University of Toronto
Faculty of Arts and Science

Midterm Examination — February 24th, 2000

CSC209S

Duration — 50 Minutes

Examiner: W. James MacLean

PLEASE HAND IN WHEN DONE

Instructions

- **No aids allowed.**
- Check to make sure you have all 7 pages.
- On the back page a list of UNIX function prototypes has been provided to assist you. You may detach this sheet (last page only).
- Read the entire exam paper before you start.
- Answer all questions in the space provided.
- Attempt answers to **all** questions.
- Not all questions are of equal value, so budget your time accordingly.
- All shell questions assume `csh` and all programming questions are in ANSI C.
- When writing C programs, you are not expected to remember (or mention) the name of include files used by system calls
- There is a total of 45 marks.

Please Complete This Section

Name	Family Name:	
	Given Names:	**SOLUTIONS**
	Student Number:	

Marks

Q1	7.4/15
Q2	5.0/10
Q3	5.3/10
Q4	5.5/10

Total 51.6%

1. [15 Marks] Recall that in a UNIX filesystem, a file may have more than one name. Write a 'C' program to find all filenames in a directory that refer to the same file. The program takes one (non-optional) command line argument; the name of a file. Your program will then find all filenames in the same directory that refer to the same file.

```
#include <stdio.h>      // many of you did not read the instructions
#include <sys/types.h>  // regarding include files
#include <sys/stat.h>
#include <dirent.h>
#include <string.h>

int main(int argc❶, char *argv[]❶) // it's a program, need to declare
{                                  // main()
  char           fileName[256] = ""  ;
  char           dirName [256] = "." ;
  int            lastSlash    ;
  struct stat    buf          ; ❶ struct stat *buf wrong, but got mark
  DIR           *dir          ; ❶
  struct dirent *entry        ; ❶

  if (argc != 2) ❶
  {
    fprintf(stderr,"Usage: %s <fileName>\n", argv[0]);
    return 1 ;
  }

  lastSlash = strlen(argv[1]) - 1; ❷ for separating file/directory name
  while (argv[1][lastSlash] != '/' && lastSlash >= 0) lastSlash-- ;
  if (lastSlash >= 0) {
    strncpy(dirName, argv[1],  lastSlash + 1);
    dirName[lastSlash + 1] = 0 ;
  }
  strcpy(fileName, argv[1] + lastSlash + 1);

  if (stat(argv[1], &buf) ❶ == -1❶)
  {
    fprintf(stderr,"Unable to stat() %s!\n", argv[1]);
    return 2 ;
  }

  dir = opendir(dirName); ❶
  if (dir == (DIR *)NULL) ❶
  {
    fprintf(stderr,"Unable to open directory %s for reading!\n",
dirName);
    return 3;
  }
  while ((entry = readdir(dir)) != (struct dirent *)NULL) ❶
    if (entry->d_ino == buf.st_ino) ❶
      printf("%s\n", entry->d_name);
  closedir(dir); ❶

  return 0 ;
}
```

- You didn't need a program as complete as what I have shown here, but it had to contain certain key points …
- Alternate method: scan directory to find a name matching the one given, record the inode number, rewind the directory and then look for matches
- If you could describe the basic algorithm but did not give code, you got 3 marks
- Many did opendir(argv[1]) instead of parsing to get dirname/filename
- Many thought this question was just about symbolic links—this is wrong
- Many people used strcmp() to compare names … this does not do what the question asked
- The system call system(const char *command) does **not** return the output from the command executed
- You can't get the current working directory from getenv()
- There was no need to use open()/fopen() for this question
- Comparing two files byte-for-byte is wrong: they could be identical but different

2. [10 Marks] Consider the output from the UNIX utility "df" below:

```
Filesystem            kbytes    used   avail capacity  Mounted on
/dev/dsk/c0t0d0s0     369639  230289  102390    70%    /
/proc                      0       0       0     0%    /proc
fd                         0       0       0     0%    /dev/fd
/dev/dsk/c0t0d0s6     369639   20977  311702     7%    /var
/dev/dsk/c0t0d0s7     123455   78102   33013    71%    /cache
swap                  604568    9168  595400     2%    /tmp
```

Write a CSH script named `dfCheck` to do the following:

1) Calculate the total capacity of all mounted filesystems,
2) Calculate the total available capacity of all mounted file systems,
3) Calculate the average total and available capacities of all mounted filesystems.

