Assignment 1

COMP 2320 --- 1st Semester, 2006-2007
Due: Wednesday, 1st of November, 2006 (5:00pm)

Objective
In this assignment, you are going to learn about multi-process programming using fork( ) and semaphores and multi-thread programming using the POSIX pthread lib and mutex.

1. You have to write a reader/writer program with reader's priority as described in Figure 5.22 and Page 242-243 in our textbook. (50 marks)

   Program Description:
   The program you are going to write will use the fork( ) function call and spawn four processes: two readers and two writers. All four processes will read/write a common file called "SharedFile". The access of this "SharedFile" and the coordination among the four processes are controlled by 2 semaphores as described in Figure 5.22. Beside the SharedFile, the file "ReadCount" is to keep track of the number of readers waiting to read the file, and the file "ProcessDone" is to synchronize all the processes and let the parent process to close the 3 semaphores properly.

   The operations of the reader and writer can be briefly described as follows:

Reader	**Writer**

Repeat for 20 times
  <enter Critical Section x>
  Implement Reader's Priority by "ReadCount"
  <exit Critical Section x>
  Read the number from the file "SharedFile"
  Report the action and the SharedFile situation
  <enter Critical Section x>
  Implement Reader's Priority by "ReadCount"
  <exit Critical Section x>
  sleep for (100 to 200 msec)
	Repeat for 20 times
	  <enter Critical Section wsem>
	  Read the number from the file "SharedFile"
	  Increment the number randomly (from 1 to 6)
	  sleep for (100 to 200 msec)
	  Write back the number to the file "SharedFile"
	  Report the action and the SharedFile situation
	  <exit Critical Section wsem>
	  sleep for (100 to 400 msec)

   Implementation Details:
   - You should use the semaphore package "sem_pack.h" available at the webpage of this class -- (http://www.comp.hkbu.edu.hk/~jng/comp2320.html) for this exercise. Study the sample programs on the usage of the package. Use the last 5 digits of your student ID as your first semaphore, and add 1 to the number for your second and third semaphore (you only need 2 semaphores for the reader/writer synchronization, the third semaphore is for proper closing of all 3 semaphores by the parent process).
   - You should use the *fork( )* system call to spawn out the four processes. That is, when you run your program, four processes will be created – 2 readers and 2 writers.
   - Your program should also generate the THREE files: SharedFile, ReadCount, and ProcessDone, all with '0' as the initial value.
   - A demo program will be available to show the look and feel of this exercise.
   - I assume you run your program on the cslinux machine.

```
/* program readersandwriters */
int readcount;
semaphore x = 1, wsem = 1;
void reader()
{
    while (true)
    {
        semWait (x);
        readcount++;
        if (readcount == 1)
            semWait (wsem);
        semSignal (x);
        READUNIT();
        semWait (x);
        readcount--;
        if (readcount == 0)
            semSignal (wsem);
        semSignal (x);
    }
}
void writer()
{
    while (true)
    {
        semWait (wsem);
        WRITEUNIT();
        semSignal (wsem);
    }
}
void main()
{
    readcount = 0;
    parbegin (reader, writer);
    }
```

Figure 5.22 A Solution to the Readers/Writers Problem Using Semaphores: Readers Have Priority

Readers Have Priority

Figure 5.22 is a solution using semaphores, showing one instance each of a reader and a writer; the solution does not change for multiple readers and writers. The writer process is simple. The semaphore **`wsem`** is used to enforce mutual exclusion. As long as one writer is accessing the shared data area, no other writers and no readers may access it. The reader process also makes use of **`wsem`** to enforce mutual exclusion. However, to allow multiple readers, we require that, when there are no readers reading, the first reader that attempts to read should wait on **`wsem`**. When there is already at least one reader reading, subsequent readers need not wait before entering. The global variable **`readcount`** is used to keep track of the number of readers, and the semaphore **`x`** is used to assure that **`readcount`** is updated properly.

