PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

DECEMBER EXAMINATIONS 2001

CSC 209H1 F
Duration — 3 hours

Examination Aids: ***None***

Student Number: |_|_|_|_|_|_|_|_|_|

Last Name: ____________________

First Name: ____________________

Lecture Section: L0101 **Lecturer: Reid**

---

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above,
and read the instructions below *carefully*.)

This final examination consists of 9 questions on 13 pages (including this one), *When you receive the signal to start, please make sure that your copy of the examination is complete.* Answer each question directly on the examination paper, in the space provided.

Be aware that concise, well thought-out answers will be rewarded over long rambling ones. Also, unreadable answers will be given zero so write legibly.

You do **not** need to include header files or do error checking in C programs except where specifically mentioned in a question. You do not need to include the "#!" line in shell or perl programs. The last two pages of this exam contain a list of C function prototypes structs, and some Perl and Bourne shell details.

# 1:	______/10
# 2:	______/ 5
# 3:	______/ 9
# 4:	______/ 6
# 5:	______/12
# 6:	______/10
# 7:	______/12
# 8:	______/12
# 9:	______/14
TOTAL:	______/90

Good Luck!

Question 1. [10 MARKS]

Circle the correct answer for the following questions:

TRUE	FALSE	A process running in the background is an orphaned process.
TRUE	FALSE	Signals are used to send several bytes of data from one process to another process.
TRUE	FALSE	A process is in the blocked state if it is waiting for a child to terminate (i.e., it has called `wait(&status)`).
TRUE	FALSE	An inode stores the contents of a file.
TRUE	FALSE	An inode contains a process id.
TRUE	FALSE	A signal can be generated by a software or hardware event.
TRUE	FALSE	The kill command is only used to terminate processes.
TRUE	FALSE	The following two commands will display the same list of file names: `find . -name "a209*" -print`, `ls ./a209*`
TRUE	FALSE	A new process is created in Unix only when a running process calls `fork()`

.

Question 2. [5 MARKS]

What is the output of the following set of Bourne shell commands, given that the current working directory contains the following files? (' is a forward quote, and ` is a backward quote).

```
a.c    b.h    d.ch   f.c
```

cmd="ls"

```
echo $cmd *         ____________________
echo *.[ch]         ____________________
echo "$cmd *.c"     ____________________
echo '$cmd *.c'     ____________________
echo `$cmd *.c`     ____________________
```

Question 3. [9 MARKS]

In the program below, assume that all processes run until they terminate normally, and the shell that executes the program does not terminate.

```
int main() {
    int status, p1, p2;
    p1 = fork(); /*child 1*/

    if(p1 == 0) {
        printf("A\n");
        p2 = fork();/*child 2*/
        if(p2 == 0) {
            printf("C\n");
        } else {
            printf("D\n");
        }
        wait(&status);

    } else {
        printf("B\n");
    }

    printf("All done\n"); exit(0);
}
```

Part (a) [3 MARKS]

Write the output of this program in a valid order.

Part (b) [3 MARKS]

For each process in the above program, state whether it is possible for the process to become a zombie. If it is possible for the process to become a zombie, explain the circumstances under which it could become a zombie, otherwise explain why the process could not become a zombie.

Part (c) [3 MARKS]

For each process in the above program, state whether it is possible for the process to become an orphan. If the process can become an orphan explain the circumstances under which it would be orphaned, otherwise explain why the process could not become an orphan.

Question 4. [6 MARKS]

When the following Perl patterns are applied to the string `$s1`, write the value of `$1` if the pattern matches the string. If the pattern does not match the string, write "undef"

```
$s2 = "Don't Cry Joe 3:03 1955";

                     /(\w+)/  $1 = ____________________

                    /(\w+)$/  $1 = ____________________

                     /(\d+)/  $1 = ____________________

                /(\d:\d\d)$/  $1 = ____________________

                   /([\d:]+)/ $1 = ____________________

/([\w\s]+)\s+\d+:\d+\s*\d?/   $1 = ____________________
```

Question 5. [12 MARKS]

Part (a) [3 MARKS]

List three types of software tools, other than an editor and a compiler, that a professional programmer should be able to use well, and give a one-sentence description of the purpose of each.

Part (b) [4 MARKS]

A programmer chooses a language based on the type of problem to be solved. Give one reason why a programmer might choose a compiled language over an interpreted language, and one reason why a programmer might choose an interpreted language over a compiled language for a particular problem.

