University of Toronto Scarborough

CSCB09 Summer 2020 Final Examination

Duration - 2 hour 30 minutes

Aids allow: Open book

Solution and Marking

1. Miscelleneous short answers. Please enter all parts into the same Q1.txt.

   (a) [1 mark] Alan has a file named `dec25`. He wants to create a link `xmas` to `dec25`, such that if `dec25` is deleted and then a new `dec25` is created with new content, `xmas` refers to the new version. Advise Alan on whether the link should be a hard link or symbolic link.

   Answer: symbolic/soft

   (b) [1 mark] Anne has a file named `oct31`. She wants to create a link `hlw` to `oct31`, such that if `oct31` is deleted and then a new `oct31` is created with new content, `hlw` refers to the old version. Advise Anne on whether the link should be a hard link or symbolic link.

   Answer: hard

   (c) [2 marks] Manfred and Natasah are implementing pipelining in C, equivalent to the shell command

   ```
   prog1 | prog2
   ```

   Their code fragments are below. What's wrong in each case? Assume that code not shown but outlined in comments (the "ellided") is correct.

Manfred's version	Natasha's version

`// pipe creation ellided`
`pid_t p1, p2;`
`int s1, s2;`
`p1 = fork();`
`if (p1 == 0) {`
`  // close and dup2 ellided`
`  execlp("prog1", "prog1",`
`  (char*)NULL);`
`} else if (p1 > 0) {`
`  // close ellided`
`  waitpid(p1, &s1);`
`  p2 = fork():`
`  // rest of code ellided`
	`// pipe creation ellided`
	`pid_t p1, p2;`
	`int s1, s2;`
	`p2 = fork();`
	`if (p2 == 0) {`
	`  // close and dup2 ellided`
	`  execlp("prog2", "prog2",`
	`  (char*)NULL);`
	`} else if (p2 > 0) {`
	`  // close ellided`
	`  waitpid(p2, &s2);`
	`  p1 = fork();`
	`  // rest of code ellided`

   Answer:

   Manfred's: prog2 isn't started until prog1 terminates.
   Many acceptable variations, e.g.:
   If prog1 doesn't terminate, prog2 never started.
   If prog1 has a lot to write, blocked because prog2 isn't around to read, parent waits and doesn't start prog2, completing the impasse. (Very good concrete scenerio.)

Natasha's: prog1 isn't started until prog2 terminates.
Many acceptable variations, e.g.:
If prog2 doesn't terminate, prog1 never started.
If prog2 reads, blocked because prog1 isn't around to write, parent waits and doesn't start prog1, completing the impasse. (Very good concrete scenerio.)

Marking: You could call it 1 for Manfred and 1 for Natasha.
Or you could consider them related and mark based on total quality.

But don't accept trivial issues like "no error checking".

(d) [2 marks] Tom and Jerry both want their programs to, upon the signal `SIGINT`, continue normal running and not respond to the signal. Tom sets the signal action to `SIG_IGN`; Jerry sets the signal action to this no-op handler:

```
void jerry_handler(int sig) { }
```

What happens when Tom's program calls `select` and it blocks, and then `SIGINT` arrives? What happens in Jerry's case?

Answer: Tom's still blocks. Jerry's returns -1. (I don't require "`errno=EINTR`".)

(e) [4 marks] In an 8-bit binary number $b_7b_6b_5b_4b_3b_2b_1b_0$, we say "bit #0" for b_0. E.g., in 00000001, bit #0 is 1; in 11111110, bit #0 is 0. Let `x` be an `unsigned char` variable. Match the code fragments on the left below to the effects on the right (you may just write the labels "Or", "Comp", etc.):

Or. `x = x | 1;`	Zero. Change bit #0 of `x` to 1; other bits unchanged.
Xor. `x = x ^ 1;`	One. Change bit #0 of `x` to 0; other bits unchanged.
And. `x = x & 1;`	Comp. Complement bit #0 of `x`, i.e., if 0 change to 1, if 1 change to 0; other bits unchanged.
AndC. `x = x & ~1;`	Else. Something else.

Answer: Firstly, my typo! I mislabelled Zero and One. But no one raised it during exam. I also sampled some student answers and most seem to obey the bad labelling and perfectly answered:

Or-Zero, Xor-Comp, And-Else, AndC-One

Marking: Basically 1 mark per pair I guess.

If you believe that a student is auto-correcting my labelling, and their "Zero" means "change to 0", "One" means "change to 1", please accomodate. I think I saw a few rare cases where it's obvious, but pretty rare.

