Readers Writers Problem

Christian Carreras
CSC510 Kutztown University

Introduction

Readers Writers is an example of a synchronization problem

Several variants to the problem depending on preference

In this context, there are two processes: readers and writers

Accessing the critical section one process at a time is suboptimal

What is the solution?

Concurrency

A sharable resource open to multiple processes should allow concurrent access under certain conditions

Readers only read data, allow concurrency (simultaneous reading)

Writers edit data, ensure mutual-exclusion

Multiple readers or only one writer at a time in the critical section

If reading, ensure no writers. If writing, ensure no readers/writers

Variants

There are five main variants to the R/W Problem:

- Fair Preference
- Weak Reader Preference (WRP)
- Strong Reader Preference (SRP)
- Weak Writer Preference (WWP)
- Strong Writer Preference (SWP)

Fair Preference

Whoever arrives next goes in next

Uses a FIFO queue to determine order

Concurrent readers are allowed if they are lined up in the queue together

Since readers and writers are not organized, there are penalties to performance

Time grows more linearly when exposed to a higher percentage of writers compared to other variants

Pros:

- Concurrent readers
- No starvation
- Decent speed

Cons:

- Slowest compared to other variants
- No organization of R/W

FAIR READERS-WRITERS PROBLEM with Semaphores -

```
Semaphore ReaderCount, Mutex, Order;

Reader:                                   Writer:
P(Order)
P(ReaderCount)                                P(Order)
#Readers += 1                             P(Mutex)
if (#Readers == 1) P(Mutex)               V(Order)
V(Order)
V(ReaderCount)                                Critical Section (Write)
 Critical Section (Read)
P(ReaderCount)                                V(Mutex)
#Readers -= 1
if (#Readers == 0) V(Mutex)
V(ReaderCount)
```

http://faculty.kutztown.edu/spiegel/CSc552/Examples/ReadersWriters/Fair.Semaphore.htm

Weak Reader Preference

When a reader is in the critical section, writers must wait until all readers exit

Once a reader holds the critical section, readers can keep entering until they all finish

Basically fair preference without the queue

The organizing of readers leads to a significant time deduction in most cases

Pros:

- Concurrent readers
- Organizes readers for faster results
- Good speed

Cons:

- Possible starvation (writers)
- Performs worse than fair with a high amount of readers (80-100%)

```
writer() {
    resource.P();          //Lock the shared file for a writer

    <CRITICAL Section>
    // Writing is done

    <EXIT Section>
    resource.V();          //Release the shared file for use by other readers. Writers are allowed if there are no readers requesting it.
}

reader() {
    rmutex.P();           //Ensure that no other reader can execute the <Entry> section while you are in it
    <CRITICAL Section>
    readcount++;          //Indicate that you are a reader trying to enter the Critical Section
    if (readcount == 1)   //Checks if you are the first reader trying to enter CS
        resource.P();     //If you are the first reader, lock the resource from writers. Resource stays reserved for subsequent readers
    <EXIT CRITICAL Section>
    rmutex.V();           //Release

    // Do the Reading

    rmutex.P();           //Ensure that no other reader can execute the <Exit> section while you are in it
    <CRITICAL Section>
    readcount--;          //Indicate that you are no longer needing the shared resource. One less readers
    if (readcount == 0)   //Checks if you are the last (only) reader who is reading the shared file
        resource.V();     //If you are last reader, then you can unlock the resource. This makes it available to writers.
    <EXIT CRITICAL Section>
    rmutex.V();           //Release
}
```

https://en.wikipedia.org/wiki/Readers%E2%80%93writers_problem

Weak Writer Preference

If a writer is writing, then waiting writers go first (before readers)

Whoever holds the critical section will continue to do so until all of that type finish

Hits the goldilocks zone between performance and starvation risk

The organizing of both readers and writers leads to a tremendous time deduction

Critical section will “flip-flop” between R/W

Pros:

- Concurrent readers
- Organizes readers and writers
- Best speed

Cons:

