OS LAB8

NAME :SAGNIK CHATTERJEE

ROLL NO: 61

SEC : B

REG :180905478

Q1

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

int prevsum;//shred value by the threads

void \*runner(void \*param)

{

prevsum = fibonacci((int)param);

pthread\_exit(0);

}

int fibonacci (int x)

{

if (x <= 1) {

return 1;

}

return fibonacci(x-1) + fibonacci(x-2);

}

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

{

int count, i;

pthread\_attr\_t attr;

if (argc != 2) {

fprintf(stderr,"usage: pthreads <integer value>\n");

exit(1);

}

count = atoi(argv[1]);

if (count < 1) {

fprintf(stderr,"%d must be>= 1\n", count);

exit(1);

}

pthread\_attr\_init(&attr);

for(i=1;i<=count;i++){

pthread\_t thread;

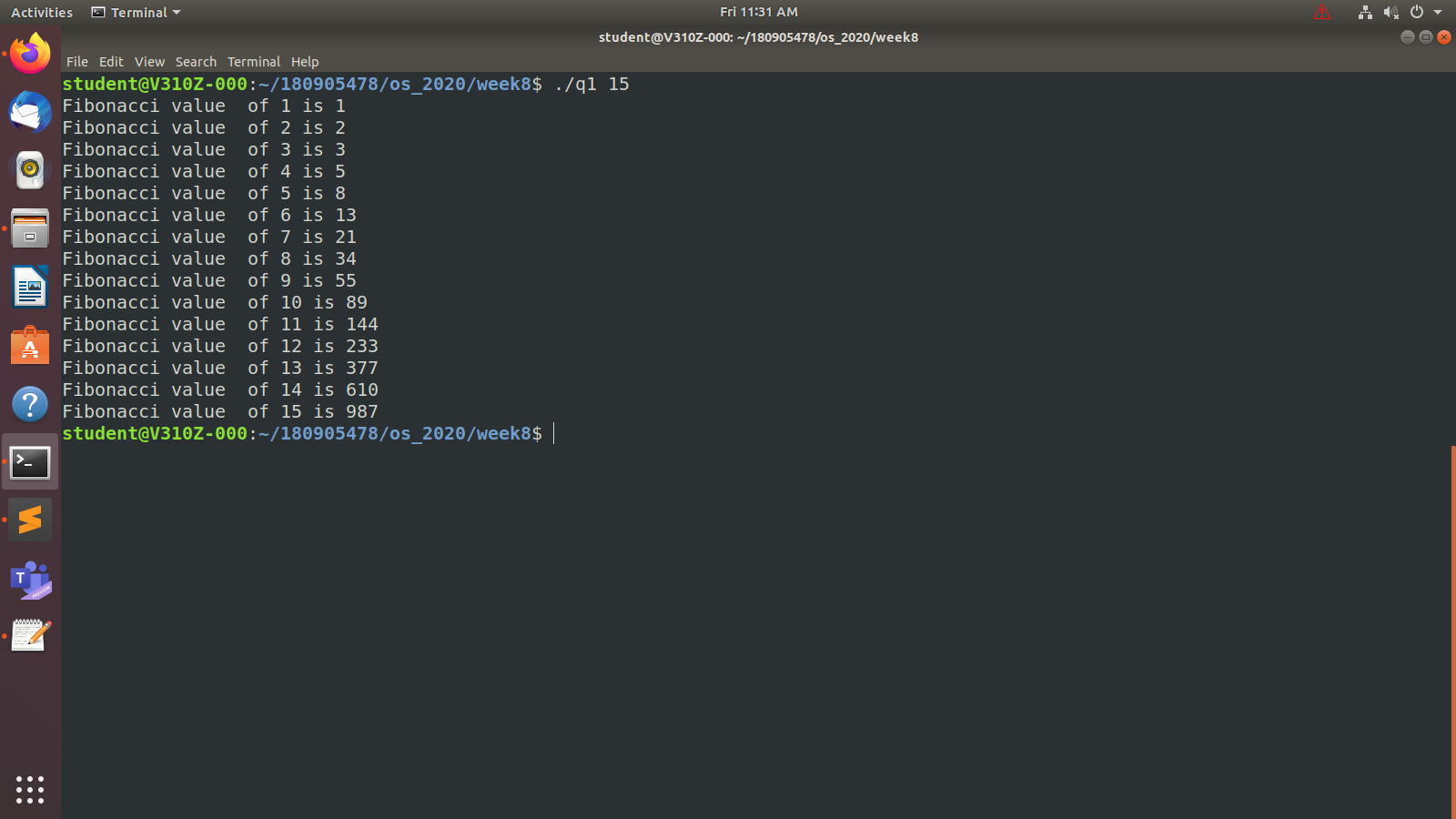
pthread\_create(&thread,&attr,runner,(void\*)i);

