**Lab Report-04**

|  |  |
| --- | --- |
| Date: 9 April 2021 | Course Code: CSE325(1) |
| Course instructor: Tanni Mittra | Course Title: Operating System |
| Name: Adri Saha | Id: 2019-1-60-024 |

**Problem01:**

Write a C program that creates a child process. Then child sends an integer to the parent process.

Parent reads the value, and then determines whether the integer is an odd or an even number. The

result must be sent to the child again who eventually reads the value (odd or even). You may use atoi()

to convert a string into an integer. The following shows a sample interaction of the program.

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/wait.h>

int main()

{

int n,pipeDC1[2], pipeDC2[2];

char \*message;

if(pipe(pipeDC1) == -1) {

printf("Finding Error in Pipe Descriptor 1\n");

exit(1);

}

if(pipe(pipeDC2) == -1) {

printf("inding Error in Pipe Descriptor 2\n");

exit(1);

}

pid\_t pid = fork();

if(pid < 0) {

printf("Failed to create child\n");

return 1;

}

if(pid == 0) {

close(pipeDC1[0]);

printf("Inside child process %d\n",(int)getpid());

printf("Enter the integer number: ");

scanf("%d", &n);

printf("\nSending to the parent...\n");

write(pipeDC1[1], &n, sizeof(n));

close(pipeDC1[1]);

wait(NULL);

printf("Inside child process %d\n", (int)getpid());

close(pipeDC2[1]);

char \*result = calloc(256, 4);

read(pipeDC2[0], result, sizeof(result));

printf("Received result: %d is an %s number\n", n, result);

close(pipeDC2[0]);

printf("Child Terminated\n");

exit(1);

}

else {

sleep(5);

close(pipeDC1[1]);

printf("Inside parent process %d\n", (int)getpid());

read(pipeDC1[0], &n, sizeof(n));

printf("Received result: %d\n", n);

close(pipeDC1[0]);

if(n%2!= 0) {

message = "odd ";

}

else {

message = "even ";

}

close(pipeDC2[0]);

printf("\nSending result to the child...\n");

write(pipeDC2[1], message, sizeof(message));

close(pipeDC2[1]);

wait(NULL);

printf("Parent terminated\n");

}

return 0;

}

**Output:**

Text

Description automatically generated

**THREADS**

**Problem 2.** Write a C program that creates multiple threads with NULL as parameter sent to the thread

execution function, runner. Each thread inside the runner function must reads an integer value, n,

and then produce sum of values from 1 to n(inclusive) and prints the sum. Ensure that the main()

must wait for the termination of these two threads.

**Code:**

#include <stdio.h>

#include <pthread.h>

#include<stdlib.h>

#include<unistd.h>

int sum=0;

void \*runner(void \*no)

{

int \*number,i;

number=(int\*)no;

for(i=1; i<=\*number; i++)

{

sum=sum + i;

}

printf("Final Sum is : %d \n",sum);

// pthread\_exit(NULL);

}

int main()

{

int n;

pthread\_t tid [2];

//pthread\_attr\_t attr[2];

printf("Enter Number Up to Which You want to Sum :");

scanf("%d",&n);

int i,rc;

for(i=0; i < 2; i++)

{

printf("Creating thread %d\n",i);

rc = pthread\_create(&tid[i],NULL,runner,(void \*)&n);

pthread\_join(tid[i], NULL);

printf("Sum of thread %d is: %d \n",i,sum);

if (rc)

{

printf("ERROR! return code from pthread\_create() is %d\n", rc);

//exit(NULL);

}

}

printf("\n");

}

**Output**Text

Description automatically generated

**Problem 3:** This is an extension of the previous problem. The two threads now only calculate the sum but don’t print it. Instead the main() must print the sum of these two sums produced by these two threads.

Modify the solution of the previous problem accordingly to solve this one.

**Code:**

#include <stdio.h>

#include <pthread.h>

#include<stdlib.h>

#include<unistd.h>

int sum=0;

void \*runner(void \*no)

{

int \*number,i;

number=(int\*)no;

for(i=1; i<=\*number; i++)

{ sum=sum + i;

}

// pthread\_exit(NULL);

}

int main()

{

int n;

pthread\_t tid [2];

//pthread\_attr\_t attr[2];

printf("Enter Number Up to Which You want to Sum :");

scanf("%d",&n);

int i,rc;

for(i=0; i<2; i++)

{

//printf("Creating thread %d\n",i);

rc = pthread\_create(&tid[i],NULL,runner,(void \*)&n);

pthread\_join(tid[i], NULL);

printf("Sum of thread %d is: %d \n",i,sum);

if (rc)

{

printf("ERROR! return code from pthread\_create() is %d\n", rc);

//exit(NULL);

}

}

printf("\n");

printf("Final Sum by 2 thread is : %d \n",sum);

// return 0;

}

**Output:**

Text

Description automatically generated