PLEASE HAND IN

PLEASE HAND IN

**UNIVERSITY OF TORONTO**
**Faculty of Arts and Science**

**APRIL EXAMINATIONS 2005**

**CSC 209H1 S**
**St. George Campus**

**Duration — 3 hours**

**Examination aids: One 8.5 x 11 sheet of paper**

| | |
|---|---|
| **Student Number:** | |
| **Last Name:** | SOLUTIONS |
| **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 11 questions on 19 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: | ______/16 |
| # 2: | ______/ 5 |
| # 3: | ______/10 |
| # 4: | ______/ 6 |
| # 5: | ______/ 7 |
| # 6: | ______/ 7 |
| # 7: | ______/ 5 |
| # 8: | ______/ 8 |
| # 9: | ______/ 8 |
| # 10: | ______/10 |
| # 11: | ______/ 6 |
| TOTAL: | ______/88 |

*Good Luck!*

## Question 1. [16 marks]

### Part (a) [5 marks]

Circle the correct answer below.

[✓ TRUE] FALSE — An environment variable can be accessed by a shell program or by a C program.

TRUE [✓ FALSE] — When following code is executed, I can be sure that the null termination character is copied. (Assume that argv[1] holds a valid string.)

```
    char buf[256];
    strncpy(buf, argv[1], sizeof(buf));
```

[✓ TRUE] FALSE — If a client's request for a socket connection arrives before the server has called `listen`, the connection will be denied.

TRUE [✓ FALSE] — If the fields of the struct passed in as an argument in the following function call are modified inside the function, the modifications will still be visible when the function returns.

```
  void f(struct record r)
```

TRUE [✓ FALSE] — The function `g` defined below has the effect of setting the parameter `s` to point to the string "Hello".

```
  void g(char *s) {
      s = malloc(30);
      strncpy(s, "Hello", 30);
  }
```

### Part (b) [4 marks]

Given the two declarations below, the C statements that follow will compile without warning or error messages:

```
int *p;
int i = 1000;
```

[✓ TRUE] FALSE `char q = i;`

TRUE [✓ FALSE] `char *c = p;`

[✓ TRUE] FALSE `double d = i;`

TRUE [✓ FALSE] `double *f = &i;`

**Part (c)** [7 marks]

The current working directory contains 3 files: `a`, `ab`, and `b`. The contents of each file are shown below:

| a | ab | b |
|---|---|---|
| apple<br>banana | banana<br>apple | banana<br>orange |

When `grep` is run with the `-l` option it prints the name of each input file that contains a match.

For each of the following lines, give the output that would appear when the command is run. Note that `set` does not produce output. (' is a single quote and ` is a back quote.)

```
cmd="grep -l"



echo $cmd banana *    grep -l banana a ab b


echo "$cmd banana *"  grep -l banana *


echo '$cmd banana *'  $cmd banana *


echo `$cmd banana *`  a ab b



set `cat a`
$cmd $1 ??            ab



set `$cmd banana *`
echo `$cmd apple $*`  a ab
```

## Question 2. [5 MARKS]

### Part (a) [2 MARKS]

When a process writes to a pipe that has already been closed it gets a SIGPIPE signal and the process terminates. Write a C program fragment that would cause the process to ignore the SIGPIPE signal.

```
  sigaction(SIGQUIT, SIG_IGN, NULL);
```

(Students might also declare a variable of type `struct sigaction` and pass its address in as the third argument.)

### Part (b) [1 MARK]

Explain why it would not be a good idea to ignore SIGPIPE.

*Ignoring SIGPIPE hides the error from the programmer.*

### Part (c) [2 MARKS]

Describe in English what a better solution would be if you did not want the process to terminate when the SIGPIPE signal is received.

*Install a signal handling function that prints an error message, and returns control to the process.*

## Question 3. [10 MARKS]

### Part (a) [2 MARKS]

Explain what it means when a global variable is defined as `static`.

*The variable can only be used in the file in which it is declared.*

### Part (b) [2 MARKS]

Explain what it means when a global variable is defined as `extern`.

*The variable is declared in another file. The extern modifier is used to indicate the type of the variable to the compiler so that the file may be compiled separately. In particular no memory is allocated for the extern declaration.*

### Part (c) [6 MARKS]

Complete the function below that looks for one string inside another. The first argument is the string that is to be searched, and the second argument is the string you are interested in finding. If the specified string is found, `findString` returns a pointer to the location in the source string where the string was found. It returns NULL if the string was not found. You may **not** use any string functions other than `strlen`.

