ARIZONA STATE UNIVERSITY

CSE 430, SLN 14814 — **Operating Systems** — Spring 2015

Instructor: Dr. Violet R. Syrotiuk

Assignment #2

Available Wednesday 02/18/2015; due Monday 03/02/2015

This assignment had questions related to process aynchronization and the `pthreads` library.

1. 

This question has been on a midterm exam in the past. Therefore you should be able to do it without looking for help online or otherwise!

Huey, Dewey, and Louie are Donald Duck's mischievous nephews. Huey is the leader of the group. Dewey is the brains of the group and so follows Huey, while Louie is laid-back and follows behind Dewey.

Introduce semaphore(s) to synchronize processes Huey, Dewey, and Louie such that `Huey(i)` always gets executed before `Dewey(i)` which always gets executed before `Louie(i)`, for `i` ≥ 0. If $\Rightarrow$ denotes an "executes before" relation, then the processes should be synchronized such that:

`Huey(0)` $\Rightarrow$ `Dewey(0)` $\Rightarrow$ `Louie(0)` $\Rightarrow$ `Huey(1)` $\Rightarrow$ `Dewey(1)` $\Rightarrow$ `Louie(1)` $\Rightarrow \ldots$

```
while( true ){
   Huey(i);
   i++;
}
```

```
while( true ){
   Dewey(i);
}
```

```
while( true ){
   Louie(i);
}
```

Variable `i` is global, and is initalized to zero. Be sure to initialize any semaphore(s) that you introduce!

2. Consider the following solution to the infinite buffer producer/consumer problem we discussed in class.

```
int n = 0; /* global variable */
semaphore mutex, produced;

void Producer(){
   while(true){
      produce();
      wait(mutex);
      append();
      n++;
      if(n == 1)
         signal(produced);
      signal(mutex);
   }
}
```

```
void Consumer(){
   while( true ){
      wait( mutex );
      take();
      n--;
      signal( mutex );
      consume();
      if(n==0)
         wait(produced);
   }
}
```

```
void main(){
   init(mutex,1);
   init(produced,0);
   parbegin(Producer(),Consumer());
}
```

Give an interleaving of the statements of the producer and consumer illustrating that the consumer can consume an item in the buffer that does not exist. One way to do this is to fill in a table such as the one that follows, stopping when the condition occurs.

```
Producer     Consumer     mutex      n     produced
produce();                  1        0        0
wait(mutex);                0        0        0
n++;                        0        1        0
...
```

3. **The roller coaster problem.** Suppose that there are n passenger threads and a roller coaster car thread. The passengers repeatedly wait to take rides in the car which can hold C passengers, $C < n$. The car can go around the tracks only when it is full.

   Here are some additional constraints:

   - `Passengers` should invoke `board` and `unload`.
   - The car should invoke `load`, `run`, and `unload`.
   - `Passengers` cannot board until the car has invoked `load`.
   - The car cannot depart until C passengers have boarded.
   - `Passengers` cannot unload until the car has invoked `unload`.

   Write pseudocode using semaphores for synchronization and mutual exclusion for the passengers and car that enforces these constraints.

4. There is a useful website on `pthreads` at `https://computing.llnl.gov/tutorials/pthreads/`. Use it to familiarize yourself with creating threads, and manipulating mutex variables. In the following problems, you are expected to use *only* the `pthreads` library on a Linux machine to produce a solution.

   (a) Introduce a mutex into the program `inc.cc` to eliminate the race condition on the variable `counter` in the function `increment()`. Include the modified program as your solution.

   (b) Now, modify the program `inc.cc` to see how many threads you can spawn on your machine. This is implementation dependent; check the return code to determine when spawning the thread fails. Include your revised program, and output showing when you reach the thread limit. Since the limit is implementation dependent, report the configuration of your hardware (e.g., machine, number of cores, etc.) and your software (e.g., operating system, compiler, version, etc.).

   (c) Write a program using `pthreads` to implement your solution to Question 1 synchronizing Huey, Dewey, and Louie. Include your code, as well as sample output of your program, for up to `i` $= 5$.

   (d) Write a program using `pthreads` to implement your solution to the roller coaster problem in Question 3. Assume a car can hold $C = 4$ passengers, and that there are $n = 10$ passenger threads. Let the coaster run 5 times, indicating which threads board the car in each run around the tracks.