2. [6 marks] Shell pipelining. Please submit solution as `Q2.sh`

The Grande Online-Only Cafe wants to give out prizes to a few of its registered customers by a lucky draw! They already have a customer file like this:

```
743 Blanchett Anne anne.b@gmail.com
89 Hathaway Cate meow32@hotmail.com
```

etc., there are hundreds of more lines. Each line is a customer record with customer ID, family name (restricted to one word), given name (again one word), and email address, using a single space to separate two fields.

Assume that there is a program `shuf` (actually it exists) that reads lines from stdin and outputs a random permutation of the lines to stdout. Write a shell pipeline that picks 5 random customer lines and outputs them. The output should be alphabetically ordered by family names; if tie, by given names (if still tie, no further tie-breaking necessary).

The complete customer file will come from stdin, and your output should go to stdout. Don't worry about duplicate customer records.

Note that only a shell pipeline is accepted.

```
Answer:

shuf | head -5 | sort -k2,2 -k3,3

First, no marks at all if not pure pipeline!
e.g., "foo <(bar)" or temp files!

1 mark: shuf
1 mark: head -5  (or equiv eg head -n 5)
1 mark: sort
1 mark: -k2,2 -k3,3.  Note: -k2,3 is different, no mark here!
1 mark: shuf before head or sort
1 mark: head before sort

It's OK to say "shuf -", "head -5 -", "sort ... -",
i.e., use "-" to say "read from stdin".

It's OK to s/head/last/ because it receives shuffled file anyway.

It's OK to add -b to sort.
-b is more robust about extra spacing though not needed here
```

3. [6 mark] Shell scripting. Please submit solution as `Q3.sh`

Macy's school computer doesn't have `make` installed for some reason! She needs your help writing a shell script that performs a small but important job of `make`. She has a lot of TeX files (filenames end with ".tex"), and there is a program `pdflatex` for generating corresponding PDF files. Using "A1.tex" as an example, "`pdflatex A1.tex`" generats "A1.pdf". Clearly, she wants to generate if and only if one of:

- A1.tex exists but A1.pdf doesn't
- both exists, but A1.tex is newer

"A1.tex" is just an example—this applies to all TeX and PDF files in the current directory. Implement a Bourne shell script to do this.

The command "`basename A1.tex .tex`" outputs `A1` to help you.

Answer:

```
for i in *.tex; do
    if [ -e "$i" ]; then
        j=‘basename "$i" .tex‘.pdf
        if [ ! -e "$j" -o "$i" -nt "$j" ]; then
            pdflatex "$i"
        fi
    fi
done
```

1 mark: compute pdf filename

1 mark: All necessary quotings are done.
You don't have to give if the rest is poorly done.
Below, don't deduct marks for lacking quoting.

Many ways to implement the main logic, so I guess feature-based marking.
You can deduct if you think "this doesn't count".
Accept -e for existence check, I don't insist -f, -r, etc.
1 mark: if foo.pdf DNE, pdflatex foo.tex
1 mark: if both exists and foo.tex newer, pdflatex foo.tex
1 mark: if foo.tex DNE, don't generate (esp. when foo.pdf exists)
1 mark: if both exists and foo.tex not newer, don't generate

I use for-loop over the *.tex pattern. This can break if no match, $i
becomes literally '*.tex'. This is why I need my [-e "$i"] guard.
Another good way: for loop over *, then case *.tex).

Not sure what to do if they don't have a loop or the loop is trash.
Maybe just deduct some marks.

4. [11 marks] If you can look up in RAM quickly, you can look up on disk slowly! Please submit solution as `Q4.c` (starter file provided).

   In this question, you will write code for binary search tree lookup, but the tree is stored in a binary file on disk! The file consists of 0 or more nodes defined by this struct:

```
typedef struct {
  int key;
  long left, right;  // file positions (absolute) or -1L
} node;
```

   `key` is a key as usual. Note that instead of pointers to children, we have file positions (offsets) of children, so their types are `long` as in `fseek`; accordingly, when there is no left/right child, we use `-1L` (-1 but type `long`).

   The file may be empty, meaning the tree is empty; but if not, we know that the root node is at the beginning—position 0. Other nodes may be anywhere else, we only know that positions of nodes are non-negative multiples of `sizeof(node)`.

   Implement lookup:

```
int is_present(FILE *f, int needle);
```

   This looks for `needle` in the tree in `f`. It should return 1 (C true) if `needle` is a key in the tree, 0 (C false) if not.

   You do not know where the current file position is before this function begins. You may assume that `f` allows `fseek`, and `fread` on `f` either hits EOF or succeeds. You may assume that if `left` is non-negative, the left child node exists at that position; similarly for `right`.

```
int is_present(FILE *f, int needle)
{
  long pos;
  node v;

  pos = 0;
  while (pos >= 0) {
    fseek(f, pos, SEEK_SET);
    if (fread(&v, sizeof(node), 1, f) != 1) return 0;
    if (v.key == needle) return 1;
    else if (v.key < needle) {
      pos = v.right;
    } else {
      pos = v.left;
    }
  }
  return 0;

}
```

As usual there are many correct logics, some more redundant than others.
So I guess feature-based marking. The following points may not be code order.

