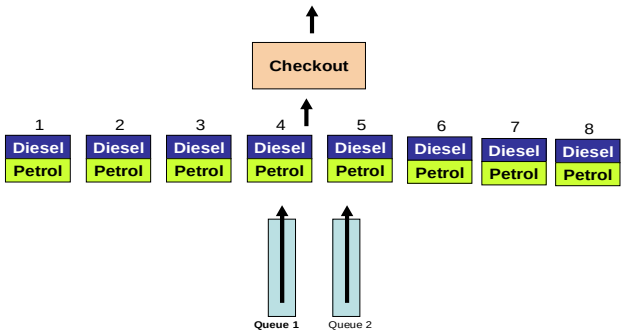
System and device programming
11 September 2015

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

Write a concurrent C program in the Unix environment that simulates the operation of a service station for fuel delivery. The station is equipped with 8 pumps for dispensing fuel and two lanes for the queuing of the cars.



The main thread creates a **checkout thread**, and 8 **pumps threads**.
It also creates **forever** a random number of **car threads** at a time (the max number of **`arrivals`** at a time is given as the first argument of the command line) at random intervals between 1 and **`intervals`** seconds (**`intervals`** is the second argument of the command line), with an increasing integer, starting from 1, as car identifier.

A car thread selects at random the type of fuel it needs, then it waits for a free pump in a randomly chosen queue. The maximum length of each queue is 20.
Pumps 1 to 4 serve preferably cars queued at Queue 1, and if it is empty, cars queued at Queue 2.
Pumps 5 to 8 serve preferably cars queued at Queue 2, and if it is empty, cars queued at Queue 1.

Once a car has access to a pump, the time required for the fuel delivery (generated by the pump at random) is between 1 and **`service`** seconds (given as the third argument of the command line).
After refueling, the pump sends to the checkout thread these data:

- The car thread identifier.
- The type of fuel delivered.
- The delivering time.

The car **has to wait** until the checkout thread computes the cost of the fuel, and allows it to pay and proceed.
The cash amount to be paid, is equal to the delivery time multiplied by 1.6 and 1.4 € for petrol and diesel, respectively.
The checkout thread prints on the screen the thread car id, the number of the queue where the car has waited, and the amount to pay. It then allows the car thread to proceed and to terminate.
Recall that:
`sem_trywait()` is the same as **`sem_wait(),`** except that if it would block, then it does not decrement its counter, instead it returns -1 and sets the **`errno`** variable to **`EAGAIN`** and does not block.
Example of use:

```
result = sem_trywait(...);
if (result == -1 && errno == EAGAIN) { ... }
else {...}
```

System and device programming
11 September 2015

(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) Write the solution with semaphores of the Readers & Writers problem **with Readers precedence, but allows a writer to reserve its entrance to its critical region, blocking the access to new readers, if the last writing operation was performed by more than 10 seconds.** The Writer, obviously, has to wait the end of all the current reading operations.
   Recall that you can use **`sem_timedwait(),`** which is the same as **`sem_wait(),`** except that a second arguments specifies a limit on the amount of time that the call should block if the decrement cannot be immediately performed. Example of use of **`sem_timedwait`** :
   ```
   clock_gettime(CLOCK_REALTIME, &ts); ts.tv_sec += seconds;
   r = sem_timedwait(s_full, &ts);
   ```
   the return value is **`r=-1` if timedout** (the semaphore counter is not decremented) or **`r=0` on success** (the semaphore counter is decremented).
2. (3.0 marks) What is a priority semaphore? Write the functions wait and signal for a priority semaphore.
3. (3.5 marks) Describe the differences between mutexes and semaphores in Win32. What kind of functions can be used in order to synchronize with them ?
   Provide a user implementation of semaphores, corresponding to the following definitions/prototypes:

```
typedef struct {
  …
} USER_SEMAPHORE, *LPUSER_SEMAPHORE;
LPUSER_SEMAPHORE createSem(int initCnt,int maxCnt);
BOOL releaseSem(LPUSER_SEMAPHORE sem);
BOOL waitSem(LPUSER_SEMAPHORE sem);
```

   Definition of the struct and implementation of functions is needed.
   Assume a simple implementation where both `releaseSem` and `waitSem` decrement/increment the semaphore counter by 1.
4. (2.0 marks) What's the difference between a user-generated exception and a system exception? How can a user exception be generated, and how can it be detected ? Briefly describe the encoding used for integer numbers representing exceptions.
5. (3.5 marks) Given the prototype of the (Win32) `CreateProcess` function and the definition of `PROCESS_INFORMATION`

```
BOOL
CreateProcess (LPCTSTR lpImageName,
  LPTSTR lpCommandLine,
  LPSECURITY_ATTRIBUTES lpsaProcess,
  LPSECURITY_ATTRIBUTES lpsaThread,
  BOOL bInheritHandles, DWORD dwCreate,
  LPVOID lpvEnvironment, LPCTSTR lpCurDir,
  LPSTARTUPINFO lpsiStartInfo,
  LPPROCESS_INFORMATION lppiProcInfo);
```

```
typedef struct
_PROCESS_INFORMATION {
      HANDLE hProcess;
      HANDLE hThread;
      DWORD dwProcessId;
      DWORD dwThreadId;
      } PROCESS_INFORMATION;
```

   explain the difference between the `lpImageName` and `lpCommandLine` parameters. Are the two parameters mandatory, or can a caller pass NULL for one of them (motivate the answer)? Why does `PROCESS_INFORMATION` contain two handle and two Id fields? Are `hProcess` and `hThread` pointers to the same object ? Are process/thread Ids and handles equivalent? Given a process/thread, is its Id unique? And are two handles for the same process (e.g. returned by `OpenProcess/OpenThread` or `DuplicateHandle`) equal (i.e. same pointer)?