PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL EXAMINATIONS 2007

CSC 209H1 S
St. George Campus

Duration — 3 hours

Examination aids: One 8.5 x 11 sheet of paper (double-sided)

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

Last Name: _____

First Name: _____

Instructor: _____

---

*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.)

This examination consists of 9 questions on 18 pages (including this one). *When you receive the signal to start, please make sure that your copy of the examination is complete and fill in your student number on every page.* If you need more space for one of your solutions, use the last pages of the exam or the back of this page and *indicate clearly the part of your work that should be marked.*

1: _____/ 9

2: _____/ 6

3: _____/ 6

4: _____/ 7

5: _____/24

6: _____/ 6

7: _____/ 9

8: _____/11

9: _____/10

TOTAL: _____/88

Good Luck!

Question 1. [9 MARKS]

Part (a) [5 MARKS]

Circle the correct answer below.

TRUE	FALSE	A process whose parent has not called wait is an orphaned process.
TRUE	FALSE	A read on a pipe will block when there is nothing to ready to read on the pipe.
TRUE	FALSE	If a client's request for a socket connection arrives between the times that a server has called `listen` and `accept`, the connection will be denied.
TRUE	FALSE	If a struct is passed in as an argument in a C function, any modifications made to the contents of the struct will be visible when the function returns
TRUE	FALSE	Read and execute permissions are needed on a shell program to execute it.

Part (b) [2 MARKS]

Fill in the third argument to `strncat`. Assume that `str1` and `str2` both contain valid strings.

```
char str1[SIZE1];
char str2[SIZE2];

strncat(str1, str2,_____________________________________);
```

Part (c) [2 MARKS] Write the output of the following C snippet.

```
char a[10];
char *b = malloc(10 * sizeof(char));
printf("A = %d\n", sizeof(a));
printf("B = %d\n", sizeof(b));
```

Question 2. [6 MARKS]

The current working directory contains 3 files: `hop`, `skip`, and `jump`. The contents of each file are shown below:

hop

```
#!/bin/sh
arg=$1
echo "Running hop"
if [ -x $1 ]; then
      $1
fi
```

skip

```
#!/bin/sh
echo "Running skip"
```

jump

```
#!/bin/sh
echo "Running jump"
```

For each of the following commands, give the output that would appear when the command is run. (' is a single quote and ‘ is a back quote.)

`run="hop"` `$run`	_____
`echo "$run hop"`	_____
`echo '*'`	_____
`echo ‘*‘`	_____
`[sh]*`	_____
`set ???p` `echo $2`	_____

Question 3. [6 MARKS]

Part (a) [2 MARKS]

I told my friend who has an account on CDF that I have a program she can run in `~reid/bin/prog`. When I run `ls -l ~reid/bin/prog` I see `-rwxr-xr-x` for the permissions on the file. When she runs `~reid/bin/prog` she gets "Permission denied." What could be the cause?

Part (b) [2 MARKS]

What are `htonl()` and `ntohl()` for? Why do we need to use them on the port number when creating a socket but not on character arrays that are read and written on the socket?

Part (c) [1 MARK]

What could go wrong if the reader of a pipe forgets to close `fd[1]`?

Part (d) [1 MARK] Explain why the following code is incorrect (aside from the lack of error checking).

```
char line[10]
int fd[2];
int newfd;
pipe(fd);
newfd = fd[0];
read(newfd, line, 10);
```

Question 4. [7 MARKS]

Write a C program that takes a string and a directory name as arguments and prints the names of the files in that directory whose names end with the string. For example, if the program is called with `.html foo`, then it will print the names of all of the files in `foo` that end with `.html`. (Do not create another process.)

Question 5. [24 MARKS]

The code below is the start of a very simple implementation of a shell. The only functionality in this shell is the ability to read lines from standard input and execute a program in the background or in the foreground. Recall that when a program is executing in the foreground, the shell user cannot start another command until the program has terminated. If a program is started in the background by appending '&' to the command string, then the shell user can execute another command before the first one terminates.