- Possible starvation (both)

```
//READER
reader() {
<ENTRY Section>
  readTry.P();
  rmutex.P();
  readcount++;
  if (readcount == 1)
    resource.P();
  rmutex.V();
  readTry.V();

<CRITICAL Section>
//reading is performed

<EXIT Section>
  rmutex.P();
  readcount--;
  if (readcount == 0)
    resource.V();
  rmutex.V();
}
```

```
//WRITER
writer() {
<ENTRY Section>
  wmutex.P();
  writecount++;
  if (writecount == 1)
    readTry.P();
  wmutex.V();

<CRITICAL Section>
  resource.P();
  //writing is performed
  resource.V();

<EXIT Section>
  wmutex.P();
  writecount--;
  if (writecount == 0)
    readTry.V();
  wmutex.V();
}
```

https://en.wikipedia.org/wiki/Readers%E2%80%93writers_problem

Strong Reader Preference

If a writer is writing and another writer is waiting, an arriving reader will go next

Readers will only have to wait for one writer if they are not in control of the crit. section

Writers can only enter the crit. section if there is no active or waiting readers

Tries to get the most out of concurrency at the cost of starvation

Pros:

- Concurrent readers
- Groups readers together
- Good for unbalanced R/W ratios

Cons:

- Starvation is almost definite (writers)

```
procedure entry StartRead;
begin
 if Writing then begin
  ReaderWaiting:=True;
  ReaderGate.Wait
 end;
 Inc(NumReaders);
 if ReaderWaiting>=1 then begin
  Dec(ReaderWaiting);
  if ReaderWaiting=1 then
   NoWriter.Signal
  else ReaderGate.Signal;
 end
end;
```

```
procedure entry FinishRead;
begin
 Dec(NumReaders);
 if NumReaders=0 then
     NoReaders.Signal;
end;
```

```
procedure entry StartWrite;
begin
 Inc(TotalWriters);
 if TotalWriters>1 then
  NoWriter.Wait;
 if NumReaders>0 then
  NoReaders.Wait;
 Writing:=True;
end;
```

```
procedure entry FinishWrite;
begin
 Dec(TotalWriters);
 if ReaderWaiting>0 then begin
     Writing:=False;
     ReaderGate.Signal
 end
 else NoWriter.Signal;
end;
```

http://faculty.kutztown.edu/spiegel/CSc552/Examples/ReadersWriters/StrongReaderPref.Monitor.htm

Strong Writer Preference

Arriving writers go next

Readers will have to wait for no active or waiting writers to enter the crit. section

Readers still have concurrency until a new writer arrives, then it will have to give up the crit. section to the writers

Pros:

- Concurrent readers
- Groups writers together
- Good for unbalanced R/W ratios

Cons:

- Starvation is almost definite (readers)
- Hinders concurrency performance

```
Reader:

P(CGuard);
ar+=1;
if (aw==0)
{rr+=1;
 V(R); }
V(CGuard);
P(R);

Critical Section

P(CGuard);
rr-=1;
ar-=1;
if (rr==0)
 while (ww<aw)
 {ww+=1;
  V(W);}
V(CGuard);
```

```
Writers:

P(CGuard);
aw+=1;
if (rr==0)
{ww+=1;
V(W); }
V(CGuard);
P(W);
P(WGuard);

Critical Section

V(WGuard);
P(CGuard);
ww-=1;
aw-=1;
if (aw==0)
while (rr<ar)
{rr+=1;
  V(R); }
V(CGuard);
```

http://faculty.kutztown.edu/spiegel/CSc552/Examples/ReadersWriters/StrongWriterPref.Semaphore.htm

