The Midterm Strikes Back!

Grading Structure

The midterm has 4 questions. The grade break down is given as follows:

Questions	Points
Q1	30
Q2.1	10
Q2.2	30
Q2.3	30
Total	100

Note that *each question* is covered by many grading scripts and that you can earn partial credit. Some are easier to pass then others. I want to emphasize, once again, that the *solutions are short.* If you find yourself writing a substantial amount of code, you are most likely making this too hard on yourself. Note that we provide a `Makefile` as well as startup code. Do check it out before starting.

Q1. Pipeline command.

Complete the main() function in *Q1.c* to implement the following pipeline command:

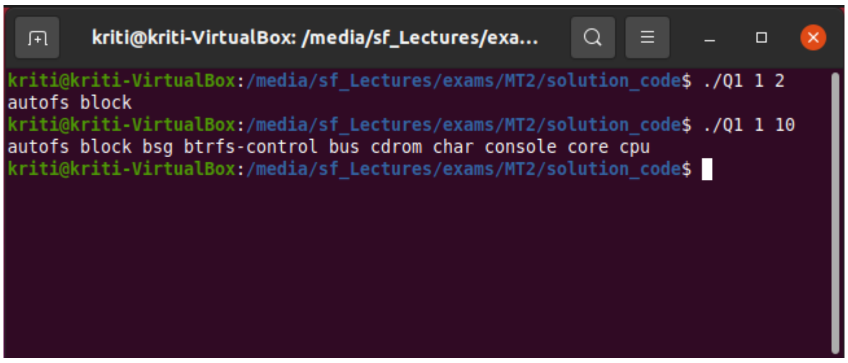
```
ls /dev | xargs | cut -d' ' -f<a>-<b>
```

where <a> and <b> are positive integers specified as command line arguments of your program

In this instance, the pipeline consists of three external commands, which can take zero or more arguments, chained together to form a single instruction. The pipeline has 3 stages:

1. The first stage outputs the list of files in `/dev`.
2. The second stage takes the output of stage 1 on its standard input, delimits it by blanks, and echos them on its standard output.
3. The third stage takes the output of stage 2 as its standard input and prints the selected fields (<a>-<b>) on the standard output delimited by a blank. Note that the pair of quotes after `-d` are not needed when passing arguments through `execl`.

Below are sample outputs from running the program on my system. Note: the output can vary on each machine. The length of the output, however, should be the same as the below examples.



```
kriti@kriti-VirtualBox:/media/sf_Lectures/exams/MT2/solution_code$ ./Q1 1 2
autofs block
kriti@kriti-VirtualBox:/media/sf_Lectures/exams/MT2/solution_code$ ./Q1 1 10
autofs block bsg btrfs-control bus cdrom char console core cpu
kriti@kriti-VirtualBox:/media/sf_Lectures/exams/MT2/solution_code$
```

Figure 1: A screenshot of a computer Description automatically generated with medium confidence

Q2. A client-server application

Complete the code in *Q2service.c, Q2client.c and Q2server.c* to implement a client-server application for SQLite. SQLite is a relational database management system that takes as input a query, consults a database and returns with an appropriate response. The functionality to be implemented in each of the above programs is as follows:

1. *Q2server.c*: This is a server process that runs on the localhost over TCP. The server is listening *on two ports: 9000 and 9001* and must respond to inbound chat request in the following way:

    a. Inbound requests on port 9000: run an instance of SQLite3 for each client, which reads and executes the stream of SQLite commands coming over the socket, and sends back the response. Note: the executable for SQLite is sqlite3. It takes as input a database file. You have been provided with foobar.db in the starter code. An example .csv file containing test data has also been given.

    b. Inbound requests on port 9001: read commands from the socket and terminate the server if the command is '$die!'. Any other request should be met by producing the message (on a single line)

    ```
    bad command [<command that was sent>]\n
    ```

   c. reap all the dead children.

2. *Q2client.c*: This is a client process that must

    a. connect to the server process (via port 9000) running on the hostname provided by the user. For testing purposes, your server (Q2server) is running on localhost.

