Readers-Writers Problem

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.

The readers-writers problem is used to manage synchronization so that there are no problems with the shared data.

Solution Outlines:

- One set of data is shared among a number of processes.
- Once a writer is ready, it performs its write. Only one writer may write at a time.
- If a process is writing, no other process can read it .
- If at least one reader is reading, no other process can write.
- Readers may not write and only read.

1. Solution Using Reader's Priority:

Here priority means, no reader should wait if the share data is currently opened for reading.

Program Code:

```
#include<iostream>
#include<semaphore.h>
#include<pthread.h>
#include<string>
#include<fstream>
using namespace std;

int rcnt  = 0;
sem_t wsem;
sem_t x;

void* reader(void *rno){

   sem_wait(&x);

   rcnt++;
   if(rcnt==1){
      sem_wait(&wsem);
   }
   sem_post(&x);

  //critical section started
```

```
ifstream fin;
    fin.open("readpro.txt",ios::in);  //Opening of file for reading
    string line;

     cout <<"\nReader"<<*((int *)rno)<<" is reading: \n";
    // Execute a loop until EOF (End of File)
    while(fin){
            getline(fin, line); // Reading a Line from File
        cout << line<<endl;
    }
    fin.close();  //closing of file
    //critical section ended

    sem_wait(&x);
    rcnt--;
    if(rcnt==0){
        sem_post(&wsem);
    }
    sem_post(&x);
}

void* writer(void* wno){

    sem_wait(&wsem);
     //critical section started
            string name;
            cout<<"Enter ur name to write "; //input from console
            getline(cin,name);
            //cout<<name;
            ofstream fout;
            fout.open("readpro.txt",ios::app); // appending to its existing contents.
            fout <<"Name= "<< name<< "\n";
            cout<<"\nwriter"<<*((int *)wno)<<" has written: "<<name<<endl;
            fout.close();
    //critical section ended
    sem_post(&wsem);
}
int main(){

    ofstream fout;
    fout.open("readpro.txt",ios::trunc); //deleting all conetent before write
    fout.close();

            pthread_t read[3], wrt[3]; //three threads for both reader and writer
            sem_init(&x, 0, 1); //initialisation of semaphore x
            sem_init(&wsem, 0, 1); //initialisation of semaphore wsem

    int a[3] = {1,2,3}; //Just used for numbering the Reader and Writer.

    for(int i = 0 ; i < 3 ; i++){
        pthread_create(&wrt[i],NULL,writer,(void *)&a[i]);
            pthread_create(&read[i],NULL,reader,(void *)&a[i]);
    }
```

```
for(int i = 0 ; i < 3 ; i++){
        pthread_join(read[i],NULL);
            pthread_join(wrt[i],NULL);
    }

    sem_destroy(&wsem);
    sem_destroy(&x);
}
```

<u>Decription of program:</u>

Here we used three variables: **x, wsem, rcnt** to implement solution

1. **semaphore** x;   // semaphore **x** is used to ensure mutual exclusion when **rcnt** is updated i.e. when any reader enters or exit from the critical section
2. **semaphore** wsem; // semaphore **wsem** is used by both readers and writers. the semaphore **wsem** is queued on both readers and writers in a manner such that preference is given to readers if writers are also there. Both are initially 1.
3. **int** rcnt;   // **rcnt** tells the number of processes performing read in the critical section, initially 0.

We used **6 pthreads (3 for each Reader and Writer process)**. For that we used c++ pthread library.

And **shared resourse is a file named as "readpro.txt".**

We also used fstream library to do in and out in file.

We are taking input from console to write to the file. We are appending each name that writer writes to the file. After restart of program, all file data vanishes, and 1st writer writes a fresh content.

Functions for sempahore :

– sem_wait() : decrements the semaphore value.

– sem_post() : increments the semaphore value.

These both functions are imported from semaphore library.

Writer Function:

1. Writer requests the entry to critical section.
2. If allowed i.e. wait() gives a true value, it enters and performs the write. If not allowed, it keeps on waiting.
3. It exits the critical section.

