

COLLEGE OF ENGINEERING, TRIVANDRUM

EXPERIMENT 4

Network Programming Lab

Author:
Alan Anto

Registration Number :
TVE16CS09

February 11, 2019

Contents

1	First Readers-Writers Problem	2
1.1	AIM	2
1.2	Theory	2
	1.2.1 Introduction	2
	1.2.2 Solution	2
1.3	Algorithm	3
1.4	Program	3
1.5	Output	5
1.6	Result	5

1 First Readers-Writers Problem

1.1 AIM

To implement first readers - writers problem.

1.2 Theory

1.2.1 Introduction

The readers-writers problem relates to an object such as a file that is shared between multiple processes. Some of these processes are readers i.e. they only want to read the data from the object and some of the processes are writers i.e. they want to write into the object. 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. If the data is already being used by a reader say R1, and another reader R2 requests for access, it'll be forced to wait until the R1 finishes the reading. It is not efficient to wait till this happens . So in first readers-writers problem a constraint will be added so that no reader shall be kept waiting if it is opened for reading by some other reader .

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 process will now need to enter the ENTRY Section. Once it enters the ENTRY Section, it locks the ENTRY Section. ENTRY section shall remain locked until they are done with it. They're locking the ENTRY Section so that no other user can enter the section while they are using it. It will stop the race condition of users. The same is valid for EXIT section as well. Once they finishes execution, the EXIT section will also be marked. Only one user is allowed in EXIT Section at a time.

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

The following program was written based on the algorithm.

```

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

sem_t mutex, writeblock;
int data = 0, rcount = 0;

void *reader(void *arg)

```

```
{
    int f;
    f = ((int) arg);
    sem_wait(&mutex);
    rcount = rcount + 1;
    if(rcount==1)
        sem_wait(&writeblock);
    sem_post(&mutex);
    printf("Reading by thread %i \n", arg);
    sleep(1);
    sem_wait(&mutex);
    rcount = rcount - 1;
    if(rcount==0)
        sem_post(&writeblock);
    sem_post(&mutex);

    printf("thread %i finished reading \n", arg);
}

void *writer(void *arg)
{
    int f;
    f = ((int) arg);
    sem_wait(&writeblock);
    data++;
    printf("Writing by thread %i \n", arg);
    sleep(1);
    sem_post(&writeblock);

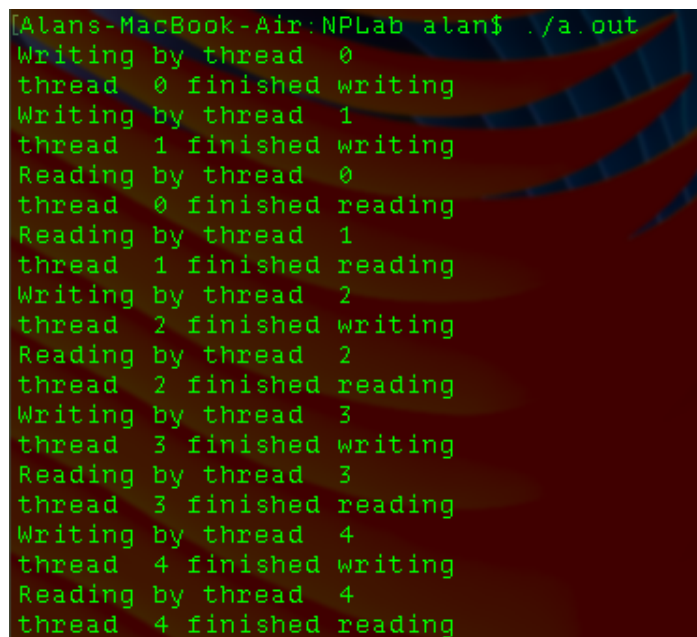
    printf("thread %i finished writing \n", arg);
}

int main()
{
    int i, b;
    pthread_t rtid[5], wtid[5];
    sem_init(&mutex, 0, 1);
    sem_init(&writeblock, 0, 1);
    for(i=0; i<=4; i++)
    {
```

```
        pthread_create(&wtid[i],NULL,writer,(void *)i);
        pthread_create(&rtid[i],NULL,reader,(void *)i);
    }
    for(i=0;i<=4;i++)
    {
        pthread_join(wtid[i],NULL);
        pthread_join(rtid[i],NULL);
    }
    return 0;
}
```

1.5 Output

The following output was obtained on running the program

A terminal window with a dark background and green text. The prompt is '[Alans-MacBook-Air:NPLab alan\$./a.out'. The output shows five threads (0-4) each performing a write and a read operation. The sequence of operations is interleaved: thread 0 writes, thread 1 writes, thread 0 reads, thread 1 reads, thread 2 writes, thread 2 reads, thread 3 writes, thread 3 reads, thread 4 writes, and thread 4 reads. Each thread's write and read operations are separated by a 'finished' status line.

```
[Alans-MacBook-Air:NPLab alan$ ./a.out
Writing by thread 0
thread 0 finished writing
Writing by thread 1
thread 1 finished writing
Reading by thread 0
thread 0 finished reading
Reading by thread 1
thread 1 finished reading
Writing by thread 2
thread 2 finished writing
Reading by thread 2
thread 2 finished reading
Writing by thread 3
thread 3 finished writing
Reading by thread 3
thread 3 finished reading
Writing by thread 4
thread 4 finished writing
Reading by thread 4
thread 4 finished reading
```

1.6 Result

The familiarization of first readers-writers problem was completed and the program executed successfully.