Concurrent Programming ID1217
Report III

Emil Ståhl

February 17, 2020

1 Introduction

This report covers the implementation of a number of programs written in C that utilizes parallell execution with multiple threads. Synchronization is done solely with semaphores. In computer science, a semaphore is a variable or abstract data type used to control access to a common resource by multiple processes in a concurrent system such as a multitasking operating system. A semaphore is simply a variable. This variable is used to solve critical section problems and to achieve process synchronization in the multi processing environment. A trivial semaphore is a plain variable that is changed (for example, incremented or decremented, or toggled) depending on programmer-defined conditions.[1] The main topic covered in this work is to get a deeper understanding of the problems that a multithreaded program results in and how they can be solved with semaphores.

2 The programs and its purposes

The work consists of three different multithreaded programs written in C. Below is a description of each program and its purposes.

2.1 The Unisex Bathroom Problem

This program simulates a unisex bathroom that is used by an arbitrary number of men and women. The men and women are represented as threads that work (sleeps) and uses the bathroom for a random amount of time. Synchronization is done solely with semaphores. The persons will sleep for a random amount of time between visits to the bathroom as well as sleep for a smaller random amount of time to simulate the time it takes to be in the bathroom. The program prints a trace of interesting simulation events and then terminates when every person has used the bathroom for a specified number of times.

[1] A pragmatic, historically oriented survey on the universality of synchronization primitives

2.2 The Hungry Birds Problem

This program simulates the concept of one producer and multiple consumers. Given are n baby birds and one parent bird. The baby birds eat out of a common dish that initially contains W worms. Each baby bird repeatedly takes a worm, eats it, sleeps for a while, takes another worm, and so on. If the dish is empty, the baby bird who discovers the empty dish chirps real loud to awaken the parent bird. The parent bird flies off and gathers W more worms, puts them in the dish, and then waits for the dish to be empty again. This pattern repeats forever. The birds are represented as concurrent threads (i.e. array of "babyBird" threads and a "parentBird" thread), and the dish as a critical shared resource that can be accessed by at most one bird at a time. Only semaphores are used for synchronization.

2.3 The Bear and Honeybees Problem

This program simulates the concept of multiple producers and a single consumer. Given are n honeybees and a hungry bear. They share a pot of honey. The pot is initially empty; its capacity is H portions of honey. The bear sleeps until the pot is full, then eats all the honey and goes back to sleep. Each bee repeatedly gathers one portion of honey and puts it in the pot; the bee who fills the pot awakens the bear. The bees are represented as concurrent threads (i.e. array of "bees" threads and a "bear" thread), and the pot as a critical shared resource that can be accessed by at most one thread at a time. Only semaphores are used for synchronization.

3 Main problems and solutions

Below is a description of the different problems that each implementation resulted in and how they were solved.

3.1 The Unisex Bathroom Problem

The program consists of three methods, a main method that reads command line arguments, creates and join threads. Two different methods to represent males and females, the logic behind these two methods are extremely similar. Furthermore, three semaphores are used which are `lock`, `waiting_male`, `waiting_female`. The logic behind the program can be described like this:

- Get hold of the critical section (lock) semaphore.
- If there are people of the opposite gender inside we enter the waiting FIFO queue for our gender.
- Once we get hold of the lock semaphore we let all waiting persons of the same gender inside the bathroom.

- When all persons of the same gender are inside we release the lock
- Use the bathroom
- We then once again require the semaphore lock and all persons leave the bathroom

When the bathroom is empty and there are persons of the opposite gender waiting we give them priority in order to ensure fairness and avoid starvation. Since the waiting queue is FIFO that will ensure fairness. In addition to this, since the program won't terminate until every person has used the bathroom an equal number of times this will work as a proof for fairness if the program terminates.

3.2 The Hungry Birds Problem

The program consists of one producer thread and multiple consumer threads that all manipulates a shared global variable `worms`. The consumer threads will wait for the semaphore `full` and decrement the global variable. If the global variable is zero the thread that discovers it will signal the producer with the semaphore `empty` to start producing. When the producing thread has produced enough it will signal with the semaphore `full` to let the consumers know that they can start consuming again. The program seems to be starvation free but there is a discrepancy in how much each thread gets to consume.

