Networking Lab Assignment 5

Second Readers-Writers problem

Albin Antony

20 February 2019

1 Second Readers-Writers problem

1.1 Aim

To Implement the Second Readers-Writers problem.

1.2 Theory

The first solution is suboptimal, because it is possible that a reader R1 might have the lock, a writer W be waiting for the lock, and then a reader R2 requests access. It would be unfair for R2 to jump in immediately, ahead of W; if that happened often enough, W would starve. Instead, W should start as soon as possible. This is the motivation for the second readers-writers problem, in which the constraint is added that no writer, once added to the queue, shall be kept waiting longer than absolutely necessary. This is also called writers-preference.

In this solution, preference is given to the writers. This is accomplished by forcing every reader to lock and release the readtry semaphore individually. The writers on the other hand don't need to lock it individually. Only the first writer will lock the readtry and then all subsequent writers can simply use the resource as it gets freed by the previous writer. The very last writer must release the readtry semaphore, thus opening the gate for readers to try reading.

1.3 Algorithm

Algorithm 1 Algorithm for second Readers Writers Problem

```
semaphore resource=1;
semaphore rmutex=1;
semaphore wmutex=1;
semaphore resource=1;
readcount=0;
writecount = 0;
procedure READER
<ENTRY Section>
readTry.P()
rmutex.P()
readcount++
if readcount == 1 then
resource.P()
end if
rmutex.V()
readTry.V()
<CRITICAL Section>
<EXIT Section>
rmutex.P()
 readcount
if readcount == 0 then
resource.V()
end if
rmutex.V()
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 wmutex;
pthread_mutex_t readTry;
pthread_mutex_t resources;
int readcount=0;
int writecount=0;

void *Writer(void* arg){
    long t=(long)arg;
    pthread_mutex_lock(&wmutex);
    writecount++;
    if (writecount==1)
        pthread_mutex_lock(&readTry);
    pthread_mutex_unlock(&wmutex);
    pthread_mutex_lock(&resources);
    printf("Thread %d writing....\n",t);
    printf("Thread %d finished writing\n",t);
    sleep(2);
    pthread_mutex_unlock(&resources);
    pthread_mutex_lock(&wmutex);
    writecount--;
    if(writecount==0)
        pthread_mutex_unlock(&readTry);
    pthread_mutex_unlock(&wmutex);
    pthread_exit(NULL);
}

void *Reader(void* arg){
    long t=(long)arg;
    pthread_mutex_lock(&readTry);
    pthread_mutex_lock(&rmutex);
    readcount++;
    if(readcount==1){
        pthread_mutex_lock(&resources);
    }
    pthread_mutex_unlock(&rmutex);
    pthread_mutex_unlock(&readTry);
    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 reading.....
Thread 2 reading.....
Thread 1 finished reading
Thread 2 finished reading
Thread 3 writing....
Thread 3 finished writing
Thread 4 writing....
Thread 4 finished writing
Thread 5 writing....
Thread 5 finished writing
Thread 8 writing....
Thread 8 finished writing
Thread 9 writing....
Thread 9 finished writing
Thread 6 reading.....
Thread 7 reading.....
Thread 7 finished reading
Thread 6 finished reading
```

1.6 Result

Implemented second Readers Writers problem in cpp compiled on g++ 6.3.0 and executed on Debian 4.9 Kernel 4.9 and outputs were verified.