PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

St. George Campus

APRIL 2014 EXAMINATIONS

CSC 209H1S
Instructor — Karen Reid
Duration — 3 hours

Examination Aids: One double-sided 8.5x11 sheet of paper. No electronic aids.

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

Last (Family) Name(s): _____

First (Given) Name(s): _____

---

*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 8 questions on 18 pages. A mark of at least 28 out of 72 on this exam is required to pass this course. *When you receive the signal to start, please make sure that your copy of the examination is complete.*
You are not required to add any `#include` lines, and unless otherwise specified, you may assume a reasonable maximum for character arrays or other structures. Error checking is not necessary unless it is required for correctness.

MARKING GUIDE

1: _____/10
2: _____/ 3
3: _____/ 9
4: _____/10
5: _____/10
6: _____/ 7
7: _____/ 9
8: _____/14

TOTAL: _____/72

Good Luck!

Question 1. [10 MARKS]

Part (a) [2 MARKS]

A running process terminates with the error message: `Broken pipe`. Explain what would happen in the program code to cause this message, and why it terminates.

Part (b) [2 MARKS]

Consider the following code snippet where `fd`, `buf`, and `MAXBUF` are correctly initialized. Note that `strchr` returns a pointer to the first occurrence of `'r'` in the string `buf` or `NULL` if `'r'` is not in `buf`.

```
int nbytes = read(fd, buf, MAXBUF);

char *ptr = strchr(buf, '\r');
```

i) Explain what is potentially wrong with this code snippet. ii) Insert one line of code to the snippet above to fix this potential problem

Part (c) [2 MARKS]

Describe the purpose of the different types of sockets that a server process (such as the one in question 8) must manage.

Part (d) [2 MARKS]

Why is it important to close pipe descriptors that are not being used or are no longer used?

Part (e) [1 MARK] When is a signal handling function called?

Part (f) [1 MARK]

Explain why we must use global variables if we want a signal handling function to modify program state (such as the `numkids` variable in exercise 6).

Question 2. [3 MARKS]

A student tells me, "My program works fine on my machine at home, but when I run it on CDF, I get a segmentation fault. Even stranger, it also works fine when I run it in the debugger."

Part (a) [1 MARK] Is there a bug in the student's code? Circle the correct answer.

Yes No

Part (b) [2 MARKS] Explain why it works in some cases but not in others, and what the most likely issue is.

Question 3. [9 MARKS]

Part (a) [8 MARKS]

Complete the statements in the code below so that it compiles without error or warning and uses the library functions correctly. State any assumptions you think you need to make. **Read carefully!**

```
struct games {
    char *name;
    char results[4];
};

struct games **teams = malloc(_____________________________________);


_________________________ t;

t.name = "Dickson";
t.results[0] = 'w';
t.results[1] = 'l';

// QUESTION: Are the types in the next statement below correct?
// Circle the correct answer and if necessary, fix the statement

//     YES         NO


teams[0] = t;

teams[1] = malloc(_____________________________________);


teams[1]->name = malloc(_____________________________);

printf("Enter team name:\n");

fgets(teams[1]->name, _____________________________, stdin);


strncpy(teams[1]->name, str, _____________________________);

teams[1]->results[0] = 'w';
teams[1]->results[1] = 'w';

char *banner = malloc(sizeof(char) * MAXLINE);


strncpy(banner, teams[0]->name, _____________________________);


strncat(banner, " vs. ", _____________________________________);


strncat(banner, teams[1]->name, _____________________________________);
```

Question 3. (CONTINUED)

The code below continues from the previous page. Fill in the correct type of each variable.

```
__________________ a = &(teams[0]->name);

__________________ b = teams[0]->results[0];

__________________ c = teams[0]->results;

__________________ d = *teams[1];

__________________ e = *(teams[1]->name);

__________________ f = &(teams + 1);
```

Part (b) [1 MARK]

What assumption(s) did you neeed to make to be certain that the code on both pages of this question will work properly? (Be as specific as possible.)

Question 4. [10 MARKS]

All the subquestions below concern this `Makefile` and the programs it builds.

