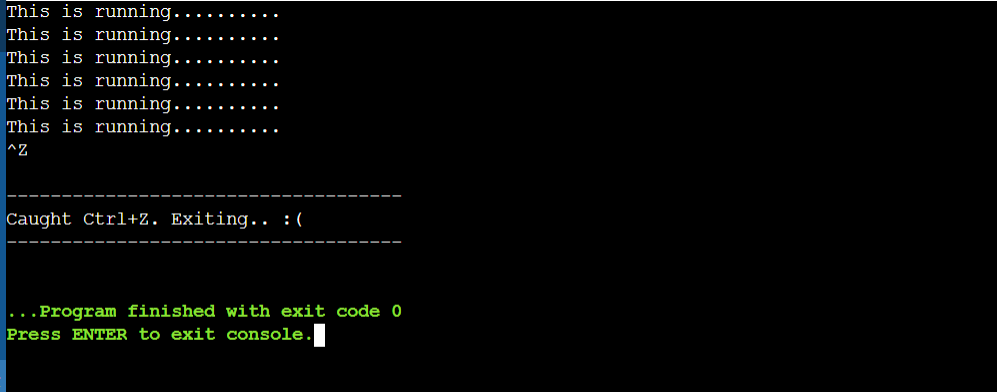
**sleep(1);**

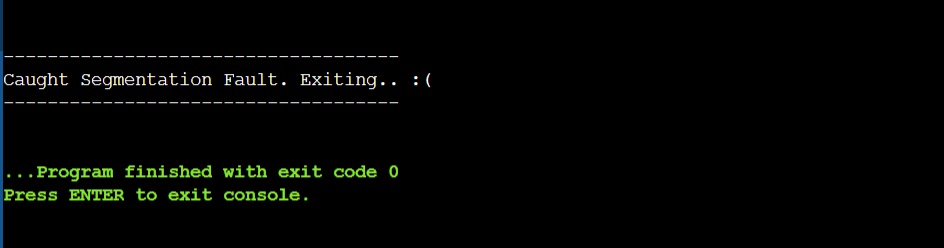
**}**

**return 0;**

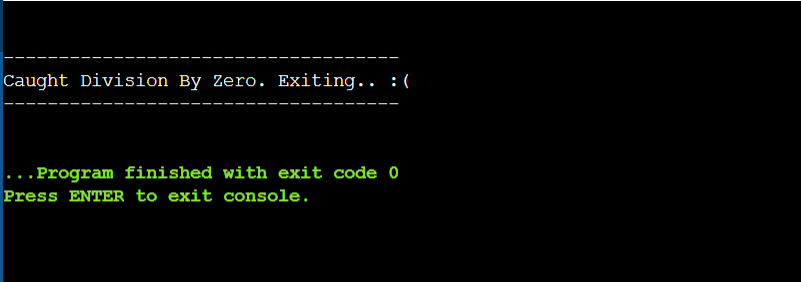
**}**

**OUTPUT:**

****

****

**3**

****

1. Write a program which creates a child process and continues to run along with its child (choose any small task of your own). Once the child completes its task, it should send a signal to parent which in turn terminates the parent. (Expected output: output of the task carried out by the child process, termination of parent)

