ASSIGNMENT- IV

NAME: SOUMYADIP GHOSH

STREAM: CSE-A

ROLL NUMBER: 1951007

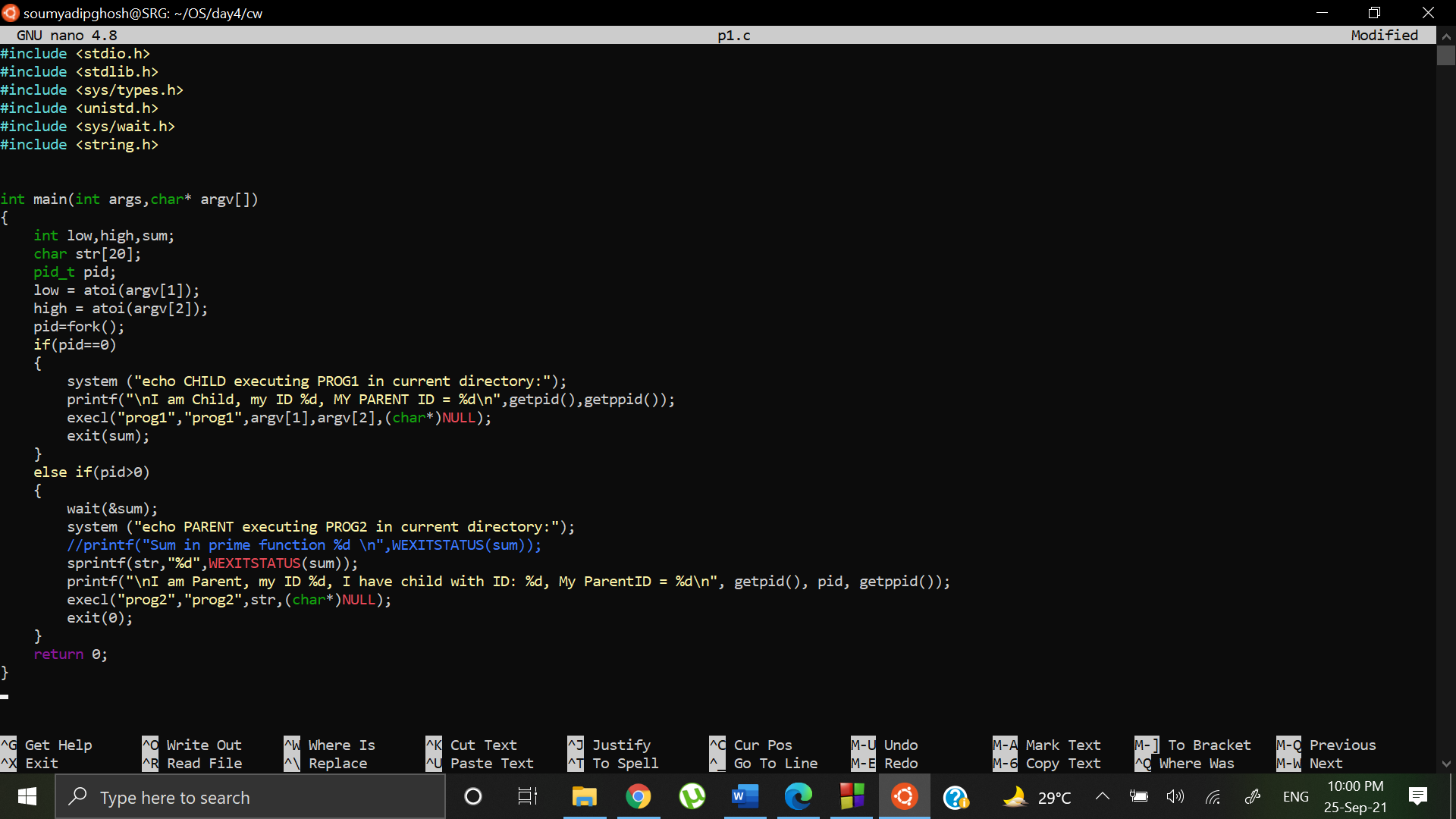
SUBJECT: OS LAB

Classwork

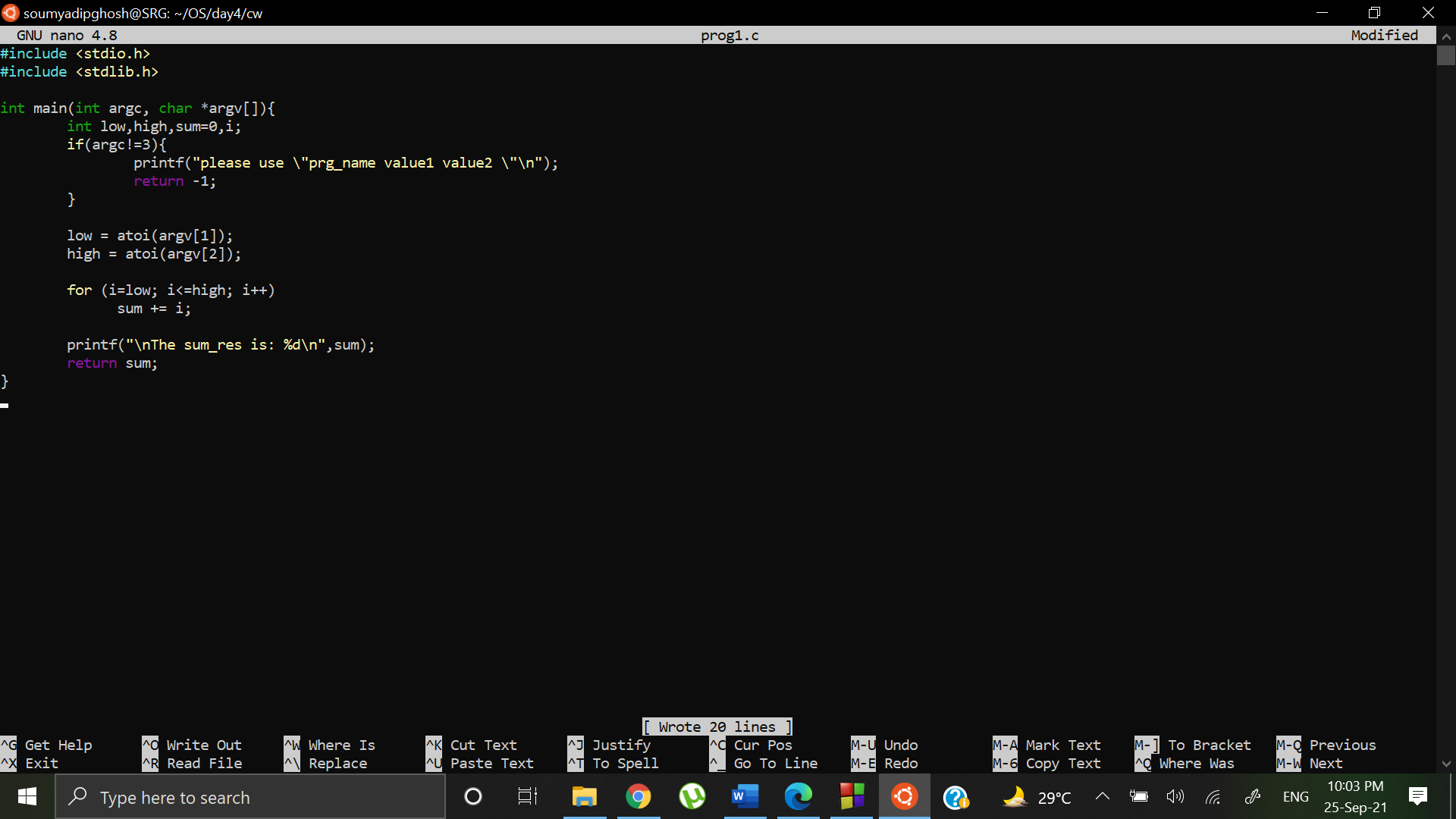
Question 1: a) Write a program [named demo.c] that accepts two integers (low, high) as command line argument. demo.c in turn should call two other programs pro1 and pro2. pro1 should calculate the summation of all integers between (low, high) as sum\_res. pro2 should evaluate whether sum\_res is prime or not.

Code:

