**Operating System Lab**

**Lab Exercise #10**



**Session: 2021**

**Submitted by:**

**Wali Muhammad 2021-SE-39**

**Submitted to:**

**Mr. Waqas Ali**

Department of Computer Science, New Campus

**University of Engineering and Technology Lahore, Pakistan**

**Task 1**

**Description**

Write a simple program that create two threads sender and receiver. The sender thread sends a message string to receiver thread. This program uses condition variables to achieve synchronization.

**Solution**

**Code**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <string.h>

pthread\_cond\_t cv = PTHREAD\_COND\_INITIALIZER;

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

char \*message = NULL;

bool message\_ready = false;

void \*sender(void \*arg)

{

char \*msg = (char \*) arg;

message = malloc(strlen(msg) + 1);

strcpy(message, msg);

message\_ready = true;

pthread\_cond\_signal(&cv);

return NULL;

}

void \*receiver(void \*arg)

{

pthread\_mutex\_lock(&mutex);

while (!message\_ready)

{

pthread\_cond\_wait(&cv, &mutex);

}

printf("Receiver received message: %s\n", message);

pthread\_mutex\_unlock(&mutex);

return NULL;

}

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

{

if (argc < 2)

{

printf("Usage: %s message\n", argv[0]);

return 1;

}

//Creating 2 threads for sending and reciving

pthread\_t sender\_thread, receiver\_thread;

if (pthread\_create(&sender\_thread, NULL, sender, argv[1]) != 0) {

fprintf(stderr, "Error creating sender thread\n");

exit(1);

}

if (pthread\_create(&receiver\_thread, NULL, receiver, NULL) != 0) {

fprintf(stderr, "Error creating receiver thread\n");

exit(1);

}

if (pthread\_join(sender\_thread, NULL) != 0) {

fprintf(stderr, "Error joining sender thread\n");

exit(1);

}

if (pthread\_join(receiver\_thread, NULL) != 0) {

fprintf(stderr, "Error joining receiver thread\n");

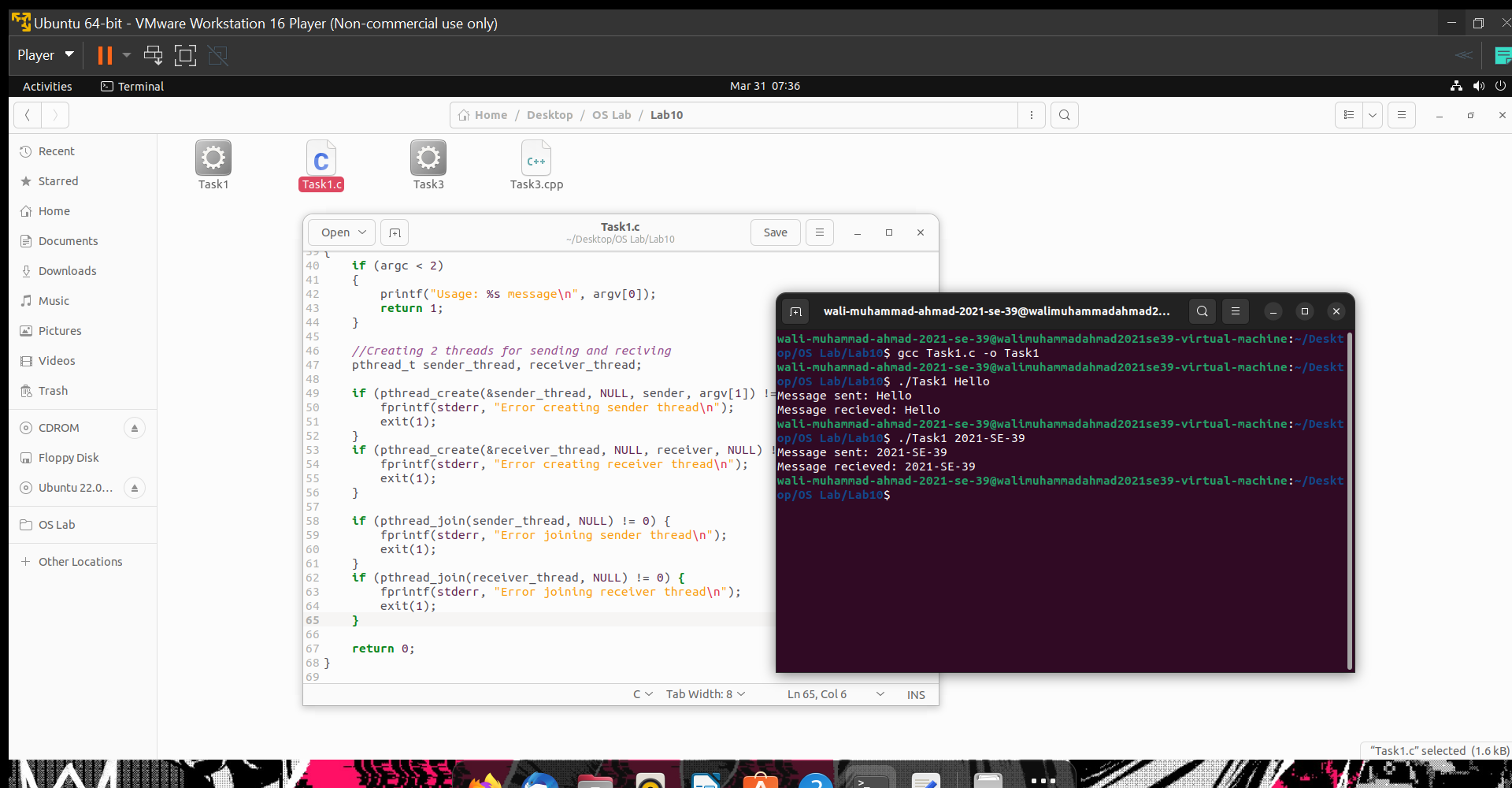
exit(1);

