CS 3423 Test 2

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

   ```
   int extractrec(char *fdb, int num, char *irec, int rsz);
   ```

   This function will `extract` record `num` from the file whose pathname is given by `fdb`, writing it to the memory location given by `irec`. Note that the file will be one record shorter.

   You may assume that both files 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 reads/writes **must** be record sized. If successful, **extractrec()** will return a 0 else **extractrec()** 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 want a program that will return a list of all of the user names on a system which is running `ldap`. Since the system is running `ldap`, **/etc/passwd** doesn't contain the users so in order to get the users you need to execute `getent passwd` which will print on stdout the (essentially) normal `passwd` contents which you can then pipe into sed to pick out the user names from each line. For example on the command line you could execute

   ```
   getent passwd | sed 's/:.*//'
   ```

   Write a **C program** called `getusers.c` which will output a list of usernames, i.e. write a C program that does the same as the above command using the unix programs `getent` and `sed`. Note that the single quotes in the `sed` command are only there to protect the string from the shell and are not seen by `sed`.

3. [**25**] Write a C program which will create a **total** of 37 processes all of which will have their own separate pipe, i.e. all processes will need to be able to write to all other processes but each process will only need to read from its own pipe. Each process must be assigned a unique index 0, 1, ..., 36. 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 and messages don't get mixed up!

(OVER)

4. **[25]** Write a `function`, called `xcmd()`, which will read all messages from a pipe. The message structure is given below and the read file descriptor of the pipe is passed to the function. Each message will contain the name of an executable and the names of two fifos. The function will cause the execution of the executable with no arguments and the stdin and stdout of the process should be connected to the input (data) FIFO, `dfifo`, and the output (return) FIFO, `rfifo`. The function `xcmd()` has a prototype of:
`int xcmd(int fd);`
and the message structure is:

```
struct command
  {
    char cmd[64];      /* Null-terminated name of executable          */
    char dfifo[64];    /* Name of data FIFO                           */
    char rfifo[64];    /* Name of return FIFO                         */
  };
```