PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

DECEMBER EXAMINATIONS 2002

CSC 209H1 F
Duration — 3 hours

Examination Aids: ***None***

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

Last Name: SOLUTIONS

First Name: _____

Lecture Section: _____

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 10 questions on 16 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 or where required for the correct execution of the program. 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:	_____/ 13
# 4:	_____/ 9
# 5:	_____/ 8
# 6:	_____/ 14
# 7:	_____/ 14
# 8:	_____/ 9
# 9:	_____/ 8
# 10:	_____/ 10
TOTAL:	_____/100

CONT'D...

Question 1. [10 MARKS]

Circle the correct answer for the following questions:

[TRUE]	FALSE	When `getopt` is used to read in options, it allows the options to appear in any order.
TRUE	[FALSE]	If a pointer is written through a pipe to a child process, it can be used to access memory in the parent process.
TRUE	[FALSE]	To satsify the three properties for a correct solution to the critical section problem, we require atomic hardware instructions.
TRUE	[FALSE]	If the parent's process id of a process is 1, the process must be a zombie.
[TRUE]	FALSE	If I don't have read permissions on a shell script then I cannot execute the shell script.
[TRUE]	FALSE	The `stat` function gets most of the information it returns from a file's inode.
TRUE	[FALSE]	The `stat` function will return an error if it is given a directory name as an argument rather than a regular file.
[TRUE]	FALSE	Shared libraries can reduce total memory usage if multiple processes use the same library.
[TRUE]	FALSE	A client can communicate with multiple servers through a single socket connection.
TRUE	[FALSE]	If a process contains the code sequence `fork(); fork(); fork();`, after there will be 4 processes.

Question 2. [5 MARKS]

Given the string below, write the value of $1 when each of the following Perl patterns is applied to the string. Write "no match" if the string does not match the pattern.

```
6:16pm up 5 days, 23:26, 6 users, load average:  0.37, 0.20, 0.1
```

`/([\d:,.]+)/`	6:16
`/(\w+)$/`	1
`/(\w+:\s*\d+\.\d+)/`	average: 0.37
`/((\d+\.\d+,\s*){1,3})/`	0.37, 0.20,
`/\s+(\d+\s+\w+)\s+/`	no match

Question 3. [13 MARKS]

Part (a) [2 MARKS]

How does creating additional processes allow a server to support multiple socket connections reliably? What problem can be solved by having the extra processes?

Each process can handle a single connection so that if a read blocks, other connections don't block unnecessarily.

Part (b) [2 MARKS]

What is the main drawback of the approach to supporting multiple connections described in part (a)?

Collecting data from the clients and combining it together.

Part (c) [2 MARKS]

How does `select` allow a server to support multiple socket connections reliably?

It allows us to block on a set of file descriptors. We only call read or accept when we know the data is ready so a misbehaving client cannot cause the server to block indefinitely.

Part (d) [2 MARKS]

Explain how a process knows or can find out the process id of its child.

The return value of the `fork` call is the process id of the child that was created.

Part (e) [2 MARKS]

Write two different system calls that would cause a process to terminate. If there are arguments to the system call that are required to make a process terminate, state what the values of the arguments must be.

(The system calls should be truly different, not just variations on the same operation. E.g., `fopen` and `open` are too similar, and neither should appear in your answer)

`exit(1)` and `kill(pid, SIGKILL)`

Part (f) [3 MARKS]

Write three different system calls that would cause a process to block. If there are arguments to the system call that are required to make a process block, state what the values of the arguments must be.

(The system calls should be truly different, not just variations on the same operation. E.g., `fopen` and `open` are too similar, and neither should appear in your answer.)

