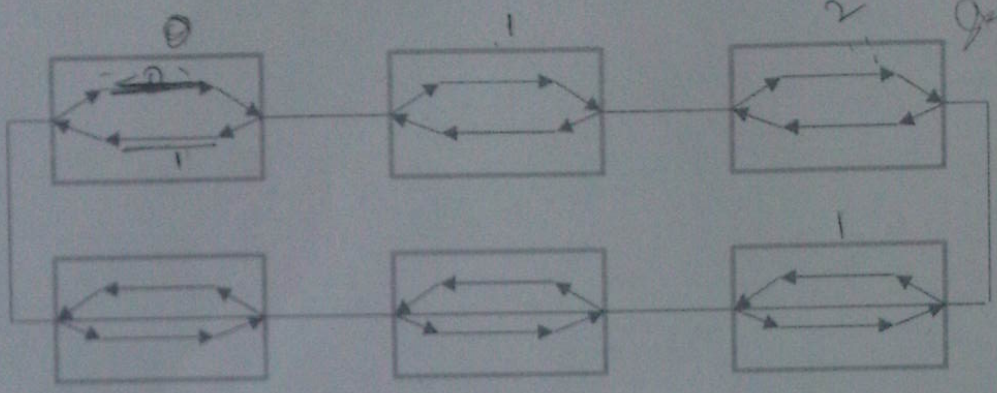
System and Device Programming
Examination Test – Programming Part
22 July 2013

Examination Time: 1h 45min. Evaluation. 18 marks.
Textbooks and/or course material allowed.
*The final mark is the sum of the 1st and the 2nd parts; **it cannot be refused** (no retry for marks >= 18).*

A circle underground line has a number of stations connected by a single track. Every station has two tracks, track 0 devoted to the trains traveling clockwise, and track 1 devoted to the trains traveling counterclockwise, as shown in the figure.



Implement a concurrent C program that takes two command line arguments, the number of stations in the circle line, and the number of trains (less than the number of stations).
The main thread must randomly setup the initial condition, i.e., the station and track where every train is, and create a thread for every train. Notice that the direction of the train depends on the track it is resting. A train traveling clockwise will always use track 0 of any station, and train traveling counterclockwise will always use track 1 of any station.
When all the data structures representing trains, the stations, and the connection tracks between two stations have been set, the train thread can start. Every train stay at the station for a random number of seconds (max 5), then, if it is possible it goes to the next station, occupying the connection track for 10 seconds. The movement of the train (always in the same original direction) is possible if the destination station track is free and the connection track is free.
Every thread must print its current condition in a common log file.
See this example of log for 10 stations and 3 trains:

```
Train n. 0, in station 5 going COUNTERCLOCKWISE
Train n. 2, in station 6 going CLOCKWISE
Train n. 1, in station 2 going CLOCKWISE
Train n. 2, traveling toward station 7
Train n. 1, traveling toward station 3
Train n. 0, traveling toward station 4
Train n. 2, arrived at station 7
Train n. 2, in station 7 going CLOCKWISE
Train n. 2, traveling toward station 8
Train n. 1, arrived at station 3
Train n. 1, in station 3 going CLOCKWISE
Train n. 0, arrived at station 4
Train n. 0, in station 4 going COUNTERCLOCKWISE
```

Suppose that you have available a function that returns a different pair (station, track) every time it is called:

```
void select_station_and_track(int *station, int *track)
```

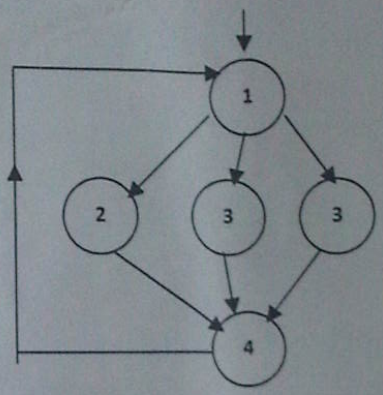
System and Device Programming
22 July 2013

(Theory: no textbooks and/or course material allowed)

(15 marks) The final mark is the sum of the 1st > 8 and the 2nd part > 10

The final mark cannot be refused, it will be registered (no retry for marks >= 18

1. (3.0 marks) Implement this precedence graph **with the minimum number of semaphores**. The threads are 4, and they are **cyclic**. **There are 2 instances of the same threads** (number 3).



2. (3.0 marks) List and explain the steps, the files, and the processes involved in the Unix OS to make the login prompt appear in your screen.
   List the steps, the files and the processes involved **during** the login to finally obtain your shell prompt.

3. (3.0 marks) Explain the role of filter expressions in try...except blocks. Why the following block calls the Filter function ?

```
_try {
      . . .
}
_except (Filter (GetExceptionCode ()))
```

   Is `Filter` a system routine or a user-generated function ? What is the role of `GetExceptionCode ()` ?
   Is it possible to raise a user-generated exception ? (If yes, how and for what purpose; if no, why ?).

4. (3.0 marks) Explain the support for heap management in WIN32 and motivate the advantage of using multiple heaps over libc memory management (based on malloc/calloc). Describe an example where multiple heaps can provide a better solution that a single heap.

5. (3.0 marks)
   a) Describe events in the Windows system and their related system calls. Describe the 4 cases of event signal/release, related to manual/auto reset and set/pulse event conditions.
   b) Write a C code implementation of the a pulseEvent manualReset event (using only mutexes and/or semaphores), under the condition that the number of concurrently waiting threads is <= 32).
   Prothotypes of mutex/semaphores functions are reported below:

```
HANDLE CreateSemaphore (LPSECURITY_ATTRIBUTES lpsa, LONG cSemInitial, LONG cSemMax, LPCTSTR lpszSemName);
HANDLE CreateMutex (LPSECURITY_ATTRIBUTES lpsa,  BOOL fInitialOwner, LPCTSTR lpszMutexName);
BOOL ReleaseSemaphore (HANDLE hSemaphore, LONG cReleaseCount, LPLONG lpPreviousCount);
BOOL ReleaseMutex (HANDLE hMutex);
```