Assignment 3 - Semaphores

Emanuel Paraschiv& Mohammad Asfour

February 2025

1 Introduction

The scope of this assignment is to use semaphores to synchronize threads in a concurrent program with shared variables and to control access to shared resources.

All following programs are developed in c using pthreads along with semaphores, representing classic consumer-producer scenarios.

2 Hungry birds problem

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. When the dish is empty, the baby bird, who discovers the empty dish, chirps loudly to awaken the parent bird. The parent bird flies off, gathers W more worms, puts them in the dish, and then waits for the dish to be empty again. This pattern repeats forever.

In this case we deal with one producer and multiple consumers.

The Producer will simply wait until a consumer signals the empty dish, then updates it to full:

```
    void *Parent_Bird(void *arg) {
    while (1) {
        sem_wait(empty);  // Wait for a baby bird to signal an empty dish
        printf("Refilling dish\n");
        worm_count = WORM_NUM;
    }
    return NULL;
}
```

Each consumer will fetch a portion of the dish and the last to take, noticing the end will signal the producer:

```c
void *Baby_Birds(void *arg) {
    int id = *(int *)arg;
    free(arg);

    while (1) {
        sem_wait(mutex);  // Enter critical section
        if (worm_count > 0) {
            worm_count--;
            printf("Baby bird %d is eating a worm. Worms left: %d\n", id, worm_count);

            if (worm_count == 0) {
                printf("Baby bird %d signals that the dish is empty.\n", id);
                sem_post(empty);  // Signal the parent bird when the dish is empty
            }
        }
        sem_post(mutex);  // Leave critical section

        sleep(1);
    }
    return NULL;
}
```

Baby birds can continuously access the dish as long as worms are available. They only block each other when one is currently accessing the dish (due to mutex), but otherwise, they don't wait unnecessarily.

The solution is fair because the producer only acts when the dish is empty, ensuring it doesn't interfere with the baby birds eating. The consumers eat as long as worms are present, but they must wait for a refill when the dish is empty. There's no priority given to either eating or refilling . The action depends solely on the state of the shared resource worm_count.

3 Bees and bear problem

This is a multiple producers - single consumer type of problem. n number of bees (producers) produce H amount of honey and fill a pot, and when the pot is filled a bear (consumer) will come and empty the pot. This will signal the bees to start producing again, and so on. The honey pot (shared memory) is protected by semaphores. Mainly 2 semaphores are used: `mutex` which acts like a lock (initialized to 1), and `fill_pot` which acts like a condition variable (initialized to 0).

Each bee that enters the function `_Bees`() will acquire the lock `mutex` will fill the pot with one portion of honey, then check if the pot is filled. If it is filled, it will signal the `full_pot` condition in bear:

```c
if (H < capac) {
    printf("Bee number %d is putting one portion of honey in the pot\n", worker_id);
```

```
    H++;
    if (H >= capac) {
        printf("Bee number %d has filled the pot and waking up the bear\n", worker_id);
        sem_post(&full_pot);
    }
}
```

The thread that controls the bear will then be signaled to consume, which will empty the pot, sleep for some time, then wait until the pot is full again:

```
while(1) {
    sem_wait(&full_pot);
    printf("Bear has been woken and will eat the honey\n");
    H = 0;
    Sleep( 10);
}
```

The solution is mostly fair, as the bear will not interfere with the locks until the pot is filled, and the bees block each other only when modifying the shared resource, otherwise there is no unnecessary waiting. The only negative thing is that since we are using semaphores, the bees will be managed using a FIFO queue, so the bees that are more active will contribute more to the pot.