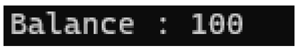
Experiment 7:

Code 1:

```
//SOLUTION OF RACE (AROUND) CONDITION
//USING SEMAPHORE VARIABLE 0
//MUTUAL EXCLUSION
//CONTINUING WITH EXP 7
//LINK -pthread

#include<stdio.h>
#include<unistd.h>
#include<pthread.h>
#include <stdlib.h>
#include <semaphore.h>



int balance = 100;
sem_t S;


void* withdraw(void* args);
void* deposit(void* args);

pthread_mutex_t mutex;





int main()
{
    pthread_mutex_init(&mutex, NULL);
    //sem_init(&S,0,1);//as we are using thread of same process so second arg
0


    pthread_t t1;
    pthread_t t2;

    pthread_create(&t1,NULL,withdraw,NULL);
    pthread_create(&t2,NULL,deposit,NULL);
    pthread_join(t1,NULL);
    pthread_join(t2,NULL);//now t2 will run seprately from any other thread
    printf("Balance : %d\n",balance);
}

void* withdraw(void* args)
    {

        //sem_wait(&S);
        pthread_mutex_lock(&mutex);
        int i = balance;
        i = i - 1;
        sleep(1);
        balance = i;
        //printf("Balance : %d\n",balance);
        pthread_mutex_unlock(&mutex);
        //sem_post(&S);
    }

void* deposit(void* args)
    {

        //sem_wait(&S);
        pthread_mutex_lock(&mutex);
        int i = balance;
        i = i + 1;
        sleep(1);
        balance = i;
        //printf("Balance : %d\n",balance);
        pthread_mutex_unlock(&mutex);
        //sem_post(&S);
    }
```

Output 1:



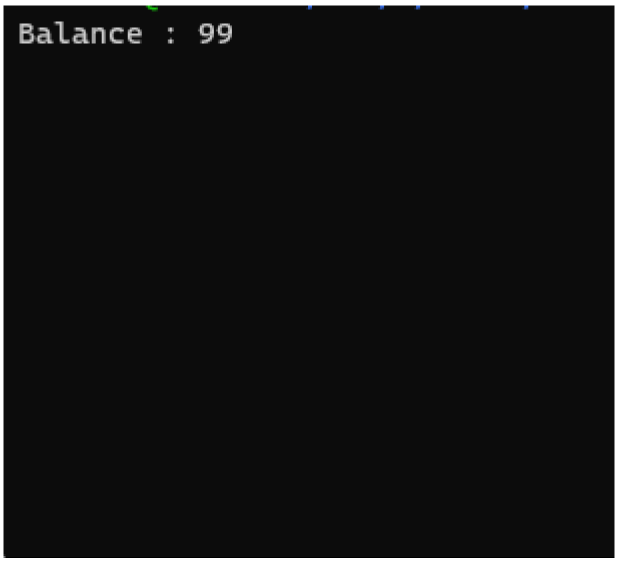
```
Balance : 100
```

Code 2:

```
// deadloc with two mutex
//SOLUTION OF RACE (AROUND) CONDITION
//USING SEMAPHORE VARIABLE 0
//MUTUAL EXCLUSION
//CONTINUING WITH EXP 7A
//LINK -pthread
```

```c
#include<stdio.h>
#include<unistd.h>
#include<pthread.h>
#include <stdlib.h>
#include <semaphore.h>




int balance = 100;
sem_t S;


void* withdraw(void* args);
void* deposit(void* args);

pthread_mutex_t mutex1;
pthread_mutex_t mutex2;





int main()
{
    pthread_mutex_init(&mutex1, NULL);
    pthread_mutex_init(&mutex2, NULL);
    //sem_init(&S,0,1);//as we are using thread of same process so second arg
0


    pthread_t t1;
    pthread_t t2;

    pthread_create(&t1,NULL,withdraw,NULL);
    pthread_create(&t2,NULL,deposit,NULL);

    pthread_join(t1,NULL);
    pthread_join(t2,NULL);//now t2 will run seprately from any other thread
    //printf("Balance : %d\n",balance);
}

void* withdraw(void* args)
    {

        //sem_wait(&S);
        pthread_mutex_lock(&mutex1);
        int i = balance;
        i = i - 1;
        sleep(1);
        balance = i;
        printf("Balance : %d\n",balance);
        pthread_mutex_unlock(&mutex2);
        //sem_post(&S);
    }

void* deposit(void* args)
    {

        //sem_wait(&S);
        pthread_mutex_lock(&mutex1);
        int i = balance;
        i = i + 1;
        sleep(1);
        balance = i;
        //printf("Balance : %d\n",balance);
        pthread_mutex_unlock(&mutex2);
        //sem_post(&S);
    }
```