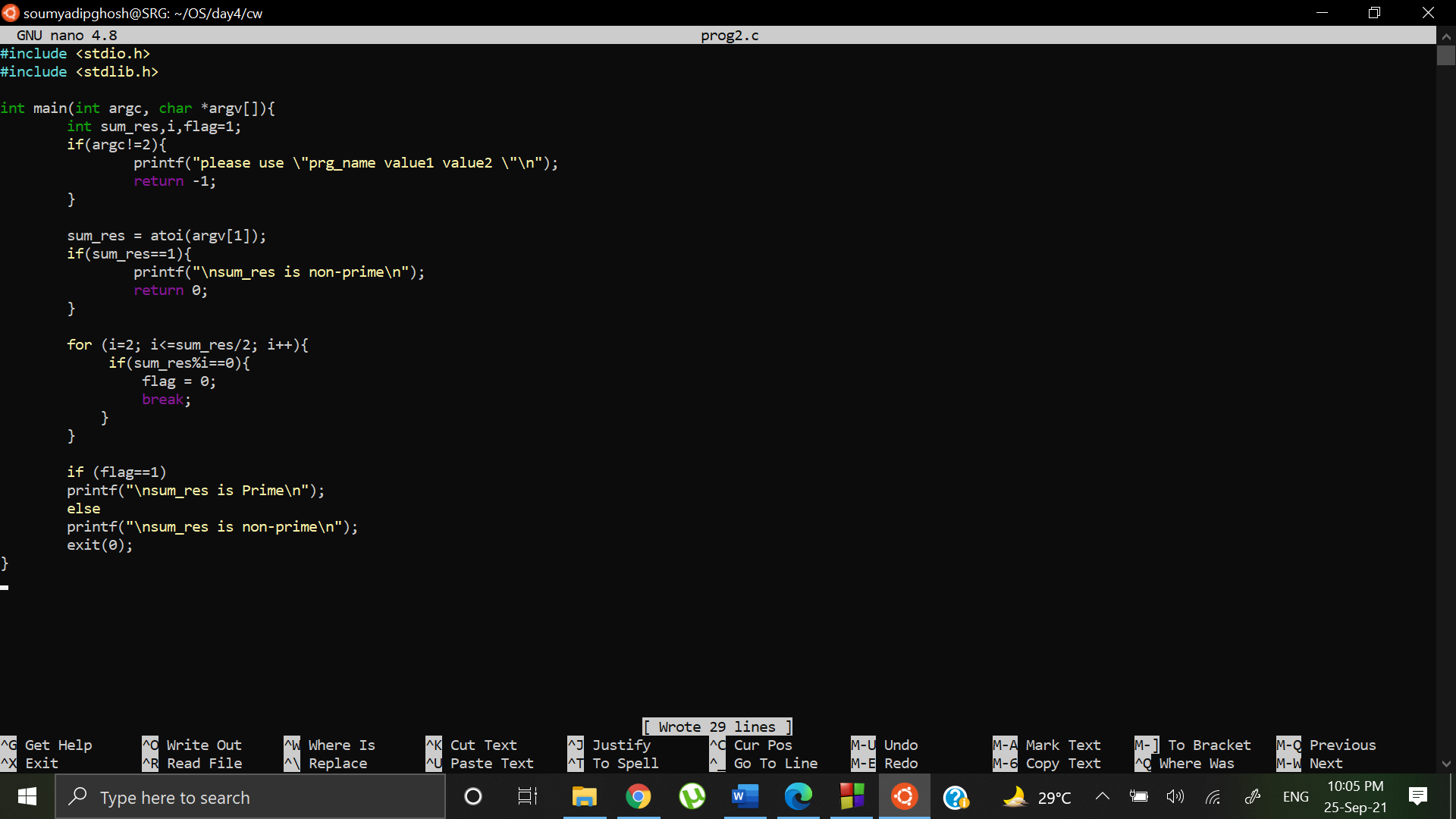
demo.c



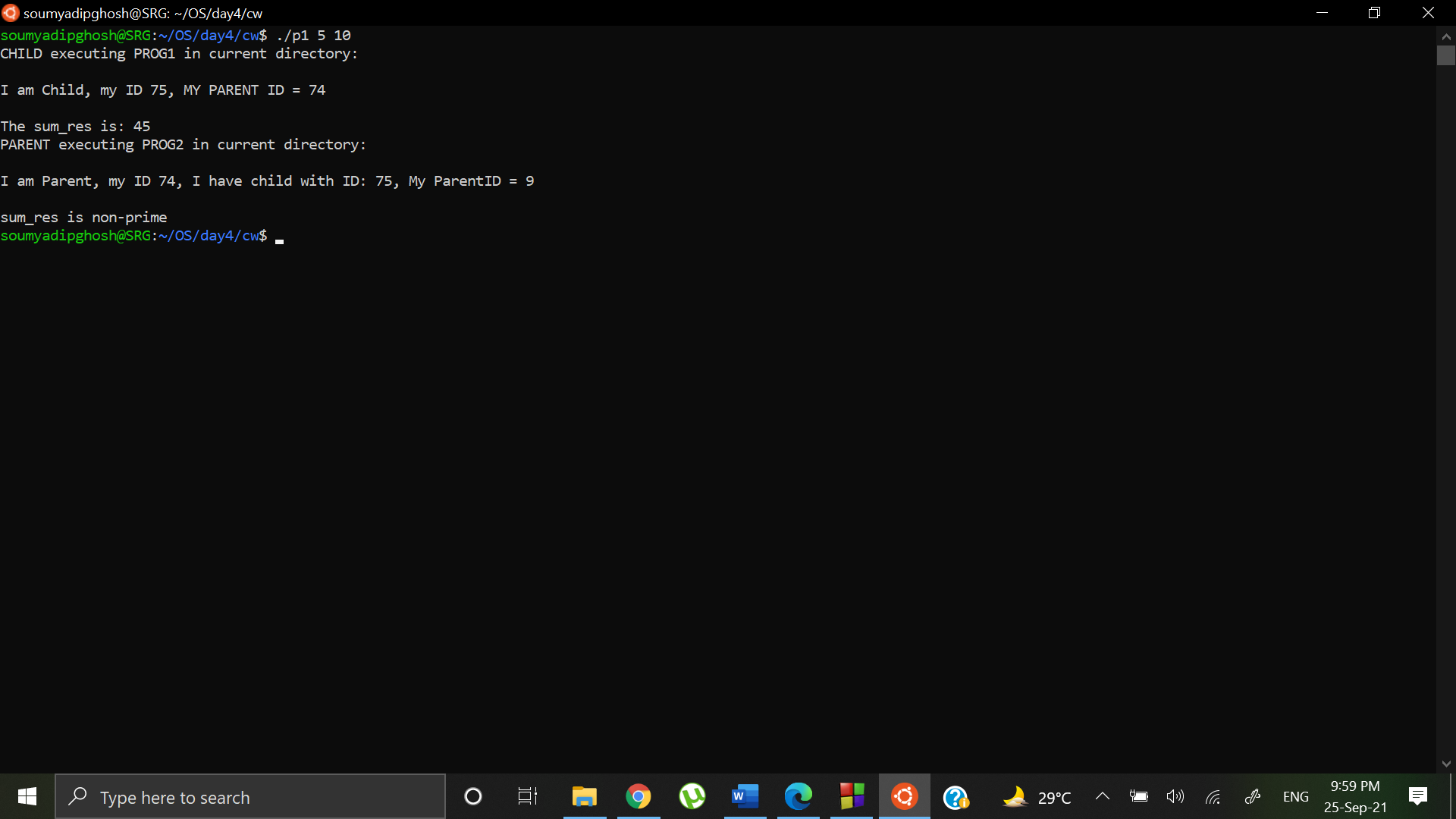
prog1.c



prog2.c



Output:



Question 2: Write a program to illustrate the Bounded Buffer or Producer/Consumer problem.

