CS 3423 Test 2

You do not need to error trap lseeks, opens, reads, writes, and pipe commands. Please error trap forks and execs.

1. **[30]** Write a C **function** with prototype:

   ```
   int mvrec(char *fdb, int num1, char *fdb2, int num2, int rsz);
   ```

   This function will extract record num1 from the file whose pathname is given by **fdb**, inserting the extracted record in the file whose pathname is given by **fdb2**. Note that the file **fdb** will shorter by one record and the file **fdb2** will be longer by one record. You may assume that **fdb** and **fdb2** have more records than both **num1** and **num2**. No records in **fdb2** should be overwritten, the record should be **inserted**.

   You may also assume that both files **fdb** and **fdb2** 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 reads/writes **must** be record sized. If successful, **mvrec()** will return a 0 else **mvrec()** 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 using a distribution of linux on your home machine which is using **systemd** so you no longer have the standard log files available and must use **journalctl** to access the equivalent data. If you have a power failure or other issues at home you netcat a short message to port 22 (ssh) on your office machine, saying "error msg" (The message would actually be more accurate). This generates a log message of the form

   ```
   Nov 21 09:40:00 exely sshd: Bad protocol 'error msg' from 187.153.134.175
   ```

   Thus we could extract and record these messages as they appear in a log file using

   ```
   journalctl -f --full | grep --line-buffered Bad >> /var/log/power.log
   ```

   Write a single C program which will accomplish the same thing as the above command using the unix programs `journalctl` and `grep`.

3. **[25]** Write a C program which will create a **total** of 33 processes, each process having its own pipe (i.e. one that it will read from). Each process must be assigned a unique index 0, 1, ..., 32. The original process will send an integer generated by **rand()** to processes, 1, 2, ..., 32. Each child will read the number from its pipe and, if even, will write to the parent (process 0) the random number read, its pid and its index. The original parent (process 0) will read each message and print a message similar to

   ```
   Original Process 0 read random number 3157 from process 35991 index 17
   ```

   Be careful that your code doesn't hang and messages don't get mixed up!

(OVER)

4. **[25]** Write a `function` with prototype

   ```
   int pdopen(char *prog1, char *arg1, char *prog2, char *arg2);
   ```

   which will cause the equivalent of the pipeline

   ```
   prog1 arg1 | prog2 arg2
   ```

   to be executed. prog1 reads its data from stdin so the original process needs to be able to write this data to prog1. **pdopen** will return the file descriptor which the original process will use to write the data to prog1.