Part (c) [5 MARKS]

Semaphores may be used keep track of the references to some resource (e.g., a file, or a shared data structure). The creator of the resource also creates and initializes the associated semaphore. The creator uses the semaphore to determine when it is safe to delete the resource and the semaphore. It is safe to delete a resource when there are no more references to it. Other processes use the sempahore to indicate when they are actively using the resource (e.g., they have a file pointer to an open file, or a pointer to a shared data structure).

Explain, in English, the semaphore operations required 1) by the creator of the resource and 2) by other processes using the resource. Be clear about when processes might block, and the value of the semaphore variable at each stage.

Question 6. [10 MARKS]

Write a C program that uses `fork` and `pipe` to set up the following pipeline and execute it.

```
grep a209 /u/csc209h/submitted | wc
```

Question 7. [12 MARKS]

A given remote data server receives a connection request from a client, and reads a file name from the client. Then it writes the contents of the file to the client, one line at a time (a line will be no longer than `LINE_SIZE`). When it finishes sending the file, the server closes the connection.

Complete the following socket **client** program that sends requests to the remote data server for the files given in `names`, creates the file, and writes the lines sent from the server to the file.

You must ensure that the client closes the socket, even when an error occurs. Recall that `htons()` must be called to convert a port into network byte order, and that the socket type will be `AF_INET`.

```
#define SERVER_PORT  30000
#define LINE_SIZE 16

char *names[5] = {"afile", "bbfile", "cfile", "df", "efile"};
char *hostname = "penguin";

int main(void) {

       struct hostent *hp = gethostbyname(hostname);
```

Question 8. [12 MARKS]

Part (a) [8 MARKS]

Write a C program that creates 4 processes. Each of the four processes calls the function `do_child()` defined below, and returns the result of `do_child()` as its termination code. The parent process will print to standard output the value of each of the child's termination code. If a child process receives a SIGTERM signal the child should call a signal handler that terminates the child process, passing the current value of `(result/count)` as the child's termination code. The parent process should not modify the behaviour of any signals. Several global variables have been declared for you. You may need to write one or more functions in addition to main.

Part (b) [4 MARKS]

Modify `do_child()` so that it blocks SIGTERM while it is executing the two lines marked with an `X`. Write the additional code beside the function `do_child()` and indicate with arrows where each piece of the added code should go.

```
int child_result;
double result;
int count;

int do_child( void ) {
    result = 0.0;
    count = 0;

    while(count < 10) {
        result += runexp();   /*X*/
        count++               /*X*/
    }
    return result / count;
}
```

(*Complete part (a) below*)

An extra page if you need it.

Question 9. [14 MARKS]

Assume you are given the source code and a Makefile which compiles an executable called `updateFile`. The specifications of `updateFile` are given below. Your job is to write two programs that can be used to test `updateFile`: a Perl program that will check if the output of `updateFile` follows the specifications, and a Bourne shell program that runs the tests. Do **not** write `updateFile`. You may assume that the source code and Makefile to produce the executable `updateFile` are in the current working directory.

The program `updateFile` creates or appends to a file the output from the parent and a child process. It is run as "`updateFile -p <pl> -c <cl> -f <fname>`" where `<pl>` is the number of lines the parent process writes to the file `<fname>` and `<cl>` is the number of lines the child process writes to `<fname>`. A line written by the parent begins with "Parent", and a line written by the child begins with "Child". `updateFile` uses a semaphore to synchronize the writes to the file.

Part (a) [5 MARKS]

Write a Perl program that takes two arguments and reads from **standard input**. The first argument is the number of lines of input expected to begin with the word "Parent" and the second argument is the number of lines expected to begin with the word "Child". It will print "Correct" to standard output if it found the correct number of lines beginning with "Parent" and "Child", and "Wrong" otherwise.

Part (b) [9 MARKS]

Write a Bourne shell program to test `updateFile`. The shell program uses the Perl program in part a) which is stored in a file called `matchOut` to check if the output produced by `updateFile` conforms to the specifications.

The shell program produces output that answers the following questions. Assume all necessary files are in the current working directory:

- Does the makefile correctly produce the executable?
- If the file "afile" exists, does "updateFile -p 5 -c 10 -f afile" produce the expected output? Assume "afile" contains 3 lines beginning with "Parent" and 4 lines beginning with "Child".
- If the file "bfile" does not exist does "updateFile -p 1 -c 1 -f bfile" produce the expected output?
- Does updateFile remove the semaphore when it terminates? (Assume that the user is not running any other programs requiring semaphores.) If the semaphore is not removed, the shell program should remove it.