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <sys/types.h>**

**#include <sys/wait.h>**

**void sig\_usr(int *signo*){**

**if(signo == SIGINT)**

**printf("Signal Received!");**

**return;**

**}**

**int main(){**

**int i, status;**

**pid\_t pid, ppid;**

**ppid = getpid();**

**printf("PARENT PROCESS CREATED\n\n");**

**printf("Running: \n");**

**printf("PARENT PROCESS STARTED\n\n");**

**pid = fork();**

**if(pid==0){**

**printf("CHILD PROCESS CREATED\n\n");**

**printf("Running: \n");**

**printf("CHILD PROCESS STARTED\n\n");**

**for(i=1; i<=50; i++)**

**if(i%2==0)**

**printf("Even: %d\n",i);**

**printf("killing parent...\n");**

**kill(ppid, SIGINT);**

**printf("CHILD PROCESS ENDED\n\n");**

**printf("PARENT PROCESS ENDED\n\n");**

**4**

**}**

**else{**

**if(pid>0)**

**pid = waitpid(pid, &status,0);**

**if(signal(SIGINT,sig\_usr) == SIG\_ERR)**

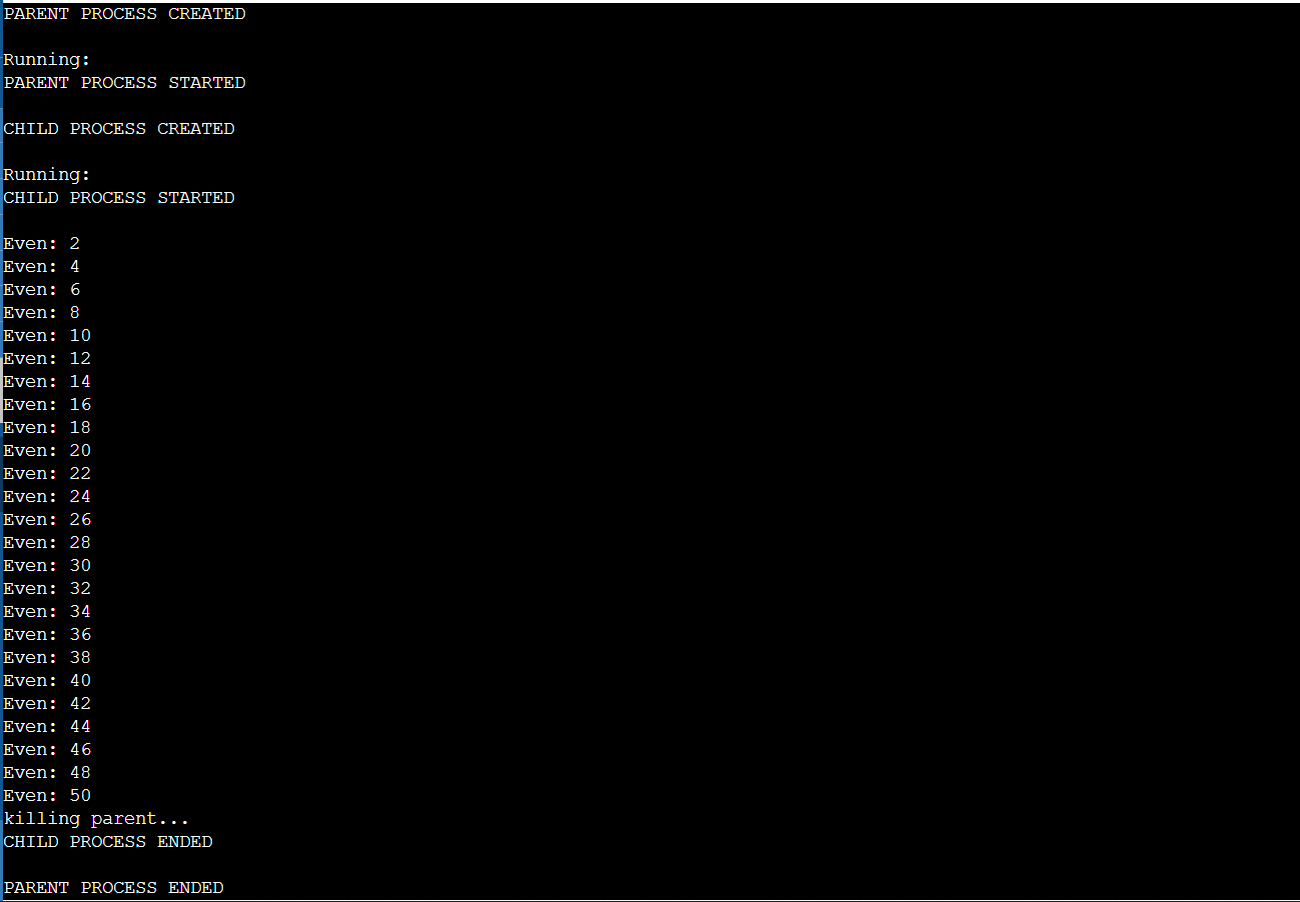
**printf("Signal processed ");**

**}**

**return 0;**

**}**

**OUTPUT:**

****

1. Write two c programs: One displaying the PID infinitely and the other program sending a signal to terminate the first program. (Note: Execute the programs in separate terminals)

