Introduction The purpose of this work is to familiarize you with creating processes using system calls fork / exec, process communication through pipes, using low-level I / O and creating bash scripts.

As part of this work you will implement a distributed travelMonitor tool that will receive requests from citizens who want to travel to other countries, check if they have been properly vaccinated, and approve whether a traveler is allowed to enter a country. Specifically, you will implement the travelMonitor application which will create a series of monitor processes that, together with the application, will answer user questions.

A) The travelMonitor application (90%)

The travelMonitor application will be used as follows:

./travelMonitor –m numMonitors -b bufferSize -s sizeOfBloom -i input_dir

where:

- The numMonitors parameter is the Monitor number of processes that the application will create. - The bufferSize parameter: is the size of the buffer for reading over pipes. The sizeOfBloom parameter specifies the size of bloom filters in * bytes *. Indicative size of the bloom filter for exercise data will be of the order of 100Kbytes. This parameter is of the same logic as the corresponding parameter of the first task.

- The input_dir parameter: is a directory that contains subdirectories with the files that will be processed by Monitor processes. Each subdirectory will have the name of a country and will contain one or more files. For example, input_dir could contain subdirectories China / Italy / and Germany / which have the following files:

-

/input_dir/China/China-1.txt /input_dir/China/China-2.txt /input_dir/China/China-3.txt
...
/input_dir/Italy/Italy-1.txt /input_dir/Italy/Italy-2.txt
...
/input_dir/France/France-1.txt /input_dir/France/France-2.txt /input_dir/France/France-3.txt
...

Each file contains a series of citizen records where each line describes a citizen's vaccination status for a specific virus. For example, if the contents of the /input_dir/France/France-1.txt .txt file are:

```
889 John Papadopoulos France 52 COVID-19 YES 27-12-2020
889 John Papadopoulos France 52 H1N1 NO
776 Maria Tortellini France 36 SARS-1 NO
125 Jon Dupont France 76 H1N1 YES 30-10-2020
```

means that in France we have a citizen (John Papadopoulos) who has been vaccinated for COVID-19 on 27-12-2020 but not for H1N1, Maria Tortellini who has not yet been vaccinated for SARS-1, and Jon Dupont who was vaccinated for H1N1. Similar to the first exercise if there are lines that are contradicting with previous ones, you can discard the records that create inconsistency in the same way as in the first exercise.

Specifically, an entry is an ASCII line of text consisting of the following elements in this order: - citizenID: a string (it can have a single digit) that defines each such

sign up.
- firstName: a string consisting of letters without spaces. - lastName: a string consisting of letters without spaces. - age: a positive (> 0), integer <= 120. - virusName: a string consisting of letters, numbers, and possibly a

hyphen "-" but without spaces.
- YES or NO: indicates whether the citizen has been vaccinated against the virus. - dateVaccinated: date the citizen was vaccinated. If the previous field is NO there is a dateVaccinated, not field in the record.

To get started, the travelMonitor application should create a set of named pipes to communicate with numMonitors child processes that it will create. Next, you need to do fork numMonitors child processes. Each child process will call the exec with an executable file a program called Monitor that you will write and that will take as arguments the paths of the named pipes that will be used for communication between the initial parent process travelMonitor and each exec'd Monitor process. Then, the parent process will inform each Monitor via named pipe about the subdirectories that the Monitor will undertake. TravelMonitor will distribute subdirectories evenly (in round-robin alphabetically) with the countries in input_dir in Monitor processes. You can assume that subdirectories will be flat, that is, they will only contain files, not subdirectories.

When the application (parent process) finishes the initialization actions, it will wait for a series of bloom filters from the Monitor processes (see below) and when it receives all the information, it will be ready to accept input (commands) from the user from the keyboard (below for commands).

Each Monitor process, for each directory assigned to it, will read all its files and fill in a series of data structures that it will use to answer questions posed by the parent process. One structure will be the bloom filter which, as in the first work, will be used to quickly check if a citizen (with a citizenID ID) has been vaccinated for this virus. Each Monitor process, after having finished reading the input files, will send through the named pipe to the parent process a bloom filter for each virus that will represent all the vaccinated citizens of the countries managed by the Monitor process. The choice of the number of named pipes between the parent process and the Monitor processes, as well as other data structures is your design choice. If, while reading files, a Monitor process detects a problematic record (eg in syntax or an inconsistent record), it will ignore the problematic record. When the Monitor process finishes reading its files and has sent all the Bloom filters to the parent, it notifies the parent process via named pipe that the Monitor is ready to accept requests.

If a Monitor process receives a SIGINT or SIGQUIT signal then it prints in a file named log_file.xxx (where xxx is its process ID) the name of the countries (subdirectories) it manages, the total number of requests received to enter countries it manages, and the total number of requests approved and rejected.

Logfile format:

```
Italy
China
Germany
TOTAL TRAVEL REQUESTS 29150
ACCEPTED 25663
REJECTED 3487
```

If a Monitor process receives a SIGUSR1 signal, it means that 1 or more new files have been placed in one of the subdirectories assigned to it. Assume that no new directories are created with countries and that existing files are not modified. That is, there will be only new files. The monitor process will check the subdirectories to find the new files, read them and update the data structures it holds in memory. After reading the new files, it will send the updated bloom filters to the parent process.

