University of Toronto
Faculty of Arts and Science

December Examinations 1998

CSC209F

Duration — 2 Hours
No Aids Allowed

Examiner: W. James MacLean

Instructions

- **No aids allowed.**
- Check to make sure you have all 12 pages.
- 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 values, so budget your time accordingly.
- All shell questions assume `csh` and all programming questions are in ANSI C.
- On the back page a list of UNIX function prototypes has been provided to assist you.
- There are a total of 100 marks.

Please Complete This Section

Name	Family Name:	
	Given Names:	
	Student Number:	

Marks

Q1	/10
Q2	/10
Q3	/15
Q4	/10
Q5	/10

Q6	/5
Q7	/10
Q8	/15
Q9	/5
Q10	/10

Total □

1. [10 Marks] Write a for loop which continually reads from two file descriptors, one character at a time. One descriptor is associated with stdin and the other, int sockFd, with a socket. You do not know which descriptor will have data ready at any time, so your solution must not block on a read() for either descriptor. All data read is to be echoed to stdout. When EOF is reached on stdin, the loop should terminate. Declare any variables you may need. You may assume that writing to stdout never blocks.

```
char c ;
int result ;
fd_set readSet ;
int fdIn = fileno(stdin);
int max_fd = max(sockFd, fdIn) + 1 ;
for (;;)
{
  FD_CLEAR(&readSet);
  FD_SET(fdIn, &readSet);
  FD_SET(sockFd, &readSet);
  result = select(max_fd, &readSet, NULL, NULL, NULL);
  if (result == -1)
  {
    fprintf(stderr, "select() error (%s)\n", strerror(errno));
    exit(1);
  }
  if (FD_ISSET(fdIn, &readSet))
  {
    result = read(fdIn, &c, 1);
    if (result == 0) break ;
    fprintf(stdout, "%c", c);
  }
  if (FD_ISSET(sockFd, &readSet))
  {
    result = read(fdIn, &c, 1);
    fprintf(stdout, "%c", c);
  }
}
```

2. [10 Marks] Write a short C program that creates two children. Each child executes a function `void DoChild(int writeFd, int readFd)` and then exits immediately (you are not to write `DoChild()`, just assume it exists). The program is to create a pair of pipes whose file descriptors are passed to the children for two-way communication between the children. The parent does not exit until both children have terminated. The parent should report any abnormal termination information for the children.

```
int main()
{
  int numChildren = 2, status, pid ;
  int pipe1[2], pipe2[2]; /* 0 = read, 1 = write */

  if (pipe(pipe1) == -1)
  {
    perror("Error creating Pipe1");
    exit(1);
  }
  if (pipe(pipe2) == -1)
  {
    perror("Error creating Pipe2");
    exit(1);
  }
  switch(fork())
  {
    case 0 :
      DoChild(pipe1[1], pipe2[0]);
      exit(0);
    case -1 :
      perror("Error fork()ing child 1");
      exit(1);
  }
switch(fork())
  {
    case 0 :
      DoChild(pipe2[1], pipe1[0]);
      exit(0);
    case -1 :
      perror("Error fork()ing child 2");
      exit(1);
  }
  while (numChildren)
  {
    pid = wait(&status);
    if (pid != -1)
    {
      if (!WIFEXITED(status))
        fprintf(stderr,"Child %d exits abnormally!\n", pid);
      numChildren-- ;
    }
  }
  return 0 ;
}
```

3. [15 Marks] Write a "lowercase server" that takes messages from a client and turns all uppercase characters into lowercase before echoing the message back to the client. Implement the server using an internet stream socket.

```
#define thePort 4242
int main()
{
  int soc, ns, result ;
  /* 1 mark each for defining peer, self; 2 marks for setting port number */
  struct sockaddr_in self = {AF_INET, htons(thePort)};
  struct sockaddr_in peer = {AF_INET};
  char c ;

  soc = socket(AF_INET, SOCK_STREAM, 0); /* 2 marks */
  if (soc == -1) { perror("Socket"); exit(1); }

  /* 2 marks for bind() */
  result = bind(soc, (struct sockaddr )&self, sizeof(self));
  if (result == -1)
  {
    perror("Bind");
    exit(1);
  }

  if (listen(soc, 1) == -1) /* 2 marks */
  {
    perror("Listen");
    exit(1);
  }

  for (;;)
  { /* 2 marks for accept() */
    ns = accept(soc, (struct sockaddr_in )&peer, sizeof(peer));
    if (ns = -1)
    {
      perror("Accept");
      exit(1);
    }

    if (fork() == 0)
    {
      while (read(ns, &c, 1)) /* 1 mark (read) */
      {
        c = toupper(c);    /* 1 mark */
        write(ns, &c, 1); /* 1 mark */
      }
      exit(0);
    }
  }
  return 0 ;
}
```

4. [10 Marks] What does this program do when invoked as follows? In order to show your understanding of its operation, add comments to important statements in the program.

```
a.out grep a209 /etc/passwd % wc -l
```