First assume non-empty file and tree (root exists):
1 mark: First node examined is at position 0.
2 mark: When position of interest is -1, don't access file, return 0
2 marks: fseek to position of interest
         deduct 1 if eg v.left*sizeof(node)
1 mark: fread node
1 mark: If key == needle, return 1
1.5 marks: If key < needle, pursue right child
        (suitable update vars and loop back, or recursion)
1.5 marks: If needle < key, pursue left child
        (likewise)

This is for empty file and tree:
1 mark: return 0 if fread root node returns 0 (hits EOF)
  I want simple code so I check EOF every fread. This is not required,
  but if you don't check other freads you have to write more code.

5. The files to submit in this question are `line.h`, `solve.h`, `Makefile`

Helen wishes to modularize the following C file into multiple C files for separate compilation.

```
#include <stdio.h>

typedef struct {
  double m, c;  // as in y=mx+c
} line;

double compute_y(const line *L, double x)
{
  return L->m * x + L->c;
}

typedef struct {
  double x, y;
} point;

int solve(point *p, const line *L1, const line *L2)
{
  double d = - L1->m + L2->m;
  if (d == 0) return 0;
  else {
    p->x = L1->c - L2->c;
    p->y = - L1->m * L2->c + L2->m * L1->c;
    return 1;
  }
}

int main(void)
{
  line L1, L2;
  point P;
  L1.m = 4;   L1.c = 1;
  L2.m = -2;  L2.c = 2;
  printf("%f\n", compute_y(&L1, 0));
  solve(&P, &L1, &L2);
  printf("%f %f\n", P.x, P.y);
  return 0;
}
```

Helen wishes to put `compute_y` in 'line.c', `solve` in 'solve.c', and `main` in 'myprog.c'; the two structs also need to be moved to suitable header files.

(continued on next page)

(a) [6 marks] Write the corresponding header files 'line.h' and 'solve.h'.

```
1 mark: #ifndef ... #define ... #endif in both headers
        (What if only one header has it? If you like to be nice,
        assume innocent forgetfulness and still award this mark)
2 marks: line.h has:
         1: the line struct
         1: compute_y prototype, BUT not implementation
3 marks: solve.h has:
         1: #include line.h
            lose this mark if anything that belongs in line.h
            appears here
         1: the point struct
         1: solve prototype, but not implementation
```

(b) [6 marks] Write a Makefile for building an executable from the C source files after the modularization. The executable is to be called 'myprog'.

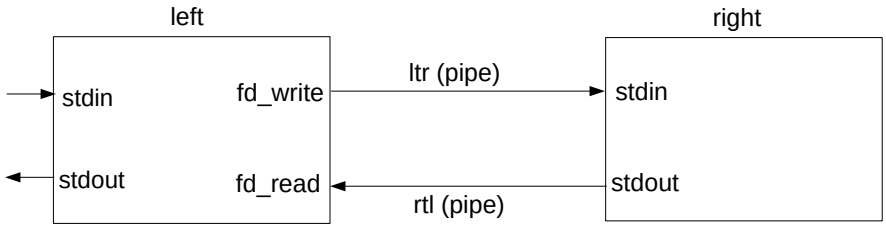
Reminder for Makefile rule syntax:
target : *prerequisite...*
    command to build target

Reminder for 'gcc' options:
-c: compile to object files, e.g., 'foo.c' to 'foo.o'
-o *name*: specify output filename

```
myprog: myprog.o solve.o line.o
        gcc -o myprog myprog.o solve.o line.o
myprog.o: myprog.c solve.h line.h
        gcc -c myprog.c
  # or: gcc -c $< [-o $@]
solve.o: solve.c solve.h line.h
        likewise
line.o: line.c line.h
        likewise

1.5 marks: exe rule
  Maybe split like: 1 for prereqs, 0.5 for recipe
2 marks: foo.o prereq on foo.c and "gcc -c" recipe
  Accept "%.o : %.c  ->  gcc -c $<"
  2 if all of myprog.o, solve.o, line.o covered
  1 if two of them covered
  0.5 if one of them covered
0.5x5 marks for prereqs on the .h files,
      eg solve.o requires solve.h and line.h
```

6. In this question, you will implement in Q6.c (starter provided) setting up the processes and pipes in this diagram (there is also a parent process lurking, not shown):



The right process will eventually exec a program that isn't aware of this setup except to read from stdin and write to stdout. Therefore before exec, you will set up stdin to be the read end of the ltr pipe, stdout to be the write end of the rtl pipe.