Makefile

```
xmodemserver : xmodemserver.o crc16.o
    gcc -Wall -g -o xmodemserver xmodemserver.o crc16.o

client1 : client1.o crc16.o
    gcc -Wall -g -o client1 client1.o crc16.o

xmodemserver.o : xmodemserver.c crc16.h xmodemserver.h
    gcc -Wall -g -c xmodemserver.c

crc16.o : crc16.c crc16.h
    gcc -Wall -g -c crc16.c

client1.o : client1.c crc16.h
    gcc -Wall -g -c client1.c

clean :
    rm *.o out client1 xmodemserver
```

Initial contents of the current working directory:

```
Makefile        crc16.c         sample.txt      xmodemserver.h
client1.c       crc16.h         xmodemserver.c
```

Part (a) [1 MARK]

What is the target of the rule that is executed when `make` is run with no arguments?

Part (b) [1 MARK]

Write a rule that will build both `xmodemserver` and `client1` with one call to `make`. (You can choose a name for the target.)

Part (c) [2 MARKS]

Write a shell command that will run `xmodemserver` in the background, with no arguments, and will redirect standard output and standard error to a file called `out`.

Part (d) [2 MARKS]

Write a rule for the makefile so that when `make test` is run, `xmodemserver` and `client1` will be compiled if necessary, and the line you wrote in part (c) will be run followed by a line that executes `client1` (Assume `client1` takes no arguments.)

Part (e) [2 MARKS]

Starting with only the initial contents of the current working directory, list the files that are created or modified when we run `make xmodemserver`.

Part (f) [2 MARKS]

Given that part (e) has been executed, list the files that are created or modified when we run `make client1`.

Question 5. [10 MARKS]

Consider the following program that compiles (with warnings).

```
struct node {
    char *name;
    double value;
    struct node *next;
};

struct node *createnode(char *name, double n){
    struct node newnode;
    newnode.name = name;
    newnode.value = n;

    return &newnode;
}

// Remove the first element from the list pointed to by head.
void delete_first(struct node *head){
    if(head !== NULL) {
        free(head);
        head = head->next;
    }
}

int main(){
    char *city1;

    strncpy(city1, "Toronto", strlen("Toronto"));

    char city2[] = "Vancouver";

    char *city3 = "Montreal";

    struct node *head = malloc(sizeof(struct node *));

    struct node *a;

    a = createnode(city1, 2.5);

    a->next = createnode(city2, 0.6);

    a->next->next = createnode(city3, 1.6);

    head = a;
    delete_first(head);

    while(head != NULL) {
        printf("%s %f\n", head->name, head->value);
        head = head->next;
    }
}
```

Part (a) [5 MARKS]

This program has six errors in it that cause it to not work correctly. Precisely describe **five** of the errors below.

Part (b) [5 MARKS]

Fix the five errors directly in the code on the previous page so that code operates as intended without any errors.

Question 6. [7 MARKS]

Consider a server whose purpose is to read a command from a client and execute the command, and send the standard output of the command to the client.

You are asked to write the following function to implement part of this server. Assume that this function is called after the client has connected. Assume that the function `mkargs` is given for you. (Do not write `mkargs`.)

```
/* mkargs takes a string as an argument and returns an array of strings where
 * each (space-separated) word in str is an element of the returned array.
 */
 char **mkargs(char *str);

/* Reads a command from soc. (Assume that the read call reads the whole line
 * including the network newline)
 * Uses mkargs to create the array of arguments.
 * Runs the command and writes the output back to the client.
 * Closes the client connection.
 */
void get_and_run_cmd(int soc) {
```

Question 7. [9 MARKS]

Part (a) [2 MARKS]

Explain what the `PATH` variable is used for.

Part (b) [4 MARKS]

Which system calls will be used in the implementation of the shell when the following command is typed:

```
grep L0101 classlist > daysection.txt
```

Part (c) [1 MARK]

When I run a program and type `^C` in the shell to terminate the process, which system call is used to implement this functionality?

Part (d) [1 MARK]

When I type `ls` in the shell, I never see the next shell prompt until after `ls` has finished executing. Which system call is used to implement this functionality?

Part (e) [1 MARK]

File permissions are stored in a bit vector of type `unsigned int`. Assume `p` hold the permission values for a file, and using the constants below as appropriate, fill in the conditional expression below so that if statement below will print "owner can execute the file" only if `p` has the execute permission bit set.

```
#define S_IRUSR 0000400  /* read permission, owner */
#define S_IWUSR 0000200  /* write permission, owner */
#define S_IXUSR 0000100  /* execute/search permission, owner */

unsigned int p;  // Assume p holds the permissions for a file.

if (                                                          ) {

    printf("owner can execute the file\n");
} else
    printf("permission denied\n");
}
```

Question 8. [14 MARKS]

