Assignment - 5:Report

Operating systems - II:CS3523

CS20BTECH11037, Patnala Vikas

Design of Program

- The statement of the reader-writers problem is that when a writer accessing the critical section no other reader or writer should have to enter the critical section and multiple readers can access that critical section same time while readers accessing the critical section no writers can enter the critical section
- Here we will use two semaphore($rw\_mutex, mutex$) initiallized to one and one global variable($read\_count$)
- If a writer wants to enter the critical section then we will lock the critical section using the semaphore rw_mutex and then in the reader thread in the entry section we will check whether rw_mutex is free or not if free then we will acquire the lock and the readers acquire critical section after that when reading finishes reader thread will release the rw_mutex.
- As the above solutions there might be a change of starving the writer process hence we implement another algorithm which will be fair i.e. it not allow any of the process to starve in this process there will be another semaphore $queue$ which is initiallized to 1 and will be used to achieve fairness
- The Design of Reader-Writers is

```
writer(){
while (true) {
    wait(rw_mutex);
    ...
    /* writing is performed */
    ...
    signal(rw_mutex);
  }
}
reader(){
  while (true) {
    wait(mutex);
    read count++;
    if (read count == 1)
```

```
        wait(rw mutex);
    signal(mutex);
    ...
    /* reading is performed */
    ...
    wait(mutex);
    read count--;
    if (read count == 0)
        signal(rw mutex);
    signal(mutex);
  }
}
```

- The design of Fair Readers-Writers

```
writer(){
while (true) {
    wait(queue);
    wait(rw_mutex);
    signal(queue);
    ...
    /* writing is performed */
    ...
    signal(rw_mutex);
  }
}
reader(){
  while (true) {
    wait(queue);
    wait(mutex);
    read_count++;
    if (read_count == 1)
        wait(rw_mutex);
    signal(queue);
    signal(mutex);
    ...
    /* reading is performed */
    ...
    wait(mutex);
    read_count--;
    if (read_count == 0)
        signal(rw_mutex);
    signal(mutex);
  }
}
```

Algorithm

- Input will be taken by the main thread and this main thread. And then we will store that input variables globally because every thread should have to access

those variables.

- After that we will declare nw writers threads and nr readers threads and then we will create those reader and writer threads and run the functions $writer\ \&\ reader$ in these threads and we will pass the id of the thread that thread which is equal to $i + 1$
- After that each reader and writer threads will call the function separately and then here our logic starts
- Here we will restore the value of the thread id and use that value to identify each thread
- $default\_random\_engine$ is used to generate random number from $exponential\_distribution$ to simulate the CRITICAL and REMAINDER section. Parameter for both critical and remainder section were $\frac{1}{\mu_{CS}}$ and $\frac{1}{\mu_{Rem}}$ respectively.
- After that we will run each reader thread Critical section kr times and each writer thread Critical section kw as given in the question
- Here every time if a process request the CS then we will make use of the function $getSysTime$ which takes the $time\_t$ variable and converts it to our required format
- After that in each ENTRY SECTION of writer thread we use semaphore locking mechanism to stop every writer except one because only one writer will have to enter the critical section at a time
- In ENTRY SECTION of each reader thread we will just increment the value of $read\_count$ by 1 because and this increment will be done using the binary semaphore $mutex$ and then as multiple readers can enter the critical section at a time we give allow to multiple thread to read the critical section and then after the reading finishes we will decrement the value of $read\_count$ by 1 using the binary semaphore $mutex$ this is the basic logic behind both the algorithms. And also while a reader reading then writer must not enter the critical section and this can be implemented easily by checking in the reader that writer is currently doing any work or not if doing then we will wait until it finishes else we will acquire the lock rw_mutex
- Here we will output to the file using $fprintf$ because by using this function we will not get the outputs which are mixed for example if two threads request the CS at the same then we may have a chance of getting the values mixed using $fprintf$ will not mix the values instead output them individually

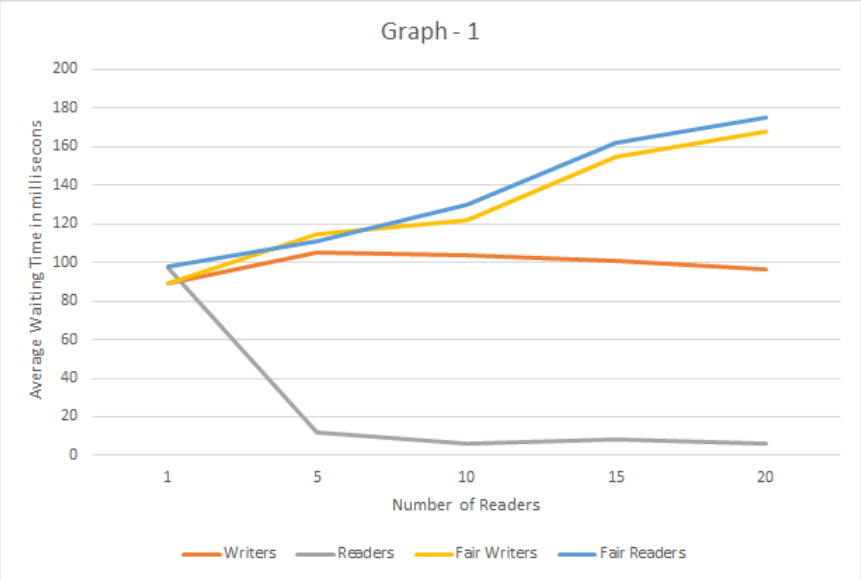
- In the above algorithms we will keep the thread which is executing the CS to sleep for certain amount of time which we will get using the random generator. After that every process will have to sleep for another certain amount of time when leaving the REMAINDER SECTION
- After that the control comes to the main function and then the main function will wait until the threads complete their execution

