Student Name: Kawshik Lodh

Student Id: 11605555

Email Address: [lodh.kawshik@gmail.com](mailto:lodh.kawshik@gmail.com)

Github Link: <https://github.com/Kawshik/Operating-System-CA2/>

Question no: 02

Code:

#include<stdio.h>

#include<string.h>

#include<pthread.h>

#include<stdlib.h>

#include<unistd.h>

#define NUM\_THREAD 3

int average, min, max;

void \*

doSomeThing(void \*param)

{

int \*argv = (int \*) param;

sleep(1);

int j;

int sum = 0;

int upper = atoi(param);

sleep(1);

pthread\_t id = pthread\_self();

unsigned long i = 0;

if (id = 1) {

int i;

for (i = 0; i < upper; i++) {

sum += argv[i];

}

printf("sum of no's is :\n", sum);

}

if (id = 2) {

printf("\n Second thread processing\n");

}

if (id = 3) {

printf("\n Third thread processing\n");

}

for (i = 0; i < -1; i++);

{

pthread\_exit(NULL);

}

}

int

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

{

pthread\_t threads[NUM\_THREAD];

pthread\_attr\_t attr;

int \*taskid[NUM\_THREAD];

int i = 0;

int t;

int err;

//int input,a;

if (argc != 2) {

fprintf(stderr, "usage: a.out <integer value>\n");

return -1;

}

pthread\_attr\_init(&attr);

for (t = 0; t < NUM\_THREAD; t++) {

taskid[t] = (int \*) malloc(sizeof(int));

\*taskid[t] = t;

printf("In main: creating thread %d\n", t);

err = pthread\_create(&threads[t], &attr, doSomeThing, argv[1]);

if (err) {

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

err);

exit(-1);

}

}

for (t = 0; t < NUM\_THREAD; t++) {

pthread\_join(threads[t], NULL);

printf("Joining thread %d\n", t);

}

pthread\_exit(NULL);

}

Description: In the following program we had to calculate various statistical values for a list of numbers. We had to create threads to create the various values such as average, minimum and maximum values.

Threads as we know are scheduled on processors and each thread can execute a set of instructions independent of other processes and threads.

So here we have created a thread to find the average, one to find the maximum and one for minimum.

Algorithm:

Step1: Take user input

Step 2: Create a thread to Calculate Average

Step 3: Create a thread to Calculate Minimum Number

Step 4: Create a thread Calculate Maximum Number

Step 5: Display data

Code:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <ctype.h>

#define P1\_READ 0

#define P2\_WRITE 1

#define P2\_READ 2

#define P1\_WRITE 3

#define NUM\_PIPES 2

char \*toggleString(char \*argv){

int i;

char \*str = malloc(sizeof(argv[1]));

strcpy(str, argv );

for(i=0;str[i]!='\0';i++)

{

if(str[i]>='A'&&str[i]<='Z')

{

str[i]+=32 ;

} else if (str[i]>='a'&&str[i]<='z')

{

str[i]-=32; //Make it uppercase

}

}

return str;

}

int inputValidation(int argc, char \*argv[]){

int i;

int c = 0;

char str[strlen(argv[1])];

strcpy(str, argv[1]);

if (argc != 2)

{

fprintf(stderr, "\nUsage: %s <string> or <'string 1, string 2', ..., string n'> for multiple strings\n", argv[0]);

exit(EXIT\_FAILURE);

} else

{

for(i=0;i<strlen(str);i++) {

if(isalpha((int) str[i]))

{

c = 1;

}

}

if(c == 0){

printf("\nSorry, The string you entered did NOT contain any Alphabetical Characters\nRun me again, with at least 1 Alphabetical character\n\n");

exit(EXIT\_FAILURE); //Exit on improper input

}

return (0);

}

}

int main(int argc, char \*argv[]) {

assert(argc>1);

int fd[2\*NUM\_PIPES];

int len, i;

pid\_t pid;

char parent[strlen(argv[1])];

char child[strlen(argv[1])];

if(inputValidation(argc, argv) == 0)

strcpy(parent, argv[1]);

for (i=0; i<NUM\_PIPES; ++i)

{

if (pipe(fd+(i\*2)) < 0)

{

perror("Failed to allocate pipes");

exit(EXIT\_FAILURE);

}

}

if ((pid = fork()) < 0)

{

perror("Failed to fork process");

return EXIT\_FAILURE;

}

if (pid == 0)

{

close(fd[P1\_READ]);

close(fd[P1\_WRITE]);

pid = getpid();

len = read(fd[P2\_READ], &child, len);

if (len < 0)

{

perror("Child: Failed to read data from pipe");

exit(EXIT\_FAILURE);

}

else if (len == 0)

{

fprintf(stderr, "Child: Read EOF from pipe");

}

else

{

printf("Child(%d): Recieved Message\n\nChild(%d): Toggling Case and Sending to Parent\n",pid, pid);

if (write(fd[P2\_WRITE], toggleString(child), strlen(child)) < 0)

{

perror("Child: Failed to write response value");

exit(EXIT\_FAILURE);

}

}

close(fd[P2\_READ]);

close(fd[P2\_WRITE]);

return EXIT\_SUCCESS;

}

close(fd[P2\_READ]);

close(fd[P2\_WRITE]);

pid = getpid();

printf("\nParent(%d): Sending %s to Child\n\n", pid, argv[1]);

if (write(fd[P1\_WRITE], argv[1], strlen(argv[1])) != strlen(argv[1]))

{

perror("Parent: Failed to send value to child ");

exit(EXIT\_FAILURE);

}

len = read(fd[P1\_READ], &parent, strlen(parent));

if (len < 0)

{

perror("Parent: failed to read value from pipe");

exit(EXIT\_FAILURE);

}

else if (len == 0)

{

fprintf(stderr, "Parent(%d): Read EOF from pipe", pid);

}

else

{

printf("\nParent(%d): Received %s from Child\n\n", pid, parent);

}

close(fd[P1\_READ]);

close(fd[P1\_WRITE]);

wait(NULL);

return EXIT\_SUCCESS;

}

Description:

Pipe is a technique for passing information from one program to another.

Here in this program we are using inter process communication using ordinary pipes, where we send a string to a process and the second process converts the case of each character and sends it back. We are using two pipes one for sending the message from the first to the second the other for getting the data back.

Algorithm:

Step 1: First take the input from the user

Step 2: Validate the inputs

Step 3: Create an array of pipes

Step 4: Assign the process to the pipes

Step 5: Close all the pipes for future use