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
  int p[2];
  int i,pid1,pid2, status;
      argv++;
      for (i = 1; i <= argc ; i++) /* oops, error here! */
          if (strcmp(argv[i],"%") == 0) /* find % in cmd line */
          { /* found it! */
              argv[i] = '\0';
              break;
          }
      pipe(p); /* create pipe for IPC */
      if ((pid1 = fork ()) == 0)
      {
          close(p[1]) ; /* don't need to write        */
          dup2(p[0],0); /* re-map read fd onto stdin  */
          close(p[0]) ; /* close extra read fd        */
          execv(argv[i+1], &argv[i+1]); /* exec args after % */
          _exit(1); /* if we get here, error! */
      }
      if ((pid2 = fork ()) == 0)
      {
          close(p[0]) ; /* don't need to read         */
          dup2(p[1],1); /* re-map write fd onto stdin */
          close(p[1]) ; /* close extra write fd       */
          execv(argv[0],argv); /* exec args before %  */
          _exit(1); /* if we get here, error! */
      }

      close(p[0]); /* close fd's in parent */
      close(p[1]);
      while (wait(&status)!=pid2); /* wait for children */
      if (status == 0) printf("child two done\n");
      else printf("child two failed\n");
      exit(0); /* terminate normally, yeah! */
}
```

This program creates a pipe between two processes: in this case it counts the lines in /etc/passwd which contain the string "a209"

5. [10 Marks] Compare UNIX's `semop()`/`semget()` functions for managing semaphores with those provided by Posix threads (Pthreads). Function prototypes have been provided below to aid your memory, **but you are not to write any code for this question** (marks will be deducted if you do).

```
int semget(key_t key, int nsems, int semflags);
int semop(int semId, stuct semops *sem_ops, int nops);
int pmutex_init(pthread_mutex_t *mt, pthread_mutexattr_t *attr);
int pmutex_destroy(pthread_mutex_t *mt);
int pmutex_lock(pthread_mutex_t *mt);
int pmutex_trylock(pthread_mutex_t *mt);
int pmutex_unlock(pthread_mutex_t *mt);
```

* pthread semaphores are much easier to use
* semop()/semget() semaphores are system-wide, pthread semaphores are local to a process
* semop() allows operations on multiple semaphores at once
* semop() gives the programmer some control over the value of the semaphore variable (*e.g.* can allow n threads into a critical section at once, not just 1)
* pthread semaphores are global vars within a process
* semop()/semget() semaphores can be shared amongst processes

Any four of these (or other valid contrasts/comparisons), gets full marks (*i.e.* 2.5 marks each)

6. [5 Marks] What is wrong with the following program skeleton? (Ignore the fact that it does not check return codes from the pthread functions.) Suggest two different ways to fix it?

```
pthread_mutex_t mutexA,  /* mutexes protecting */
                mutexB ; /* 2 resources        */

void *func1(void *)
{
  pthread_mutex_lock(&mutexA);  /* req resource A */
   ⋮
  pthread_mutex_lock(&mutexB); /* req resource B */

  /* critical section code */

  pthread_mutex_unlock(&mutexB);
   ⋮
  pthread_mutex_unlock(&mutexA);
}

void *func2(void *)
{
  pthread_mutex_lock(&mutexB); /* req resource B */
   ⋮
  pthread_mutex_lock(&mutexA); /* req resource A */

  /* critical section code */

  pthread_mutex_unlock(&mutexA);
   ⋮
  pthread_mutex_unlock(&mutexB);
}

int main()
{
  pthread_t thread1, thread2 ;

  pthread_mutex_init(&mutexA, NULL);
  pthread_mutex_init(&mutexB, NULL);

  pthread_create(&thread1, NULL, func1, NULL);
  pthread_create(&thread2, NULL, func2, NULL);

  pthread_mutex_destroy(&mutexA);
  pthread_mutex_destroy(&mutexB);
}
```

This program is susceptible to deadlock. (3 marks, -1 if "deadlock" not specifically mentioned)

1) Use only one mutex to protect both resources … (1 mark) (note: accessing mutexes in same order is equivalent to this)

2) When locking the second resource, use pthread_trylock() to see if it's available … if not release the first resource and wait a random time. (1 mark)

Some suggested associating one mutex with each critical section, but the mutexes really belong with the resources …

7. [10 Marks] You are writing a program which involves the following source files: `main.c`, `file_a.c`, `file_b.c`, `file_c.c`. All of the files include `main.h`, and `file_a.c` and `file_c.c` require `head_1.h`. Also, `file_b.c` and `file_c.c` require `head_2.h`. The program uses sockets (there are calls to `socket()`, `bind()`, `listen()` and `accept()`). Write a makefile for this project which assumes the final executable is to be named `myProg` and links any necessary libraries.