Also, the script is to take an optional parameter which, if specified, is a patterm which the filesystem name must match to be included. For example:

```
% dfCheck '/dev*'
```

only includes those filesystems whose names start with '`/dev`'.

```
#!/usr/bin/csh -f ❶
#
# CSC209S Midterm, Feb 24th, 2000
# Question #2

# get raw data, and delete header line
set data = "`df`" ❶
shift data

if ( $#argv == 1 ) then ❶ read command line parameter
 set pattern = "$argv[1]"
else
 set pattern = ""
endif

@ sumCapacity = 0 ❶ initialize variables
@ sumAvail = 0

set i = 1
while ( $i <= $#data ) ❶ loop through data
  set y = ( $data[$i] )
  if ( "$pattern" =~ "" || "$y[1]" =~ $pattern ) then ❷ compare pattern
    @ sumCapacity = $sumCapacity + $y[2] ❶ update sums
    @ sumAvail    = $sumAvail    + $y[4]
  endif
  @ i = $i + 1
end

@ aveCapacity = $sumCapacity / $#data ❶ calc averages
@ aveAvail    = $sumAvail    / $#data
```

```
echo Total capacity = $sumCapacity kbytes ❶ output results
echo Total available = $sumAvail kbytes
echo Average capacity = $aveCapacity kbytes
echo Average available = $aveAvail kbytes
```

- You didn't need script as complete as I have shown here, but it neede to contain certain key points
- csh array indices start from 1, not 0
- If $#argv == 1, then you have one parameter
- set data = "`df | grep $argv[1]`" doesn't work: it matches the pattern anywhere in the line, not just in the first field as you were supposed to do; could use "`df | grep ^$argv[1]`"
- If you use set data = "`<command>`", you can't use foreach item ($data) to loop through the data, as it destroys the line-by-line structuring

3. [10 Marks] Write a CSH script named `lls` to list only filenames that are symbolic links. The script takes one optional argument, which is the name of a directory to use when looking for the links. When no argument is specified, the search is conducted in the current working directory.

```
#!/usr/bin/csh -f ❶
#
# CSC209S Midterm, Feb 24th, 2000
# Question #3

set args = ""
if ( $#argv == 1 ) then
 if ( ! -d $argv[1] ) then ❶
   echo $argv[1] not a directory!
   exit
 endif
 set args = $argv[1] ❶
else if ( $#argv > 1 ) then ❶
  echo "Usage: $0 <file|directory>"
  exit
endif

set links = `ls -1 -aF $args ❷ | grep @ ❷ | tr -d @`

if ( $#links > 0 ) then

  echo Symbolic Links:
  foreach link ( $links ) ❶
    echo $link ❶
  end

else

  echo No symbolic links found

endif
```

- Your script didn't need to be as complete as this, but still needed to contain certain key points
- you must check that $argv[1] is a valid directory
- many people left $'s off of variable references
- if … then syntax was sloppy in most answers
- "`ls $argv[1] -aF | grep @`" ok, except error occurs if no command line parameter given …
- could also look for 'l' as the first character in the permissions field: "`ls -al $args | grep ^l`"
- "argc" doesn't exist in csh scripts
- might check if "stat $file | grep symbolic" is empty?
- "ls -s" doesn't list symbolic links
- "if (-l $file)" doesn't test for symbolic links, since "-l" isn't defined in csh

4. [10 Marks] Briefly answer the following (assume 1 mark each unless otherwise indicated)

a) What is the difference between a program and a process?

A program is an executable file, a process is an executing instance of a program.

b) What is an *inode*?

An inode is a data-structure used by a file system to store important information about a physical file on the hard disk. (need to say 'node' or 'data structure; inode is **not** a 'number')

c) How can you test whether a pathname is *absolute* or *relative*?

If the first character is / (not '\'!), then the name is an absolute path. All others are relative.

d) How can you delete the file named `fred|barney.c` ?

`rm "fred|barney.c"` or `rm fred\|barney.c`

e) Is a directory file a regular file?

No.

f) If you have execute-permission for a directory, can you delete a file in that directory (Yes/No)?

No. You need write permission on the directory. Write permission on file **not** necessary.

g) What is the purpose of the csh variable *noclobber*?

When set, it prevents accidental overwriting of files via I/O redirection.

h) How can you execute a shell script without a new shell process being created?

Use "source".

i) Define what the UNIX term *zombie* means.

A zombie is a process that has terminated but not had its return status read. A terminated child **does not** send a signal to the parent … UNIX kernel does.

j) Demonstrate briefly how to check to see if any child processes have terminated without blocking.

```
int status, pid ;
if ((pid = waitpid(-1, &status, WNOHANG)) != -1)
  printf("Child %d has exited.\n", pid);
```

Macros & Function Prototypes

I/O

```
char   *fgets(char *s, int n, FILE *stream)
FILE   *fopen(const char *file, const char *mode)
int     close(int fd)
int     dup(int fd)
int     dup2(int fd, int oldfd)
int     fclose(FILE *stream)
int     FD_ISSET(int fd, fd_set &fds)
int     feof(FILE *stream);
int     ferror(FILE *stream);
int     fflush(FILE *stream)
int     fileno(FILE *stream)
int     fprintf(FILE *stream, const char *format, …)
int     fscanf(FILE *stream, const char *format, …)
int     listen(int soc, int n)
int     open(const char *path, int oflag)
int     pipe(int filedes[2])
int     select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
               struct timeval *timeout)
int     sprintf(char *s, const char *format, …)
int     write(int fd, void *buf, int nbyte)
ssize_t read(int fd, void *buf, size_t nbyte)
void    FD_CLEAR(int fd, fd_set &fds)
void    FD_SET(int fd, fd_set &fds)
void    FD_ZERO(&fd_set)
```