}

return 0;

}

**Code Execution Screenshot**

****

**Task 2**

**Description**

Suppose, you have a bank account with some balance amount Rs. 100. It is not enough to meet one-month expenditure, so you asked your dad, brother, and cousin to deposit some money in your account. Now, suppose your dad, brother and cousin all tried to deposit money at the same time. To serve their request, server process creates three threads, one for each to let them deposit money. Suppose, to deposit money, thread updates account balance as follows: -

1. OldBalance = your\_account.balance

2. NewBalance = OldBalance + depositedAmount

3. Your\_account.balance = NewBalance

Now in your dad, brother and cousin’s case they started depositing money at the same time so all threads take OldBalance = 100. All threads do calculation of NewBalance separately as thread(1) NewBalance = 100 + 10000 ——> dad deposited Rs. 10000. thread(2) NewBalance = 100 + 5000 ——–> brother deposited Rs. 5000. thread(3) NewBalance = 100 + 5500 ——–> brother deposited Rs. 5500. Now suppose thread(2) updates your account balance as Rs. 5100, and just after that thread(1) updates your account balance to Rs. 10100. During updating thread(1) and thread(3), thread three(3) also run on the same time and it will set new balance to Rs. 5600. So, your final account balance is Rs. 5600. So, in effect, the money deposited by your brother and dad hasn’t been credited into your account.

**Solution**

**Code**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <pthread.h>

#include <semaphore.h>

#define BALANCE 100

sem\_t semaphore;

pthread\_mutex\_t mutex;

pthread\_cond\_t cond;

int account\_balance = BALANCE;

void \*deposit(void \*arg)

{

int deposit\_amount = \*(int\*) arg;

sem\_wait(&semaphore);

pthread\_mutex\_lock(&mutex);

int old\_balance = account\_balance;

int new\_balance = old\_balance + deposit\_amount;

account\_balance = new\_balance;

printf("Deposited Rs. %d. New account balance: Rs. %d\n", deposit\_amount, account\_balance);

pthread\_cond\_broadcast(&cond);

pthread\_mutex\_unlock(&mutex);

sem\_post(&semaphore);

return NULL;

}

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

{

sem\_init(&semaphore, 0, 1);

pthread\_mutex\_init(&mutex, NULL);

pthread\_cond\_init(&cond, NULL);

int dad\_deposit = atoi(argv[1]);

int brother\_deposit = atoi(argv[2]);

int cousin\_deposit = atoi(argv[3]);

pthread\_t dad\_thread, brother\_thread, cousin\_thread;

pthread\_create(&dad\_thread, NULL, deposit, &dad\_deposit);

pthread\_create(&brother\_thread, NULL, deposit, &brother\_deposit);

pthread\_create(&cousin\_thread, NULL, deposit, &cousin\_deposit);

pthread\_cond\_wait(&cond, &mutex);

pthread\_cond\_wait(&cond, &mutex);

