Homework 3 - Report

Group Members
Anton Hammar - antham@kth.se
Ludwig Flodin - ludwigfl@kth.se

The Hungry Birds Problem

The code is divided into two functions, producer and consumer. For this problem there are multiple consumers called baby birds and one producer called parent bird. To start off the program the thing that is to be consumed, the worm counter is set to a value bigger than 0 to activate the consumer part of the code.

The multiple consumers are threads which represent baby birds, each thread will try to lock the “notEmpty”, if it is already locked by another thread they will be forced to wait until the lock is released by the current thread. When a baby bird thread has acquired the lock it will decrement the total amount of worms (eat a worm), release the lock to let other threads access the worm total and sleep for a set amount of time before trying to acquire the lock again.

If a baby bird thread that has the lock notices that the worm total is not bigger than 0 it will do an unconditional increment (sem_post) to the “empty” lock. This is the lock that the producer, parent bird is waiting for. Parent bird that has been waiting for the “empty” variable to become 1 will now be able to lock it with conditional decrement (sem_wait) and refill the worm total. It will then release the lock that the baby bird threads are waiting for to indicate that there are now worms available to be eaten. The “empty” lock will now be set to 0 and will stop the parent until another baby bird thread increments it again in the future.

The Bear and Honeybees Problem

The Bear and Honeybees Problem works very similarly to The Hungry Birds Problem. It starts by creating threads for each bee. The bear uses the main thread.

Each bee has an ID which is used when printing out interesting events. This ID comes from the thread. The bee then enters an infinite while loop where it will continue to gather honey to a honeypot.

Each loop starts by calling “sem_wait” on the “notFull” semaphore which stops all other bees putting honey in the pot while another bee is doing so. If the pot is not full, the “honey” value is incremented plus 1. After this it calls “sem_post” on the “notFull” semaphore to allow other bees to add their honey to the pot. The bee then proceeds to “sleep” for a short time in order to represent the time it takes for a bee to gather honey.

If a bee adds honey to the pot and it becomes full (which is checked with a “maxHoney” variable), then the bee will awaken the bear by calling “sem_post” on the “full” semaphore and then the bee waits until the bear has finished eating all the honey.

The bear is also in an infinite while loop. In the loop it waits for the “full” semaphore to be released so the bear can eat the honey. If its released it calls “sem_wait” on the “full” semaphore to lock it again. Then it proceed to eat all the honey by setting the “honey” variable to 0. This takes two seconds by using the “sleep” function. After its done it calls “sem_post” on the “notFull” semaphore in order to release the lock and allow the bees to start filling up the pot again.

Fairness

Both the The Hungry Birds Problem and The Bear and Honeybees Problem are not “fair” with respect to both consumer and producer. In the bird problem it is unfair to the producer (the parent bird), and in the bee problem it is unfair to the consumer (the bear).