State Machine Implementation

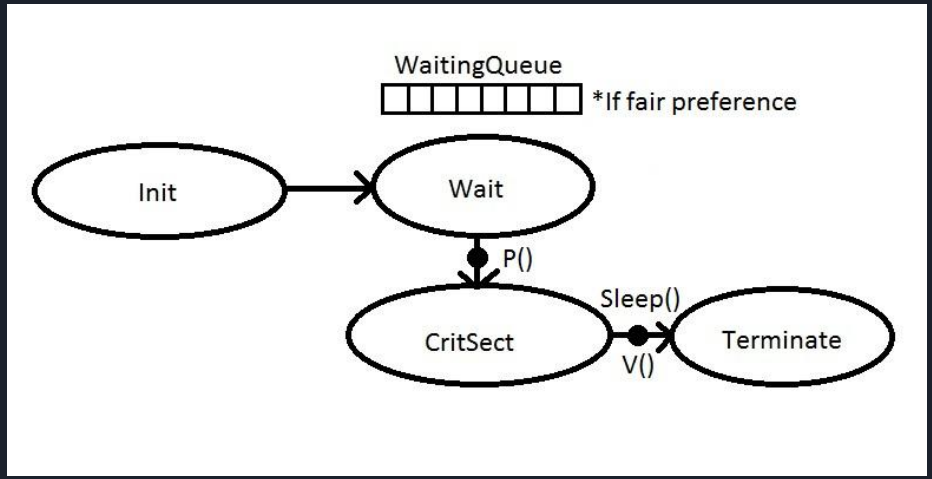
Ten threads are assigned a reader or writer role pseudo-randomly

Each thread loops through the state machine one hundred times

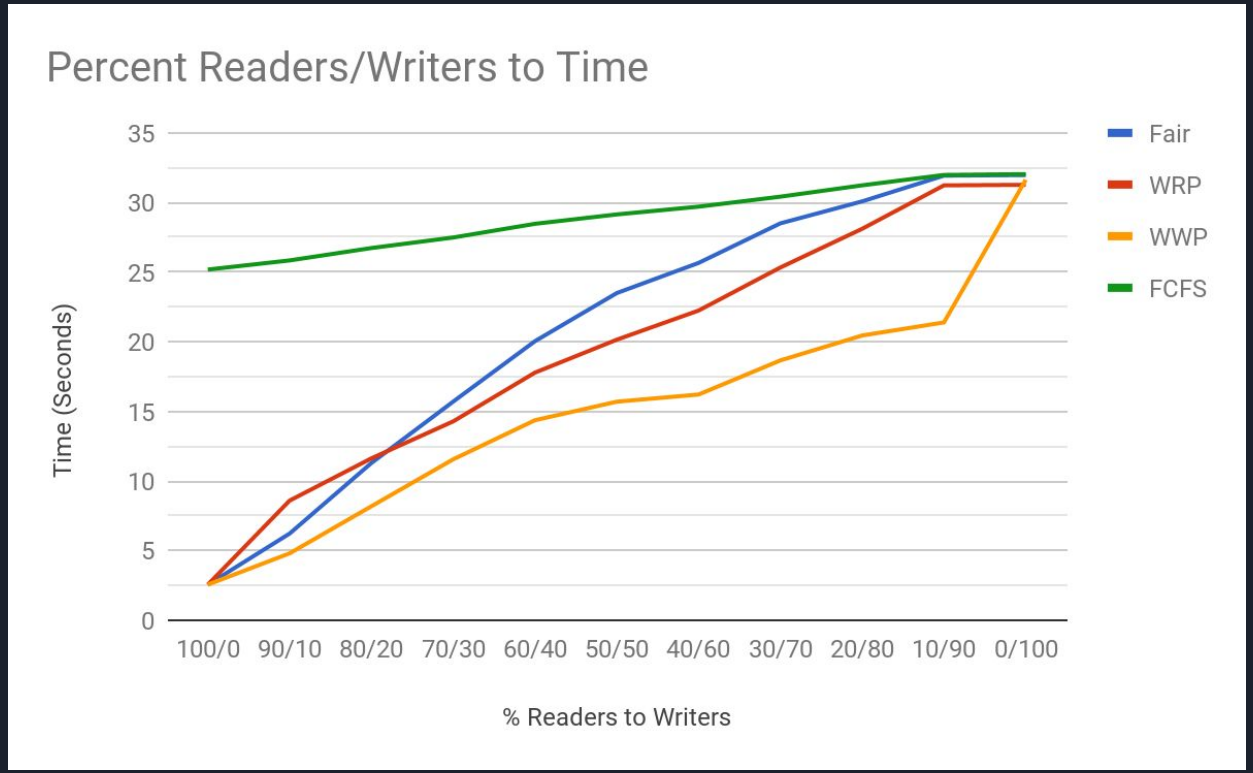
Threads wait to acquire the mutex in the wait state to go to the crit. section state