Pseudo code for writer:

```
// writer requests for critical section
    wait(wsem);

// performs the write

 // leaves the critical section
     signal(wsem);
```

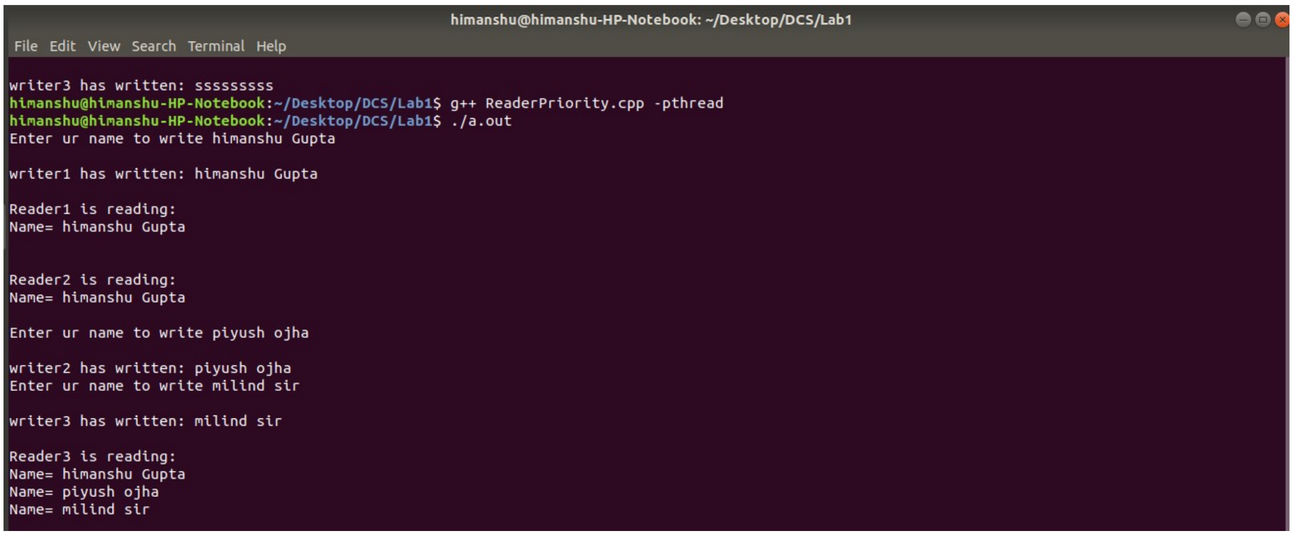
Reader Function:

1. Reader requests the entry to critical section.
2. If allowed:
    - it increments the count of number of readers inside the critical section. If this reader is the first reader entering, it locks the **wsem** semaphore to restrict the entry of writers if any reader is inside.
    - It then, signals x as any other reader is allowed to enter while others are already reading.
    - After performing reading, it exits the critical section. When exiting, it checks if no more reader is inside, it signals the semaphore “wsem” as now, writer can enter the critical section.
3. If not allowed, it keeps on waiting.
4. **Pseudo code for Reader:**

```
wait(x);
rcnt++;
if(rcnt == 1) {
    wait(wsem);
}
signal(x);
// Perform read operation

wait(x);
rcnt--;
if(rcnt == 0) {
    signal(wsrm);
}
signal(x);
```

Screenshot:



```
writer3 has written: sssssssss
himanshu@himanshu-HP-Notebook:~/Desktop/DCS/Lab1$ g++ ReaderPriority.cpp -pthread
himanshu@himanshu-HP-Notebook:~/Desktop/DCS/Lab1$ ./a.out
Enter ur name to write himanshu Gupta

writer1 has written: himanshu Gupta

Reader1 is reading:
Name= himanshu Gupta


Reader2 is reading:
Name= himanshu Gupta

Enter ur name to write piyush ojha

writer2 has written: piyush ojha
Enter ur name to write milind sir

writer3 has written: milind sir

Reader3 is reading:
Name= himanshu Gupta
Name= piyush ojha
Name= milind sir
```

