CS 2413 Test 2

1. **[25]** Write a C **function** with prototype:

   ```
   int delrecs(char *fdb, int rnum, int num, int rsz);
   ```

   This function will delete the `num` records, starting with record `rnum` in the file whose name is given by `fdb`. Notice that there cannot be holes in the file, i.e. the resulting file must contain the remaining records as contiguous records. You may assume that the file contains at least `rnum + num` records.

   Furthermore you may assume that the file consist of an integral number of records and that all records have the same size, **rsz**. 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 file and read/writes **must** be record sized. If successful, **delrecs()** will return a 0 else **delrecs()** will return -1. The function must use low-level I/O except for printing error messages and you may assume that $rsz \leq 1024$.

2. **[25]** There are constant attacks on your machine consisting of thousands of ssh login attempts. This has no chance of working since you don't use word-based passwords on your system but they irritate you and fill up your log files making it next to impossible to read your logs. You have written a program, `sshreject whitelist` which will read from stdin and if it finds 5 failed login attempts without a success from the same IP (not in the file whitelist) will reject the route. Thus your machine becomes a "black hole" to that IP. However your program has to read the log file `/var/log/auth.log` which is periodically rotated. The `tail` program can be told to ignore the rotation and continue reading from the new `auth.log` file. Thus this could be done on the command line with the command:

   ```
   tail -F /var/log/auth.log | sshreject myips
   ```

   Write a C program that will implement the above.

3. **[25]** Write a C program which will create a **total** of 40 processes all of which will have their own separate pipe, i.e. all processes will be able to write to all other processes but each process will only read from its own pipe. Each process will be assigned a unique index 0, 1, . . . , 39. Each process **with an odd pid** will write it's index and pid to process 0 (possibly including process 0). The processes with even pids will write their index and pid to all other processes (but not to themselves). Then each process (odd and even) will read all the messages written to its pipe and print on stdout, for each message, a line similar to:
   Process 35894 index 24 heard from process 35992 index 17
   Be careful that your code doesn't hang!

4. **[25]** Write a `function`, called `xmsg()`, which will read **all** messages from the pipe whose file descriptor is passed to the function. Each message will contain the name of an executable and a single argument. The function will cause the execution of the executable with the single argument and return -1 on error or 0 on reaching an end-of-file on the pipe. The function cannot wait for the executable nor do we want to create zombies. The function `xmsg()` has the following prototype:
   `int xmsg(int fd);`
   and the message structure is:

   ```
   struct command
     {
       char cmd[64];      /* Null terminated name of executable */
       char arg[64];      /* Single Argument to command         */
     };
   ```