Some commands you will find useful are "`whoami`" which prints the name of the user, "`ipcrm sem <key>`" which removes a semaphore, and "`ipcs -s`" which produces output of the form:

```
key        semid     owner     perms     nsems     status
0x0006f385 5177350    a209xxxx  666        1
```

```
int accept(int sock, struct sockaddr *addr, int addrlen)
int bind(int sock, struct sockaddr *addr, int addrlen)
int close(int fd)
int closedir(DIR *dir);
int connect(int sock, struct sockaddr addr, int addrlen)
int execl(const char *path, char *argv0, ..., (char *)0)
int execlp(const char *file, char *argv0, ..., (char *)0)
int execv(const char *path, char *argv[])
int execvp(const char *file, char *argv[])
int fclose(FILE *stream)
int FD_ISSET(int fd, fd_set &fds)
void FD_SET(int fd, fd_set &fds)
void FD_CLR(int fd, fd_set &fds)
void FD_ZERO(fd_set &fds)
char *fgets(char *s, int n, FILE *stream)
int fileno(FILE *stream)
pid_t fork(void)
FILE *fopen(const char *file, const char *mode)
int fprintf(FILE *stream, const char *format, ...)
struct hostent *gethostbyname(const char *name)
int kill(int pid, int signo)
int listen(int sock, int n)
int open(const char *path, int oflag)
DIR *opendir(const char *name);
int pclose(FILE *stream)
int pipe(int filedes[2])
FILE *popen(char *cmdstr, char *mode)
struct dirent *readdir(DIR *dir);
ssize_t Readline(int filedes, void *buf, size_t maxlen);
ssize_t Readn(int filedes, void *buf, size_t nbytes);
int select(int maxfdp1, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)
int semctl(key_t key, int semnum, int cmd, union semun arg)
        /*cmd has the value SETVAL, GETVAL, IPC_RMID */
int semget(key_t key, int nsems, int semflags)
int semop(int semId, struct semops *sem_ops, int nops)
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
int sigaddset(sigset_t *set, int signum);
int sigemptyset(sigset_t *set);
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
       /*how has the value SIG_BLOCK, SIG_UNBLOCK, or SIG_SETMASK */
unsigned int sleep(unsigned int seconds);
int socket(int family, int type, int protocol)
int sprintf(char *s, const char *format, ...)
int stat(const char *file_name, struct stat *buf);
char *strncat(char *dest, const char *src, size_t n);
int strncmp(const char *s1, const char *s2, size_t n);
char *strncpy(char *dest, const char *src, size_t n);
int wait(int &status)
int waitpid(int pid, int *stat, int options)
void Writen(int filedes, const void *buf, size_t nbytes);

WIFEXITED(status)       WEXITSTATUS(status)
WIFSIGNALED(status)     WTERMSIG(status)
WIFSTOPPED(status)      WSTOPSIG(status)
```

```
struct sockaddr_in {
  sa_family_t          sin_family;
  unsigned short int   sin_port;
  struct in_addr       sin_addr;
  unsigned char        pad[8];        /* Unused */
};

struct hostent {
   char    *h_name;        /* official name of host */
   char    **h_aliases;    /* alias list */
   int     h_addrtype;     /* host address type */
   int     h_length;       /* length of address */
   char    **h_addr_list;  /* list of addresses */
}

struct sigaction {
   void (*sa_handler)(int);
   sigset_t sa_mask;
   int sa_flags;  /* 0 */
}
```

Shell and Perl comparison operators

Shell	Perl	Description
-d filename	-d filename	Exists as a directory
-f filename	-f filename	Exists as a regular file.
-r filename	-r filename	Exists as a readable file
-w filename	-w filename	Exists as a writable file.
-x filename	-x filename	Exists as an executable file.
-z string	string eq ""	True if empty string
str1 = str2	str1 eq str2	True if str1 equals str2
str1 != str2	str1 ne str2	True if str1 not equal to str2
int1 -eq int2	int1 == int2	True if int1 equals int2
-ne, -gt, -lt, -le	!=, >, >=, <, <=	For numbers
!=, >, >=, <, <=	ne, gt, lt, le	For strings
-a, -o	&&, ||	And, or.

Perl pattern matching:

\s	space
\w	word
\d	digit
[]	one character of a set
[^]	any character not in set
+	one or more
*	zero or more
?	zero or one
.	any character

Meta characters that need to be escaped are
`\ | ( ) [ { . $ ^ ? +`

Perl functions:

```
push(ARRAY, LIST)
pop(ARRAY) -- returns LIST
sort BLOCK LIST -- returns LIST
defined(SCALAR) -- returns true if SCALAR is defined, false otherwise
split(/PATTERN/, SCALAR) - returns LIST
```