Graphs

Average Waiting Times with Constant Writers and varying Readers

In this graph, we measure the average time taken to enter the CS by reader and writer threads with a constant number of writers. Here we will vary the number of reader threads nr from 1 to 20 in the increments of 5 on the X-axis

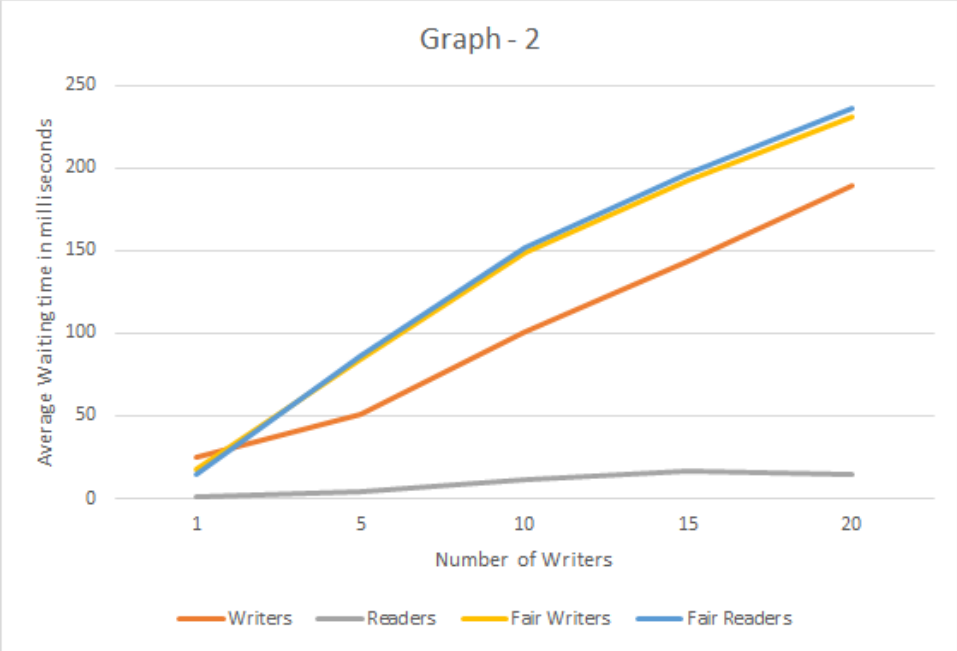
Other parameters used are $nw = 10, kr = kw = 10, \mu_{CS} = \mu_{Rem} = 10$



Average Waiting Times with Constant Readers and varying Writers

In this graph, we measure the average time taken to enter the CS by reader and writer threads with a constant number of readers. Here we will vary the number of writers threads nw from 1 to 20 in the increments of 5 on the X-axis