You do not need to provide error handling for the given code.

Part (a) [9 MARKS] Complete the loop below according to the comments.

```
int main() {

    char line[30];
    char **args;
    char *ptr;
    int background;

    while(1) {
        printf (">");
        fgets(line, 30, stdin);
        ptr= strchr(line, '\n');
        *ptr = '\0';

        /* If the user typed return, then there is no command to execute */
        if(strlen(line) == 0) {
            continue;
        }

        /*mkarray returns an array of the words in line using space as a delimiter*/
        args = mkarray(line);

        /* If the user just typed return go back to the top of the loop */
        if(args[0] == NULL) {
            continue;
        }

        /*checkbackground returns 1 if the program should be run in the background
        and 0 otherwise */
        background = checkbackground(args);

        /* Run the program specified by args.  If the program is to be run
        in the foreground, then block until the process terminates and
        print a message if the program terminated normally, otherwise return
        to the top of the loop to wait for the user to enter another command.
        Check if any background programs have terminated, and print a message
        if they terminated normally. */
```

Part (a) (CONTINUED)

Part (b) [8 MARKS]

Implement the function `char **mkarray(char *line)` that returns an array of the words in the string `line`, using a space as the delimiter.

Part (c) [3 MARKS]

Based on your implementation of `mkarray()`, is there any memory that should be freed explicitly outside of `mkarray()`? If so, show the code below that the calling process would run to free the correct memory. State where in the loop in part (a) this code to free memory should go.

Part (d) [4 MARKS]

Write the function `int checkbackground(char **args)` that returns 1 if the program should be run in the background, and 0 otherwise. (Hint: the function should modify `args` so that the array contains only the function name and the actual program arguments.)

Question 6. [6 MARKS]

Write a Bourne shell program called `makepath` that takes a pathname as a command line argument and creates all the components of that path if they don't already exist. For example,

```
makepath foo/bar/baz
```

should create the directories `foo`, `foo/bar`, and `foo/bar/baz`. You may not use `mkdir -p` in your script. You may find the following commands useful:

```
basename NAME - Print NAME with any leading directory components removed.
                (basename / produces '/')
dirname NAME - Print NAME with its trailing /component removed;
               if NAME contains no /'s print '.'  (dirname / produces '/')
```

Question 7. [9 MARKS]

Part (a) [6 MARKS]

A CSC207 student needs some help automating tests. The student has written a program called `parsewiki` that takes a file name as a command line argument and writes to standard output. The files in the directory `tests` are to be used as test input to `parsewiki`. There is a directory called `expected` that contains identically named files with the expected output.

Write a Bourne shell program that runs `parsewiki` on each of the files in `tests`, places the output in file of the same name as the input file in a directory called `actual`, and then uses `diff` to compare the actual output file to the expected output file, and will print the message "Test failed on file X" (where X is replaced with the name of the file that failed) if the expected and actual output differ. No other output should be printed.

Note that `diff` returns 0 if the files are the same and 1 if they differ.

Part (b) [3 MARKS]

Write a Bourne shell program that prints the name of each file in `tests` that does not have a matching file in `expected`.

Question 8. [11 MARKS]

Consider the following program. Assume the program runs without error. The error handling has been removed to make the code shorter.

```
void sig_int(int signo) {
    write(2, "caught SIGINT\n", sizeof("caught SIGINT\n"));
    /*pause();*/ /* Uncomment this line for part (c) */
    /*F*/
    return;
}

int main(void) {
    sigset_t newmask, oldmask, pendmask;
    struct sigaction newact;

    sigemptyset(&newact.sa_mask);
    sigaddset(&newact.sa_mask, SIGTERM);
    sigaddset(&newact.sa_mask, SIGINT);
    newact.sa_handler= sig_int;
    newact.sa_flags = 0;
    /*A*/
    sigaction(SIGINT, &newact, NULL) ;

    sigemptyset(&newmask);
    sigaddset(&newmask, SIGTERM);
    sigaddset(&newmask, SIGINT);

    /*B*/
    printf("Blocking signals\n");
    sigprocmask(SIG_BLOCK, &newmask, &oldmask);

    /*C*/
    sigpending(&pendmask);
    if(sigismember(&pendmask, SIGTERM))
     printf("SIGTERM pending\n");
    if(sigismember(&pendmask, SIGINT))
     printf("SIGINT pending\n");

    /*D*/
    sigprocmask(SIG_SETMASK, &oldmask, NULL);
    printf("Unblocking signals\n");

    /*E*/
    printf("All Done\n");
    exit(0);
}
```

