Operating System

Project 3: Readers/ Writers Locks

Report

04/05/20

Maria Martinez

Locks are mutual extension primitives that share data across the CPU. Locks are principal used to protect data providing optimization to the program using threads, in other to control the program because using thread it can halt the progress waiting for a condition to become true. A common implementation of locks is the reader and writer locks with is a code that uses threads to access and modify data. Read and Writers locks, is a flexible locking primitive that work like this, imagine that we are reading a data, this data can only be use to read at that time while we are reading we can not write, we only can read , but multiple readers can read the data simultaneously. When we finish reading and we want to modify the data we can now use Writers. Only one writer is permitted to modify at a time, when we are writing on the data no readers are allow. Read and Writers Locks uses mutex to control the execution order creating locks() function to lock and unlocks() functions to unlock, mutex are used in critical zones to guarantee optimization. Readers acquire more often lock for the shared resource, therefore Reader and Writer locks give more priority to readers due to multiple readers can read the data at the same time, while writer do not, this creates a writer starvation.

* Pseudocode of the solution

let initial\_readers= total count of reader threads

let initial\_writers= total count of writer threads

let pending\_readers= initial\_readers

let pending\_writers= initial\_writers

let active\_readers= 0

initialize binary semaphores:

read\_mutex= 1

write\_mutex= 1

pending\_rw\_lock= 1

initialize balancers semaphores:

balancer\_0= 0 //avoid writers’ starvation priority writers

balancer\_1= 1 // avoid readers starvation, priority readers

reader\_thread:

acquire lock for pending\_rw\_lock

calculate remaining percentages for both readers and writers

if percentage\_remaining\_readers <= percentage\_remaining\_writers and pending\_writers > 0:

release lock for pending\_rw\_lock

wait on balancer\_0 to give chance for the writers to proceed

else

release lock for pending\_rw\_lock

acquire lock for read\_mutex

if active\_readers=0:

acquire lock for writer\_mutex

increment active\_readers

release lock for read\_mutex

acquire lock for read\_mutex

decrement active\_readers

acquire lock for pending\_rw\_lock

decrement pending\_readers

calculate remaining percentages for both readers and writers

if pending\_readers == 0:

signal balancer\_1 enough to make sure no writer is blocked

else if percentage\_remaining\_readers <= percentage\_remaining\_writers and th\_info->rw->pending\_writers > 0:

signal balancer\_1 one time to wake up a writer

else:

signal balancer\_o one time to wake up a reader

release lock for pending\_rw\_lock

if active\_readers == 0:

release lock for write\_mutex

release lock for read\_mutex

writer\_thread:

acquire lock for write\_mutex

acquire lock for pending\_rw\_lock

calculate remaining percentages for both readers and writers

if percentage\_remaining\_writers < percentage\_remaining\_readers and th\_info->rw->pending\_readers > 0:

release lock for pending\_rw\_lock

release lock for write\_mutex

wait on balancer\_1 to give chance for the readers to proceed

acquire lock for write\_mutex

else

release lock for pending\_rw\_lock

acquire lock for pending\_rw\_lock

decrement pending\_writers

if pending\_writers == 0:

signal balancer\_0 enough to make sure no reader is blocked

else

signal balancer\_o one time to wake up a reader

release lock for pending\_rw\_lock

release lock for write\_mutex

In order to fix the writer starvation and at the same time do not create the reader starvation. I implement two semaphores balance\_0 and balancer\_1. These semaphores are used to coordinate the balance of operation between readers and writers with the purpose of calculating the percentages for the remaining readers and writers that are waiting to be done. Showing the result of the calculations will help to decide and establish priorities. It is important to note that these calculations are done many times thought the program.

In terms of concept Reader and Writers Lock is simple, but how to set up the mutex and understanding how the correct output should be more complicated. I spend a couple of days just reading and collecting information from different sources, in other to understand the different codes and examples. In the code implementation I spend so many hours of testing, and trying stuff out within the code. From Tuesday to Friday I spend around 4 hours each day working on the project. On Sunday I spend from 11 am until 9 pm working on the project finishing the code and working on the report.