`accept`, `wait`, `select` (with no timout param), `read`, `waitpid` (as long as WNOHANG is not specified). `kill(pid, SIGSTOP)`

Question 4. [9 MARKS]

Part (a) [4 MARKS]

Assume the contents of the current working directory are

```
p1  p2  x  y  z
```

where `p1` is an executable program that prints to standard output its first command line argument. If the following Bourne shell commands are executed, what is printed? (‘ is a backquote and ’ is a single quote.)

```
a="p*"
b="?"
c=‘$a‘
d=’$b’
```

`echo $a` p1 p2

`echo $b` x y z

`echo $c` p2

`echo $d` $b

Part (b) [5 MARKS]

Assume we have a program `psieve` that takes two arguments: `-n <number of primes>` and `-n <output file name>`. Write a Bourne shell program that uses a loop to run `psieve` with the following arguments to `-n`: 5, 16, 40, 1000. The output file names should be "out.`<n>`" where `n` is the argument to `-n`. For example with a value of 2 for `n`, I would run `psieve -n 2 -f out.2`.

At the end of each iteration of the loop *either* make sure that all of the `psieve` processes have terminated *or* print a message if there are any `psieve` processes running. (Assume the script will only run on Linux machines.)

```
for n in 5 16 40 1000
do
    psieve -n $n -f out.$n
    killall psieve
    ps -axu |grep $USER | grep psieve
done
```

Just about anything with `ps | grep psieve` is acceptable.

Question 5. [8 MARKS]

Write a Perl program that reads from standard input a list with the format given below (modified output from `ps`). Your program will report the number of instances of every program that is not being run by `root`. The program being run is that last field of each line. (There may be other program names than those shown in the example input.)

Example input:

```
root     28533  ?        S    22:29   0:00 /local/sbin/sshd
g2mmmm   28536  pts/5    S    22:29   0:00 -tcsh
g2mmmm   28551  pts/7    S    22:30   0:00 -tcsh
reid     28567  ?        S    22:35   0:00 /local/sbin/sshd
reid     28568  pts/0    S    22:35   0:00 -tcsh
g2mmmm   28557  pts/7    S    22:30   0:00 nedit
root     28565  ?        S    22:35   0:00 /local/sbin/sshd
```

The above input would lead to the following output:

```
1       nedit
1       /local/sbin/sshd
3       -tcsh
```

```
#!/usr/bin/perl -w

use strict;
my $line;
my @cols;
my %progs;

while($line = <STDIN>) {
  chomp($line);
  @cols = split(/\s+/, $line);
  if($cols[0] ne "root") {
    if(defined($progs{$cols[6]})) {
      $progs{$cols[6]}++;
    } else {
      $progs{$cols[6]} = 1;
    }
  }
}

foreach my $p (keys(%progs)) {
  print("$progs{$p}\t$p \n");
}
```

Question 6. [14 MARKS]

To answer parts of this question you will find useful the wrapper functions for semaphores that we defined in class: `initSemaphore`, `acquire`, `release`, `removeSemaphore`.

Below is a C function that implements the Dining Philosopher algorithm, where `k` is the id or number of philosopher and `c_left` and `c_right` represent the chopsticks for the `kth` philosopher. Assume that `MAXTURNS` is defined, and that `getrand()` returns an appropriately-sized random number.

```
void philosopher(int k, int c_left, int c_right) {
  int i;
  for(i = 0; i < MAXTURNS; i++) {

    pickup(c_left);

    pickup(c_right);

    printf("%d is eating\n", k);    sleep(getrand());

    putdown(c_left);

    putdown(c_right);

    printf("%d is thinking\n", k);  sleep(getrand());
  }
  exit(0);
}
```

Part (a) [1 MARK]

The operations `pickup` and `putdown` must be _____ to ensure only one philosopher thinks it has a chopstick.