For example, the call `char *ptr = findString("a chatterbox", "hat")` returns a pointer to ”h” in the first string.

```
char *
findString(char *source, char *substring) {


  char *p1 = src;
  char *p2, *p3;
  int i;

  if(strlen(substring) > strlen(source)) {
    return NULL;
  }

  while(*p1 != '\0') {
    p3 = p1;
    p2 = substring;
    for(i = 0; i < strlen(substring); i++) {
      if(p3[i] == '\0' || p2[i] != p3[i]) {
        break;
      }
    }
    if(i == strlen(substring)) {
      return p1;
    }
    p1++;
  }
  return NULL;
}
```

## Question 4. [6 MARKS]

Consider the following program. Assume it runs to completion without errors.

```
int main()
{
  int i;
  int currProc = 0;
  for(i = 0; i < 3; i++) {
    if(fork() == 0) {
      printf("Child %d %d\n", i, getpid());
    } else {
      printf("Parent %d %d\n", i, getpid());
      currProc = i;
      break;
    }
  }

  printf("Done %d %d\n", currProc, getpid());
  wait(0);
  return 0;

}
```

### Part (a) [1 MARK]

How many processes, including the initial one, are created? 4

### Part (b) [5 MARKS]

Write the output of the program in a valid order. You will need to choose return values of `getpid()`. Each process must have a different value, and a given process always receives the same return value from `getpid()`.

```
Parent 0 951
Done 0 951
Child 0 952
Parent 1 952
Done 1 952
Child 1 953
Parent 2 953
Done 2 953
Child 2 954
Done 0 954
```

## Question 5. [7 MARKS]

### Part (a) [3 MARKS]

In the shell script below, circle all of the places where a new process is created. (‘ is a back quote, ’ is a single quote)

The following commands create new processes: `ls, echo, grep, echo, cut, expr`

```
#!/bin/sh

files=‘ls‘
count=0
for file in $files
do
    while read line
    do
        line=‘echo $line | grep TOTAL‘
        num=‘echo $line | cut -d " " -f 2‘
        if [ -n "$num" ]
        then
            echo $file $num
            count=‘expr $count + 1‘
        fi
    done < $file
done
echo "TOTAL: $count"
```

### Part (b) [4 MARKS]

Rewrite the program in part (a) so that it still performs the same functionality but creates as few processes as possible.

```
count=0
for file in *
do
    while read line
    do
        set $line
        if [ "$1" == "TOTAL" ]
        then
            num=$2
            echo $file $num
            count=‘expr $count + 1‘
        fi
    done < $file
done
echo "TOTAL: $count"
```

## Question 6. [7 marks]

Consider the following program. Assume the program runs without error.

```
void sig_quit(int signo)
{
  struct sigaction newact;
  newact.sa_flags = 0;
  sigemptyset(&newact.sa_mask);
  newact.sa_handler= SIG_DFL;

  printf("caught SIGQUIT\n");
  /* F */
  sigaction(SIGQUIT, &newact, NULL);
  /* G */
  return;
}

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

  sigemptyset(&newact.sa_mask);
  newact.sa_handler= sig_quit;
  newact.sa_flags = 0;
  /* A */
  sigaction(SIGQUIT, &newact, NULL);

  sigemptyset(&newmask);
  sigaddset(&newmask, SIGQUIT);
  /* B */
  sigprocmask(SIG_BLOCK, &newmask, &oldmask);

  sleep(10);
  /* C */
  if(sigismember(&pendmask, SIGQUIT))
printf("SIGQUIT pending\n");
  /* D */
  sigprocmask(SIG_SETMASK, &oldmask, NULL);
  /* E */
  exit(0);
}
```

**Part (a)** [5 MARKS]

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

| | What happens | Output |
|---|---|---|
| A | Program terminates | None |
| B | The sig_quit function is called<br>The program continues | caught SIGQUIT |
| C | SIGQUIT is blocked until<br>the program calls sigprocmask | (SIGQUIT pending)<br>caught SIGQUIT |
| D | SIGQUIT is blocked until<br>the program calls sigprocmask | (SIGQUIT pending)<br>caught SIGQUIT |
| E | calls the sig_quit function | caught SIGQUIT |

**Part (b)** [2 MARKS]

Describe what happens if the signal SIGQUIT arrives for the second time at the process at markers F and G. If any output is produced write it down, otherwise write "None." Assume the program is restarted each time.

| | What happens | Output |
|---|---|---|
| F | Since the new signal handler<br>has not been called, sig_quit<br>is called again. | caught SIGQUIT<br>caught SIGQUIT |
| G | The default behaviour of the<br>signal takes effect and the program<br>terminates | caught SIGQUIT |

## Question 7. [5 marks]

Write a Bourne shell program that takes a single command line argument. It assumes the command line argument is the name of an executable program and executes it once for each file in the current working directory that begins with "test". It runs the program on each file in sequence so that one completes before the next one starts. If the program returns 0 then it prints a message indicating that the program completed successfully, otherwise it prints a message indicating that the program failed.

For example, if the program is run as `sh_runit sim`, and the current working directory contains `test1`, `test2`, and `test3`, then the result is that `sim` is run on each of the files:

```
sim test1
sim test2
sim test3
```

If the program returned 0 for the last two instances of `sim`, but not the first, it would print:

```
sim test1 failed
sim test2 terminated successfully
sim test3 terminated successfully
```

```
for f in test*
do
    $1 $f
    if [ $? == 0 ]
    then
        echo "$1 $f terminated successfully"
    else
        echo "$1 $f failed"
    fi
done
```

## Question 8. [8 marks]

Write a C program that that takes a single command line argument. It assumes the command line argument is the name of an executable program and executes it once for each file in the current working directory that begins with "test". It runs the program on each file in sequence so that one completes before the next one starts. It prints a message indicating whether or not each program completes successfully.