```
CC = gcc
CFLAGS = -g
LFLAGS = -lsocket -lbind
OBJS = main.o file_a.o file_b.o file_c.o

all: ${OBJS}
     ${CC} -o myProg ${OBJS} ${LFLAGS}

main.o: main.c main.h
     ${CC} -c main.c ${CFLAGS}

file_a.o: file_a.c main.h head_1.h
     ${CC} -c file_a.c ${CFLAGS}

file_b.o: file_b.c main.h head_2.h
     ${CC} -c file_b.c ${CFLAGS}

file_c.o: file_c.c main.h head_1.h head_2.h
     ${CC} -c file_c.c ${CFLAGS}
```

8. [15 Marks] Write a csh script called **makepath** that, when given a pathname, creates all the components of that pathname if they don't already exist. For instance,

```
makepath foo/bar/blah
```

should create the directories foo, foo/bar, and foo/bar/blah. It must handle both absolute and relative paths. You may not use `mkdir -p` in your script. Do **not** use recursion.

```
#!/usr/bin/csh -f

if ( $#argv != 1 ) then
  echo "Usage: $0 <newpath>"
  exit 1
endif

# the following can be replaced with
# set pathList = ( `echo $1 | sed 's/\// /g'` )
set temp = $argv[1]
set pathList = ""
while ( $temp !~ "" )
  set pathList = ( $temp:t $pathList )
  if ( $temp:t !~ $temp:h )
    set temp = $temp:h
  else
    set temp = ""
  endif
end

if ( $argv[1] =~ /* ) cd / # move to root dir

foreach dir ( $pathList )
  if ( -f $dir )
    echo "$cwd/$dir is a file, exiting ...
    exit 1
  endif
  if ( -d $dir )
    cd $dir
    continue # this part of path already exists
  endif
  echo "making $dir \( $cwd/$dir \)"
  mkdir $dir
  if ( $status != 0 ) # check mkdir status
    echo "Unable to create $cwd/$dir, exiting ...
    exit 1
  endif
end
```

9. [5 Marks] Show a simple implementation for UNIX's `sleep()` function using `alarm()` and `pause()`.

```
int sleep(int iNumSecs)
{
  if (signal(SIGALRM, sig_alrm) == SIG_ERR)
    return(iNumSecs);
  alarm(iNumSecs);  /* set alarm */
  pause();          /* wait for signal*/
  return(alarm(0)); /* turn off alarm in case      */
                    /* other sig received, unslept */
                    /* secs are returned           */
}

void sig_alrm(){/* do nothing */}
```

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

a) Why is `vfork()` more efficient than `fork()` if the child is just going to call `exec()` immediately?

It doesn't copy the parent's variables (*i.e.* the parent's address space) … this makes it far quicker …

b) What is the `tar` utility used for?

"tape archive" utility: used for combining multiple files/directories into one file for ease of transport, tape backup, subsequent compression, *etc.*

c) [2 Marks] Define "indefinite postponement".

A process/thread waits for an event which could possibly happen, but never does.

d) [2 Marks] Define "deadlock".

A process/thread waits for an event which will never happen.

e) *True or False*: race conditions are a form of non-determinism in a program.

True

f) *True or False*: semaphores allocated with `semget()` are a system-wide resource

True

g) Name two different socket families.

AF_INET
AF_UNIX

h) Name two different socket types.

SOCK_STREAM
SOCK_DGRAM

Function Prototypes

The following list is sorted alphabetically …

```
char *fgets(char *s, int n, FILE *stream)
FILE *fopen(const char *file, const char *mode)
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 close(int fd)
int connect(int soc, struct sockaddr *addr, int addrlen)
int dup(int fd)
int dup2(int fd, int oldfd)
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 fclose(FILE *stream)
int FD_ISSET(int fd, fd_set &fds)
int fflush(FILE *stream)
int fileno(FILE *stream)
int fprintf(FILE *stream, const char *format, …)
int fscanf(FILE *stream, const char *format, …)
int kill(int pid, int signo)
int listen(int soc, int n)
int open(const char *path, int oflag)
int pause(void)
int pipe(int filedes[2])
int plcose(FILE *stream)
int pmutex_destroy(pthread_mutex_t *mutex)
int pmutex_lock(pthread_mutex_t *mutex)
int pmutex_unlock(pthread_mutex_t *mutex)
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutex_attr_t *attr)
int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
           struct timeval *timeout)
int semget(key_t key, int nsems, int semflags);
int semop(int semId, stuct semops *sem_ops, int nops);
int socket(int family, int type, int protocol)
int sprintf(char *s, const char *format, …)
int wait(int &status)
int waitpid(int pid, int *stat, int options)
int write(int fd, void *buf, int nbyte)
pid_t fork(void)
ssize_t read(int fd, void *buf, size_t nbyte)
unsigned alarm(unsigned nsec)
void (*signal(int sig, void (*disp)(int)))(int)
void (*sigset(int sig, void (*disp)(int)))(int)
void FD_CLEAR(fd_set &fds)
void FD_SET(int fd, fd_set &fds)
void FD_ZERO(&fd_set)
```