CS 2413 Test 2

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

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

   This function will insert the record stored at memory location `rec` **AFTER** record number `num` in the file whose name is given by `fdb`. Notice that a hole in the file must be made in order to insert the new record in its current position. You may assume that the file contains at least `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, **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 $rsz \leq 1024$.

2. [**25**] You are running `portsentry` on your machine which detects probes at various ports and will block the prober. You want a program which will report the attacks which occured after the last report. The last report was based on the file which is currently named `/var/log/ps.udp.1` while the current set of attacks (which includes the old attacks) is stored in `/var/log/ps.udp.0`. Write a C program that will do a diff between these two files and email root the results. This could be written on the command line as below (but don't).

   ```
   diff /var/log/ps.udp.1 /var/log/ps.udp.0 | mail -s Attack root
   ```

3. [**25**] Write a C program which will create a **total** of 35 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, ..., 34. Each process **with an odd pid** will write it's index and pid to every other process (but not to itself). The processes with even pids will not write any messages. 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**] 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 client will write a message 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 making certain that stdin and stdout are connected correctly. 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 return fifo, created by the client and that the server's `stdin` is redirected to the data fifo also created by the client and both 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 fifo name	512-767

   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.