Code:

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

int mutex= 1;

int buff\_size = 5;

int product= 0;

void \*consumer(void \*arg){

pthread\_t \*thd\_id = (pthread\_t\*)arg;

pthread\_t sid;

if(product>0 && mutex==1)

{

mutex--;

sid = pthread\_self();

printf("\nHello from new thread %lu got %lu!",sid,\*thd\_id);

product--;

printf("\nProduct count: %d",product);

mutex++;

}

else{

if(mutex==0)

printf("\nProducer is executing Critical Section");

else

printf("\nBuffer Empty");

}

pthread\_exit(NULL);

}

void \*producer(void \*arg){

pthread\_t \*thd\_id = (pthread\_t\*)arg;

pthread\_t sid;

if(product!=buff\_size && mutex==1)

{

mutex--;

sid = pthread\_self();

printf("\nHello from new thread %lu got %lu!",sid,\*thd\_id);

product++;

printf("\nProduct count: %d",product);

mutex++;

}

else{

if(mutex==0)

printf("\nConsumer is executing Critical Section");

else

printf("\nBuffer Full");

}

pthread\_exit(NULL);

}

int main(){

int rc1, rc2;

pthread\_t tid1, tid2, sid;

int n= 100;

int ch;

while(n--){

ch=rand()%2;

if(ch==0)

{

if( (rc1=pthread\_create( &tid1, NULL, consumer, (void \*)&tid1)))

{

printf("Thread creation failed: %d\n", rc1);

exit (1);

}

}

else

{

if( (rc2=pthread\_create( &tid2, NULL, producer, (void \*)&tid2)))

{

printf("Thread creation failed: %d\n", rc2);

exit (1);

}

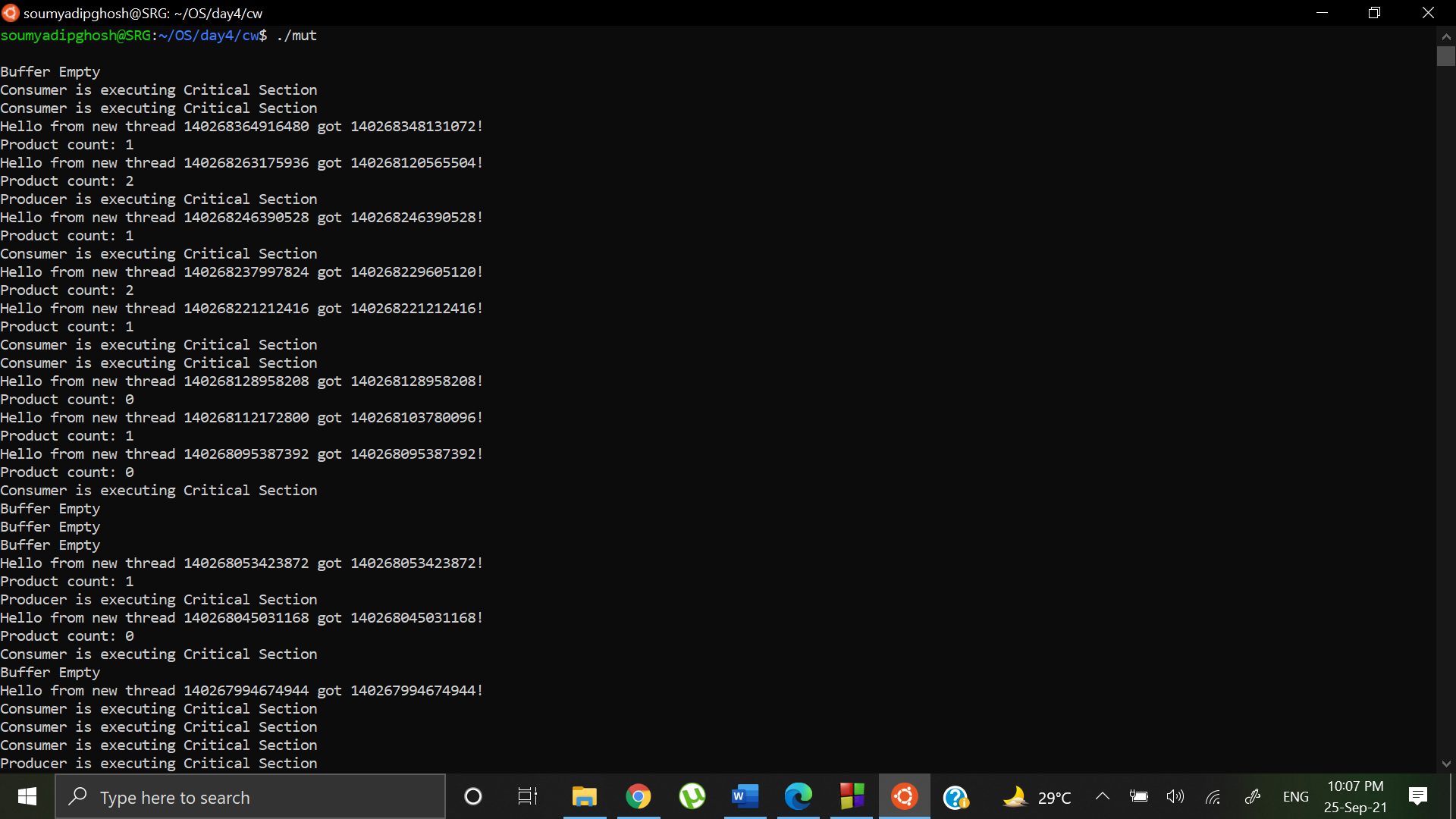
}

}

return 0;

}

Output:



Homework

Question: Write a multi-threaded program such that it creates 10 more threads. Each thread should print Hello from nth thread along with the argument received from the main thread. Each thread should return its own id and a unique value to main using pthread\_exit(). Main thread should be able to print this returned message identifying which thread ended within its own thread. Use mutex for synchronization and compare the results with the previous week’s assignment.