The left process won't exec, instead call

```
void do_left(int fd_read, int fd_write);
```

Therefore you will call `do_left` so that `fd_read` is the read end of the rtl pipe, `fd_write` is the write end of the ltr pipe. Note that the left process inherits stdin and stdout from the parent, and `do_left` can perform I/O on all 4 distinct FDs.

Both left and right are child processes of a common parent. The code is structured as follows:

```
void leftright(void)
{
  // parent begins
  int ltr[2];
  int rtl[2];

  pipe(ltr);
  pipe(rtl);

  pid_t left = fork();
  if (left == 0) {
    // left child code, part (a)
  } else {
    pid_t right = fork();
    if (right == 0) {
      // right child code, part (b)
      execlp("right", "right", (char*)NULL);
      exit(1);
    } else {
      // parent code after both forks, part (c)
      wait(NULL);
      wait(NULL);
```

```
    }
  }
}
```

(a) [4 marks] Complete the code for the left child.

```
close(ltr[0]);
close(rtl[1]);
do_left(rtl[0], ltr[1]);

1 mark per close. But only before calling do_left
1 mark per do_left parameter
Deduct 0.5 marks per other close
```

(b) [5 marks] Complete the code for the right child.

```
dup2(ltr[0], 0);
dup2(rtl[1], 1);
close(ltr[0]);   // only after dup2(ltr[0], ...)
close(ltr[1]);
close(rtl[0]);
close(rtl[1]);   // only after dup2(rtl[1], ...)

1.5 marks per dup2. Also accept eg "close(0); dup(ltr[0]);"
0.5 marks per close.
Order doesn't matter apart from the comments above.
```

(c) [2 marks] Complete the code for the parent after forking and before waiting.

```
close all four ltr[0,1], rtl[0,1], any order
0.5 marks per close
```

7. [16 marks] Please submit solution to this question as Q7.c

The `do_left` from the last question

```
void do_left(int fd_read, int fd_write);
```

actually does this job:

- Copy data from `fd_read` to stdout, verbatim.
- Copy data from stdin to `fd_write` **but** change all lower case letters to upper case (in the sense of `toupper`).

Since there are two data sources with unknown arrival times, `select` is a simple way to wait for data, given that the two input FDs won't change.

Implement `do_left`. Use buffer sizes of 512 bytes. You may assume that `write` is successful and doesn't block. You may assume `fd_read` $\neq 0$. When any one of the input sources gives EOF, call `exit(0)`.

```
void do_left(int fd_read, int fd_write)
{
  fd_set rs;

  for (;;) {
    FD_ZERO(&rs);
    FD_SET(0, &rs);
    FD_SET(fd_read, &rs);
    select(fd_read + 1, &rs, NULL, NULL, NULL);
    if (FD_ISSET(0, &rs)) {
      char buf[512];
      int r = read(0, buf, 512);
      if (r == 0) exit(0);
      if (r > 0) {  // this check is optional
        for (int i = 0; i < r; i++) {
          buf[i] = toupper(buf[i]);
        }
        write(fd_write, buf, r);
      }
    }
    if (FD_ISSET(fd_read, &rs)) {
      char buf[512];
      int r = read(fd_read, buf, 512);
      if (r == 0) exit(0);
      if (r > 0) {  // this check is optional
        write(1, buf, r);
      }
    }
  }
```

```
}


2 marks: outer loop around the following
         (you don't have to give this if you think the rest is trash)

2 marks: set up read_fd_set to have 0 and fd_read (nothing else)
         before select
         Note: there are two style. My style is FD_ZERO then FD_SET.
         Another style has two fd_sets, preset one outside the loop,
         never change it again; in the loop, copy preset
         to 2nd fd_set, give that to select.
         Both are OK. In the 2nd style, even using "=" to copy is OK.

1 mark: select(fd_read + 1, &read_fd_set, NULL, NULL, NULL)
        It is OK to assume select returns +ve

When fd_read is ready:
  1 mark: test that fd_read is ready
  1 mark: read 512 bytes
  1 mark: exit(0) if read returns 0
  If read returns non-0: (I decided not to insist +ve)
  1.5 marks: write to 1, but watch out
             how many bytes (1 mark for correct count)

When 0 is ready:
  1 mark: test that 0 is ready
  1 mark: read 512 bytes
  1 mark: exit(0) if read returns 0
  If read returns non-0:  (I decided not to insist +ve)
  2 marks: toupper every byte
           I guess harmless to convert all 512 bytes
           But just don't exceed buffer size
  1.5 marks: write to fd_write, but watch out
             how many bytes (1 mark for correct count)

What if student reads 1 byte at a time, not 512 bytes?
Maybe deduct 2 marks.
```

(End of questions.)