CPSC 545 Computing Systems
Project 2: Hungary Eagles Need Feeding
Due: April 27th, 23:59PM

1. Goal

To develop a C/C++ multi-threading program that uses Pthread synchronization mechanisms to solve real-world problems.

2. Description

A family of eagles has one mother eagle, ***n*** baby eagles, and ***m*** feeding pots initially empty, where 0 < ***m*** <= ***n***. Each baby eagle must eat using a feeding pot and each feeding pot can only serve one baby eagle at any time. Therefore, no more than ***m*** baby eagles can eat at the same time. The mother eagle eats elsewhere and does not require a feeding pot. **All feeding pots are assumed to be empty at the very beginning and the mother eagle is sleeping.**

Each baby eagle plays for a while and eats. Before a baby eagle can eat, he must wait until a feeding pot has food. After eating, the corresponding feeding pot becomes empty and has to be refilled by the mother eagle.

The mother eagle is very tired and needs sleep. So, she goes to sleep until a baby eagle wakes her up. Should this happen, the mother eagle hunts for food and fill ***all*** feeding pots. Then, she sleeps again. It is possible that when a baby eagle wants to eat and finds out that all feeding pots are empty. This baby eagle should wake up the mother eagle. As mentioned earlier, the mother eagle takes some time to hunt for food and fill all feeding pots. Once all feeding pots are filled, no more than ***m*** waiting hungry baby eagles can eat. Since the mother eagle does not want to wake up too often, she insists that exactly one baby eagle who found out all feeding pots being empty can wake her up. More precisely, even though two or more hungry baby eagles may find out all feeding pots being empty, only one of them can wake her up and all the others just wait until food will become available.

Note that a baby eagle should not wake up the mother if some baby eagles are eating and the remaining pots are empty.

After providing the ***t***-th meal, the mother eagle retires and sends all of her baby eagles away to start an independent life. Perhaps, she needs a vacation, and, therefore, the feeding game ends!

In the system, eagles are simulated by threads. Initially, the ***m*** feeding pots are all empty. Each baby eagle thread has the following pattern:

```
while (1) {
    Delay();          // play for a while
    ready_to_eat(...);  // get hungry
    Delay();          // eat for a while
    finish_eating(...); // finish eating
                        // do some other thing
```

```
}
```

The mother eagle thread has the following pattern:

```
while (not time to retire) {
   goto_sleep(...);   // take a nap
   Delay();          // prepare food
   food_ready(...);   // make food available
   Delay();          // do something else
}
```

Here, functions ready_to_eat(), finish_eating(), goto_sleep() and food_ready() are functions written by you. They have the following meaning:

- ready_to_eat() blocks the caller, a baby eagle, if all feeding pots are empty. One and only one baby eagle who finds out all feeding pots being empty should wake up the mother eagle. This function returns only if there is a feeding pot with food available to this baby eagle.
- finish_eating() should be called when a baby eagle finishes his meal.
- goto_sleep() is only called by the mother eagle when she wants to take a nap.
- food_ready() is called when the mother eagle has finished adding food in all ***m*** feeding pots.

In this way, all controls and synchronization mechanisms are localized in these four functions and the eagle threads look very clean.

2.1 Handle Important situations correctly

Your implementation must correctly handle the following **important situations** among others:

- At any time, there are no more than ***m*** baby eagles eating.
- A baby eagle must wait when he wants to eat but has no free feeding pot and/or all free feeding pots are empty.
- If there is a non-empty feeding pot, a hungry and ready-to-eat baby eagle can eat.
- No hungry baby eagle will eat using an empty feeding pot.
- At any time, a feeding pot can only be used by one eating baby eagle.
- Only one baby eagle among all baby eagles who want to eat can wake up the mother eagle.
- The mother eagle does not do her work until a baby eagle wakes her up.
- While the mother eagle is preparing food, no baby eagle can wake up the mother again until the mother goes back to take a nap.
- Before all ***m*** feeding pots are filled, no hungry baby eagle can eat.
- Once the feeding pots are refilled, the mother eagle must allow baby eagles to eat. Then, she goes back to sleep.
- You must terminate your program gracefully. More precisely, if ***t*** feedings are needed, then your program **cannot** terminate right after the mother eagle refills the feeding pots ***t*** times. Instead, your program must wait until all feeding pots become empty, even though there may be baby eagles waiting for food.