2. <u>Solution Using Writer's Priority:</u>

<u>Program Code:</u>

```
#include<iostream>
#include<semaphore.h>
#include<pthread.h>
#include<string>
#include<fstream>
using namespace std;


int shareV = 1, rcnt = 0, wcnt = 0;
sem_t wsem;
sem_t x,y;
sem_t rsem;

void* reader(void *rno){
    sem_wait(&rsem);
            sem_wait(&x);

            rcnt++;
            if(rcnt==1){
            sem_wait(&wsem);
            }
            sem_post(&x);
    sem_post(&rsem);

    //critical section started

    ifstream fin;
    fin.open("writepro.txt");
    string line;

    cout <<"\nReader"<<*((int *)rno)<<" is reading: \n";
    // Execute a loop until EOF (End of File)
    while(fin){
            getline(fin, line);  // Reading a Line from File
        cout << line << endl;
    }

    fin.close();
    //critical section ended

    sem_wait(&x);
    rcnt--;
    if(rcnt==0){
        sem_post(&wsem);
    }
    sem_post(&x);

}
```

```
void* writer(void* wno){

    sem_wait(&y);
    wcnt++;
    if(wcnt == 1){
        sem_wait(&rsem);
    }
    sem_post(&y);

    //critical section started
    sem_wait(&wsem);

            shareV*=2;
            ofstream fout;
            fout.open("writepro.txt",ios::trunc); // ios::trunc mode delete all conetent before open
            fout << "share variable= " << shareV << "\n";
            cout<<"\nwriter"<<*((int *)wno)<<" has modified share variable to "<<shareV<<endl;
            fout.close();

    sem_post(&wsem);
    //critical section ended

    sem_wait(&y);
    wcnt--;
    if(wcnt == 0){
        sem_post(&rsem);
    }
    sem_post(&y);
}

int main(){

    pthread_t read[5], wrt[5]; //five threads for both reader and writer
    sem_init(&wsem, 0, 1);  //initialisation of semaphore wsem
    sem_init(&rsem, 0, 1);  //initialisation of semaphore rsem
    sem_init(&x, 0, 1);   //initialisation of semaphore x
    sem_init(&y, 0, 1);   //initialisation of semaphore y

    int a[5] = {1,2,3,4,5};
    for(int i = 0 ; i < 5 ; i++){
            pthread_create(&wrt[i],NULL,writer,(void *)&a[i]);
            pthread_create(&read[i],NULL,reader,(void *)&a[i]);
    }
    for(int i = 0 ; i < 5 ; i++){
        pthread_join(read[i],NULL);
        pthread_join(wrt[i],NULL);
    }
    sem_destroy(&wsem);
    sem_destroy(&rsem);
    sem_destroy(&x);
    sem_destroy(&y);
}
```

<u>Decription of program:</u>

Here we used seven variables: **x, y, wsem, rsem, rcnt, wcnt, shareV** to implement solution

1. **semaphore** x;   // semaphore **x** is used to ensure mutual exclusion when **rcnt** is updated i.e. when any reader enters or exit from the critical section, initially 1.
2. **semaphore** y;   // semaphore **y** is used to ensure mutual exclusion when w**cnt** is updated i.e. when any reader enters or exit from the critical section, initially 1.
3. **semaphore** wsem; // semaphore **wsem** is used by both readers and writers. the semaphore **wsem** is queued on both readers and writers in a manner such that preference is given to readers if writers are also there, initially 1.
4. **semaphore** rsem; // semaphore **rsem** is used by both readers and writers. the semaphore r**sem** is queued on both readers and writers in a manner such that preference is given to writers if readers are also there, initially 1.
5. **int** rcnt;   // **rcnt** tells the number of processes performing read in the critical section, initially 0.
6. **int** wcnt;   // w**cnt** tells the number of processes performing write in the critical section, initially 0.
7. **int** shareV;   // **shareV** is a shared variable between readers and writers, initially 1.

We used 10 **pthreads (5 for each Reader and Writer process)**. For that we used c++ pthread library.

And **shared resourse is a file named as "writepro.txt".**

We also used fstream library to do in and out in file.

Here we are updating shared variable by multiplying by 2 and writing to the file at each write process.

Functions for sempahore :

– sem_wait() : decrements the semaphore value.

– sem_post() : increments the semaphore value.

These both functions are imported from semaphore library.
At last, we are destroying the all semaphores.

Writer Function:

1. Writer requests the entry to critical section.
2. If allowed:
   - it increments the count of number of writers inside the critical section. If this writer is the first writer entering, it locks the **rsem** semaphore to restrict the entry of reader if any writer is inside.
   - It then, signals y as any other writing is allowed to enter while others are already writing.
   - After that it waits for other writers to complete the writing.
   - After performing writing, it exits the critical section. When exiting, it checks if no more writer is inside, it signals the semaphore "rsem" as now, writer can enter the critical section.

3. If not allowed, it keeps on waiting.

Pseudo code for writer:

```
wait(y);
       writecount++;
       if (writecount==1)
            wait(rsem);
     signal(y);
     wait(wsem);
           //do Writing
     signal(wsem);
     wait(y);
       writecount--;
       if (writecount==0)
            signal(rsem);
     signal(y);
```

