CS 3423 Test 2

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

   ```
   int delrec(char *fdb, int num, int rsz);
   ```

   This function will delete the record `num` from the file `fdb`. Since this is a sequential file of records the records after record `num` must be moved up to fill in the missing record and the file shortened.

   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, **delrec()** will return a 0 else **delrec()** 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 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 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`

3. **[25]** Write a C program which will create 23 children processes and two pipes. All processes should know about both pipes. The original parent process will write 1000 messages to the pipe each consisting of a single integer starting with 0 up to 999. Each of the 23 child processes will read as many messages (one at a time) from the pipe as possible, printing out on stdout a message containing the child's pid and the message number. The general form of the message is:
   `Child 12475 received message 134`
   Each child will then write to the parent a message consisting of the number of messages that that child received. The parent will add up the total number of messages that the children claim to have received and, if the total is 1000 will print the following message to stdout:

   ```
   All messages are accounted for
   ```

   and if the total is less that 1000 (for example say 945) will print:

   ```
   945 messages are accounted for
   ```

OVER

4. **[25]** Write a `function`, called `xmsg()`, which will read **all** messages, whose structure is given below, from the pipe whose read file descriptor is passed to the function. Each message will contain the name of an executable command as well as two arguments to the command. The function will cause the execution of the command with it's two arguments and return -1 on error or 0 when an EOF is reached on the pipe. The function `xmsg()` has a prototype of:

   ```
   int xmsg(int fd);
   ```

   and the message structure is:

   ```
   struct command
     {
       char cmd[64];       /* Null terminated name of executable */
       char argv1[10];     /* Argument 1*/
       char argv2[10];     /* Argument 2*/
     };
   ```