Raut Mangesh

CS503 Final

Problem 1.

int counter = 0; //declaring count to 0

void handler (int sig) //function handler parameter sig

{ counter++; //counter increment

}

int main() { //main method

signal(SIGUSR1, handler); //calling signal

signal(SIGUSR2, handler); //calling signal

int parent = getpid(); //parent getting pid

int child = fork(); //child to fork

if (child == 0) //if child is 0

{

kill(parent, SIGUSR1); //kill parent

kill(parent, SIGUSR2); //kill parent

kill(parent, SIGUSR1); //kill parent

kill(parent, SIGUSR2); //kill parent

exit(0); //exit safe

}

sleep(10); //wait or sleep

waitpid(child, NULL, 0); //waiting for child

printf("Received %d USR 1,2 signals\n", counter);

return 0;

}

In this program we can see the SIGUSR1 and SIGUSR2 which are used here in signal they are user-defined signals, so they aren’t triggered by any particular action. We can explicitly send them programmatically like the kill (parent, SIGUSR1); here we can see there are four times kill called if the child is 0 kill the parent with the SIGUSR1 and SIGUSR2.

Where pid is the process id of the receiving process. At the receiving end we can register a signal handler. Like we did in main signal(SIGUSR1, handler); signal(SIGUSR2, handler);

This program is all about the counter and the signals. First counter is defined to 0. Then the handler function is having parameter integer sig inside function counter increment when the function handler called it takes integer value sig and counter will increment.

The kill function can be used to send a signal to another process. It can be used for a lot of things other than causing a process to terminate. The parent process starts a child to perform a task. The kill function sends the signal SIGUSR1 and SIGUSR2 to the process or process group specified by pid. Here int parent is set to getpid and we are passing that in kill function. SIGUSR1 and SIGUSR2 can also have a value of zero to check the validity of the pid. We are having if loop that checks the child == 0. All processes in the same process group as the sender. A process can send a signal to itself with a call like kill (getpid(), signum). The return value from kill is zero if the signal can be sent successfully. Otherwise, no signal is sent and a value of -1 is returned. If pid specifies sending a signal to several processes, kill succeeds if it can send the signal to at least one of them. There’s no way you can tell which of the processes got the signal or whether all of them did.

The following errno conditions are defined for this function are EINVAL, EPERM and ESRCH. EINVAL is for invalid condition when the signal argument is invalid. EPERM is generate when you do not have privilege to send a signal to the process or processes in the group named by pid. ESRCH is the pid argument does not refer to an existing process or group.

As I tried to look on this code and see what is happening, I understand it is not giving any errors running perfectly and killing the parent without any error. It received 2 as counter because it called two times in main signal handler first time 1 second time 2 value would be the counter.

That’s why it will print output like this **Received 2 USR 1,2 signals. The value of the counter can be 2.**

Problem 2.

#include "csapp.h"

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#define NTHR 16

**void** \* do\_work (**void** \* arg);

**struct** AR

{

**double** \*array;

**int** size;

**double** \*sum;

};

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

{

**double** array[1024];

**double** sum;

pthread\_t id[4];

**struct** AR \* arg;

**void** \*return\_value;

**int** i;

**int** no\_of\_threads = NTHR;

printf("Creating %d number of threads...\n", no\_of\_threads);

**for** (i = 0; i < 100; i++) array[i] = i;

arg = (**struct** AR \* ) calloc (1, **sizeof** (**struct** AR) ) ;

arg->array = array;

arg->size = 100;

arg->sum = &sum;

**for**(i = 0; i <= 4; i++)

{

pthread\_create(&id[i], **NULL**, do\_work, arg);

}

**for**(i = 0; i <= 4; i++)

{

pthread\_join(id[i], &return\_value);

}

printf("Total Sum : %f\n",sum);

}

**void** \* do\_work (**void** \* arg)

{

**struct** AR \* argument;

**int** i, size;

**double** \* array;

**double** \* sum;

**double** localsum = 0;

**int** start stop;

argument = (**struct** AR \* ) arg;

size = argument -> size;

array = argument -> array;

sum = argument -> sum;

\* sum = 0;

**for** (i = 0; i < NTHR; i ++ )

{

localsum += argument[i];

}

\*sum += localsum;

pthread\_exit(0);

**return** **NULL**;

}