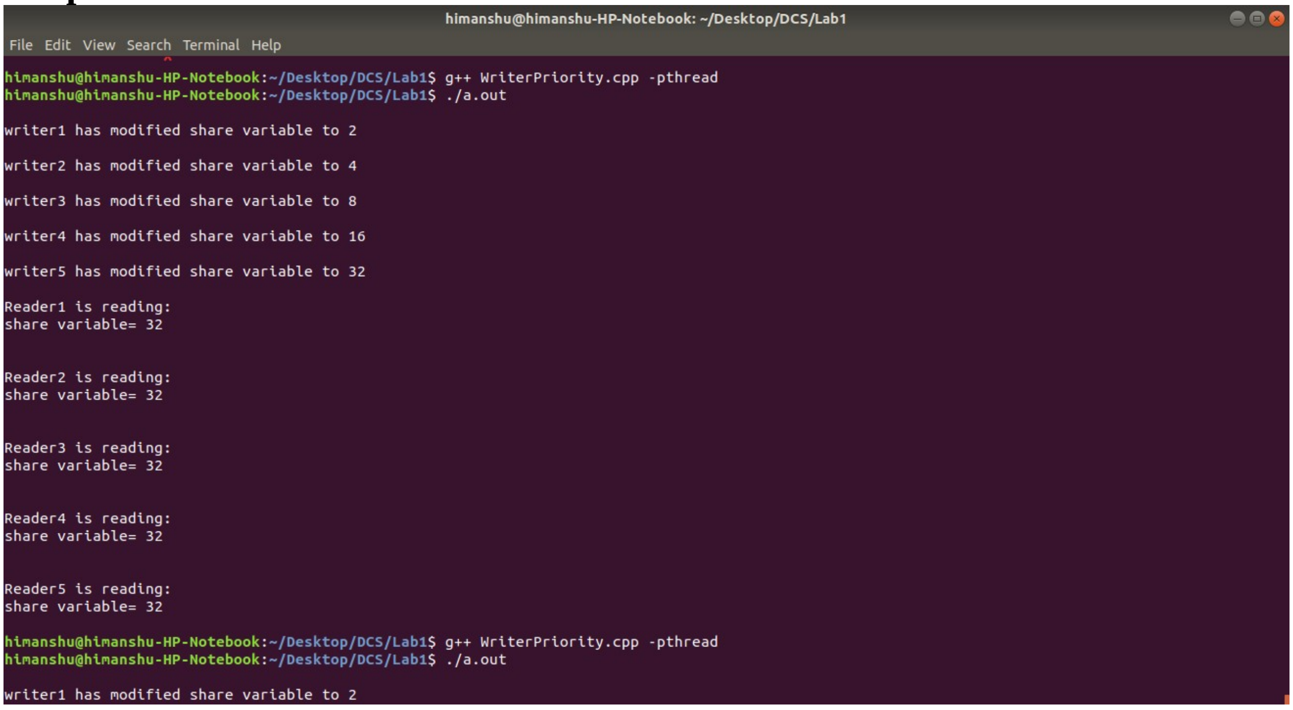
Reader Function:

1. Reader requests the entry to critical section.
2. If allowed:
   - it increments the count of number of readers inside the critical section. If this reader is the first reader entering, it locks the **wsem** semaphore to restrict the entry of writers if any reader is inside.
   - It then, signals x as any other reader is allowed to enter while others are already reading.
   - After performing reading, it exits the critical section. When exiting, it checks if no more reader is inside, it signals the semaphore **wsem** as now, writer can enter the critical section.
3. If not allowed, it keeps on waiting.

Pseudo code for Reader:

```
wait(rsem);
       wait(x);
        readcount++;
        if (readcount==1)
             wait(wsem);
       signal(x);
      signal(rsem);
             //do Reading;
     wait(x);
        readcount--;
        if (readcount==0)
             signal(wsem);
     signal(x);
```

Output Screenshot:


himanshu@himanshu-HP-Notebook: ~/Desktop/DCS/Lab1
File Edit View Search Terminal Help

```
himanshu@himanshu-HP-Notebook:~/Desktop/DCS/Lab1$ g++ WriterPriority.cpp -pthread
himanshu@himanshu-HP-Notebook:~/Desktop/DCS/Lab1$ ./a.out

writer1 has modified share variable to 2

writer2 has modified share variable to 4

writer3 has modified share variable to 8

writer4 has modified share variable to 16

writer5 has modified share variable to 32

Reader1 is reading:
share variable= 32


Reader2 is reading:
share variable= 32


Reader3 is reading:
share variable= 32


Reader4 is reading:
share variable= 32


Reader5 is reading:
share variable= 32

himanshu@himanshu-HP-Notebook:~/Desktop/DCS/Lab1$ g++ WriterPriority.cpp -pthread
himanshu@himanshu-HP-Notebook:~/Desktop/DCS/Lab1$ ./a.out

writer1 has modified share variable to 2
```

Submitted By
Piyush Ojha(BT17CSE084)
Himanshu Gupta(BT17CSE093)