Question a:

Code:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#include <string.h>

pthread\_t tid[10];

char \*ret[10];

int mutex=1;

void \*test\_thread(void \*arg)

{

unsigned long i = 0;

pthread\_t id = pthread\_self();

if(pthread\_equal(id,tid[0]))

{

printf("\n1 created with ID: %lu", pthread\_self());

mutex++;

pthread\_exit(&id);

}

else if (pthread\_equal(id,tid[1]))

{

printf("\n2 created with ID: %lu", pthread\_self());

mutex++;

pthread\_exit(&id);

}

else if (pthread\_equal(id,tid[2]))

{

printf("\n3 created with ID: %lu", pthread\_self());

mutex++;

pthread\_exit(&id);

}

else if (pthread\_equal(id,tid[3]))

{

printf("\n4 created with ID: %lu", pthread\_self());

mutex++;

pthread\_exit(&id);

}

else if (pthread\_equal(id,tid[4]))

{

printf("\n5 created with ID: %lu", pthread\_self());

mutex++;

pthread\_exit(&id);

}

else if (pthread\_equal(id,tid[5]))

{

printf("\n6 created with ID: %lu", pthread\_self());

mutex++;

pthread\_exit(&id);

}

else if (pthread\_equal(id,tid[6]))

{

printf("\n7 created with ID: %lu", pthread\_self());

mutex++;

pthread\_exit(&id);

}

else if (pthread\_equal(id,tid[7]))

{

printf("\n8 created with ID: %lu", pthread\_self());

mutex++;

pthread\_exit(&id);

}

else if (pthread\_equal(id,tid[8]))

{

printf("\n9 created with ID: %lu", pthread\_self());

mutex++;

pthread\_exit(&id);

}

else

{

printf("\n10 created with ID: %lu\n\n", pthread\_self());

mutex++;

pthread\_exit(&id);

}

}

int main()

{

int i;

int \*ptr[10];

pthread\_t sid;

sid = pthread\_self();

printf("\nMAIN thread created with ID: %lu\n",sid);

for (i = 0; i < 10; i++) {

mutex--;

if (pthread\_create(&(tid[i]), NULL, test\_thread, (void \*)&tid)){

perror ("\npthread\_create() error");

exit(1);

}

while(mutex!=1){

}

}

for(int i=0;i<10;i++){

printf ("\n%d has ended with ID %lu ", i+1,tid[i]);

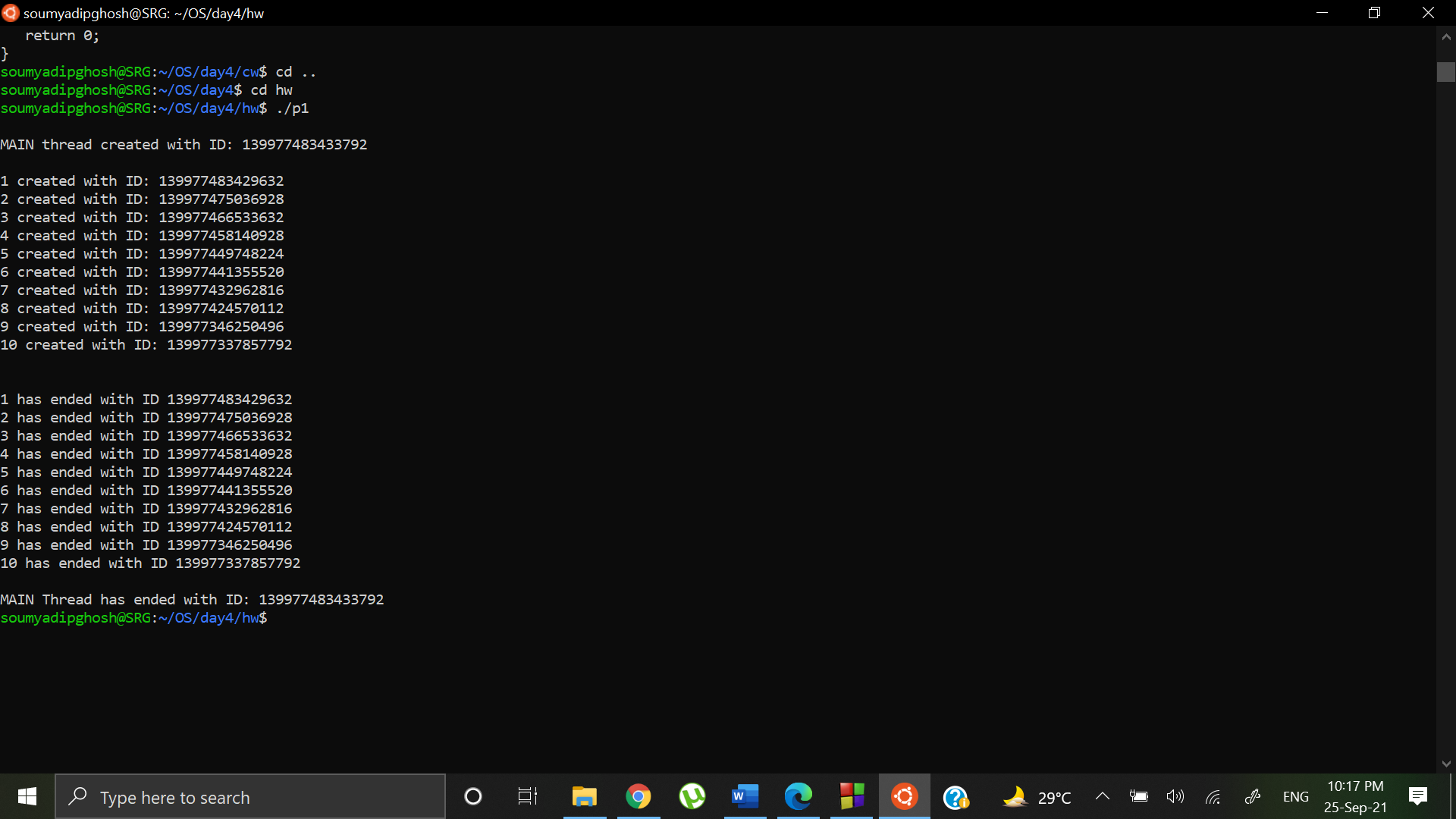
}

printf ("\n\nMAIN Thread has ended with ID: %lu\n" , pthread\_self());

return 0;