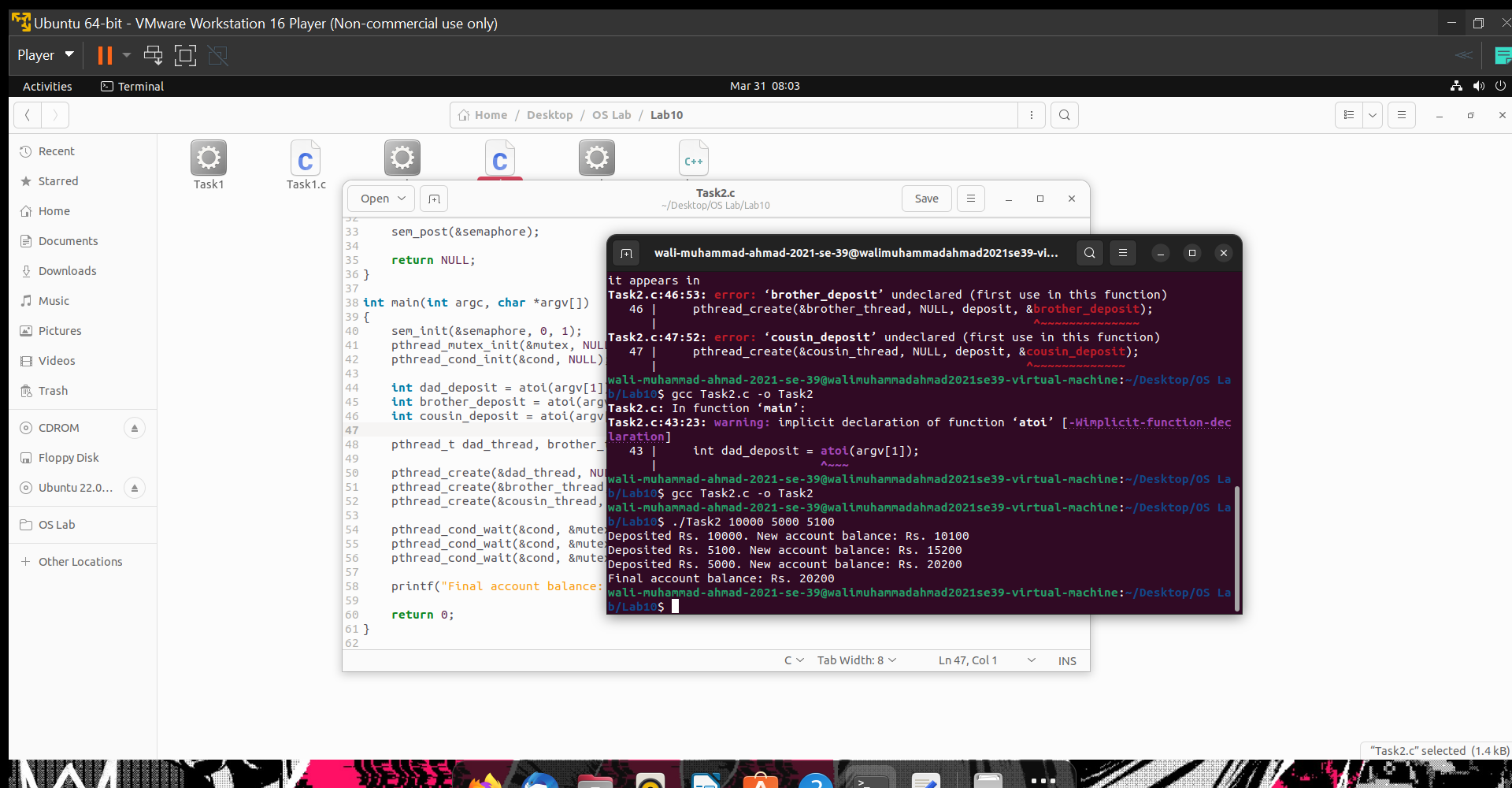
pthread\_cond\_wait(&cond, &mutex);

printf("Final account balance: Rs. %d\n", account\_balance);

return 0;

}

**Code Execution Screenshot**



**Task 3**

**Description**

Write a simple program that synchronizes threads by using semaphores.

