McMaster University
Department of Computing and Software
Dr. W. Kahl

SFWR ENG 3BB4 — Software Design III — Concurrent System Design

22 March 2007

This assignment is due on Monday, April 2, at 9:30

Your are participating in the development of a game server for a **multi-user dungeon-style game**.

In the game, there are players and "loose objects"; every loose object is, at every point in time, either located somewhere in the dungeon, or it is being carried around by a player. Players can pick up and put down loose objects; these actions are considered to be instantaneous. New players also can join the game at any time.

(a) Describe a particular game **of your own design** that preserves the essential characteristics described above and adds some interesting features. ***Read the whole assignment first, and design (a) so as to create tricky problems in (e) that you can solve nicely in (g)!***

(b) Describe **and document** the **essential data structures** that you propose to be kept in the server **for your game** from (a). These essential data structures should only concentrate on supporting the game, and not yet on synchronisation issues (but see (g) below). You should keep in mind that players usually will need fast direct access to the items they are carrying around.

- Provide a mathematical design of these data structures. Typically this will involve some base sets, and constructions like powersets, cartesian products, direct sums (disjoint unions), and sets of partial functions, total functions, or relations.
- Provide datatype definitions in C to implement this design.

(c) How do you propose to deal with players that carry around important loose objects but just stop to play without notifying the server?

(d) Give a game-independent definition of which objects are "important" in the sense of (c).

(e) Identify at least three different ways — see (f) — how players' actions might give rise to race conditions over server-side data structures, unless proper protection mechanisms are used.

(f) For each of these race conditions, state the safety properties that are violated in case of interference. ***Try to make these different!***

(g) Describe how you would prevent the interferences described in (e), taking into account that the server should be able to deal concurrently with as many player actions as possible.

- Describe the changes this implies for the data structures described in (b).
- Explain why each of the race conditions described in (e) is now avoided.
- Explain how efficiency affected the design.

(h) **Design** and implement a **proof of concept implementation** of your server design:

- There is **at least one thread per client** in the server.
- Communication with each client is via two FIFOs. This allows you to test the server using a shell script of the following shape (this example is available on the course page):

```
#!/bin/bash
for i in 1 2 3
do
   # make FIFOs for client $i:
   #
   mkfifo client$i”out client$i”in
   #
   # start observer terminal for client $i:
   #
   xterm -e "cat < client$i”out" &
done
#
# Start my server:
#
./myDungeonServer 3 &
#
# Open client input FIFOs:
#
exec 3<> client1in
exec 4<> client2in
exec 5<> client3in
#
# And play!
#
sleep 1
echo "pickup jewel" >&3; echo "turn left" >&4
sleep 1
echo "raise wand" >&4; echo "drop coat" >&3; echo "move forward" >&5
```

- You may need to arrange on your screen the `xterm` terminal windows opened in the loop, so you see what is happening on all of them.
- “`exec 3<> client1in`” makes the input FIFO of client 1 available as file descriptor 3 in the shell, and does not close it after each “`echo "xyz" >&3`”.
- “`echo "pickup jewel" >&3; echo "turn left" >&4`” feeds a line containing “`pickup jewel`” to the input of client 1 (file descriptor 3), and, “at almost the same time”, feeds a line containing “`turn left`” to the input of client 2 (file descriptor 4).

The following simplifications are allowed:

- For this proof-of-concept, you can assume the number of clients to be given as command-line argument to the server.
- Not all the functionality described in (a) needs to be implemented, but some avoidance of possible interference needs to be demonstrated.
- Client input can be designed to be easily handled by your program.

“Bells and whistles”, the option to use network sockets (textbook chapter 15) instead of FIFOs, and additional client software can get bonus marks!

Deliverables:

(1) The **log** recording your actions on this assignment, in particular any interactions with other students.

This assignment is to be solved individually!

(2) A single, well-organised document containing **your answers** to the problems above, including the **design** of the proof-of-concept implementation. This **must** be handed in **on paper**.

(3) The source code of your proof-of-concept implementation (including Makefile etc.), together with a test shell script similar to the one in the box above, bundled as a .tar.gz file containing a single top-level directory, and there at least a REDME file explaining the other files in the package. This should be sent by e-mail to “kahl@mcmaster.ca”.