pthread\_join(thread,NULL);

printf("Fibonacci value of %d is %d\n", i, prevsum);

}

}



Q2

#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

#define max\_threads 4

#define size 16

int arr[] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 };

int sum[]={0,0,0,0};

int part=0;

void \* sum\_array(){

int thread\_part =part++;

//using 4 threads so diviing into 4 parts

for(int i=thread\_part \* (size/4);i<(thread\_part+1)\*(size/4);i++){

sum[thread\_part]+=arr[i];

}

}

int main(){

printf("Numbers given for summation \n");

for(int i=0;i<16;i++){

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

}

printf("\n");

pthread\_t threads[max\_threads];

//create the 4 threads

for(int i=0;i<max\_threads;i++){

pthread\_create(&threads[i],NULL,sum\_array,(void\*)NULL);

}

//wait for all threads to complete before joining

for(int i=0;i<max\_threads;i++){

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

}

//adding sum of all 4 parts

int total\_sum=0;

for(int i=0;i<max\_threads;i++){

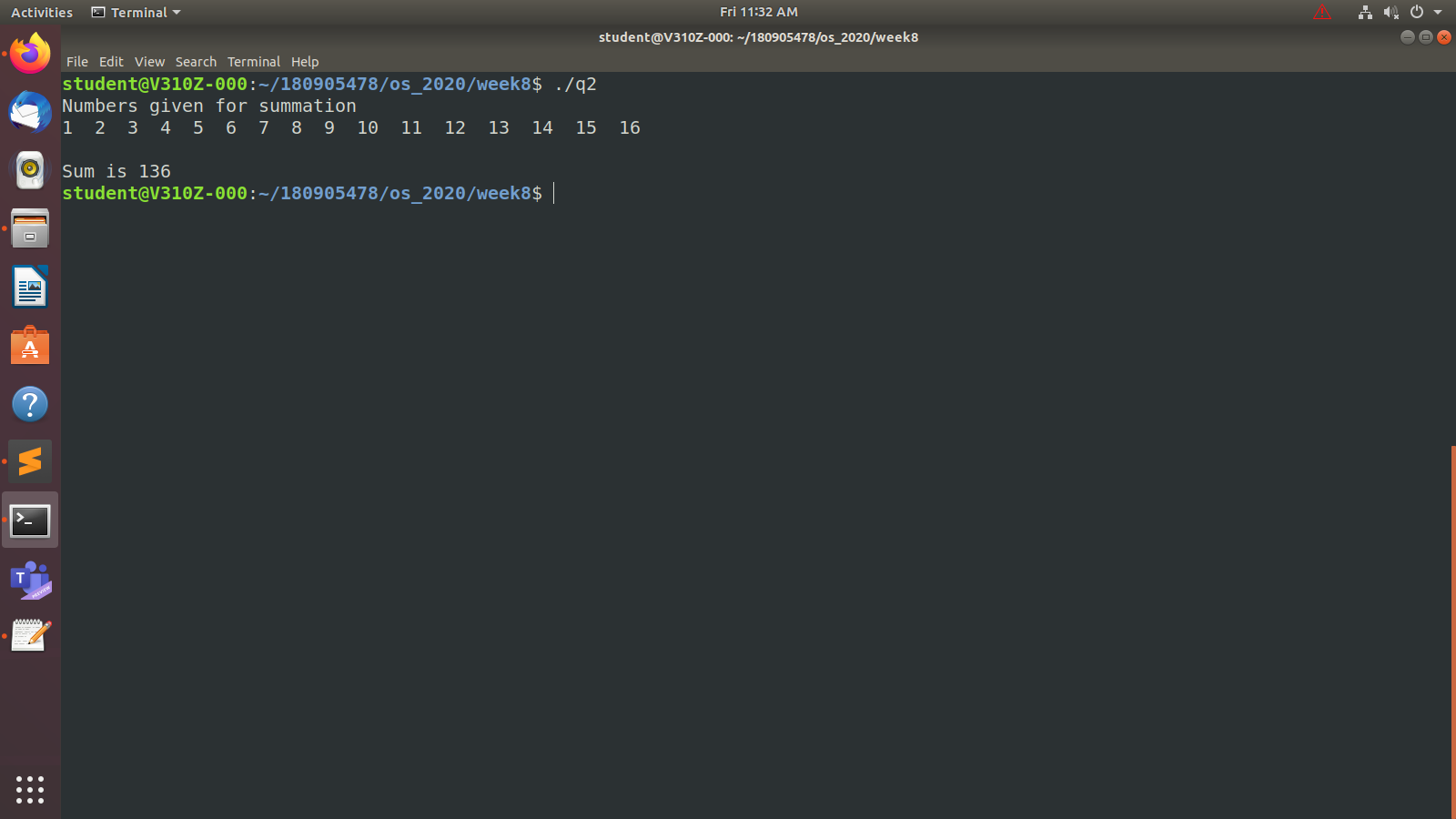
total\_sum+=sum[i];