}

Output:



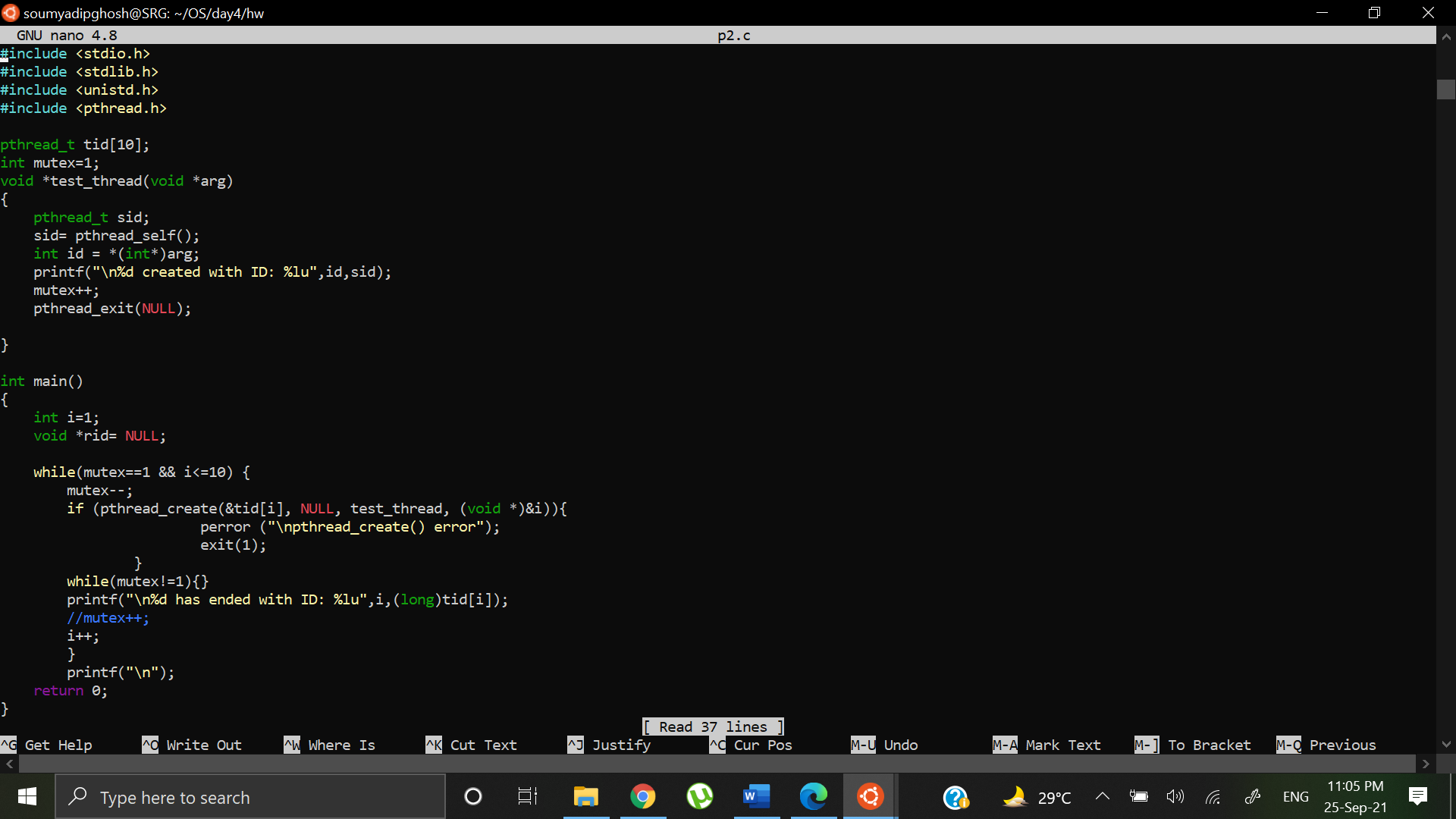
Comparison:

Week 3: In the for loop if thread was created, we were putting it to sleep for some time and meanwhile 10 threads were created. Then in the main block we were using pthread\_join() to confirm the termination of thread by returning process ID.

