Lab 8 - Programs on Threads

1. Write a multithreaded program that generates the Fibonacci series. The program should work as follows: The user will enter on the command line the number of Fibonacci numbers that the program is to generate. The program then will create a separate thread that will generate the Fibonacci numbers, placing the sequence in data that is shared by the threads (an array is probably the most convenient data structure). When the thread finishes execution, the parent will output the sequence generated by the child thread. Because the parent thread cannot begin outputting the Fibonacci sequence until the child thread finishes, this will require having the parent thread wait for the child thread to finish.

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

void\* generate\_fibonacci(void\* param)

{

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

int n = arr[0];

arr[1] = 0;

arr[2] = 1;

for(int i = 3;i <= n;i++)

{

arr[i] = arr[i-1] + arr[i-2];

}

return NULL;

}

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

{

int n;

printf("Enter no of Fibonacci numbers : \n");

scanf("%d",&n);

int\* arr = (int\*)malloc((n+1)\*sizeof(int));

arr[0] = n;

pthread\_t thread;

pthread\_create(&thread,0,&generate\_fibonacci,(void\*)arr);

pthread\_join(thread,0);

for(int i = 1;i <= n;i++)

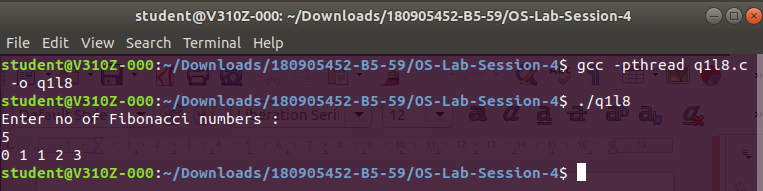
printf("%d ",arr[i]);

printf("\n");

return 0;

}

Output



2. Write a multithreaded program that calculates the summation of non-negative integers in a separate thread and passes the result to the main thread.

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

void\* summation(void\* param)

{

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

int sum = 0;

int n = arr[0];

for(int i = 1;i <= n;i++)

{

if(arr[i] > 0)

sum += arr[i];

}

return (void\*)sum;

}

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

{

int n;

printf("Enter the no. of numbers : \n");

scanf("%d",&n);

int\* arr = (int\*)malloc((n+1)\*sizeof(int));

arr[0] = n;

printf("Enter the numbers : \n");

for(int i= 1;i <= n;i++)

{

scanf("%d",&arr[i]);

}

int answer = 0;

pthread\_t thread;

pthread\_create(&thread,0,&summation,(void\*)arr);

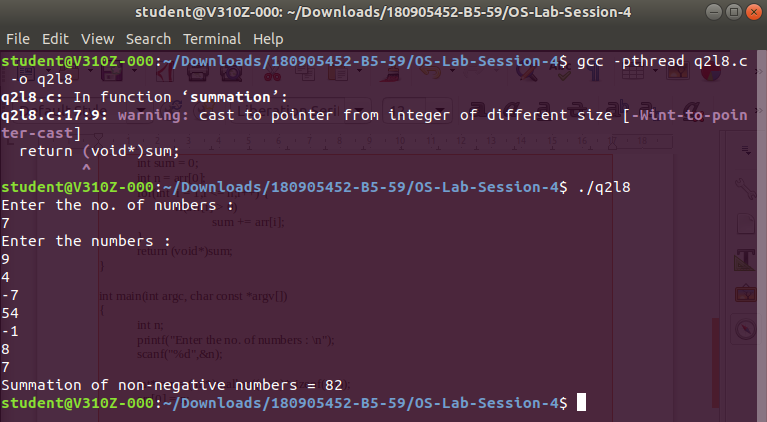
pthread\_join(thread,(void\*\*)&answer);

printf("Summation of non-negative numbers = %d\n",answer);

return 0;

}

Output



3. Write a multithreaded program for generating prime numbers from a given starting number to the given ending number.

#include<stdio.h>

#include<pthread.h>

#define N 30

#define MAX\_THREADS 4

int prime\_arr[N]={0};

void \*printprime(void \*ptr)

{

int j,flag;

int i=(int)(long long int)ptr;

while(i<N)

{

// printf("Thread id[%d] checking [%d]\n",pthread\_self(),i);

flag=0;

for(j=2;j<=i/2;j++)

{

if(i%j==0)

{

flag=1;

break;

}

}

if(flag==0 && (i>1))

{

prime\_arr[i]=1;

}

i+=MAX\_THREADS;

}

}

int main()

{

pthread\_t tid[MAX\_THREADS]={{0}};

int count=0;

printf("Enter starting and ending\n");

int st,en;

scanf("%d %d",&st,&en);

for(count=0;count<MAX\_THREADS;count++)

{

// printf("\r\n CREATING THREADS %d",count);

pthread\_create(&tid[count],NULL,printprime,(void\*)count);

