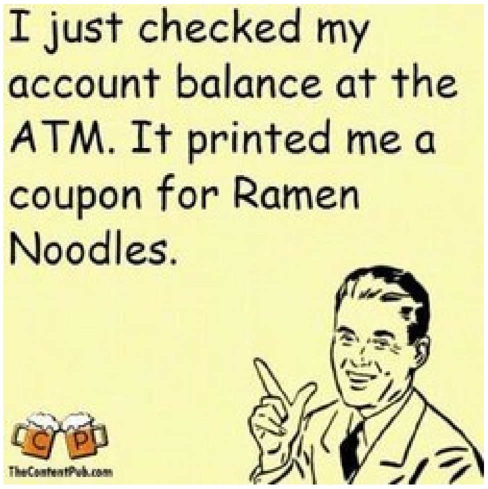
CS 370 – Project #3

Purpose: Become familiar with multi-threaded programming, semaphores, and deadlock concepts
Points: 100

Introduction:

The *Hungry Students* concurrent programming assignment using semaphores[1] is summarized below.


I just checked my account balance at the ATM. It printed me a coupon for Ramen Noodles.
TheContentPub.com

Consider a dorm floor of hungry undergraduate students. To help save students money, the dorm RA has cooked a small pot that can hold MAX_SERVINGS of Ramen Noodles[2]. When a student wants to eat, the student can get a serving from the pot, unless it is empty. If the pot is empty, the students wakes up the RA and then waits until the RA has refilled the pot.

The synchronization constraints include:

- Students can not call ***getServingFromPot()*** if the pot is empty.
- The RA can only re-fill the pot if it is empty
- The students may not wake the RA if the pot is not empty.

In this problem, the RA represents the operating system and the students represent the processes. The operating system should allocate the required resources to the processes and, at the same time, avoid deadlock.

Note, scoring will include functionality, comments, and general code quality.

Project:

Implement the Hungry Students problem in **C** (not C++). The program must compile and execute under Ubuntu 22.04 LTS (and will only be testing with Ubuntu). Assignments not executing under Ubuntu will not be scored. See additional guidance on following pages.

Submission:

When complete, submit:

- A copy of the **C** source code (not C++) file by class time.

Submissions received after the due date/time will not be accepted.

1 For more information, refer to: https://en.wikipedia.org/wiki/Semaphore_(programming)
2 For more information, refer to: https://en.wikipedia.org/wiki/Instant_noodle

Specifications:
Global variables/constants (and no others)

unsigned integers
- REFILL_SIZE = 5
- MAX_REFILLS = 5
- currPotServings
- currPotRefills

semaphores
- studentMtx
- empty pot
- full pot

The ***main()*** function should perform the following actions:

- Get and validate student count
  - must be between 3 and 20 (inclusive)
  - An array for threads must be used
    - Thread array must be dynamically allocated (i.e., use **`malloc()`**).
- Initialize semaphores
  - the student semaphore initialized to 1
  - the empty pot and fill pot semaphores initialized to 0
- Create one RA thread
- Create *student count* student threads
- Wait for the appropriate threads to complete.

The ***RA()*** function will perform the following actions:

```
while true
        P(emptyPot)
        putServingsInPot()
        V(fullPot)
```

The ***hungryStudents()*** function should perform the following actions.

```
while true
        P(studentMtx)
                if currPotServings == 0
                        V(emptyPot)
                        P(fullPot)
                getServingFromPot()
        V(studentMtx)
        studentEats(studentNumber)
```

In addition to the basic synchronization outlined, you will need to add the code to count the number of times the pot has been re-filled. When the pot has been re-filled MAX_REFILLS times, the RA and student threads should terminate.

The various support functions include

- ***putServingsInPot()***
  - update currPotServings by REFILL_SIZE
  - increment potReFills
  - display pot refilled message (see example)
  - wait a random amount of time (while pot is re-filled and re-heated)
    - **`usleep(rand() % 500000)`**
- ***studentEats()***
  - display student eats message with student number
  - waits a random amount of time (while student eats)
    - **`usleep(rand() % 1000000)`**
- ***getServingFromPot()***
  - decrements current pot servings
  - wait a random amount of time (to get a serving)
    - **`usleep(rand() % 1000);`**

The RA messages should be printed in red and the student messages printed in green.

```
printf("\033[0;31mI'm a message in red\033[0m\n", 0);     // red
printf("\033[0;32mI'm a message in green\033[0m\n", 0);   // green
```

<u>Include Files:</u>
In order to use threading functions and semaphores in C, you will need the below include files:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <pthread.h>
#include <semaphore.h>
#include <time.h>
#include <stdbool.h>
```

<u>Compilation Options:</u>
Use the following compiler options

```
gcc -Wall -pedantic -pthread -o hungryStudents hungryStudents.c
```

Point will be removed for poor coding practices and unresolved warning messages.

Example Execution:

```
ed-vm% ./hungryStudents 3
Hungry Students
  Student Count: 3
  Refill Count: 5

OK, fine, here are some more Ramen Noodles (5)...
Student 0 eating, yum...
Student 2 eating, yum...
Student 1 eating, yum...
Student 0 eating, yum...
Student 2 eating, yum...
OK, fine, here are some more Ramen Noodles (5)...
Student 1 eating, yum...
Student 2 eating, yum...
Student 0 eating, yum...
Student 2 eating, yum...
Student 0 eating, yum...
OK, fine, here are some more Ramen Noodles (5)...
Hey, this is the last of the Ramen Noodles...
Student 1 eating, yum...
Student 2 eating, yum...
Student 0 eating, yum...
Student 0 eating, yum...
Student 1 eating, yum...

Game over, thank you for playing.
ed-vm%
```

Note, actual mileage may vary.