3.3 The Bear and Honeybees Problem

The program consists of one consumer thread and multiple producer threads that all manipulates a shared global variable `honeyPot`. The consuming thread will wait for the semaphore `full` and decrement the global variable to zero. If the global variable is equal to a maximum value the thread that discovers it will signal the consumer with the semaphore full to start consuming. When the consuming thread has consumed enough it will signal with the semaphore `empty` to let the producing threads know that they can start producing again. In the printout shown below it can be seen that each producer thread adds one unit to the global variable, the program is therefore fair from a producer side point of view.

4 Evaluation

A test was performed for each program:

4.1 The Unisex Bathroom Problem

```
emilstahl$ ./bathroom 5 5 2
Man 1 enters the bathroom - Visit: #1
Women 0 enters the bathroom - Visit: #1
Man 2 enters the bathroom - Visit: #1
Women 1 enters the bathroom - Visit: #1
Man 2 enters the bathroom - Visit: #2
Women 4 enters the bathroom - Visit: #1
Man 0 enters the bathroom - Visit: #1
Women 3 enters the bathroom - Visit: #1
Man 4 enters the bathroom - Visit: #1
Women 2 enters the bathroom - Visit: #1
Man 3 enters the bathroom - Visit: #1
Women 1 enters the bathroom - Visit: #2
Women 2 enters the bathroom - Visit: #2
Man 1 enters the bathroom - Visit: #2
Man 4 enters the bathroom - Visit: #2
Man 3 enters the bathroom - Visit: #2
Women 0 enters the bathroom - Visit: #2
Women 4 enters the bathroom - Visit: #2
Women 3 enters the bathroom - Visit: #2
```

4.2 The Hungry Birds Problem

```
emilstahl$ ./hungryBirds 5 5 5 1
Bird 4 ate worm 3
Bird 0 sleeps
Bird 0 ate worm 2
Bird 1 sleeps
Bird 3 sleeps
Bird 1 ate worm 1
Bird 3 ate worm 0
Bird 2 sleeps
Bird 2 SQUEELS!!!!!!!!!!!!!!!!!!!!
**********Parent bird awake**********************
**********Parent bird found 3 new worms*********
**********Parent bird sleeping********************
```

4.3 The Bear and Honeybees Problem

```
emilstahl$ ./honeyBees 10
Bee 0 is adding one portion of honey into the honeyPot(1)
Bee 1 is adding one portion of honey into the honeyPot(3)
Bee 4 is adding one portion of honey into the honeyPot(2)
Bee 3 is adding one portion of honey into the honeyPot(5)
Bee 5 is adding one portion of honey into the honeyPot(4)
Bee 6 is adding one portion of honey into the honeyPot(6)
Bee 2 is adding one portion of honey into the honeyPot(7)
Bee 7 is adding one portion of honey into the honeyPot(8)
Bee 8 is adding one portion of honey into the honeyPot(9)
Bee 9 is adding one portion of honey into the honeyPot(10)
---------------The bear is eating all the honey from the honeyPot---------------
Bee 0 is adding one portion of honey into the honeyPot(1)
Bee 3 is adding one portion of honey into the honeyPot(3)
Bee 2 is adding one portion of honey into the honeyPot(4)
Bee 1 is adding one portion of honey into the honeyPot(5)
Bee 8 is adding one portion of honey into the honeyPot(6)
Bee 7 is adding one portion of honey into the honeyPot(7)
Bee 4 is adding one portion of honey into the honeyPot(2)
Bee 5 is adding one portion of honey into the honeyPot(8)
Bee 9 is adding one portion of honey into the honeyPot(9)
Bee 6 is adding one portion of honey into the honeyPot(10)
```

5 Conclusions

This work has focused on using semaphores for synchronization in a number of multithreaded programs. The main topics covered was to better understand the advantages and problems that a multithreaded system results in.