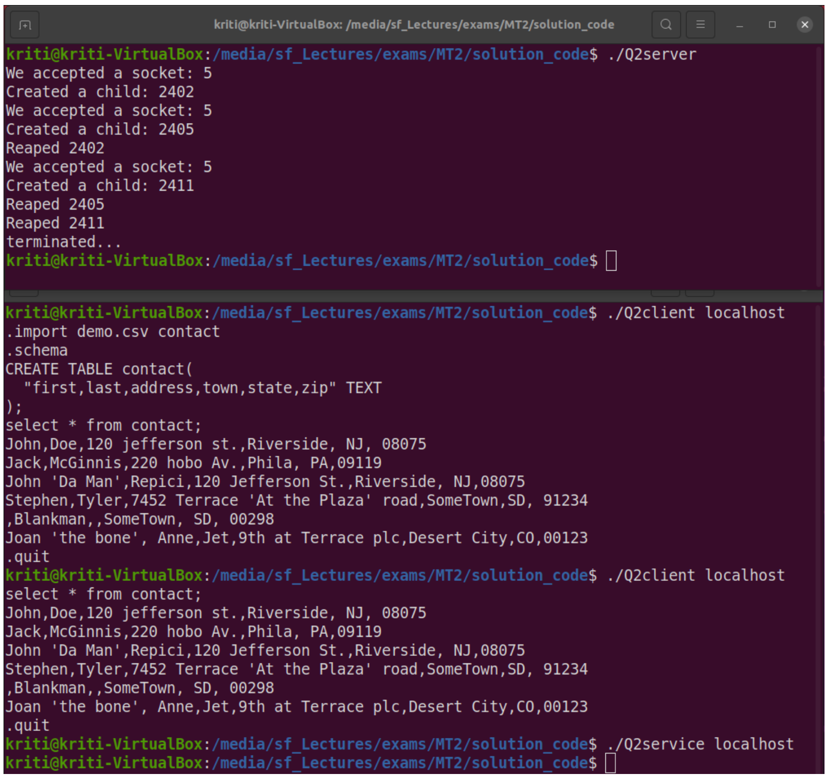
   b. read SQLite commands from stdin and send them to the server. Note: you are provided with sample SQL queries in the scr1.sql file.
   c. read the responses from the server and print them out on stdout.

3. *Q2service.c*: This is a client process that must

   a. connect to the server process (via port 9001) running on the hostname provided by the user. For testing purposes, your server (Q2server) is running on localhost.
   b. send the `'\$die!'` command to terminate the server process.

You need to complete the main() function in *Q2service.c* **(Q2.1),** *Q2client.c* **(Q2.2)** *and Q2server.c* **(Q2.3)** to implement the above functionalities. In case of *Q2client.c*, you are provided with the code to read a SQLite command from the stdin. In case of *Q2server.c*, you are provided with boilerplate code to create and bind a socket to a given address and listen for incoming TCP connect requests.