Part (b) [2 MARKS]

We can implement `pickup` and `putdown` by using chopstick as a semaphore variable. Complete the two functions below.

```
void pickup(int chopstick) {
  acquire(chopstick);
}

void putdown(int chopstick) {
  release(chopstick);
}
```

Part (c) [7 MARKS]

Complete `main` below so that a separate process is created for each philosopher, and the appropriate semaphore or semaphores are initialized.

```
#define N 5
int sem;

int main()
{
  int i;
  int chopsticks[N];
  for(i = 0; i < N; i++) {
    chopsticks[i] = initSemaphore(1);
  }
  sem = initSemaphore(N - 1);          *
  for(i = 0; i < N; i++) {
    if(fork() == 0) {
      philosopher(i, chopsticks[i],
                  chopsticks[(i+1)%5]);
    }
  }
}
```

```
#define N 5

int main() {









}
```

Part (d) [4 MARKS]

One solution to the dining philosophers problem when there are N philosophers is to ensure that only N-1 philosophers can pick up the first chopstick. Add the semaphore initialization and function calls to `philosopher` and `main` above to implement this solution. The semaphore operations should be placed so as to ensure the semaphore is held for the minimum possible amount of time, and deadlock is prevented. Hint: you either need to declare one global variable or add a parameter to `philosopher`.

Mark each line you add to `main` as part of the solution to this question with a `*`.

```
int sem; // *
void philosopher(int k, int c_left, int c_right) {
  int i;
  for(i = 0; i < MAXTURNS; i++) {

    acquire(sem);  // *
    pickup(c_left);
    pickup(c_right);
    release(sem);  // *

    printf("%d is eating\n", k); sleep(getrand());

    putdown(c_left);
    putdown(c_right);
    printf("%d is thinking\n", k); sleep(getrand());
  }
  exit(0);
}
```

Question 7. [14 MARKS]

Part (a) [8 MARKS]

Write a program whose main purpose is to call the function `doSomethingUseful()`. The function takes no arguments and the return type is `void`. When the program receives the `SIGTERM` signal the first time, it should print to standard output "Please don't kill me". When the program receives the `SIGTERM` signal the second time, it should print the message "I don't want to die" and terminate. You are not allowed to use global variables. You may need to write additional functions.

```
void whine2(int code) {
  printf("Please don't kill me.\n");
}

void whine1(int code) {
  struct sigaction sa, old;

  sa.sa_handler = whine2;
  sigemptyset(&sa.sa_mask);
  sa.sa_flags = 0;

  printf("I don't want to die.\n");
  if((sigaction(SIGTERM, &sa, &old)) < 0) {
    perror("Sigaction:");
  }
  exit(1);
}

int main() {

  struct sigaction sa, old;

  sa.sa_handler = whine1;
  sigemptyset(&sa.sa_mask);
  sa.sa_flags = 0;

  if((sigaction(SIGTERM, &sa, &old)) < 0) {
    perror("Sigaction:");
  }

  doSomethingUseful();
}
```

Part (b) [6 MARKS]

Complete the following program, so that it tries to terminate the program exec'd by the child with `SIGTERM`. After 3 seconds it checks to see if the process terminated. If not, it uses a signal that cannot be caught to terminate the process.

```
int main(int argc, char *argv[]) {

  int pid;

  if((pid = fork()) == 0) {
    execv(argv[1], &argv[1]);
    perror("exec failed");
  } else {
    sleep(1);
    kill(pid, SIGTERM);  /* 2 */

    sleep(3);            /* 1 */
    if(waitpid(pid, &status, WNOHANG) == 0) { /* 2 */
      printf("Process did not terminate\n");
      kill(pid, SIGKILL);                    /* 1 */
    } else {
      printf("Process is dead\n");
    }
  }
}
```

Question 8. [9 MARKS]

The three programs below all compile, but they all have runtime errors. You should assume that the appropriate headers are included. You should assume that no external forces cause the errors. I.e., only consider errors that result from the program itself.

For each program, explain precisely what the error is, and how it could be fixed. Vague answers will not receive full marks.