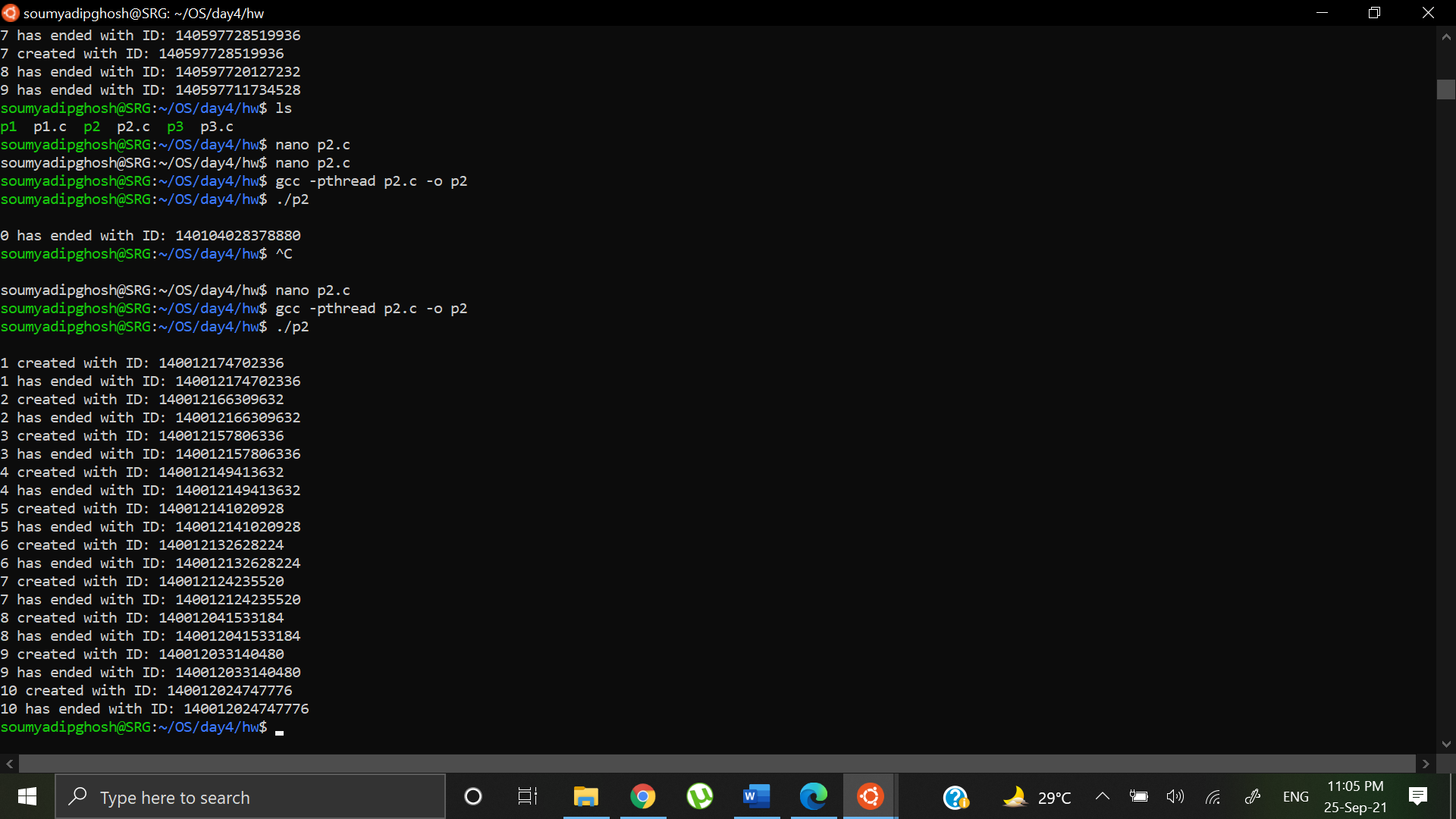
Week 4: We are taking a mutex and locking it on entering the for loop, mutex is unlocked when thread function is executed and hence unless mutex is unlocked no other thread can be created meanwhile. This keeps the order intact and then we are exiting the threads sequentially.

Question b.

Code:



Output:



Comparison:

Week 3:

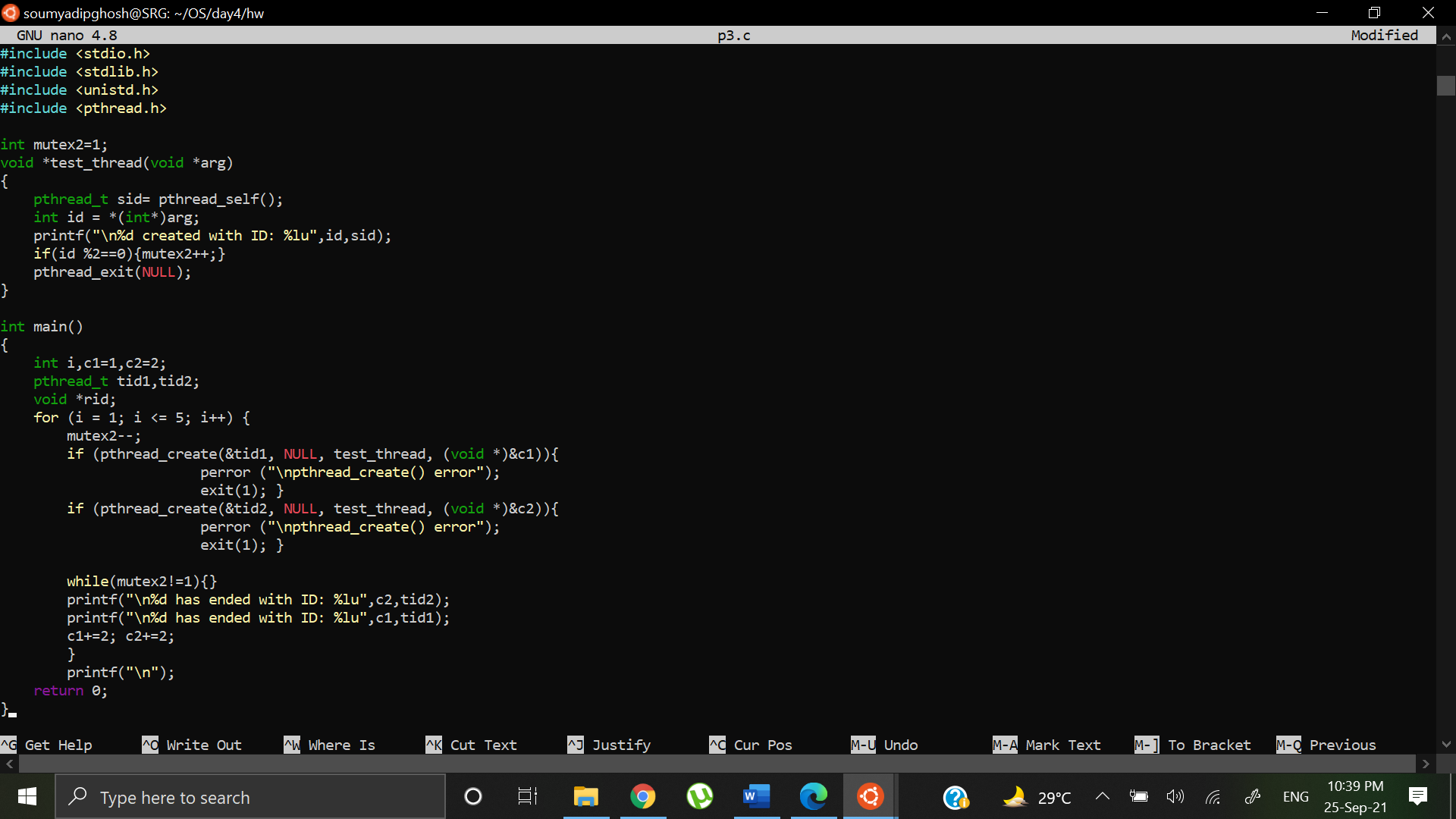
Created a thread and waited for the thread to terminate using pthread\_join and obtained the following order