}

printf("\n");

for(count=0;count<MAX\_THREADS;count++)

{

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

}

int c=0;

for(count=st;count<en;count++)

if(prime\_arr[count]==1)

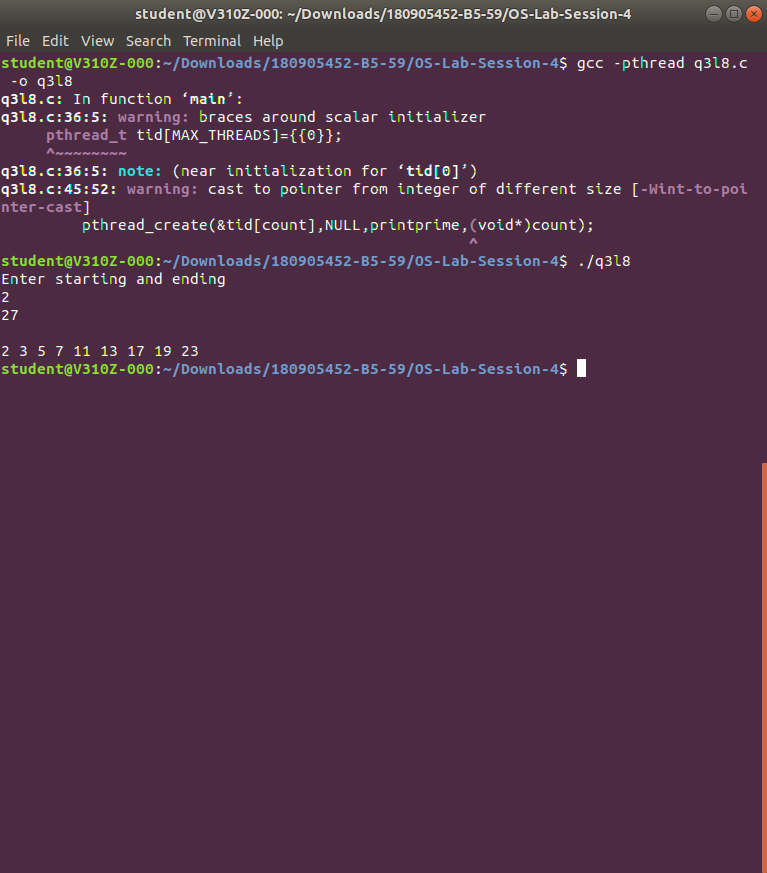
printf("%d ",count);

printf("\n");

return 0;

}

Output



4. Write a multithreaded program that performs the sum of even numbers and odd numbers in an input array. Create a separate thread to perform the sum of even numbers and odd numbers. The parent thread has to wait until both the threads are done.

#include <errno.h>

#include <ctype.h>

#include <unistd.h>

#define handle\_error\_en(en, msg) \

do { errno = en; perror(msg); exit(EXIT\_FAILURE); } while (0)

volatile int running\_threads = 0;

pthread\_t thread[1];

int numOfElements;

struct Results

{

int sum;

}Results,Results2;

void \*findsum(void \*array\_ptr)

{

int i; /\*counter\*/

int \*elements = (int\*)array\_ptr;

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

{

if(elements[i]%2==0)

Results.sum += elements[i];

else

Results2.sum += elements[i];

}

running\_threads -= 1;

return NULL;

}

int getArrayInput(int n, int \*array\_ptr)

{

int input;

int numberOfElements = 0;

printf("Creating Dynamic Array...\n-\n");

for(;;)

{

printf("Enter a positive value:\nNegative Number to Stop\n-\n");

if (scanf("%d",&input) != 1)

{

printf("\nOops that wasn't an Integer\nlets try filling the array again\nRemember INTEGERS only!\n");

exit(EXIT\_FAILURE);

}

if (input >= 0)

{

if (numberOfElements == n)

{

n += 1;

array\_ptr = realloc(array\_ptr, n \* sizeof(int));

}

array\_ptr[numberOfElements++] = input;

}

else

{

printf("\nNumber of Integers: %d\n", numberOfElements);

break;

}

}

return numberOfElements;

}

void createThreads(int \*array\_ptr)

{

int s;

s = pthread\_create(&thread[2], NULL, findsum, (void \*)array\_ptr);

if (s != 0)

{

handle\_error\_en(s, "pthread\_create");

}

running\_threads += 1;

}

int main()

{

int n = 1;

int \*array\_ptr = malloc(n \* sizeof(int));

numOfElements = getArrayInput(n, array\_ptr);

createThreads(array\_ptr);

while(running\_threads>0)

{

sleep(1);

}

printf("\nThe sum of even %d and odd %d\n",Results.sum,Results2.sum);

return(0);

}

Output