**Solution**

**Code**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

sem\_t semaphore;

void \*thread\_function(void \*arg)

{

int thread\_id = \*(int\*) arg;

sem\_wait(&semaphore);

printf("Thread #%d entered critical section\n", thread\_id);

printf("Hello from Thread #%d\n", thread\_id);

printf("Thread #%d leaving critical section\n", thread\_id);

printf("\n");

sem\_post(&semaphore);

return NULL;

}

int main() {

pthread\_t thread1, thread2;

int thread1\_id = 1;

int thread2\_id = 2;

sem\_init(&semaphore, 0, 1);

pthread\_create(&thread1, NULL, thread\_function, &thread1\_id);

pthread\_create(&thread2, NULL, thread\_function, &thread2\_id);

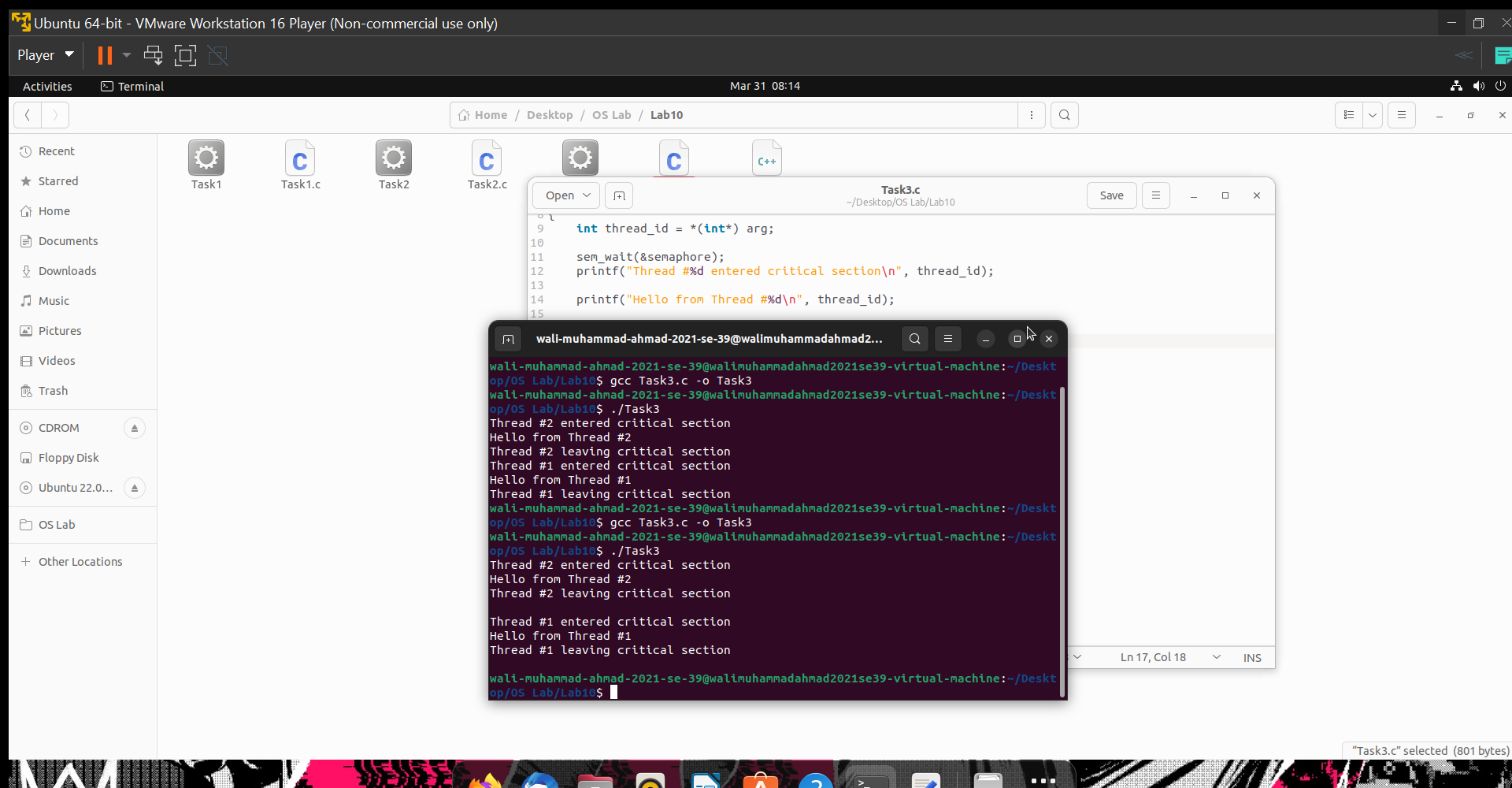
pthread\_join(thread1, NULL);

pthread\_join(thread2, NULL);

return 0;

}

**Code Execution Screenshot**



**THE END**