Week 4:

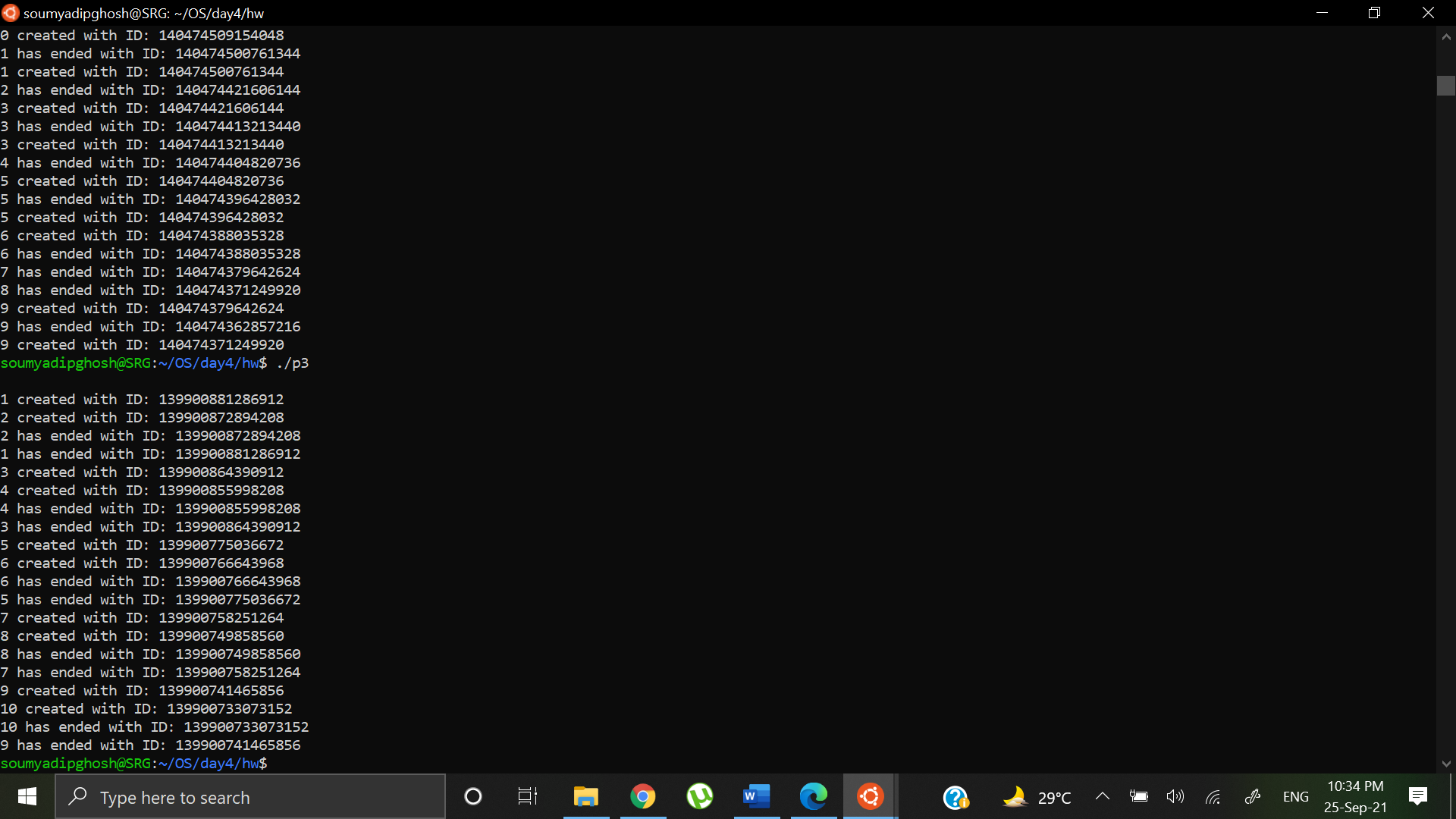
On entering the loop locked the mutex and waited for the thread to finish and then unlocked the mutex to obtain the desired output.

Question c.

Code:



Output:



Comparison:

Week 3:

Called a pair of threads simultaneously in order 1 and 2, then waited for second thread to exit using pthread\_join() and then printed it’s ID. Then executed the exit status of first thread.

Week 4:

A pair of threads are called, we are applying a mutex to the second thread of every pair and waiting for its execution to complete, then printing its return status.