Read the whole question before starting to work on it. The following code is taken from an example presented in class. It uses `select` to allow clients to connect and send messages to the server. The server prints messages received from the client to `stdout`. Assume all variables are correctly initialized.

```
int client[MAXCLIENTS]; //A) will become: struct cli client[MAXCLIENTS];
//...
FD_ZERO(&allset);
FD_SET(fd, &allset);
maxfd = fd;

while(1) {
    rset = allset;
    nready = select(maxfd + 1, &rset, NULL, NULL, NULL);

    if (FD_ISSET(fd, &rset)){
        len = sizeof q;
        clientfd = accept(fd, (struct sockaddr *)&q, &len);

        int found = 0;
        for(i = 0; i < MAXCLIENTS; i++) {
            if(client[i] == -1) {
                client[i] = clientfd;  // B) will become: init_client(&client[i]);
                FD_SET(clientfd, &allset);
                found = 1;
                if (clientfd > maxfd)
                    maxfd = clientfd;
                break;
            }
        }
        if(! found) {
            printf("Too many clients\n");
            close(clientfd);
        }
    }
    for(i = 0; i < MAXCLIENTS; i++) {

        if(client[i] != -1 && FD_ISSET(client[i], &rset)) {
            // C) Code below will need to be rewritten
            char buf[256];
            if(read(client[i], buf, 256) > 0) {
                printf("RECEIVED: %s\n",  buf);
            } else {
                /*CLEAN UP!!!*/
            }
        }
    }
}
```

Part (a) [2 MARKS]

Write the code that should be written in the else statement of the `read` call where there is a `CLEAN UP` comment.

Part (b) [12 MARKS]

It would be nice to know who is writing each message to the server, so your task in the remainder of this question is to augment this server so that:

- the first thing the server reads from a client is a user id
- the next thing it reads is a password
- the server then calls `authenticate(userid, password)` and if `authenticate` returns 0 (userid and password were **not** accepted), then the server will remove the client. If `authenticate` returns 1 then the server will continue to read messages from the client and will print *userid: message* instead of *RECEIVED: message*.

There are three parts where code will need to be changed:

Step 1) We need a struct for each client to keep track of additional information and we may need additional constants, types, or variables. Complete the struct below and add any other constants, types or variables needed. The line at `A)` will be replaced by the code in the comment.

```
struct cli {
    int fd;
```

Step 2) Initializing the client now requires more work. Write the `init_client` function below. The line at `B)` will be replaced by the call to `init_client` in the comment.

```
void init_client(struct cli *c) {
```

Step 3) The code below comment `C)` needs to be replaced. Write your code below.

This page can be used if you need additional space for your answers.

This page can be used if you need additional space for your answers.

Total Marks = 72

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);
int fseek(FILE *stream, long offset, int whence);
      /* SEEK_SET, SEEK_CUR, or SEEK_END*/
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
unsigned long int htonl(unsigned long int hostlong) /* 4 bytes */
unsigned short int htons(unsigned short int hostshort) /* 2 bytes */
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)
       /* oflag is O_WRONLY | O_CREAT for write and O_RDONLY for read */
DIR *opendir(const char *name)
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) /* family=PF_INET, type=SOCK_STREAM, protocol=0 */
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)
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 hostent {
    char *h_name; // name of host
    char **h_aliases; // alias list
    int h_addrtype; // host address type
    int h_length; // length of address
    char *h_addr; // address
}
struct sockaddr_in {
    sa_family_t sin_family;
    unsigned short int sin_port;
    struct in_addr sin_addr;
    unsigned char pad[8]; /*Unused*/
}
```

```
struct stat {
    dev_t st_dev; /* ID of device containing file */
    ino_t st_ino; /* inode number */
    mode_t st_mode; /* protection */
    nlink_t st_nlink; /* number of hard links */
    uid_t st_uid; /* user ID of owner */
    gid_t st_gid; /* group ID of owner */
    dev_t st_rdev; /* device ID (if special file) */
    off_t st_size; /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for file system I/O */
    blkcnt_t st_blocks; /* number of 512B blocks allocated */
    time_t st_atime; /* time of last access */
    time_t st_mtime; /* time of last modification */
    time_t st_ctime; /* time of last status change */
};
```

```
Useful shell commands:
cat, cut, echo, ls, read, sort, uniq, wc
ps aux - prints the list of currently running processes
grep (returns 0 if match is found, 1 if no match was found, and 2 if there was an error)
grep -v displays lines that do not match
diff (returns 0 if the files are the same, and 1 if the files differ)
```