Readers and Writers Problem:

This problem occurs when many threads of execution try to access the same shared resources at a time. There are N readers to read data and K Writers to write data to shared resources, so in this documentation we will solve this problem.

First let's say we have a lot of cars that want to cross the intersection and we don't have a traffic light so of course they will crash into each other that's the problem we don't have a traffic light.

In our solution the traffic light is called Semaphore.

Semaphores are two-field data types, one of which is a non-negative type of integer S.V and the other is a set of processes in a queue S.L. It is used to address critical section problems by using two atomic operations, wait and signal, to synchronize processes.

What are Semaphores in operating system?

Semaphores refer to the integer variables that are primarily used to solve the critical section problem via combining two of the atomic procedures, wait and signal, for the process synchronization.

The definitions of signal and wait are given below:

Wait

It decrements the value of its A argument in case it is positive. In case S is zero or negative, then no operation would be performed.

wait(A)

{

while (A<=0);

A--.

}

### Signal

This operation increments the actual value of its argument A.

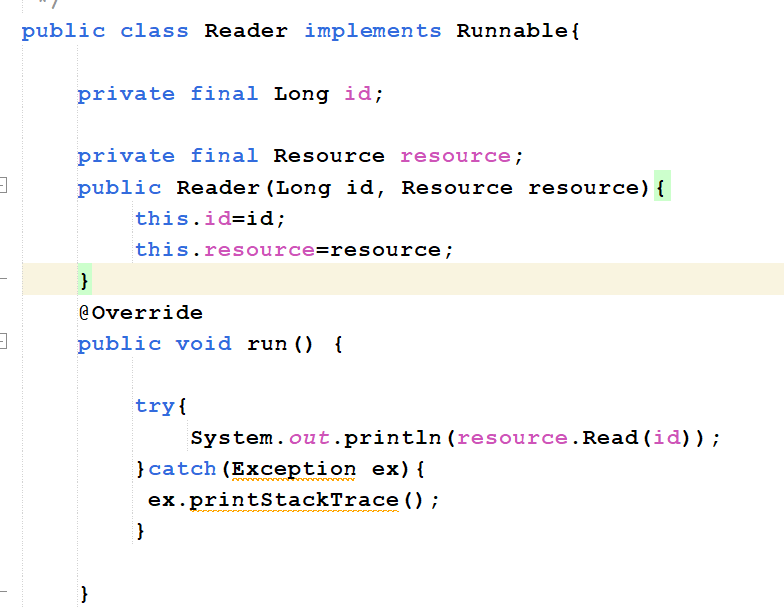
signal(A)

{

A++;

}

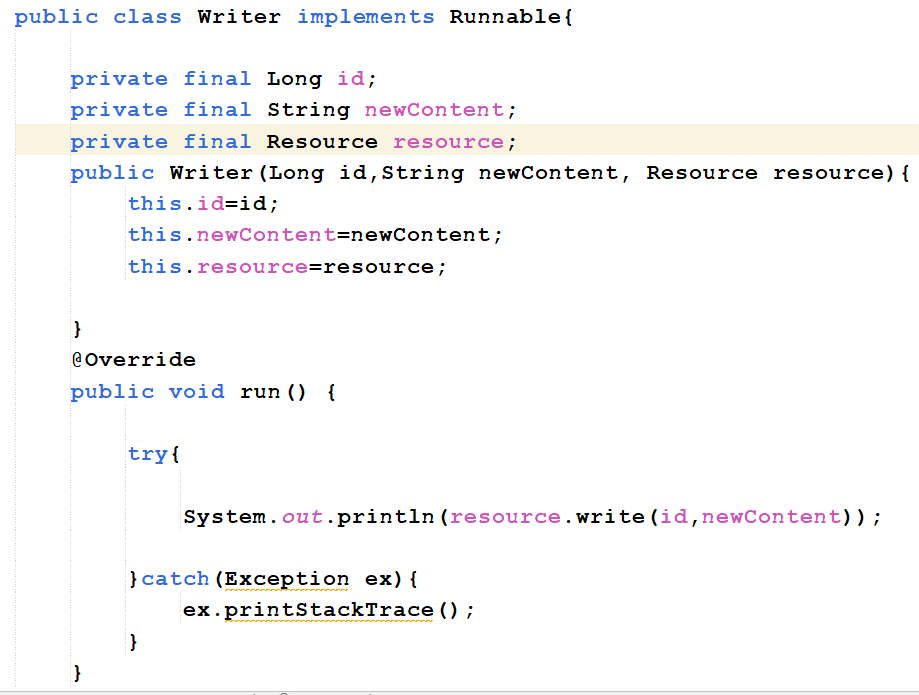
Now let's look at the code:



