# CS 2413 Test 2

1. **[25]** Write a C **program** with invocation:

   ```
   recselect filein fileout rsz rnum1 rnum2 ... rnumk
   ```

   This program will read the k records `rnum1`, ..., `rnumk` from **filein** and append the records to **fileout**. All records in **filein** and **fileout** 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`, ... The function must both open and close the files. If successful, **recselect** will return a 0 else **recselect** will return -1. The program should use low-level I/O except for error messages and you may assume that `rsz` $\leq$ 1024. Notice that `rnum1`, ..., `rnumk` need not be in sequential order,
   7, 9, 345, 12 is valid. A correct solution that works for any size record will earn extra credit. On error, return -1.

2. **[25]** You are writing a program that needs a list of all of the users on the system. The program `dispuid` will return a list of valid users, one per line. So write a function with the prototype:

   ```
   int users(void);
   ```

   which returns a file descriptor from which the main program can read the list of valid users for the system and -1 on error.

3. **[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`. Thus you want a program that will do a diff between these two files and email root the results. Written on the command line the following would do what you want your program to do:

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

4. **[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 will write it's index and pid to every other process (but not to itself). Then each process 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!