You can only use Pthread semaphores and mutex locks. Use other mechanisms will risk low or even no credit. Simply using mutex locks without semaphores will result in zero credit.

2.2 Input

The input to your program consists of number of feeding pots ***m***, number of baby eagles ***n***, and number of feedings ***t.*** Your program needs to receive them from the command line arguments as follows:

./eaglefeed m n t

Thus, `eaglefeed 8 15 12` means there are 8 feeding pots, 15 baby eagles and 12 feedings. If any one of these command line arguments is 0, the default value 10 should be used. For example, `eaglefeed 3 0 0` means that there are 3 feeding pots, 10 baby eagles and 10 feedings. **You can assume that command line arguments satisfy 0 < *m* <= *n* <= 20 and *t* > 0.**

2.3 Output

Your program should generate an output similar to the following:

```
MAIN: There are 10 baby eagles, 5 feeding pots, and 8 feedings.
MAIN: Game starts!!!!!
        ......
  Baby eagle 3 started.
        ......
Mother eagle started.
        ......
     Baby eagle 6 started.
        ......
  Baby eagle 3 is ready to eat.
        ......
Mother eagle takes a nap.
        ......
    Baby eagle 5 is ready to eat.
        ......
 Baby eagle 2 sees all feeding pots are empty and wakes up the mother.
       Baby eagle 8 is ready to eat.
Mother eagle is awoke by baby eagle 2 and starts preparing food.
 Baby eagle 1 is ready to eat.
        ......
Mother eagle says "Feeding (1)"
        ......
    Baby eagle 5 is eating using feeding pot 3.
        ......
Mother eagle takes a nap.
        ......
  Baby eagle 3 is eating using feeding pot 1.
        ......
    Baby eagle 5 finishes eating.
```

.....
.....
Mother eagle says "Feeding (8)"
Baby eagle 1 is ready to eat.
.....
Mother eagle retires after serving 8 feedings. Game ends!!!

- All output lines from the mother eagle starts on column 1 and baby eagle *i*'s output lines have an indentation of *i* spaces.
- The following output line tells us that baby eagle 3 has started:
  Baby eagle 3 started.
- The following output line tells us that baby eagle 5 is ready to eat:
  Baby eagle 5 is ready to eat.
- The following output line tells us that baby eagle 5 is eating using feeding pot 3:
  Baby eagle 5 is eating using feeding pot 3.
- The following output line tells us that baby eagle 2 sees all feeding pots being empty and wakes up the mother eagle.
  Baby eagle 2 sees all feeding pots are empty and wakes up the mother.
- The following output line tells us that baby eagle 5 finishes eating:
  Baby eagle 5 finishes eating.
- The following output line tells us that the mother eagle has started:

Mother eagle started.

- The following output line tells us that the mother eagle takes a nap:

Mother eagle takes a nap.

- The following output line tells us that baby eagle 2 wakes up the mother eagle and the mother starts preparing food:

Mother eagle is awoke by baby eagle 2 and starts preparing food.

- The following output line tells us that the mother eagle has finished food preparation. The number in () is the feeding count. This is the third feeding.

Mother eagle says "Feeding (3)"

- The following output line tells us that the required number of feedings have completed and the mother retires. The feeding game ends here. **But, your program should keep running until all baby eagles finish eating even though the mother retires. In other words, your system should end gracefully.**

Mother eagle retires after serving 8 feedings. Game ends!!!

- Please note the indentation in the output. For easy grading purpose, use the above output style. **Do not invent your own output, because the grader does not have enough time to digest your output.**

2.4 Delay()

In the delay(), you cannot simply call sleep() system call. The following code can be used to support delay().

