CSCE 351: Project 2

Levi Hassel, Eugene Kuznetsov and Colben Aldrich

1. Introduction

a. Project Goals

The goal of our project 2 is to solve the *Bear and the Honey Bees* problem. We will be using the Prototype OS/scheduler given to us on Blackboard. In conjunction with this, we will design a semaphore to provide mutual exclusion and synchronization for our threads.

b. Project Description

For this project, we will be using the base code provided by the instructor as a solution to Project 1. From there, we followed the video to correctly set up and prepare the code to implement Project 2. To accomplish the simulation of the Bear and Bees, we will create 10 'bee' threads and 1 'bear' thread that will interact with a pot of honey. We will design and implement a semaphore structure to restrict behavior of the threads interacting with the pot of honey. Each semaphore structure will be given its own space on the stack and freed upon completion. The structure will contain a blocking/waiting queue along with the current semaphore value. Upon each interaction there will be a print out as to if the action was successful or the type of encounter with the semaphore.

c. Project Management

I. Timeline -

- Download and configure Project 1 solution
- Laid out Semaphore and added required operations
- Design and create logic for Bear and Bee interaction
- Built queue for semaphores
- Finalized semaphore and queue functions
- Track and print interaction results

II. Teamwork -

Work was evenly divided, with every teammate putting in the time to do their best work, despite all members being busy with other classes and projects. We stuck with our designated meeting times from the previous project and used the instructor as a resource.

III. Risks Anticipated -

- Correctly implementing pointers and pointer functions in C
- Getting the logic to work correctly for thread interactions
- Correctly implementing the methods for the semaphores

IV. Resources -

- *Nios-II Software Developer's Handbook*
- *StackOverflow*
- *Given Resources: Project Description, Project Setup Guide, Help Powerpoint etc.*

2. Key Ideas

The key ideas for accomplishing this project are primarily from the understanding of how a thread should interact with a semaphore. From the previous project we were able to schedule threads, so here we want to be able to block and queue threads based off parameters within the project. These parameters will be the foundation of our logic for our semaphores and prevent deadlock and race conditions in our Bear and Bee threads. With our semaphore methods created correctly, our project will schedule and restrict our threads to our parameters.

3. Accomplishments

With the start of this project, the instructor provided completed Project 1 code and a video on how to configure it for Project 2. Our group opted to use this code and followed the provided videos to revamp the project to meet the needed starting point for Project 2. With the scheduler and working as to standards in Project 1, we could begin working on designing our semaphore and creating threads for our consumer and producer conditions.

By building onto Project 1 with the basic pseudo code of Homework 2 (also involving bears and bees), we adapted the code to create threads corresponding to Bees and Bears. While creating these threads, we had to correctly structure our *mysem_up* and *mysem_down* to logically lock threads from accessing the pot of honey while handling interrupts. As the first thread accesses our semaphore, we will lock the semaphore and allow the thread to finish until *mysem_up.* Any threads accessing *mysem_down* while the semaphore is locked will be blocked until the current active thread has completed. The locking mechanism we chose to use in our structure was a simple while loop. We simply looped the thread to run out the time quantum to keep it from continuing until the semaphore was unlocked.

To create the semaphore, we use a *mysem_create* function. This function creates a semaphore structure and allocates memory for it on the board. The semaphore value is initialized to the provided argument, the threads waiting counter is set to zero, and the queue is set to empty.

Our semaphore structure also contain a blocking/waiting queue. This queue was created similar to the queue given in the project files. We implemented a *sem_queue* and *sem_dequeue* function to allow interaction with the queue. The *mysem_down* and *mysem_up* functions specifically use these queue functions to add and remove threads as they are locked and unlocked.

Upon completion, the semaphore is deleted. We removed the structure and the queue associated with the semaphore using a *mysem_delete* function. This function simply frees the allocated memory that was given to the semaphore when it was created.

We also have a few functions that were made to simplify our logic. *mysem_waitCount* is used to return the number of thread waiting on the semaphore. *Mysem_value* returns the value of the given semaphore. These are used to allow other functions to access the given content in the structure.

4. Understanding

In the development of this project, we were able to better verse ourselves in the topics discussed in class. Thanks to our experiences in the previous project, we had a much better foundation in our understanding when beginning this one. We were able to successfully implement structures for the semaphores and create room for them in the stack. The use of the semaphores required knowledge of the way the thread scheduler worked. Because we were using code that was given to us, we took time to get a full understanding of the methods used to implement the scheduler. Through this, we were able to work with the same data structures in the given scheduler and use similar methods in our semaphores. After successfully implementing these structures, the rest of the project was completed successfully using our previous knowledge of correctly implementing semaphores in threads.

5. Evaluation

We are very happy with the final product. This project seemed much simpler than the previous one. The biggest reason for this was due to the amount of understanding we had gained from the previous project. We ran into a number of errors with correctly implementing pointers and data structures. If we would do this over again, we would take time in figuring out how we were going to use the pointers before we began designing. Overall, we gained a better understanding of semaphores and what it takes to make them work correctly.

6. Improvements

We have no complaints in regards to this project. The documentation was great, the process was straightforward and we learned a lot. It did seem easier after the insanity that was Project 1, but we wouldn't change that either. The learning curve for the class is just right.

7. References

All content is *original*.