5 Second Readers-Writers Problem

5.1 Aim

Implement the Second Readers-Writers Problem

5.2 Theory

The solution to the first Readers-Writers problem is sub-optimal because if a reader is reading the shared resource,a writer thread comes to write the resource and when another reader comes, the writer process will be forced to wait until the reader process completes reading.This can lead to the writer thread starving.In the solution to the second readers-writers problem this is solved by adding a write mutex and readTry mutex which blocks readers when a writer thread is waiting.

5.3 Algorithm

Algorithm 1 ReaderCallBack Procedure

```
procedure READERCALLBACK(ID)                ▷ ID is the thread number
    wait(readmutex)                         ▷ readmutex is the mutex for readcount
    readcount++
    if readcount == 1 then
        wait(resource)
    end if
    signal(readmutex)
    CRITICAL SECTION
    wait(readmutex)
    readcount–
    if readcount == 0 then
        signal(resource)                    ▷ resource is the mutex for shared resource
    end if
    signal(readmutex)
    exitThread
end procedure
```

Algorithm 2 WriterCallBack Procedure

```
procedure WRITERCALLBACK($ID$)                    ▷ ID is the thread number
    wait(writemutex)                   ▷ writemutex is the mutex for writecount
    writecount++
    if  writecount == 1 then
        wait(readTry)
    end if
    signal(writemutex)
    wait(resource)
    CRITICAL SECTION
    signal(resource)
    wait(writemutex)
    writecount–
    if writecount == 0 then
        signal(readTry)          ▷ resrouce is the mutex for shared resource
    end if
    signal(writemutex)
    exitThread
end procedure
```

5.4 Code

```
#include<pthread.h>
#include<stdio.h>
#include<semaphore.h>
#include<cstdlib>
#include<time.h>
using namespace std;
sem_t resource,rmutex,wmutex,readTry;
int readcount = 0;
int writecount = 0;
int shared_resource = 0;
void *ReaderCallBack(void * thread_no){
    sem_wait(&readTry);
    sem_wait(&rmutex);
    readcount++;
    if(readcount == 1)
        sem_wait(&resource);
    sem_post(&rmutex);
    sem_post(&readTry);
    printf("Thread Number :%ld ,Shared Resource : %ld \n",(long)
    thread_no,shared_resource);
    sem_wait(&rmutex);
    readcount--;
    if(readcount == 0)
        sem_post(&resource);
    sem_post(&rmutex);
    pthread_exit(NULL);
}
void *WriterCallBack(void * thread_no){
    sem_wait(&wmutex);
```

```c
    writecount++;
    if(writecount == 1)
        sem_wait(&readTry);
    sem_post(&wmutex);
    sem_wait(&resource);
    shared_resource++;
    printf("Shared resource was modified by thread number : %ld \n",(long)thread_no);
    sem_post(&resource);
    sem_wait(&wmutex);
    writecount--;
    if(writecount == 0)
        sem_post(&readTry);
    sem_post(&wmutex);

    pthread_exit(NULL);
}
int main(){
    sem_init(&resource,0,1);
    sem_init(&rmutex,0,1);
    sem_init(&wmutex,0,1);
    sem_init(&readTry,0,1);
    srand (time(NULL));

    for(int i = 0;i<14;i++){
        int randomNo = rand()%100+1;
        if(randomNo <50){
            pthread_t reader;
            int iret;
            iret = pthread_create(&reader,NULL,ReaderCallBack,(void *)i);
            if(iret){
                return 1;
            }
            }
        else{
            pthread_t writer;
            int iret;
            iret = pthread_create(&writer,NULL,WriterCallBack,(void *)i);
            if(iret)
                return 1;

            }

    }

    return 0;
}
```

5.5 Output

```
Thread Number :0 ,Shared Resource : 0
Shared resource was modified by thread number : 1
```

```
Thread Number :3 ,Shared Resource : 1
Shared resource was modified by thread number : 2
Thread Number :4 ,Shared Resource : 2
Shared resource was modified by thread number : 5
Shared resource was modified by thread number : 6
Thread Number :7 ,Shared Resource : 4
Shared resource was modified by thread number : 8
Shared resource was modified by thread number : 9
Thread Number :10 ,Shared Resource : 6
Thread Number :11 ,Shared Resource : 6
Shared resource was modified by thread number : 12
Thread Number :13 ,Shared Resource : 7
```

5.6 Result

Implemented a program to demonstrate the second readers-writers problem in cpp and compiled using g++ version 8.2.1 on arch linux(kernel 4.20.6).The above output was obtained