A:

```
int main(){

    int fd[2];
    char *buf; int i;

    pipe(fd);
    if(fork() == 0) {
        close(fd[1]);
        for(i = 0; i < 20; i++) {
            read(fd[0], buf, 30);
            printf("%s\n", buf);
        }
        close(fd[0]);
    } else {
        close(fd[0]);
        buf = "a";
        for(i = 0; i < 20; i++)
            write(fd[1], buf, strlen(buf));
        close(fd[1]);
    }
    wait(NULL);
}
```

Answer for A

B:

```
int main(){
    int fd[2];
    int i = 10, x = 20;

    pipe(fd);
    if(fork() == 0) {
        close(fd[1]);
        for(i = 0; i < 3; i++) {
            read(fd[0], &x, sizeof(int));
            printf("%d\n", i);
        }
        close(fd[0]);
    } else {
        close(fd[0]);
        for(i = 0; i < 10; i++) {
            write(fd[1], &i, sizeof(int));
            sleep(1);
        }
        close(fd[0]);
    }
    wait(NULL);
}
```

Answer for B

CONT'D...

C:	Answer for C:

```
int main(){

    int fd[2];
    int i = 10;

    pipe(fd);
    if(fork() == 0) {
        close(fd[1]);
        while(read(fd[0], &i, sizeof(int)) != 0)
            printf("%d\n", i);
    } else {
        close(fd[0]);
        for(i = 0; i < 5; i++)
            write(fd[1], &i, sizeof(int));
    }
    wait(NULL);
}
```

Question 9. [8 MARKS]

Part (a) [6 MARKS]

Print the output of the following program. Assume the program runs to completion and no errors occur.

```
int main() {
    int fd[2];
    int x = 11; int y = 22; int z = 33;

    pipe(fd);
    if(fork() > 0) {
        z = 77;
        read(fd[0], &y, sizeof(int));
        printf("Parent: x is %d, y is %d\n", x, y);

    } else {
        x = 55;
        write(fd[1], &x, sizeof(int));
        printf("Child: x is %d, y is %d\n", x, y);

    }
    printf("Done: z is %d\n", z);
}
```

```
Child: x is 55, y is 22
Done: z is 33
Parent: x is 11, y is 55
Done: z is 77
```

Part (b) [2 MARKS]

Is there another valid ordering of the output?

YES NO

If yes, give another valid ordering. If no, explain why.

Question 10. [10 MARKS]

Complete the server program below that establishes a connection with one client at a time. It reads a name from the client and then sends a message greeting the client and telling it what its client number is. The number assigned to each client is simply the order in which they make their connection to the server.

For example, if "Joe" was the 3rd client to connect to the server he would receive the following message from the server: "Hi Joe. You are client 3".

```
int main()
{

    struct sockaddr_in self;

    self.sin_family = AF_INET;
    self.sin_port = htons(SERVER_PORT);
    self.sin_addr.s_addr = INADDR_ANY;
    bzero(&(self.sin_zero), 8);
```

```
    /* set up listening socket soc */
    soc = socket(AF_INET, SOCK_STREAM, 0);
    if (soc < 0) {
        perror("server:socket");
        exit(1);
    }

    if (bind(soc, (struct sockaddr *)&self, sizeof(self)) == -1) {
         perror("server:bind"); close(soc);
         exit(1);
    }
    listen(soc, 1);
    /* accept connection request */

    while(1) {
        if((ns = accept(soc, (struct sockaddr *)&peer, &peer_len)) == -1) {
            perror("server:accept");
            close(soc);
            exit(1);
        }

        if((k = Readline(ns, buf, sizeof(buf))) == 1) {
                    break;
        }
        printf("Got %s\n", buf);
        count++;
        /* The print statement does not strip out the newline at the
         * end of buf.  That's okay.
         */
        sprintf(outbuf, "Hello, %s. You are client %d\n", buf, count);
        Writen(ns, outbuf, strlen(outbuf));

        close(ns);
    }
    close(soc);
    return(0);
}
```