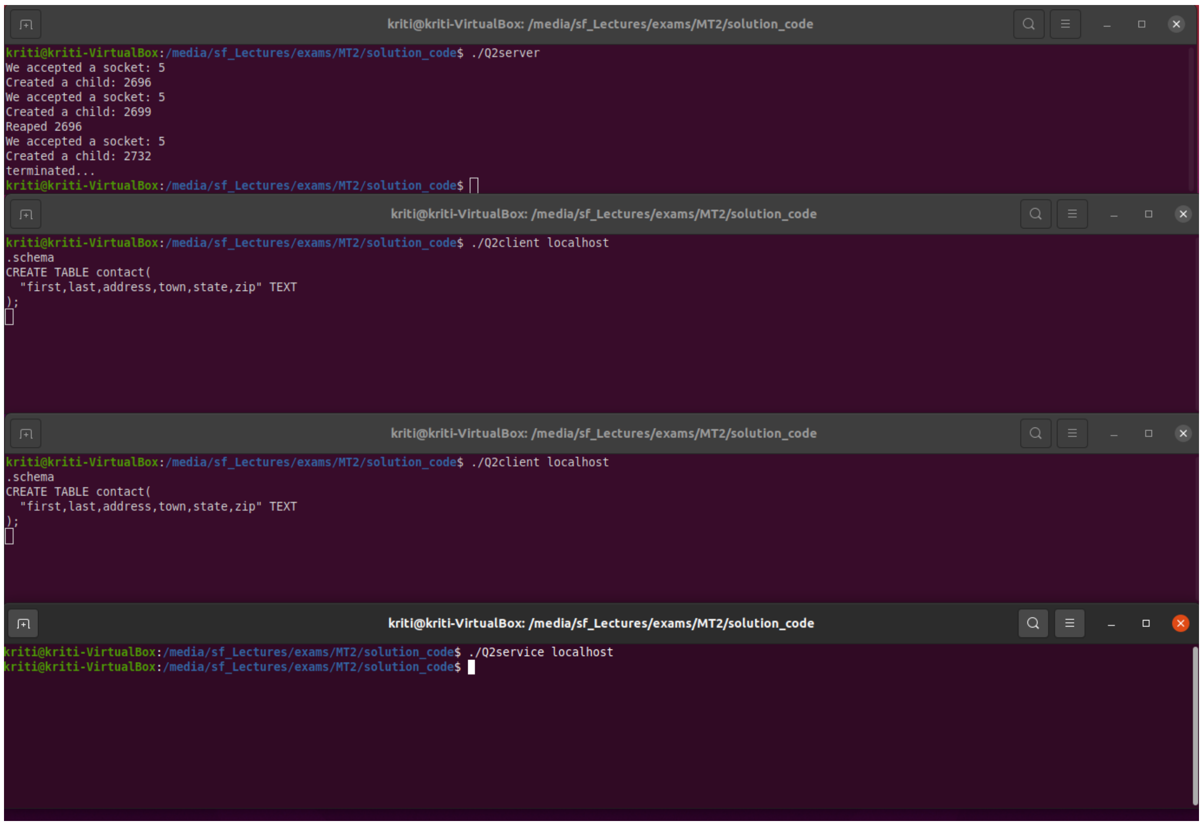
Below are sample sessions running the applications.



```
kriti@kriti-VirtualBox:/media/sf_Lectures/exams/MT2/solution_code$ ./Q2server
We accepted a socket: 5
Created a child: 2402
We accepted a socket: 5
Created a child: 2405
Reaped 2402
We accepted a socket: 5
Created a child: 2411
Reaped 2405
Reaped 2411
terminated...
kriti@kriti-VirtualBox:/media/sf_Lectures/exams/MT2/solution_code$
```

```
kriti@kriti-VirtualBox:/media/sf_Lectures/exams/MT2/solution_code$ ./Q2client localhost
.import demo.csv contact
.schema
CREATE TABLE contact(
  "first,last,address,town,state,zip" TEXT
);
select * from contact;
John,Doe,120 jefferson st.,Riverside, NJ, 08075
Jack,McGinnis,220 hobo Av.,Phila, PA,09119
John 'Da Man',Repici,120 Jefferson St.,Riverside, NJ,08075
Stephen,Tyler,7452 Terrace 'At the Plaza' road,SomeTown,SD, 91234
,Blankman,,SomeTown, SD, 00298
Joan 'the bone', Anne,Jet,9th at Terrace plc,Desert City,CO,00123
.quit
kriti@kriti-VirtualBox:/media/sf_Lectures/exams/MT2/solution_code$ ./Q2client localhost
select * from contact;
John,Doe,120 jefferson st.,Riverside, NJ, 08075
Jack,McGinnis,220 hobo Av.,Phila, PA,09119
John 'Da Man',Repici,120 Jefferson St.,Riverside, NJ,08075
Stephen,Tyler,7452 Terrace 'At the Plaza' road,SomeTown,SD, 91234
,Blankman,,SomeTown, SD, 00298
Joan 'the bone', Anne,Jet,9th at Terrace plc,Desert City,CO,00123
.quit
kriti@kriti-VirtualBox:/media/sf_Lectures/exams/MT2/solution_code$ ./Q2service localhost
kriti@kriti-VirtualBox:/media/sf_Lectures/exams/MT2/solution_code$
```

Figure 2: The Server, two clients and a run of the service app.


kriti@kriti-VirtualBox: /media/sf_Lectures/exams/MT2/solution_code

```
kriti@kriti-VirtualBox:/media/sf_Lectures/exams/MT2/solution_code$ ./Q2server
We accepted a socket: 5
Created a child: 2696
We accepted a socket: 5
Created a child: 2699
Reaped 2696
We accepted a socket: 5
Created a child: 2732
terminated...
kriti@kriti-VirtualBox:/media/sf_Lectures/exams/MT2/solution_code$
```

```
kriti@kriti-VirtualBox:/media/sf_Lectures/exams/MT2/solution_code$ ./Q2client localhost
.schema
CREATE TABLE contact(
  "first,last,address,town,state,zip" TEXT
);
```

```
kriti@kriti-VirtualBox:/media/sf_Lectures/exams/MT2/solution_code$ ./Q2client localhost
.schema
CREATE TABLE contact(
  "first,last,address,town,state,zip" TEXT
);
```

```
kriti@kriti-VirtualBox:/media/sf_Lectures/exams/MT2/solution_code$ ./Q2service localhost
kriti@kriti-VirtualBox:/media/sf_Lectures/exams/MT2/solution_code$
```

Figure 3: Another run with two clients listing the database schema