CS 2413 Final

1. [**20**] Suppose a Perl program, `port`, is invoked with the command:

   ```
   port -axt file1 file2 file3
   ```

   and suppose `file1` contains the lines "`dir1`" and "`dir2`", while `file 2` contains "`dir3`", "`dir4`" and "`dir5`", and `file3` contains "`dir6`" and "`dir7`".

   At the start of the program:

   (a) What is the list value of `@ARGV`?

   (b) What is the value of `$#ARGV`?

   (c) What value is assigned to `$bat`:

   ```
   $bat = @ARGV;
   ```

   (d) After the following statements have been executed,

   ```
   $a = shift;
   $b = shift;
   push(@ARGV,$b);
   $c = <>;
   ```

   i. What is the value of `$b`?

   ii. What is the value of `$c`?

   iii. What is the value of `$ARGV`?

   iv. What is the list value of `@ARGV`?

   (e) On the other hand if the following statements were to be executed first:

   ```
   $a = shift;
   $b = shift;
   @c = <>;
   @ARGV = ( "file1" );
   @d = <>;
   ```

   i. What is the value of `$b`?

   ii. What is the value of `@c`?

   iii. What is the value of `@d`?

2. [**20**] Write a Perl script, called `ileaf`, which will interleave the lines of a file with those of another file writing the result into a third file. If the files are a different length then the excess lines are written at the end. A sample invocation:
   `ileaf file1 file2 outfile`

3. [**20**] Write a Perl script, `pgrep`, which will search all **TEXT** files whose names are on the command line or which are under any of the directories whose names are on the command line for the regular expression which is given as the first argument on the command line. A sample invocation:
`pgrep 'cat.*dog' file1 dir1 dir2 file2 file3 dir3`

4. [**20**] You are in a very insecure university environment and need to watch your security yourself so you run `snort` to notify you when you are under attack and you are particularly concerned about attacks on RPC (remote procedure call - port 111) from outside the UTSA domain (129.115.x.x). So you want to write a Perl script which will scan the log file `/var/log/portscan.log` and print to `stdout` those lines which match a given date and are either sent to port 111 on your machine `129.115.11.32:111` or broadcast `129.115.255.255:111`. The following line from the log file shows that there was a connection on Dec 12 sent from port 7500 on ip `129.115.110.101` to port 111 on ip `129.115.255.255` (broadcast) (important fields are indicated). This is allowed at UTSA.

```
Dec 12 13:51:46 129.115.110.101:7500 -> 129.115.255.255:111 UDP
^^^ ^^          ^^^^^^^^                ^^^^^^^^^^^^^^^^^^^
```

Assuming that the log file contains the following lines:

```
Dec 11 13:44:56 129.115.239.229:2620 -> 129.115.241.187:139 SYN ******S*
Dec 12 13:51:46 129.115.110.101:7500 -> 129.115.255.255:111 UDP
Dec 12 13:47:19 210.172.44.33:1025 -> 129.115.255.255:111 UDP
Dec 12 13:51:47 129.115.110.101:2297 -> 129.115.101.23:161 UDP
Dec 12 16:57:00 193.251.77.118:1100 -> 129.115.243.138:21 SYN ******S*
Dec 12 17:45:29 129.115.11.19:38449 -> 209.67.50.208:53 UDP
Dec 12 17:45:30 129.115.11.19:40069 -> 129.115.11.32:111 UDP
Dec 12 20:47:19 210.172.44.33:1025 -> 129.115.255.255:111 UDP
Dec 12 23:03:43 210.172.44.34:1091 -> 129.115.11.32:111 SYN ******S*
Dec 13 03:03:43 210.172.44.34:4675 -> 129.115.11.32:111 SYN ******S*
Dec 13 05:09:26 129.115.11.97:46562 -> 129.115.11.181:753 SYN ******S*
Dec 13 03:47:19 210.172.44.33:1025 -> 129.115.255.255:111 UDP
```

then the following invocation
`portscanlog Dec 12`
would output the following lines

```
Dec 12 13:47:19 210.172.44.33:1025 -> 129.115.255.255:111 UDP
Dec 12 20:47:19 210.172.44.33:1025 -> 129.115.255.255:111 UDP
Dec 12 23:03:43 210.172.44.34:1091 -> 129.115.11.32:111 SYN ******S*
```

5. [**25**] Write a C **function** with prototype:

```
int delblk(char *file, unsigned int num1, unsigned int num2, unsigned int rsz);
```

A file can be considered to be a sequence of record of a fixed size, say **rsz**, where the first **rsz** bytes are to be considered record 0, the next **rsz** bytes are record 1, ... The function **delblk** will delete records `num1` to `num2` inclusively from `file1`. The records must be read/written one at a time but you may assume that they are less than 4096 in size. The function must also both open and close the files. If successful, **delblk** will return a 0 else **delblk** will return -1. The function should use low-level I/O. A useful function is

```
int ftruncate(int fd, off_t length)
```

which will set the size of the file referenced by `fd` to a size of precisely `length` bytes. Points will be allocated for efficiency of solution in terms of number of records written.

6. [**25**] Write a `function` with prototype

```
int pd2open(char *prog, char *argv[], int *fdread, int *fdwrite);
```

which will fork and exec the program with name, `prog`, and with the given `argv` array. Stdin and stdout of `prog` should be hooked to pipes with `fdwrite` being set to the write end of the stdin pipe and `fdread` being set to the read end of the stdout pipe. The function should return -1 on error and 0 otherwise.

7. [**25**] You are on a system in which the `finger` program has been disabled and you want a quicky finger type program and you decide that greping `/etc/passwd` would be sufficient. However the system that you are on uses `nis+` and so nothing is in the password file. In order to essentially get the contents of the password file you can execute `niscat passwd.org_dir`. Thus you decide to write a C program, `myfinger`, that will execute `niscat passwd.org_dir` and then pipe the stdout of the `niscat` program into `egrep rexp` where `rexp` is the regular expression given on the command line. Write the `myfinger` program which can then be used, for example, to find information about a user named George by executing:
`myfinger George`

8. [**25**] Write a C `program` which will fork 11 children. The original process and the first child will be readers while the remaining children will be the writers. There should be two pipes, one for the writers to read from and one for the readers (parent and first child) to read from. The parent will write 1000 integers, 1 to 1000, to the writers' pipe. The writers will each read as many integers as they can, counting the number read. When the pipe is empty each writer will write to the readers his pid and the number of integers read. The readers will read individual writers' reports and print on stdout a line similar to

   ```
   Process0: Child 47836 read 132 integers
   ```

   Note that the original process should call itself Process0 while the first child will call itself Process1.

9. [**30**] In the system that you are writing, many programs (called clients) will be executing and will need other programs, called servers, to process some data. In this version of the system each program will write to a well known fifo (named pipe) with the name "`/tmp/dispatch`". You should assume that this fifo already exists. The dispatcher program reads a message from the fifo, execs the appropriate server and then writes the data to the server. The server is expecting its data via stdin and will write its data to stdout. The dispatcher must be certain that the server's stdout is redirected to the fifo, created by the client, named in the message. The format of the message is:

Data	**Field (first byte - last byte)**
program name	0-255
return fifo name	256-511
data	512-1023

   Write the dispatcher program (in C) for this system assuming that there may be several dispatcher programs executing at the same time. There should be no long-lived zombies created nor should the dispatcher wait for a potentially long time. The dispatcher should follow the current path (bad idea) to find the program when exec'ing.

   Note that a dispatcher does not terminate but constantly reads messages from the dispatch fifo.