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
```

```c
/*************************************************************************
pthread_sleep takes an integer number of seconds to pause the current thread We
provide this function because one does not exist in the standard pthreads library. We
simply use a function that has a timeout.
*************************************************************************/
int pthread_sleep (int seconds)
{
	pthread_mutex_t mutex;
	pthread_cond_t conditionvar;
	struct timespec timetoexpire;
	if(pthread_mutex_init(&mutex,NULL))
	{
		return -1;
	}
	if(pthread_cond_init(&conditionvar,NULL))
	{
		return -1;
	}
	//When to expire is an absolute time, so get the current time and add
	//it to our delay time
	timetoexpire.tv_sec = (unsigned int)time(NULL) + seconds;
	timetoexpire.tv_nsec = 0;
	return pthread_cond_timedwait(&conditionvar, &mutex, &timetoexpire);
}
/*************************************************************************
* This is an example function that becomes a thread. It takes a pointer
* parameter so we could pass in an array or structure.
*************************************************************************/
int *worker(void *arg)
{
	while(1)
	{
		printf("Thread Running\n");
		fflush(stdout);
		pthread_sleep(1);
	}
}
```

```
/*****************************************************************************
* The main function is just an infinite loop that spawns off a second thread
* that also is an infinite loop. We should see two messages from the worker
* for every one from main.
*****************************************************************************/
int main()
{
	pthread_t t_id;
	if ( -1 == pthread_create(&t_id, NULL, worker, NULL) )
	{
		perror("pthread_create");
		return -1;
	}
	while(1)
	{
		printf("Main Running\n");
		fflush(stdout);
		pthread_sleep(2);
	}
	return 0;
}
```

3. Documentation

You *must* include a README (txt) file which describes your program. It needs to contain the following:

- The purpose of your program
- How to compile the program
- How to use the program from the shell (syntax)
- What exactly your program does

The README file does not have to be very long, as long as it properly describes the above points. Proper in this case means that a first-time user will be able to answer the above questions without any confusion.

Within your code you should use one or two sentences to describe each function that you write. You do not need to comment every line of your code. However, you might want to comment portions of your code to increase readability.

At the top of your README file and main C/C++ source file please include the following comment:

```
/* CPSC545  Spring2011 Project 2
* login: cs1_login_name (login used to submit)
* Linux
* date: mm/dd/yy
* name: full_name1, full_name2 (for partner(s))
* emails: your and your partners' emails */
```

In particular, in your source file, you need to describe the purposes of semaphores, mutex locks and variables used for synchronization. Put the descriptions right besides the declarations of these semaphores, mutex locks and variables. Failure to do this will risk low credit!

4. Grading

2 pts README file
2 pts Makefile
4 pts Documentation within code, coding, and style (indentations, readability of code, use of defined constants rather than numbers)
12 pts Test cases (correctness, error handling, meeting the specifications)

- Please make sure to pay attention to documentation and coding style. A perfectly working program will not receive full credit if it is undocumented and very difficult to read.
- The test cases will not be given to you upfront. They will be designed to test how well your program adheres to the specifications. So make sure that you read the specifications very carefully. If there is anything that is not clear to you, you should ask for a clarification.
- Make sure your code compiles and run on cs1.seattleu.edu. Failures to compile and execute you program result in zero in test cases.

5. Deliverables

The following deliverables must be submitted:

- Source code files
- A README file
- A Makefile that will compile your code and produce a program called **eaglefeed**. Note: this Makefile will be used by us to compile your program with the make utility.

All files should be made a package by tar and submitted using the SUBMIT utility. This is your official submission that we will grade. Please note that future submissions under the same homework title OVERWRITE previous submissions; we can only grade the most recent submission. You can NOT submit your code after the deadline is passed.

6. Submission

You should connect to cs1.seattleu.edu using your account to submit your project deliverables. You can submit multiple times before the deadline. The system keeps the most recent submission. You should follow the steps to submit your project:

- Make a tar package of your deliverables (one example below)
  tar -cvf p2.tar README Makefile eaglefeed.h eaglefeed.cpp
- Submit your tar package
  /home/fac/testzhuy/CPSC545/upload p2 p2.tar