Part (a) [5 MARKS]

Describe what happens if the signal SIGINT arrives at the process at each of the letter markers A, B, D, E, and F. If any output is produced write it down, otherwise write "None." Assume the program is restarted each time.

	What happens	Output
A		
B		
D		
E		
F		

Part (b) [2 MARKS]

Describe what happens if the signal SIGTERM arrives at the process a markers B and D. If any output is produced write it down, otherwise write "None." Assume the program is restarted each time.

	What happens	Output
B		
D		

Part (c) [4 MARKS]

Suppose that the line with `pause()` is uncommented in the `sig_int()` function, and that SIGINT arrives such that `sig_int` is called. What happens if SIGTERM arrives after the program calls `pause()`?

What if SIGQUIT arrives after the program calls `pause()`? (Assume the program is restarted and SIGINT arrives such that `sig_int()` is called.)

Question 9. [10 MARKS]

Complete the function below that takes an array of two open socket descriptors as an argument. The function implements a subtraction game where the players are given a number to start with. They each take turns subtracting 1, 2 or 3 from `number`. The person that causes the total to reach 0 wins.

In this implementation, the players write each move on the file descriptor as an `int`. You can assume that no other type of data is sent, and you do not need to verify that the number is in the correct range. You must use `select` to avoid blocking.

When a player wins, write a message to both players indicating that the game has been won.

When a player sends a message out of turn, write a message on the socket descriptor telling the player, "It's not your turn." If either player closes their socket, then write a message to the other player, close the sockets, and return.

```
void play(int fd[]) {
    int number = 21;
```

This page is provided for rough work and any answers that didn't fit.

C function prototypes and structs:

```
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 dup2(int oldfd, int newfd)
int execlp(const char *file, char *argv0, ..., (char *)0)
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)
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
struct hostent *gethostbyname(const char *name)
unsigned long int htonl(unsigned long int hostlong)
unsigned short int htons(unsigned short int hostshort)
char *index(const char *s, int c)
int kill(int pid, int signo)
int listen(int sock, int n)
unsigned long int ntohl(unsigned long int netlong)
unsigned short int ntohs(unsigned short int netshort)
int open(const char *path, int oflag)
DIR *opendir(const char *name)
int pause(void);
int pclose(FILE *stream)
int pipe(int filedes[2])
FILE *popen(char *cmdstr, char *mode)
ssize_t read(int d, void *buf, size_t nbytes);
struct dirent *readdir(DIR *dir)
int select(int maxfdp1, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact)
       /* actions include SIG_DFL and SIG_IGN */
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 *strchr(const char *s, int c)
size_t strlen(const char *s)
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)
char *strrchr(const char *s, int c)
```

```
char *strstr(const char *haystack, const char *needle)
int wait(int *status)
int waitpid(int pid, int *stat, int options) /* options = 0 or WNOHANG*/
ssize_t write(int d, const void *buf, size_t nbytes);

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

Useful structs

```
struct sigaction {
    void (*sa_handler)(int);
    sigset_t sa_mask;
    int sa_flags;
}
struct dirent {
    int d_namelen;
    int d_name[MAXNAMELEN];
}
struct sockaddr_in {
    sa_family_t sin_family;
    unsigned short int sin_port;
    struct in_addr sin_addr;
    unsigned char pad[8]; /*Unused*/
}
```

Shell comparison operators

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

Total Marks = 88