**OS Lab**

**Mutex/Semaphore handling**

**Answer all Qs**

**MUTEX**

Using the file below, create the file **mutexb.c**, compile and run it

1. What should it output?
2. Do you always get the same output?

To compile it, type gcc -pthread -o mutexb mutexb.c

To run it, type ./mutexb

Hopefully you can see there is a race condition problem and we need a lock to protect the critical section

#include <stdio.h>

#include <pthread.h>

int sum = 0; //shared

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

static void GetLock()

{

    pthread\_mutex\_lock(&m);

}

static void ReleaseLock()

{

    pthread\_mutex\_unlock(&m);

}

void \*count(void \*param)   
{

    for (int i = 0; i < 10000000; i++)

{

        sum += 1;

    }

    return NULL;

}

int main()

{

    // create 2 threads

    pthread\_t tid1, tid2;

    pthread\_create(&tid1, NULL, count, NULL);

    pthread\_create(&tid2, NULL, count, NULL);

    //Wait for both threads to finish:

    pthread\_join(tid1, NULL);

    pthread\_join(tid2, NULL);

    // lets see what we got

    printf("ARRRRG sum is %d\n", sum);

    return 0;

}

so we need to add the mutex lock

1. In order to fix the problem, add the GetLock() and ReleaseLock() function calls to a suitable location (in a new file called mutexg.c).   
   NOTE: The GetLock and ReleaseLock functions/methods are ALREADY in mutexb.c  
   You just need to add a call to them in the correct place   
   (i.e. where the critical section is)
2. Why did it fix the problem?
3. What is the critical section?
4. What is the minimum critical section?

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

**SEMAPHORES**

Now let’s try this with a semaphore  
A semaphore is a bit like a mutex lock

It starts off life at value =1

When the lock is attained the value=0

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

**FIRST**, Copy mutexb.c to sem1.c

In this file called sem1.c

1. Replace the mutex calls with the code below ….. **in the correct locations**

sem\_t s;

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

sem\_wait(&s);

// Critical Section

sem\_post(&s);

you will also need to add the following lines to the file header section

#include <signal.h>

#include <semaphore.h>

#include <unistd.h>

remember to test the original mutexb.c and then fix it with the semaphore additions

i.e. create sem1.c to fix the problems.

Did it all work ok?

1. Now get familiar with threads by adding 2 more count threads tid4 and tid3

Retest with and without the locking to ensure you understand

NOTE: in this example we just used semaphores instead of a simple mutex to stop parallel access to critical sections. But semaphores are more powerful than that as we have COUNTING semaphores (not just a value of 1 and 0) which we can also use to signal between 2 threads.

**More complex example**

Implementation of simple stack using semaphores and mutexes

Consider the code below for adding data to and from a stack

#include <stdio.h>

#include <pthread.h>

#include <signal.h>

#include <semaphore.h>

#include <unistd.h>

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

int count = 0;

double values[10];

sem\_t sitems, sremains;

void init() {

sem\_init(&sitems, 0, 0); // 0 items

sem\_init(&sremains, 0, 10); // 10 spaces

}

double pop() {

// Wait until there's at least one item

sem\_wait(&sitems);

pthread\_mutex\_lock(&m); // CRITICAL SECTION

double v= values[--count];

pthread\_mutex\_unlock(&m);

sem\_post(&sremains); // Hey world, there's at least one space

return v;

}

void push(double v) {

// Wait until there's at least one space

sem\_wait(&sremains);

pthread\_mutex\_lock(&m); // CRITICAL SECTION

values[count++] = v;

pthread\_mutex\_unlock(&m);

sem\_post(&sitems); // Hey world, there's at least one item

}

Mutex is used to protect the count and access for the stack of 10 doubles (floating point numbers)

1. How many semaphores are there?

What are the semaphore(s) used for?

What type of semaphore(s) are these?

1. Create a c file called stack.c with code above.  
   Add a main (see below) to the file to test the stack.

Add code to this main to push and pop a value

int main()

{

double d;

init();

// add code here to push a value to stack

// add code here to pop a value

printf("value popped is ... %f \n\n", d);

}

1. push and pop a few more items
2. is the stack FIFO or LIFO?

So this is all very well but there is no concurrency here so we need to test using 2 threads

e.g. a producer and a consumer, or a pusher and popper

1. add this thread functionality to the file   
   you need to add the pusher and popper thread functions as below

and you need to create these threads in main   
(see mutexb.c or sem1.c examples for the thread syntax)

static void\* thepusher()

{

for(int i=0;i<1;i++)

{

push(i);

printf("pushing ... %f \n",(double)i);

sleep(1);

}

}

static void\* thepopper()

{

for(int i=0;i<1;i++)

{

printf("popped val is ... %f \n",pop());

sleep(1);

}

}

1. test the functions by adding a pusher and popper function call to main
2. change your main test code to
   1. make the pusher and popper do 20 not 1 items
   2. start the popper 3 seconds before the pusher  
      the popper should do nothing until the pusher starts to make items available

and then you should see them alternate operations

1. change your main test code to
   1. start the pusher 12 seconds before the popper
   2. you should see the pusher only be able to push 10 items until the popper the starts and then you should see them alternate operations

So we have seen semaphores control parallel access to a buffer/array and we have seen mutexes protect items. Wasn’t that fun? ☺