STARVE-FREE READERS WRITER PROBLEM

OPERATING SYSTEMS

CSN - 232

JINENDRA VERMA
19114038
SUB BATCH O2

What is the reader writer problem?

A reader writer problem is a situation when multiple processes are trying to access(read) and edit(write) the same data structure or shared file simultaneously. In the classical solution of this problem, there is starvation for either reader or writer. So in a starve free solution of the problem to avoid this starvation, only one writer is allowed to access the critical section at any point of time and when there is no writer active then any number of readers can access the critical section.
In our solution we have used three semaphores namely **chance_queue,req** and **r_mutex.** chance_queue represents the chance of the next process to enter the critical section, req is the semaphore required to access the critical section and r_mutex is required to change the read_count variable.

Code explanation:

Initialization :

```
// set read count variable=0 which represents the number of active readers
int read_count = 0;
int data = 1;

// declaring semaphores
sem_t chance_queue,req,r_mutex;
```

Reader's code :

// ENTRY SECTION

wait for its chance

```
    sem_wait(&chance_queue);
```

requesting access to change read_count

```
    sem_wait(&r_mutex);
```

increase read_count by 1

```
    read_count++;
```

if the current reader is the first reader wait till Other writers release the resource semaphore

```
    if(read_count == 1)
        sem_wait(&req);
```

Release the chance_queue semaphore for other process

```
    sem_post(&chance_queue);
```

Release access to he read count

```
    sem_post(&r_mutex);

    //  CRITICAL SECTION
    printf("Reader %d: read data as %d\n",*((int *)rid),data);


    //  EXIT SECTION
```

Requesting access to change read_count

```
    sem_wait(&r_mutex);
```

Decreasing read count after reading is done

```
    read_count--;
```

If no other reader is remaining then release req semaphore

```
    if(read_count == 0)
```

```
        sem_post(&req);
```

Allow other readers to edit read_count

```
    sem_post(&r_mutex);
```

Writer’s code :

```
//  ENTRY SECTION
```

wait for its chance

```
    sem_wait(&chance_queue);
```

requesting access to change read_count

```
    sem_wait(&req);
```

Release the chance_queue semaphore for other process

```
    sem_post(&chance_queue);

    //  CRITICAL SECTION
        printf("Writer %d modified data from %d to
        %d\n",(*((int*)wid)),data,data+2);
        data += 2;



    //  EXIT SECTION
```

Release req semaphore for next process

```
    sem_post(&req);
```