Threads sleep in the critical section

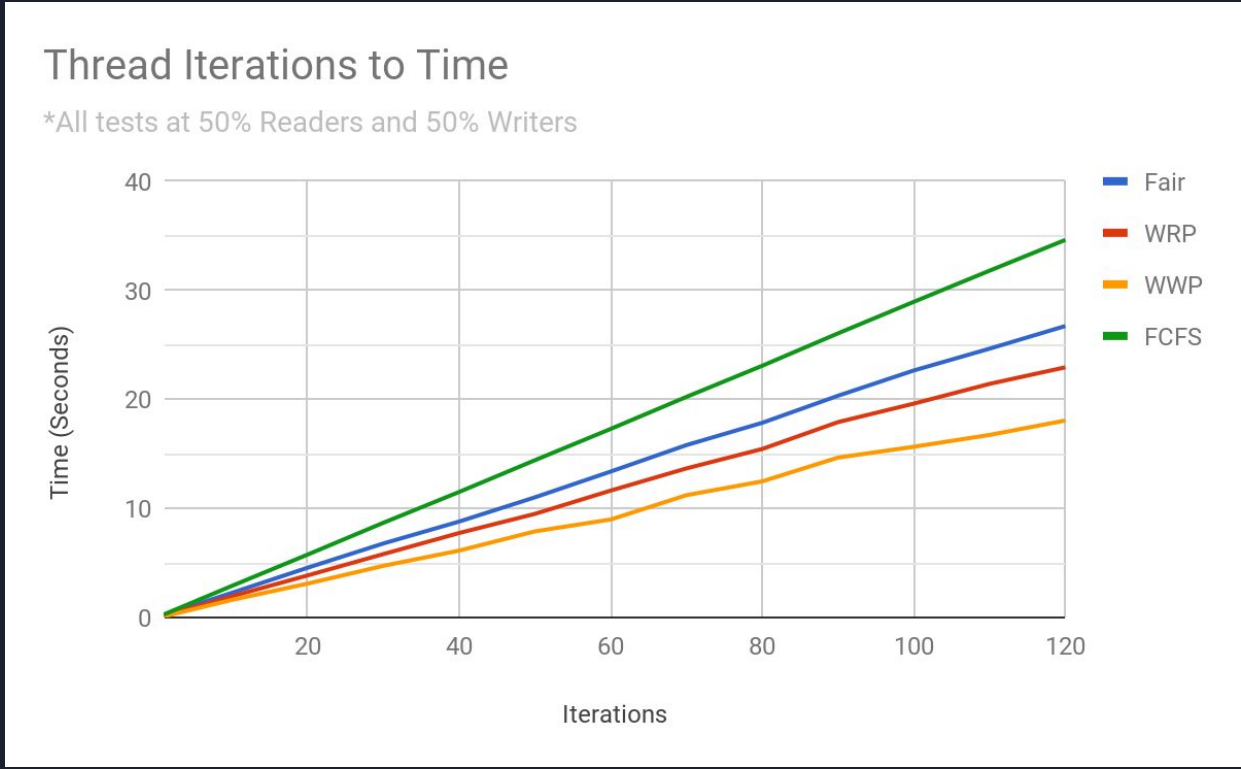
Writers sleep slightly longer than readers



Findings and Results



Findings and Results



Conclusions

Concurrency can lead to significant performance boosts

Organization among readers and writers can also lead to better performance if it is not taken to an extreme

Starvation can be a side-effect of focusing too much on concurrency or a single role in the Readers Writers Problem

A good variant finds balance between performance and starvation

References

- http://faculty.kutztown.edu/spiegel/CSc552/Examples/ReadersWriters/
- http://www.studytonight.com/operating-system/readers-writer-problem
- https://en.wikipedia.org/wiki/Readers%E2%80%93writers_problem