}

printf("\nSum is %d\n",total\_sum);

return 0;

}



Q3

#include<stdio.h>

#include<pthread.h>

#define N 10

#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[%ld] 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(int argc ,char \*\*argv)

{

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

printf("The numbers added are :-");

int a =atoi(argv[1]);

int b =atoi(argv[2]);

printf("%d\n",a);

printf("%d\n",b);

int count=0;

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;

printf("The prime numbers found are :\n");

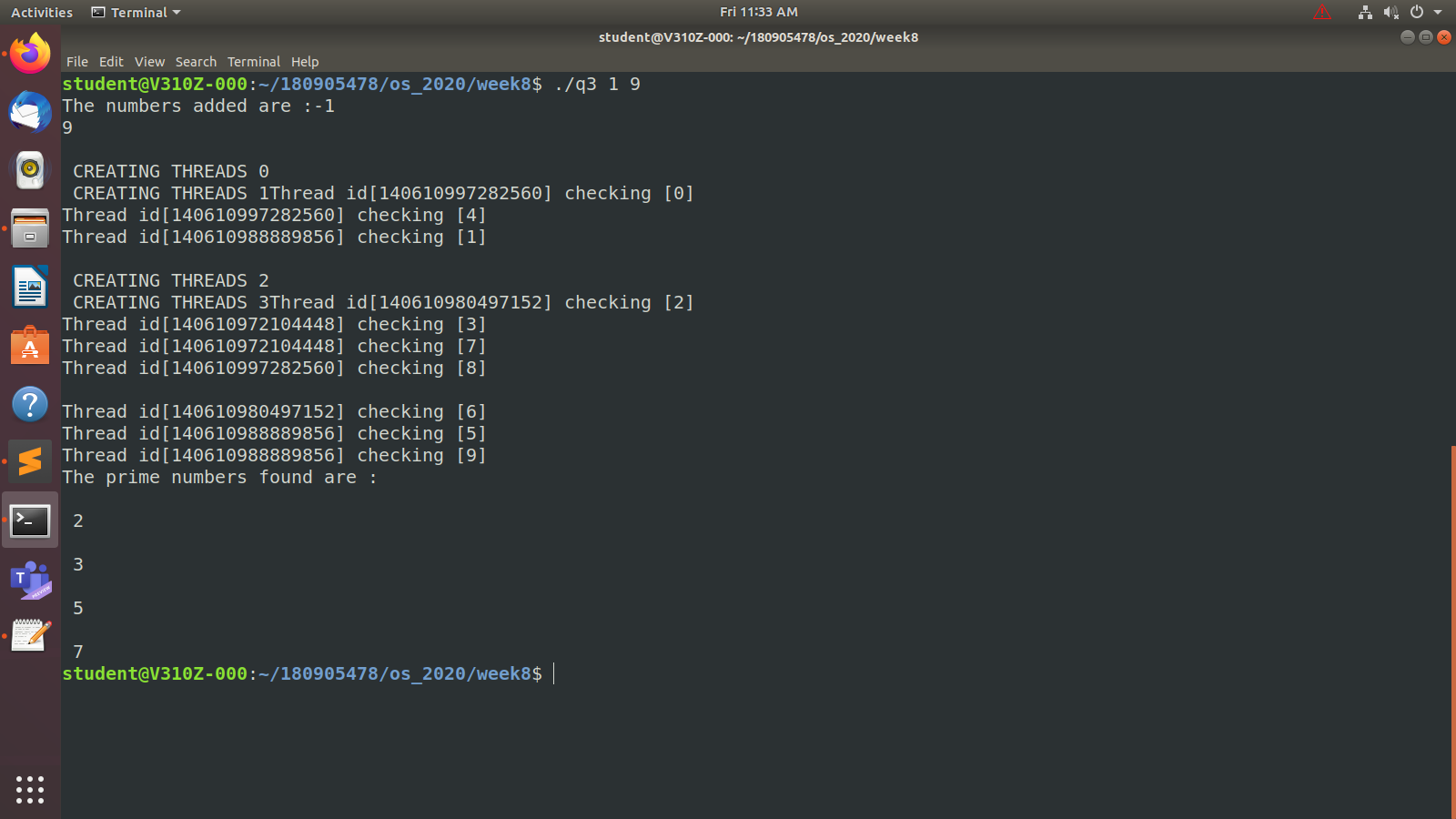
for(count=a;count<=b;count++)

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

printf("\n %d \n",count);

return 0;

}



Q4

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

pthread\_mutex\_t count\_mutex = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_cond\_t condition\_var = PTHREAD\_COND\_INITIALIZER;

int count = 0;

//checking for even and odd nums till this range

#define COUNT\_DONE 500

// print odd numbers

void \*oddNums(void\* args)

{

for(;;) {

// Lock mutex and then wait for signal to relase mutex

