CSE 42 DIS, Distributed Computing. Assignment No. 1.
Uncontrolled Simulation Version
Due date and time: TBA (Tentatively, 22nd of August, Monday, 9.30 a.m. 2005)

1. **Objectives:** The purpose of the assignment is to write a program that creates a number of processes which can communicate with each other. These processes will send messages to each other in some random order and after random intervals of time. The exercise should make the programmer more familiar with system calls such as **fork(), open(), close(), pipe(), write(), read(), etc.**,. and also give experience in the use of Unix **sleep(), rand() etc.**. The multiple processes will be created using the fork() system call (see below for more details).

This assignment will also prepare a prototype that can be used in the second assignment as a starting point. **Therefore preserve your assignment CODE carefully. Back it up on a floppy, if necessary. The final program for this assignment should not need more than a couple of hundred lines of code (including comments etc.) whether you use C, or C++ (a subset of C++ will do, as no objects are really required here). I recommend you use simple C. If you don't have a working assignment 1, you cannot hope to get assignment 2 working. So please ensure you get this one working. (start now).**

2. Each of the processes created must run continuously, repeatedly carrying out the following sequence of actions.

(a) Send a message to a *randomly chosen* "colleague".
(b) Do nothing (*sleep()* system call) for a *randomly generated* length of time.
(c) On waking , check to see if any messages are waiting. If so "consume" them (read them from the pipe and print on the log file described in para 3 below).
(d) Go back to step (a) and continue.

For choosing random numbers make use of ready-made functions available in the library.

In the step 2(c), take care to read any or all messages that may be waiting. After sending 20 messages, and consuming whatever has arrived, the process should terminate. Notice, that the number of messages received by each process is randomly determined, and therefore the process cannot wait indefinitely for more messages to consume. Some messages may therefore never be consumed, because they arrive after the destination process has terminated.

****Hint:** in step 2(c), you may have a problem because the process blocks when the communication channel (the pipe) is empty. Read up on reading from a pipe to find out why this happens and what to do about it.. The final answer will be found by using non-blocking versions of these calls.

3. Each process must write into a log file, details of each message sent, and each message "consumed" (i.e. that was read from the pipe). **This log file must be common to all the processes. You will find it necessary to take special care to see that the ordering of the messages in the file reflects the ordering of events "as they occurred".** This has to do with the write buffers for the file. You are expected to find out more by reading about i/o buffers in Unix and what to do about them. (**Hint:** The log file will also help you debug your program.)

4. In order that messages are uniquely identified, they must consist of at least three fields viz. **sender, receiver, sequence number (from the sender).**

5. To create the processes you need to use the fork() system call which will create a copy of the parent as a child process. To make *n* processes, you may let each child create its own child or let the parent create many children. This can be done in a loop which runs *n-1* times or even *n* times (if you choose to let the parent not take part in the communication).

Please experiment carefully with the fork() call creating only two processes before you run fork() in a loop! It is easy to misunderstand the semantics of fork() and create an "endless" number of children, causing you (and the system admin) lots of grief! Similarly, it is easy to create a process and leave it hanging around in the system, so please ensure that all your processes terminate properly.

To organise the communication amongst processes, you will need to create "pipes" between the processes. To understand how to do this, you need to understand the pipe() system call and how it works. (any book on Unix programming e.g. Stevens will do). Basically you only need to create n pipes if there are n processes. All process can write into or read from any of the pipes, but they should follow a simple convention: all messages sent to process "i" will be written into pipe "i", and only process "i" will read from that pipe and only from that pipe. The pipes must be created before the process forks, so that all pipes are available to each child as well as the parent.

6. Your submission must run on the machine **latcs6** (for obvious reasons of standardisation).

Electronic submission: Please attach all your code, makefiles, and a copy of the logfile etc., most conveniently in a single directory, and use the "submit" command on the system. Something like submit cse42dis <directory_name>, will do the trick (please check for the latest version of this command). The logfile should contain the results of running your program with $n = 4$ or 5.

Electronic submission from your authenticated account IS DEEMED to comply with the University secretary's requirement as printed in the green forms that are attached to hard-copy submissions. This means that you are claiming that the work is entirely your own. Please see the note regarding plagiarism below.

8. **Marking Scheme:**

(a) processes correctly created, and log file set up **4 marks**

(b) processes also sending and receiving messages correctly (as determined from running the code, or from other evidence) **4 marks**

(c) log file properly organised and showing events roughly as they occur. **2 marks**

(d) Obviously, a program that does not compile will get zero marks (this is a 4th year subject!).

9. You should consult Unix (Linux) programming books (e.g Stevens) and the "manual pages" for specific information on how the system calls work. In fact the manual pages give examples of how the calls are usually done. Use the **man** command: **man** <command>® tells you all about <command> . **man** man ® will tell you how to use the **man** command. Other on-line documentation is also available under Linux.e.g. "info".

The final program must be general enough to run with n =20, if required. It might be a good idea to read *n*, the number of processes from the command-line arguments using $argv.

10. Plagiarism: You are quite free to discuss amongst yourselves and to learn from each other. Indeed discussion/study groups are encouraged. However, **you must write your own code**. If the code you submit could have been obtained by copying someone else's submission and making some cosmetic changes with an editor, you are liable to be suspected of plagiarism. **Assignment 1 is the same as given to some students last year.** However, all of those are stored with me and will be compared. **For detailed school policies please refer to the Policies document available from the general office. The university's plagiarism policy is also on the University web-site.**

11. IMPORTANT: After the assignments have been marked, the list of marks will also be displayed in the 4th year subject web page against your student number. **Please verify that your marks have been correctly entered in the list, and report any errors or omissions to the lecturer before the end of the semester. If anyone has privacy concerns about this procedure, please e-mail me beforehand and your marks will not be displayed.**