CS 3423 Test 2

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

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

   This function will delete the block of `num` records starting at record `first` from file whose name is given by `fdb`. Notice that the records at the end of the file must be moved up to fill the positions of the deleted records and the file shortened. You may assume that all records exist in the files.

   Furthermore 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 files 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**] 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 normal `passwd` contents which you can then pipe into sed to pick of the user names. 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.

3. [**25**] Write a C `program` which will fork 23 children assigning each process a unique index from 0 to 23. After creation the original process (process 0) will write his pid to each child process. Each child process (processes 1, ..., 23) will read the message from process 0 and those with an **odd pid** will write a message back to process 0 that consists of the received pid and the child's pid. Process 0 will read all the messages written to him and will print on stdout a string similar to `"process 23456 received message from 98765\n"`.

(OVER)

4. [**30**] Write a `function`, called `xcmd()`, which will read all messages from a pipe. The message structure is given below and the open 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                         */
  };
```

The extra 5 points is for correct zombie handling.