pthread\_mutex\_lock( &count\_mutex );

if ( count % 2 != 0 ) {

pthread\_cond\_wait( &condition\_var, &count\_mutex );

}

count++;

printf("Counter value oddSums: %d\n",count);

pthread\_cond\_signal( &condition\_var );

if ( count >= COUNT\_DONE ) {

pthread\_mutex\_unlock( &count\_mutex );

return(NULL);

}

pthread\_mutex\_unlock( &count\_mutex );

}

}

// print even numbers

void \*evenNums(void\* args)

{

for(;;) {

// Lock mutex and then wait for signal to release mutex

pthread\_mutex\_lock( &count\_mutex );

if ( count % 2 == 0 ) {

pthread\_cond\_wait( &condition\_var, &count\_mutex );

}

count++;

printf("Counter value evenSum: %d\n",count);

pthread\_cond\_signal( &condition\_var );

if( count >= COUNT\_DONE ) {

pthread\_mutex\_unlock( &count\_mutex );

return(NULL);

}

pthread\_mutex\_unlock( &count\_mutex );

}

}

int main()

{

pthread\_t thread1, thread2;

pthread\_create(&thread1, NULL, oddNums, NULL);

pthread\_create(&thread2, NULL, evenNums, NULL);

pthread\_join(thread1, NULL);

pthread\_join(thread2, NULL);

return 0;

}

//in the screenshot only shown till 33

