Networking Lab Assignment 4

First Readers-Writers Problem.

Albin Antony

11 February 2019

1 First Readers-Writers Problem

1.1 Aim

To implement First Readers-Writers Problem.

1.2 Theory

1.2.1 Introduction

It is possible to protect the shared data behind a mutual exclusion mutex, in which case no two threads can access the data at the same time. However, this solution is suboptimal, because it is possible that a reader R1 might have the lock, and then another reader R2 requests access. It would be foolish for R2 to wait until R1 was done before starting its own read operation; instead, R2. This is the motivation for the first readers-writers problem, in which the constraint is added that no reader shall be kept waiting if the share is currently opened for reading. This is also called readers-preference

1.2.2 Solution

In this solution of the readers/writers problem, the first reader must lock the resource (shared file) if such is available. Once the file is locked from writers, it may be used by many subsequent readers without having them to re-lock it again.

Before entering the critical section, every new reader must go through the entry section. However, there may only be a single reader in the entry section at a time. This is done to avoid race conditions on the readers (e.g. two readers increment the readcount at the same time, and both try to lock the resource, causing one reader to block). To accomplish this, every reader which enters the ¡ENTRY Section¿ will lock the ¡ENTRY Section¿ for themselves until they are done with it. At this point the readers are not locking the resource. They are only locking the entry section so no other reader can enter it while they are in it. Once the reader is done executing the entry section, it will unlock it by signalling the mutex. Signalling it is equivalent to: mutex.V() in the above code. Same is valid for the ¡EXIT Section¿. There can be no more than a single reader in the exit section at a time, therefore, every reader must claim and lock the Exit section for themselves before using it.

1.3 Algorithm

Algorithm 1 Algorithm First Readers Writers Problem

```
1 semaphore resource=1;
2 semaphore rmutex=1;
3 readcount=0;
4 procedure WRITER
5 resource.P(); . resource.P() is equivalent to wait(resource)
6 <CRITICAL Section>
7 <EXIT Section>
8 resource.V();
9 end procedure
10 procedure READER
11 rmutex.P(); . rmutex.P() is equivalent to wait(rmutex)
12 <CRITICAL Section>
13 readcount++;
14 if readcount == 1 then
15 resource.P();
16 end if
17 <EXIT CRITICAL Section>
18 rmutex.V(); . resource.V() is equivalent to signal(resource)
19 rmutex.P();
20 <CRITICAL Section>
21 readcount ;
22 if readcount == 0) then
23 resource.V();
24 end if
25 <EXIT CRITICAL Section>
26 rmutex.V(); . rmutex.V() is equivalent to signal(rmutex)
27 end procedure
```

1.4 Program

```
#include<iostream>
#include<semaphore.h>
#include<cstdlib>
#include<pthread.h>
#include<unistd.h>
#include<time.h>

using namespace std;
pthread_mutex_t rmutex;
pthread_mutex_t resources;
int readcount=0;

void *Writer(void* arg){
    long t=(long)arg;
    pthread_mutex_lock(&resources);
    printf("Thread %d writing....\n",t);
    printf("Thread %d finished writing\n",t);
    sleep(2);
    pthread_mutex_unlock(&resources);
    pthread_exit(NULL);
}

void *Reader(void* arg){
    long t=(long)arg;
    pthread_mutex_lock(&rmutex);
    readcount++;
    if(readcount==1){
        pthread_mutex_lock(&resources);
    }
    pthread_mutex_unlock(&rmutex);
    printf("Thread %d reading.....\n",t);
    sleep(3);
    printf("Thread %d finished reading\n",t);
    pthread_mutex_lock(&rmutex);
    readcount--;
    if(readcount==0){
        pthread_mutex_unlock(&resources);
    }
    pthread_mutex_unlock(&rmutex);
    pthread_exit(NULL);
}


int main(){
```

```
    pthread_t t1,t2,t3,t4;
    int rc;
    srand(time(0));
    for(int i=1;i<10;i++){
        if(rand()%2==0)
            rc=pthread_create(&t1,NULL,Reader,(void *)i);
        else
             rc=pthread_create(&t1,NULL,Writer,(void *)i);
    }

    pthread_exit(NULL);
}
```

1.5 Output

```
Thread 1 writing....
Thread 1 finished writing
Thread 2 reading.....
Thread 4 reading.....
Thread 5 reading.....
Thread 7 reading.....
Thread 8 reading.....
Thread 9 reading.....
Thread 2 finished reading
Thread 8 finished reading
Thread 5 finished reading
Thread 4 finished reading
Thread 7 finished reading
Thread 9 finished reading
Thread 3 writing....
Thread 3 finished writing
Thread 6 writing....
Thread 6 finished writing
```

1.6 Result

Implemented First Readers Writers Problem in C ,compiled on gcc 6.3.0 and executed on Debian 4.9 Kernel 4.9 and ouputs were verified.