Name:-Anupriya

Registration no:-11804641

Email address:-anupriya1738@gmail.com

GitHub Link: https://github.com/Anupriya1738/OS_ASSIGNMENT

Code:

```
#include<semaphore.h>

#include<stdio.h>

#include<pthread.h>

# include<bits/stdc++.h>

using namespace std;


void reader(void);

void writer(void);


int Read_c=0,Write_c;

semaphore x,y,z,rsem,wsem;

pthread_t r[4],w[3];


void reader( void i)

{


        wait(z);

        wait(rsem);

        wait(&x);

        Read_c++;

        if(Read_c==1)

            wait(wsem);


        wait(x);

        Read_c--;

        if(Read_c==0)

         signal(wsem);

           signal(x);
```

```
}

void *writer(void *i)
{
        cout << "\n\n writer-" << i << "is writing";
        sem_wait(&y);
        writecount++;
        if(writecount==1)
        sem_wait(&rsem);
        sem_post(&y);
        sem_wait(&wsem);

        sh_var=sh_var+5;
        sem_post(&wsem);
        sem_wait(&y);
        writecount--;
        if(writecount==0)
        sem_post(&rsem);
        sem_post(&y);
}

int main()
{
        sem_init(&x,0,1);
        sem_init(&wsem,0,1);
        sem_init(&y,0,1);
        sem_init(&z,0,1);
        sem_init(&rsem,0,1);

        pthread_create(&r[0],NULL,(void *)reader,(void *)0);
        pthread_create(&w[0],NULL,(void *)writer,(void *)0);
        pthread_create(&r[1],NULL,(void *)reader,(void *)1);
        pthread_create(&r[2],NULL,(void *)reader,(void *)2);
        pthread_create(&r[3],NULL,(void *)reader,(void *)3);
```

```
    pthread_create(&w[1],NULL,(void *)writer,(void *)3);
    pthread_create(&r[4],NULL,(void *)reader,(void *)4);

    pthread_join(r[0],NULL);
    pthread_join(w[0],NULL);
    pthread_join(r[1],NULL);
    pthread_join(r[2],NULL);
    pthread_join(r[3],NULL);
    pthread_join(w[1],NULL);
    pthread_join(r[4],NULL);

    return(0);
}
```

INTRODUCTION

EXPLAINATION OF THE GIVEN PROBLEM

- In an Operating System, we deal with various processes and these processes may use files that are present in the system. Basically, we perform two operations on a file i.e. read and write.
- This is a Readers and Writers problem in operating system.
- Readers Writers problem is very important in order to solve conflicts that arises when same object data is used by more than one and it is used for proper synchronization of the process
- For example consider a situation where a single object data has to be read by two or more Readers then its fine but if two or more Writers wants to write into the same object data then it will create a problem so in such cases we need to mak sure that writer has exclusive access to the object data.
- Here we will implement the reader writer algo for the following situation:-
- Several reader and writer processes wish to access a critical section. Because readers do not modify the critical section, multiple readers can access the critical section concurrently. However, a writer can access the critical section only when no other reader or writer is concurrently accessing it. We wish to implement locking/synchronization between readers and writers, while giving preference to writers, where no waiting writer should be kept waiting for longer than necessary. For example, suppose reader process R1 is actively reading. And a writer process W1 and reader process R2 arrive while R1 is reading. While it might be fine to allow R2 in, this could prolong the waiting time of W1 beyond the absolute minimum of waiting until R1 finishes. Therefore, if we want writer preference, R2 should not be allowed before W1.
- our goal is to write down pseudocode for read lock, read unlock, write lock, and write unlock functions that the processes should call, in order to realize read/write locks with writer preference.

GOALS:-

- 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
- Writer should be given preference over readers.

ALGORITHM

VARIABLES USED

Semaphore mutex, wrt;

int Read_c; //   Read_c tells the number of processes performing read in the critical section,

Functions for sempahore :

Wait: **Decrements** the value as soon as it would become non-negative(greater than or equal to 1)

iSignal: **Increments** the value as there is no more process blocked on the queue.

Writer process:

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 is given preference over Reader.

```
1)  do: writer requests for critical section
    wait(wrt);

2) performs the write

3) leaves the critical section
    signal(wrt);
    while(true)
```

Reader process:

1. Reader requests the entry to critical section.
2. If allowed:
   - It increments the count of number of readers inside the critical section.
   - It then, signals mutex as any other reader is allowed to enter while others are already reading. But only if writer doesn’t arrive otherwise writer is given preference over reader waiting outside.
3. If not allowed, it keeps on waiting.

ALGO

```
1) do: wait(mutex)[Reader wants to enter the critical section]
2) Read_c; ++; [The number of readers has now increased by 1]
3) signal(mutex); [other readers can read if there is no writer waiting and only
   readers reading]
4) wait(mutex); [a reader wants to leave]
5) readcnt--;
6)  if (Read_c; == 0) [that is, no reader is left in the critical section]
        signal(wrt); [ writers can enter]

7)signal(mutex); // reader leaves

    while(true);
```

COMPLEXITY

- Let number of entries in writer section be n and the complexity of
  writer is O(f(n)),
  then the complexity of the reader's exit section is Ω(log (n /f(n))).
- The asymptotic complexity of Readers Writers problem is logarithmic and
  that is Ω max(log n, log m)

CODE SNIPPET

```
#include<semaphore.h>
#include<stdio.h>
#include<pthread.h>
int  ReadCount, WriteCount = 0;
semaphore  x,y,z,rsem,wsem = 1;
void main()
{
   int p = fork();
   if(p)
   reader();
   else
    writer();
}

void reader(){
   while(1){
    wait(z);
    wait(rsem);
    wait(x);
    ReadCount++;
    if (ReadCount ==1)
    wait(wsem);
    signal(x);
    signal(rsem);
    signal(z);
    doReading();
    wait(x);
    signal(z);
    doReading();
    wait(x);
    ReadCount --;
    if (ReadCount ==0)
    signal(wsem);
    signal(x);
   }
}
void writer(){
  while(1){
    wait(y);
    WriteCount++;
    if (WriteCount ==1)
    wait(rsem);
    signal(y);
    wait(wsem);
    doWriting();
    signal(wsem);
    wait(y);
    WriteCount --;
    if (WriteCount ==0)
    signal(rsem);
    signal(y);
   }
}
```

BOUNDARY CONDITIONS

Writer will stop writing when the write semaphore has reached 0.

Reader will stop reading when the read semaphore has reached 0.

TEST CASES

1)Single reader R1 enters the critical section

>In this case since R1 is the first one to ask for critical section so it will be allocated with the resources

2)A reader R2 tries to enter the critical section when R1 was already there

>more than one readers are allowed to read the object at same time when no other writer process is there

3)First writer W1 enters the critical section when no other read or write process was there

>since resources are free right now so W1 will get the resource easily

4)W1 enters when R1 was already there

>here W1 will have to wait until R1 is finished and only after that it can enter critical section

5)W1 and R2 arrives when R1 is already reading the object data

>As this program is Writers preference program so after completion of R1 ,

first W1 is given the resource.