Other parameters used are $nr = 10, kr = kw = 10, \mu_{CS} = \mu_{Rem} = 10$



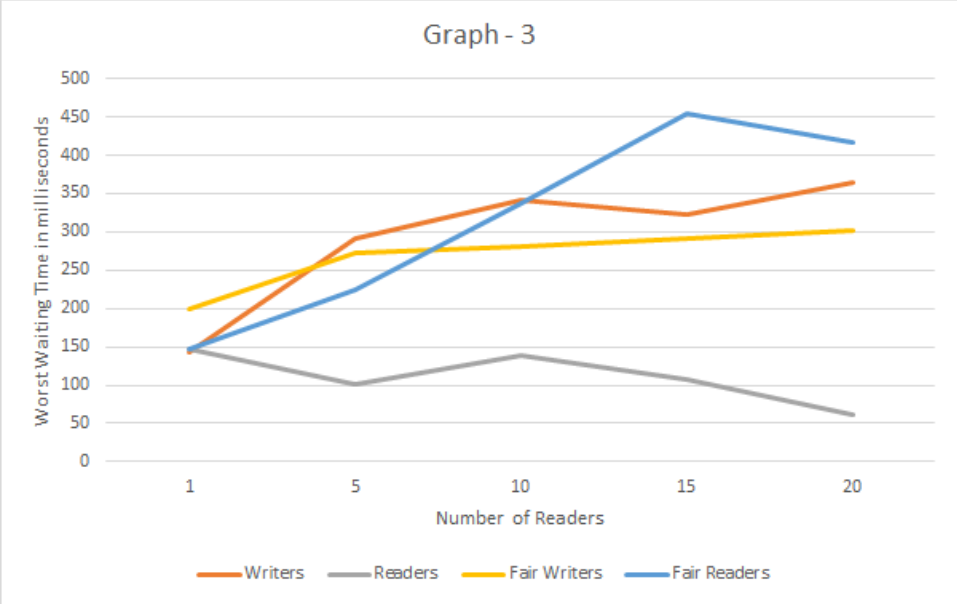
Analysis of Graph 1 and Graph 2

As we can see that Average Waiting time in the case of Fair writers and Fair Readers is more than the normal writers and readers this is because as the fair algorithms will provide fairness of writers and readers so here reader threads that are not waited in normal RW algorithm will have to wait so this will leads to increase in the waiting time of threads. We can also see that average waiting time of normal RW algorithm Readers is almost converges to 0 and making normal writers to have more average waiting time.

Worst-case Waiting Times with Constant Writers and varying Readers

In this graph, you measure the worst-case waiting time taken to enter the CS by reader and writer threads with a constant number of writers. Here we will vary the number of reader threads nr from 1 to 20 in the increments of 5 on the X-axis

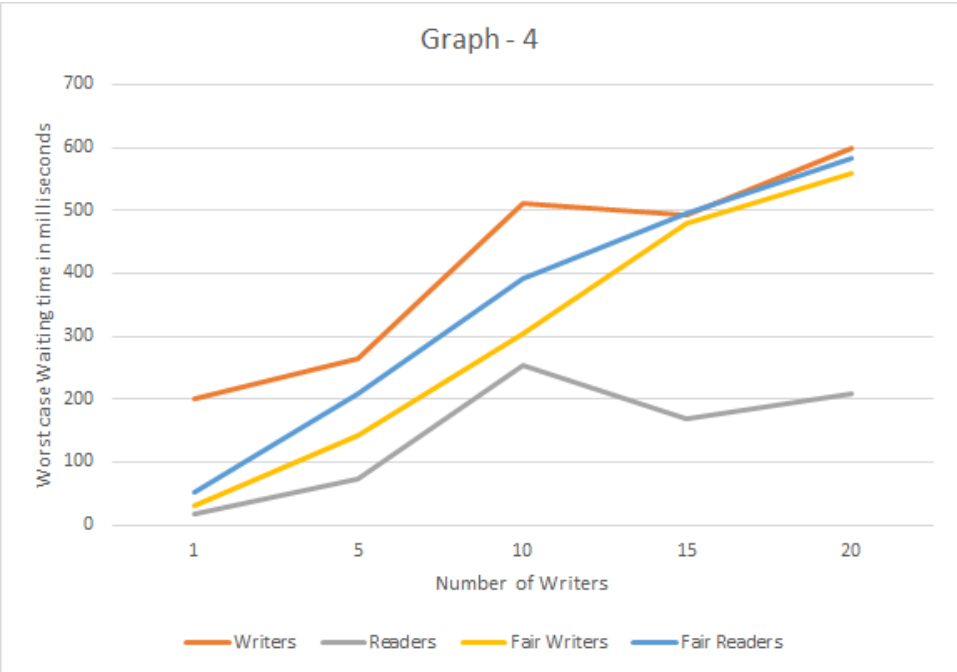
Other parameters used in these threads are $nr = 10, kr = kw = 10, \mu_{CS} = \mu_{Rem} = 10$



Worst-case Waiting Times with Constant Readers and varying Writers

In this graph, you measure the worst-case waiting time taken to enter the CS by reader and writer threads with a constant number of readers. Here we will vary the number of writers threads nw from 1 to 20 in the increments of 5 on the X-axis

Other parameters used in these threads are $nw = 10, kr = kw = 10, \mu_{CS} = \mu_{Rem} = 10$



Analysis of Graph 3 and Graph 4

As we can see that the writers in the normal RW algorithm are starving due to the high preference to the readers. So we can see in the graph that Fair RW writers are performing better than the normal RW writers. The worst case time of readers is more in case of Fair RW because in ensuring fairness to writers some readers may have to wait more time than it should in standard RW

Conclusion

- From the above graphs we can say that implementing Fair RW will ensures that no thread will have to starve but at the same time the cost of average waiting time will be increased on implementing the Fair RW algorithm
- For writers we can see that the decrease in the worst case waiting time which signifies that starving is decrease in the Fair RW algorithm for writers. And using Fair RW will increase the worst case waiting time of the readers which may lead to increase in the average waiting time of the readers.