Homework 1
Operating Systems, V22.0202
Fall 2006, Professor Yap

Due: Wed Sep 20, in class

INSTRUCTIONS:

- Please read questions carefully. When in doubt, please ask.

---

1. (2 Points each)

   (a) Exercise 1.1 (p.26), 2 purposes of an OS.

   (b) Exercise 1.3 (p.26), advantages of multiprogramming.

   (c) Exercise 2.9 (p.56), use of caches.

   (d) Exercise 3.6 (p.98), OS services.

2. (10 Points)
   An alert reviewer notices a consistent spelling error in the manuscript on Operating Systems textbook, about to go to press. The book has about 700 pages, each with 50 lines of 80 characters each. How long would it take to process (i.e., make one pass through) the manuscript, assuming that the manuscript is in the following levels of the memory hierarchy:

	TYPE	CAPACITY	ACCESS TIME
(a)	Registers	< 1KB	1 nsec
(b)	Cache	1 MB	2 nsec
(c)	Main Memory	1 GB	10 nsec
(d)	Magnetic Tape	50 GB	10 msec
(e)	Magnetic Tape	100 GB	100 sec

   For internal storage methods, assume the access time is per character, for disk devices, assume the access time is per block of 1024 characters, and for tape assume the access time is to the time to get to the start of the data, with subsequent access time similar to disk access.

3. (62 Points)
   This question assumes that you have downloaded Cygwin on your computer. Also, you have downloaded gcc, make and tar programs. Please look at our class links for this information.

   Enclosed is a simple program called `fork.c` and `test.c` that calls fork() and exec*(). They are put into a tar file called "homework1.tar". You are to write a variation of these programs.

   - 0. Read our quick introductions to Cygwin, C programming and Make program. These are links under RESOURCES on class page.
   - 1. Untar the file homework1.tar (move the file to a directory and type "tar -xvf homework1.tar"). Among the files, you will see "fork.c" and "Makefile". Please compile fork.c by typing "make". If gcc compiler creates an executable file called "a.out", you can execute the program by typing "a.out". [Note: a.out is the standard compiler output on UNIX]
   - 2. Please create a variation of the fork.c program which will create an arbitrary number of processes until your system crashes (there might be some other possibilities, depending on your system).
     BE SURE THAT ALL YOUR PROCESSES ARE KEPT AROUND FOR THE DURATION OF THE EXECUTION, NOT JUST CREATED TO DIE IMMEDIATELY. For safety, we suggest stopping all other programs before you run the test.
   - 3. There is a simple make program also provided here. You must use this to help you programming and testing.

- 4. Read the following submission rules carefully, as up to 15 percent of this question may be deducted for non-compliance.
  – Make sure that your program is well-documented (long and short comments)
  – Make sure your name, contact information, and any other information you like, is given in a README file.
  – ANY ACKNOWLEDGEMENT OF SOURCES SHOULD BE EXPLICITLY MADE HERE. Otherwise, it is considered plagiarism.
  – Your C program file name MUST be called h1.c.
  – You must put h1.c, README, Makefile, and any other necessary files into a tar file called h1.tar. (Type "tar cvf h1.tar h1.c Makefile ..." to create this tar file).
  – Send the tar file to the grader (but cc to me). After we untar your file, we expect to type "make" to compile it, and to type "make test" to run the compiled program.

```
// file: test.c
// Comp.Sys.Org.II, Spring 2005, Yap.
// The following code is a simple modification of the "fork.c" code from
//
// Dr Ian G Graham, Department of Information Technology, GUGC
// Copyright  2000-2003
//
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <sys/wait.h>

extern int errno;     /* system error number */

void syserr(char* );  /* error report, abort routine */

int main(int argc, char *argv[])
{
   pid_t pid;
   int rc;

   pid = getpid();
   printf("Process ID before fork: %d\n", pid);

   switch (fork()) {
      case -1:
         syserr("fork");
      case 0:

/* execution in child process */
         pid = getpid();
         printf("Process ID in child after fork: %d\n", pid);
         execlp("echo", "0", "Child to world: Hello!",NULL);
         syserr("execl");
   }

/* continued execution in parent process */

   pid = getpid();
   printf("Process ID in parent after fork: %d\n", pid);

   //getchar(); // hang until I type something (alternatively,
    // let parent wait for return of child process.
   exit(0);
}

void syserr(char * msg)
{
   fprintf(stderr,"%s: %s", strerror(errno), msg);
   //abort(errno); // variant
   abort();
}
```