2. You have to write a game similar to “Typing of the Dead” and practice multi-thread programming. For multi-thread programming, you will need to call functions from the POSIX “pthread” library and use “mutex” to enforce mutual exclusion. (50 marks)

   Program Description:
   The program you are going to write will use functions from the POSIX pthread library and mutex for enforcing mutual exclusion to access shared data structures. You also need to use the “ncurses” library to update the tty screen. Basically, in this exercise, you program will have to create 2 threads, one for updating the screen, and one of capturing the typing from the user. There should be some data structures or variables to keep track of the content of the screen and mutual exclusion have to be enforced.

   We assume you are working on a 80 columns by 25 rows TTY terminal. You define a window/screen of 40 columns by 20 rows for the dropping of the 32 C language keywords (see the appendix). Every 2 seconds, a random keyword at a random horizontal position will be dropped at the top of the screen. You can type the keyword at the bottom of the screen and if you type it right, the keyword(s) will disappear. And if you type the letter ‘Q’, the program will quit. I will show a demo version of this program to give you the look and feel of this game.

   A lot of improvement can be done around this game, the followings are just some of them:
   - Backspace key: I have not implemented the “backspace” key for the user, so the user must type every character correctly;
   - Speed of Drop: The original game will increase the speed of drop as you play along;
   - Color keywords: It is an easy extension for the ncurses library;
   - Scoring: I didn’t put much thought on the scoring of the game. I only implement a penalty when a keyword hits the bottom of the screen;
   - More or Speed: Someone suggest each word has a different dropping speed;
   - Keyword generation: Someone suggest that keyword generate at the top of the screen is subjected to a probability, hence, not every time (2 seconds), a keyword is generated;
   - Better display and outlook: The look and efficiency of the game can always be improved;

Appendix: The keywords for the C language.
"auto", "break", "case", "char", "const", "continue", "default", "do", "double", "else", "enum", "extern", "float", "for", "goto", "if", "int", "long", "register", "return", "short", "signed", "sizeof", "static", "struct", "switch", "typedef", "union", "unsigned", "void", "volatile", "while"

Some useful hints for this assignment
I have use the following functions for the TWO demo programs:

Reader/Writer:

#include "sem_pack.h"	*#include <stdio.h>*	*#include <stdlib.h>*	*#include <sys/types.h>*
sem_create()	fopen()	srand()	fork()
sem_close()	fclose()	rand()	getpid()
sem_wait()	fscanf()	#include <time.h>	usleep()
sem_signal()	fprintf()	time()	

Typing of the Dead:

#include <ncurses.h>	*ncurses (continue)*	*#include <pthread.h>*	*Others*
initscr()	clear()	pthread_create()	srand()
raw()	refresh()	pthread_join()	rand()
keypad()	endwin()	pthread_mutex_lock()	usleep()
noecho()	mvprintw()	pthread_mutex_unlock()	

Some useful commands that you may need for this assignment
You should check out the manual page for these commands (e.g. "%man ipcs") for their usage.

ipcs	provide information on ipc (semaphore) facilities
ipcrm	remove the resource(s) (e.g. semaphore) specified by id
ps	list the information about all processes under your login
kill	kill the process specified by pid

To compile with the pthread and ncurses library, you have to include the lib into the gcc command:
gcc –o totd totd.c –lpthread -lncurses

How to submit your program
Create a directory called comp2320 under the "Home" directory of your CSLINUX account. And create a directory called assign1 under comp2320. Upload your source programs xxxxxxxx-totd.c and xxxxxxxx-readerwriter.c, where xxxxxxxx is your login ID.

Reference Links for this assignment

In general, you should go to do a google search on "POSIX pthread", and "ncurses". But here are the two links I use for writing the demo program.

Linux Tutorial: **POSIX** Threads
http://yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html

NCURSES Programming HOWTO
http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/

For the use of the UNIX fork( ) function/system call, and the semaphore package, please refer to the class web page at http://www.comp.hkbu.edu.hk/~jng/comp2320.html