IPC

```
FILE *popen(char *cmdStr, char *mode)
int   accept(int soc, struct sockaddr *addr, int addrlen)
int   bind(int soc, struct sockaddr *addr, int addrlen)
int   connect(int soc, struct sockaddr *addr, int addrlen)
int   pclose(FILE *stream)
int   semctl(int semid, int semnum, int cmd, ...);
int   semget(key_t key, int nsems, int semflags);
int   semop(int semId, stuct semops *sem_ops, int nops);
int   shmget(key_t key, size_t size, int shmflg);
int   shmctl(int shmid, int cmd, struct shmid_ds *buf);
int   shmdt(void *shmaddr);
int   socket(int family, int type, int protocol)
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

Process Management

```
int   execl(const char *path, char *argv0, …, (char *)0)
int   execle(const char *path, char *argv0, …, (char *)0, const char *envp[])
int   execlp(const char *file, char *argv0, …, (char *)0)
int   execv(const char *path, char *argv[])
int   execve(const char *path, char *argv[], const char *envp[])
int   execvp(const char *file, char *argv[])
int   kill(int pid, int signo)
int   wait(int &status)
int   waitpid(int pid, int *stat, int options)
int   WIFEXITED(int status)
int   WIFSTOPPED(int status)
int   WIFSIGNALLED(int status)
int   WEXITSTATUS(status)
int   WTERMSIG(int status)
int   WSTOPSIG(int status)
pid_t fork(void)
```

Signals

```
int      pause(void)
unsigned alarm(unsigned nsec)
void (*signal(int sig, void (*disp)(int)))(int)
void (*sigset(int sig, void (*disp)(int)))(int)
```

Threads

```
int pmutex_destroy(pthread_mutex_t *mutex)
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutex_attr_t *attr)
int pmutex_lock(pthread_mutex_t *mutex)
int pmutex_unlock(pthread_mutex_t *mutex)
int pthread_cond_init(pthread_cond_t * const pthread_condattr_t *attr);
int pthread_cond_wait(pthread_cond_t *cond pthread_mutex_t *mutex);
int pthread_cond_timedwait(pthread_cond_t * pthread_mutex_t *mutex,
                           const struct timespec *abstime);
int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
int pthread_cond_destroy(pthread_cond_t *cond);
int pthread_create(pthread_t *new_thread_ID, const pthread_attr_t *attr,
                   void * (*start_func)(void *), void *arg);
int pthread_detach(pthread_t threadID);
int pthread_join(pthread_t target_thread, void **status);
int pthread_key_create(pthread_key_t *keyp, void (*destructor)(void *value));
int pthread_key_delete(pthread_key_t key);
int pthread_once(pthread_once_t *once_control, void (*init_routine)(void));
int pthread_setspecific(pthread_key_t key, const void *value);
pthread_t pthread_self(void);
void pthread_exit(void *status);
void *pthread_getspecific(pthread_key_t key);
```

String Handling

```
char *strtok(char *s, const char *delim)
char *strcpy(char *dest, const char *srce)
char *strncpy(char *dest, const char *srce, int count)
int   strlen(const char *s)
int   strcmp(const char *s1, const char *s2)
int   strncmp(const char *s1, const char *s2, int count)
```

Time

```
char      *asctime(const struct tm *tm);
char      *ctime(const time_t *clock);
struct tm *gmtime(const time_t *clock);
struct tm *localtime(const time_t *clock);
time_t     time(time_t *tloc);
```

```
struct timeval {
  unsigned long tv_sec  ;
  unsigned long tv_usec ;
}
```

Directory Structure

```
DIR           *opendir(const char *filename);
int            access(const char *path, int amode);
int            closedir(DIR *dirp);
int            lstat(const char *path, struct stat *buf);
int            S_ISDIR(mode);
int            S_ISREG(mode);
int            stat(const char *path, struct stat *buf);
long           telldir(DIR *dirp);
struct dirent *readdir(DIR *dirp);
struct dirent *readdir_r(DIR *dirp, struct dirent *entry);
void           rewinddir(DIR *dirp);
void           seekdir(DIR *dirp, long loc);
```

```
struct dirent {
  ino_t            d_ino;
  off_t            d_off;
  unsigned short   d_reclen;
  char             d_name[1];
}
```

```
struct stat {
  mode_t   st_mode;     /* File mode (see mknod(2)) */
  ino_t    st_ino;      /* inode of file */
  time_t   st_atime;    /* Time of last access */
  time_t   st_mtime;    /* Time of last data modification */
  time_t   st_ctime;    /* Time of last file status change */
  off_t    st_size;     /* File size in bytes */
  nlink_t  st_nlink;    /* Number of links */
  uid_t    st_uid;      /* User ID of the file's owner */
  gid_t    st_gid;      /* Group ID of the file's group */
};
```