Output 2: (Deadlock / Not Stopping)



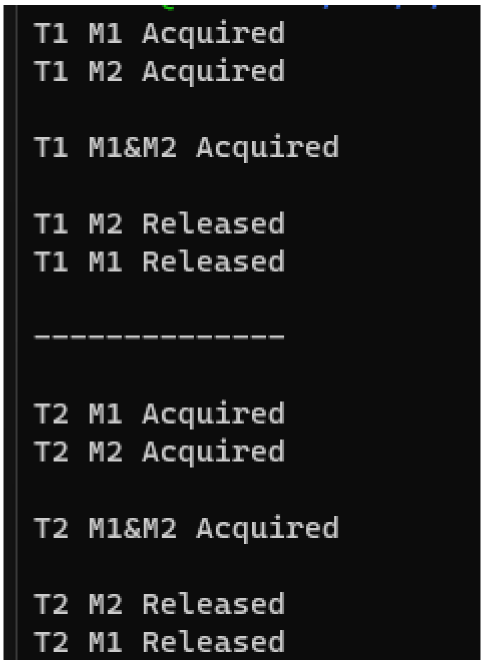
```
Balance : 99
```

Code 3:

```c
// deadloc with two mutex
//SOLUTION OF RACE (AROUND) CONDITION
//USING SEMAPHORE VARIABLE 0
//MUTUAL EXCLUSION
//CONTINUING WITH EXP 7A
//LINK -pthread
```

```c
#include<stdio.h>
#include<unistd.h>
#include<pthread.h>
#include <stdlib.h>
#include <semaphore.h>




int balance = 100;
sem_t S;


void* withdraw(void* args);
void* deposit(void* args);

pthread_mutex_t mutex1;
pthread_mutex_t mutex2;





int main()
{
    pthread_mutex_init(&mutex1, NULL);
    pthread_mutex_init(&mutex2, NULL);

    pthread_t t1;
    pthread_t t2;

    pthread_create(&t1,NULL,withdraw,NULL);
    pthread_create(&t2,NULL,deposit,NULL);

    pthread_join(t1,NULL);
    pthread_join(t2,NULL);
}

void* withdraw(void* args)
    {


        /*int i = balance;
        i = i - 1;
        sleep(1);
        balance = i;
        printf("Balance : %d\n",balance);*/
        pthread_mutex_lock(&mutex1);
        printf("T1 M1 Acquired\n");
        pthread_mutex_lock(&mutex2);
        printf("T1 M2 Acquired\n\n");

        sleep(1);
        printf("T1 M1&M2 Acquired\n\n");

        pthread_mutex_unlock(&mutex2);
        printf("T1 M2 Released\n");
        pthread_mutex_unlock(&mutex1);
        printf("T1 M1 Released\n");

        printf("\n--------------\n\n");

    }

void* deposit(void* args)
    {

        /*int i = balance;
        i = i + 1;
        sleep(1);
        balance = i;
        //printf("Balance : %d\n",balance);*/


        pthread_mutex_lock(&mutex1);
        printf("T2 M1 Acquired\n");
        pthread_mutex_lock(&mutex2);
        printf("T2 M2 Acquired\n\n");

        sleep(1);
        printf("T2 M1&M2 Acquired\n\n");

        pthread_mutex_unlock(&mutex2);
        printf("T2 M2 Released\n");
        pthread_mutex_unlock(&mutex1);
        printf("T2 M1 Released\n");



    }
```

Output 3:



```
T1 M1 Acquired
T1 M2 Acquired

T1 M1&M2 Acquired

T1 M2 Released
T1 M1 Released

--------------

T2 M1 Acquired
T2 M2 Acquired

T2 M1&M2 Acquired

T2 M2 Released
T2 M1 Released
```