For example, if the program is run as `runit sim`, and the current working directory contains `test1`, `test2`, and `test3`, then the result is that `sim` is run on each of the files:

```
sim test1
sim test2
sim test3
```

If the program completes the last two instances of `sim`, but not the first, it would print:

```
sim test1 failed
sim test2 terminated successfully
sim test3 terminated successfully
```

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

  DIR *dir;
  struct dirent *dp;
  int status;


  if((dir = opendir(".")) == NULL) {
    perror("opendir");
    exit(1);
  }

  while((dp = readdir(dir)) != NULL) {
    if(strncmp(dp->d_name, "test", strlen("test")) == 0) {
      /* run the test */
      if(fork() == 0) {
        execl(argv[1], argv[1], dp->d_name, 0);
        fprintf(stderr, "Exec failed\n");
      }else {
        if(wait(&status) == -1) {
          perror("wait");
        } else {
          if(WIFEXITED(status)) {
            printf("%s %s terminated successfully\n", argv[1], dp->d_name);
          } else {
             printf("%s %s failed\n", argv[1], dp->d_name);
          }
        }
      }
    }
  }
  return 0;
}
```

**Marking:**

- 1 for opendir
- 2 for readdir loop
- 1 for fork
- 1 for exec
- 1 for arguments to exec (mostly correct)
- 1 for wait in the right spot
- 1 for printing the final message

## Question 9. [8 MARKS]

The `popen` function "opens" a process by creating a pipe and executing the `command` that is given as an argument. The `type` (either "r" or "w") determines whether the process that calls `popen` will read from the pipe or write to it. It returns a pointer to a file object created from the file descriptor. Assume that the command has no arguments.

Write the `popen` function that is started below. Recall that the function `FILE *fdopen(int filedes, char *mode)` associates a file object with an existing file descriptor.

```
FILE *
popen(char *command, char *type) {

  int fd[0];

  pipe(fd);

  if(fork() == 0) {
    if( strcmp(type, "r") == 0) {
      dup2(fd[1], fileno(stdout));
    }else {
      dup2(fd[0], fileno(stdin));
    }
    close(fd[0]), close(fd[1]);
    execlp(command, command, 0);
  } else {
    if( strcmp(type, "r") == 0) {
      close(fd[1]);
      FILE *fp = fdopen(fd[0], type);
      if(fp == NULL) {
        perror("fdopen");
      }
        return fp;

    }else {
      close(fd[0]);
      FILE *fp = fdopen(fd[1], type);
      return fp;
    }
  }
  return NULL;
}

{\bf Marking:}
\begin{itemize}
\item 1 for pipe
\item 1 for fork
\item 1 for exec with mostly reasonable arguments
\item 1 for comparing type
```

```
\item 1 for closing file descriptors in child
\item 1 for closing file descriptors in parent
\item 1 for calling fdopen correctly
\item 1 for returning NULL if exec fails
\end{itemize}
```

## Question 10. [10 MARKS]

Complete the client program below that sends a file to a server. The program takes two command line arguments: the first is the name of the host running the server, and the second is name of the file to send to the server.

After making the connection with the server, the client expects to receive a message from the server containing the string "Ready". If it receives a different message, it prints the message to standard error and exits.

It then sends to the server the size of the file as an `int` in network byte order. After that it reads the file in chunks of up to 256 bytes, and writes each chunk to the server. It must close the socket before it terminates.

```
#define SERVER_PORT  30000

int main(int argc, char* argv[])
{
    int soc;
    struct hostent *hp;
    struct sockaddr_in peer;

    peer.sin_family = PF_INET;
    hp = gethostbyname(argv[1]);


    peer.sin_port = htons(SERVER_PORT);
    if ( hp == NULL ) {
        fprintf(stderr, "%s: %s unknown host\n",
                argv[0], argv[1]);
        exit(1);
    }

    peer.sin_addr = *((struct in_addr *)hp->h_addr);

    /* create socket */
    soc = socket(AF_INET, SOCK_STREAM, 0);
    /* request connection to server */
    if (connect(soc, (struct sockaddr *)&peer, sizeof(peer)) == -1) {
        perror("client:connect"); close(soc);
        exit(1);
    }

    read(soc, buf, sizeof(buf));
    if(strcmp(buf, "Ready") != 0) {
      fprintf("Error in writing to server.  Try again later.\n");
      exit(1);
    }

    if(stat(argv[2], &sbuf) == -1) {
      perror("Stat");
      exit(1);
    }

    write(soc, htonl(sbuf.st_size), sizeof(long));

    fp = fopen(argv[2], "r");
    while(fgets(buf, sizeof(buf), fp) != NULL) {
      write(soc, buf, strlen(buf) +1);
    }
    close(soc);
    return(0);
}
```

## Question 11. [6 MARKS]

Complete the function below that blocks until at least one of the two file descriptors passed as parameters is ready for reading, and returns that file descriptor. If both are ready, it returns the first one.

```
int whichisready(int fd1, int fd2) {
```

See Robbins and Robbins page 110

Total Marks = 88