CS 3423 Test 2

1. [**30**] Write a C **function** with prototype:

   ```
   int insrec(char *fdb, int num, char *fdbin, int numin, int rsz);
   ```

   This function will read record `numin` from the file `fdbin` and will insert the record in position `num` in the file `fdb`. Notice that no records should be overwritten and `fdb` will be one record larger. `fdbin` will be unchanged. You may assume that all records exist in the files.

   A record is merely a consecutive block of bytes of size **rsz**. The record indexing starts at 0 so record 0 consists of bytes 0 through $rsz - 1$, record 1 consists of bytes rsz through $2rsz - 1$, and so on. The function must both open and close the files and reads/writes **must (NOTE MUST)** be record sized. If successful, **insrec()** will return a 0 else **insrec()** will return -1. The function must use low-level I/O except for printing error messages and you may assume that $rsz \leq 4096$.

2. [**25**] You are running OpenWRT on your wireless router and have ssh listening at port 222 on the wireless router. Since you have been seeing many ssh scans you are interested in when (and who) do ssh scans on your wireless router. Since you run snort on another machine that information is logged but the amount of data is staggering so you want a simple program which you can run in a window that will only report probes at port 222 on your wireless router. This could be done with the command-line pipe:

   ```
   tail -f /var/portscan.log | egrep 129.115.27.203:222
   ```

   but you want a single program. Write a C `program` which will perform this functionality.

3. [**25**] Write a C `program` which will fork **num** (where num is a command-line argument) children. Every child whose pid is odd (but not the original parent) will write his pid to every other child (but not parent). Each child will then read every message written to him, one at a time, and then write each received pid along with his pid to the original parent using the following message structure.

   ```
   struct message {
     pid_t my_pid;
     pid_t received_pid;
   }
   ```

   On receipt of each message, the original parent process will print to stdout a message similar to:

   ```
   Process 1234 has received a message from Process 4567
   ```

4. [**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.

   Basically the main program wants to execute `prog` and write data to `prog` (via `fdwrite`) and read the output from `prog` from the variable `fdread`. In particular `pd2open` returns two file descriptors by putting them in the variables whose addresses are passed to the function.