**PARENT**

**#include<signal.h>**

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<string.h>**

**#include<sys/ipc.h>**

**#include<sys/types.h>**

**#include<unistd.h>**

**5**

**#include<sys/shm.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 parent**

**void handler(int *signum*){**

***/\*if signum is SIGUSR1, then Parent is receiving a message Parent \*/***

**if(signum==SIGUSR1){**

**printf("Received Child: ");**

**puts(shmptr->buff);**

**}**

**}**

**int main(){**

***//*process id of Parent**

**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 Parent in shared memory**

**shmptr->pid1 = pid;**

**shmptr->status = NotReady;**

***//*calling the signal function using signal type SIGparent**

**signal(SIGUSR1, handler);**

**while(1){**

**sleep(1);**

***//*taking input from Child**

**printf("Parent: ");**

**fgets(shmptr->buff, 100, stdin);**

**shmptr->status = FILLED;**

***//*sending the message to Parent using kill function**

**kill(shmptr->pid2,SIGUSR2);**

**}**

**shmdt((void\*)shmptr);**

**shmctl(shmid, IPC\_RMID, NULL);**

**return 0;**

**}**

**CHILD**

**#include<signal.h>**

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<string.h>**

**6**

**#include<sys/ipc.h>**

**#include<sys/types.h>**

**#include<unistd.h>**

**#include<sys/shm.h>**

**#define FILLED 0**

**#define Ready 1**

**#define NotReady -1**

**struct memory {**

**char buff[100];**

**int status, pid1, pid2;**

**};**

**struct memory\* shmptr;**

**void sig\_usr(int *signo*){**

**if(signo == SIGINT)**

**printf("Signal Received!");**

**return;**

**}**

***//*handler function to print message received from parent**

**void handler(int *signum*){**

***/\*if signum is SIGUSR1, then user 1 is receiving a message user 1 \*/***

**if(signum==SIGUSR2){**

**printf("\nKill Signal Received............\n");**

**kill(shmptr->pid1,SIGINT);**

**printf("Child killed :(\n");**

**exit(1);**

**}**

**}**

**int main(){**

***//*process id of user 1**

**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 user 1 in shared memory**

**shmptr->pid2 = pid;**

**shmptr->status = NotReady;**

***//*calling the signal function using signal type SIGparent**

**signal(SIGUSR2, handler);**

**while(1){**

***//*taking input from child**

**printf("Child: ");**

**shmptr->status = Ready;**

**kill(shmptr->pid1,SIGUSR1);**

**while(shmptr->status ==Ready){**

**7**

**printf("\nProcess ID: %d\n",shmptr->pid2);**

**sleep(2);**

**continue;**

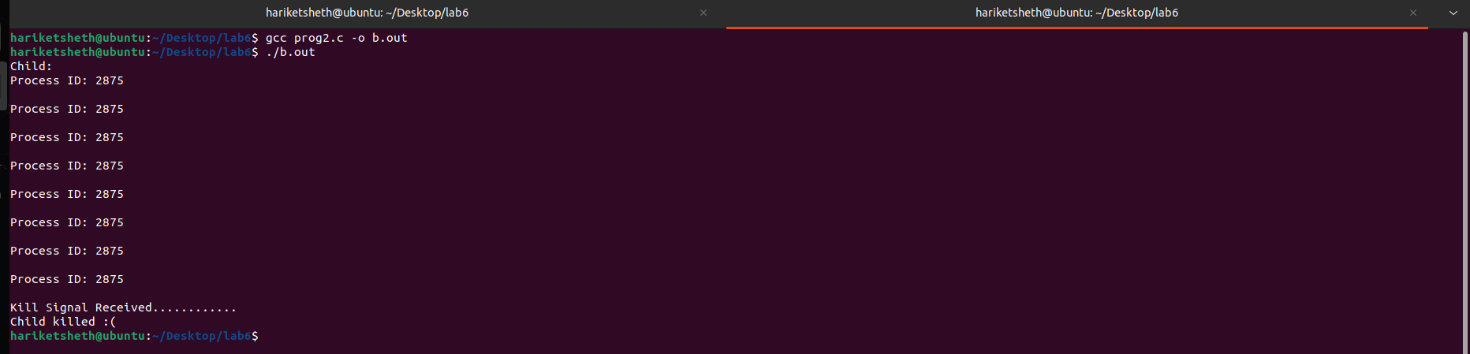
**}**

**}**

**shmdt((void\*)shmptr);**

**return 0;**

**}**

******OUTPUT:**

**8**