Here is Reader class:

1. Define two attributes integer id of the reader and object from resource.
2. Call the constructor and put the id of the reader and the resource that he wants to read.
3. In function run it will try to read the resource.
4. In any case of failure will catch and print the reason of the failure.

Second class is writer:



1. Define three attributes integer id of the writer, string new content and object from resource.
2. Call the constructor and put the id of the writer, the resource that he wants to write on it and the new content.
3. In function run it will try to write on it and print it after writing.
4. In any case of failure will catch and print the reason of the failure.

Third class is switch:

Function lock:

Graphical user interface, text, application, email

Description automatically generated

1. Define variable counter to count how many people in reading room and semaphore to protect from race condition and allow 5 people to enter the critical section to read.
2. In function lock, mutex ask to enter critical section.
3. After entering the critical section, increment the counter by 1.
4. If the counter is equal 1 then the reading room will be reserved and prevent any one to write on it.
5. After that release the mutex.

Function unlock in switch class:

Graphical user interface, text, application

Description automatically generated

1. Mutex ask to enter the critical section to unlock position in the queue.
2. After entering the critical section, the counter decrease by 1.
3. If the counter is equal 0 so there is no one in reading room.
4. Release the semaphore to allow any one to write in this resource.

Fourth class is resource:

Function read:



1. Define variable content to save the content of the resource in it.
2. Define object from semaphore (turn) to know who is the next to enter the critical section.
3. Define object from Switch (readSwitch) to allow people to read the resource.
4. Define object from semaphore (readRoom) to know that if anyone is reading the resource.
5. In function read, turn ask to enter to the critical section, readSwitch calls lock to lock the reading room and increment the number of readers.
6. Release the turn to allow anyone to ask to read again.
7. Define variable currentContent to put the content in it.
8. Put the content and id in the currentContent.
9. Finally, readSwitch calls unlock function to decrease the number of readers.
10. Return the current content.

Write function in resource class:

Graphical user interface, text

Description automatically generated

1. Writer asks to enter to the critical section.
2. Writer asks to lock the reading room.
3. After entering the critical section , the content will be equal the new content.
4. Finally, release the turn and the reading room to allow anyone to write or read.
5. Return the new content.

The five class is the main class:

Graphical user interface, text, application, email

Description automatically generated

1. Define the number of the readers.
2. Define the number of the writers.
3. Define object from resource to be the resource that all writers and all readers try to access it.
4. Loop from 0 to number of the readers.
5. Thread to each reader to access the shared resource.
6. Loop from 0 to number of the writers.
7. Thread to each writer to access the shared resource.

Example when the deadlock can occur:

When we request to enter and lock the critical section, a deadlock may occur if we do not open the critical section, any writer or reader will request to enter the critical section, will find that the critical section is busy but there is no one in the critical section.

In semaphore after we exit from the critical section then semaphore release the critical section to allow anyone to enter to it.

Example when the starvation can occur:

we have writers and readers so if there is any writer who wants to get into the critical section, he must ask to write and then lock the critical section to prevent any other reader or writer to write so if we lock the reading room no one can read from this resource but after this writer ends. Any writer else can write to it but, the reader can’t read.

In semaphore we have the turn that gives the writer or the reader the right to access the critical section, so that any writer or reader can access the critical section without starvation happened.

In the real world each writer or reader has a turn that gives them the right to access the critical section and do whatever they want to do and no one can enter the critical section if anyone else inside it and if any writer enter the critical section the reading room is locked to prevent any reader from reading and opening again after the writer finishes.