RPM - Second assignment

Develop, test and execute a program in C/C++ that simulates a simplified Publish/Subscribe communication pattern (see the scheme taken from the 3rd batch of slides).

The program is composed by 2 publishers (producers), 3 subscribers (consumers) plus one mediator for storing and dispatching the items.

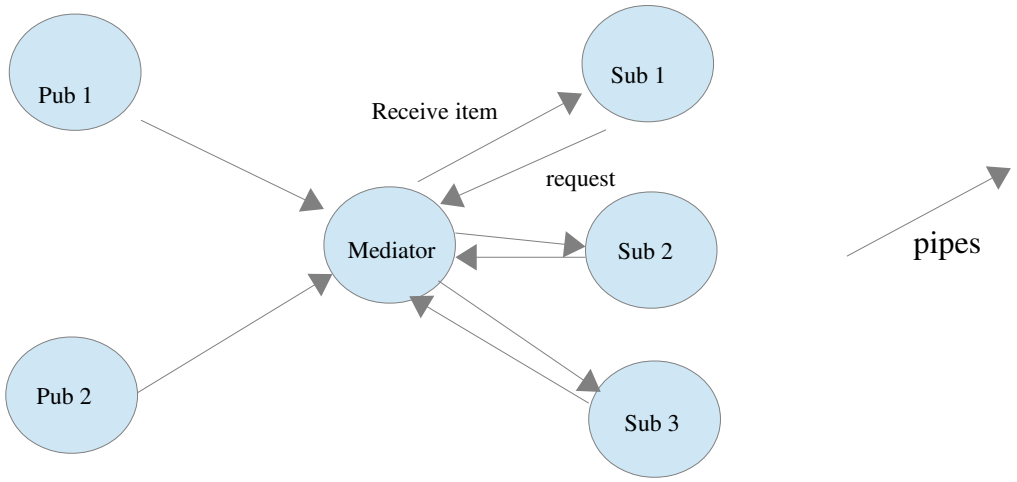
The 6 components are implemented as processes. An additional “init” process begins the execution and successively forks 6 times generating the 6 processes, then waits for completion of them all and exits. Before forking the init process generates a proper number of pipes that will be used for communication between the processes.

Publishers are cyclic as well as the subscribers. All cycle times are different; for example, 1, 2, 3, 4, 5 seconds. The mediator is cyclic as well, but runs at maximum possible speed. For simplicity we assume that subscription is active since the beginning, and that all subscribers did subscribe to all items of both publishers.

Publisher #1 generates items of type “lower case chars”, whereas publisher #2 generates “upper case chars”. All processes write on the screen what they are doing, for checking that the program works fine.

After that, develop the same model using threads instead of processes.

A possible scheme depicting the software components is the following:



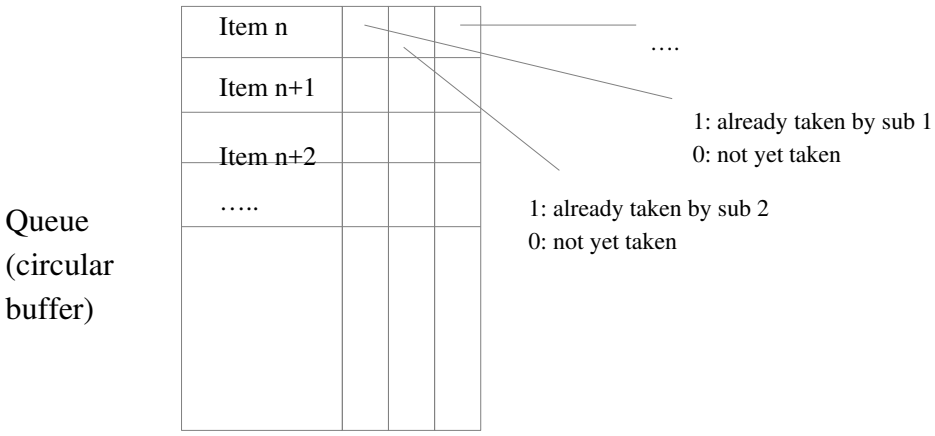
To get an item each subscriber first send a request to mediator, and immediately after waits for the item sent by the mediator as response to the previous request.

Hints(1)

Use the following Posix system calls:

- fork() to generate processes; exec() to execute the code of each process
- pipe() to generate pipes
- wait() to wait for child completion
- read(), write() to read/write from/to a pipe

The mediator stores items produced by the subscribers. A possible data structure is the following (for items #1, a second copy is needed for items #2).



Hints(2)

- The circular buffer can be easily implemented using a vector with two pointers to implement a queue
- Choose a suitable length for the queue
- In an advanced solution items “too old” (after many cycles of the mediator) not taken by subscribers might be removed
- The mediator cyclically tries to read from pipes connected to subscribers a “request” to satisfy. *How can the mediator listen to all subscribers to accept the next request? The mediator MUST NOT read in deterministic sequence the pipes carrying the requests of the subscribers.*

The correct answer is: use the ***select()*** system call. It is described in many documents in the net. See for example the attached article.

Pthreads version:

No pipes are needed; threads communicate through shared memory.
Threads are exactly the same as the previous processes.

Use the following Posix system calls:

- pthread_create() to generate threads
- pthread_join() to wait for child completion
- pthread_mutex_lock/unlock for mutual exclusion and synchronisation