If a Monitor process terminates abruptly, the parent process will have to fork a new Monitor process to replace it. Therefore, the parent process should handle the SIGCHLD signal, as well as SIGINT and SIGQUIT.

If the parent process receives SIGINT or SIGQUIT, it must first finish processing the current command from the user and after responding to the user, it will send a SIGKILL signal to the Monitors, wait for them to terminate, and finally print a file named log_file.xxxx where xxx is its process ID, the name of all the countries (subdirectories) that participated in the data application, the total number of requests to enter the countries, and the total number of requests approved and were rejected.

Logfile format:

```
Italy
China
Germany
TOTAL TRAVEL REQUESTS 883832
ACCEPTED 881818
REJECTED 2014
```

The user will be able to give the following commands to the application:

Travel / travelRequest citizenID date countryFrom countryTo virusName The application will first check the bloom filter sent to it by the Monitor process that manages the countryFrom country. If the bloom filter indicates that the citizenID citizen has not been vaccinated against the virusName prints

```
REQUEST REJECTED - YOU ARE NOT VACCINATED
```

If the bloom filter indicates that the citizenID citizen may have been vaccinated against the virusName, the application requests via named pipe from the Monitor process that manages the countryFrom country if the citizenID has actually been vaccinated. The Monitor process responds via named pipe YES / NO where in the case of YES, it sends the date of vaccination. For the answer the Monitor process can (but it is not necessary) to use the structures you had in task 1 (eg the skip list). The application checks if the citizen has been vaccinated less than 6 months before the desired trip date and prints one of the following messages

```
REQUEST REJECTED - YOU ARE NOT VACCINATED
REQUEST REJECTED - YOU WILL NEED ANOTHER VACCINATION BEFORE TRAVEL DATE
REQUEST ACCEPTED - HAPPY TRAVELS
```

• / travelStats virusName date1 date2 [country]

If no country argument is given, the application will print the number of citizens who have applied for permission to travel through space [date1 ... date2] to countries checking for vaccination against the virusName, and the number of citizens approved and rejected. If a country argument is given, the application will print the same information but only for that country. The date1 date2 arguments will be in DD-MM-YYYY format.

Output format: Example: TOTAL
```
REQUESTS 29150 ACCEPTED
25663
REJECTED 3487
```

Add / addVaccinationRecords country With this

request the user has placed in input_dir / country one or more files for processing by the Monitor process. The parent process sends a notification via SIGUSR1 signal to the Monitor process that manages the country that there are input files to read in the directory. Monitor process reads that it finds a new file, updates the data structures and sends back to the parent process, through named pipe, the updated bloom filters that represent the total number of citizens who have been vaccinated

• / searchVaccinationStatus citizenID The parent process

forwards to all Monitor processes the request via named pipes. The Monitor process that manages the citizen with a citizenID ID sends via named pipe whatever information he has about the vaccinations that the specific citizen has done / has not done. When it receives the information, the parent prints it to stdout.

Output format: Example:

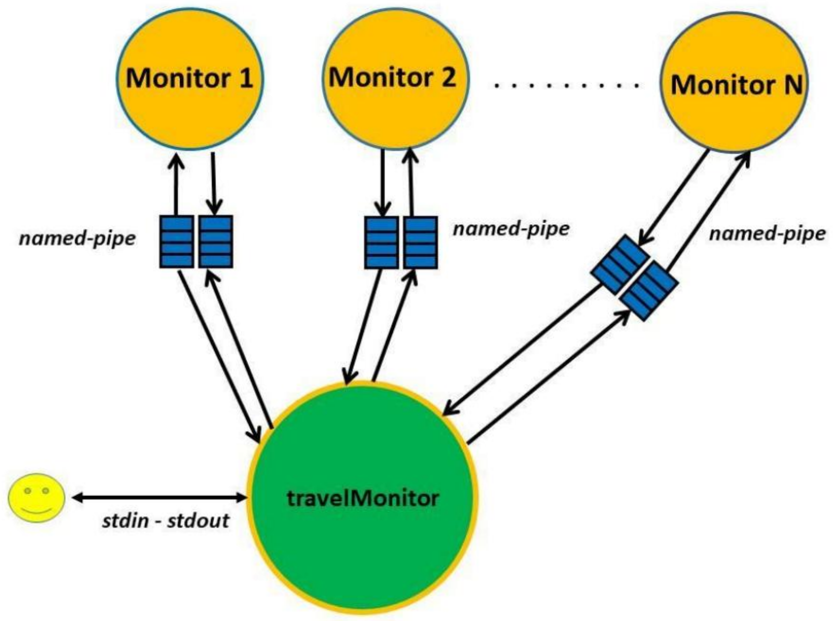
```
889 JOHN PAPADOPOULOS GREECE
AGE 52
COVID-19 VACCINATED ON 27-12-2020
H1N1 NOT YET VACCINATED
```

Exit / exit

Exit the application. The parent process sends a SIGKILL signal to the Monitors, waits for them to terminate, and prints it in a file named log_file.xxxx where xxx is its process ID, the name of all the countries (subdirectories) that participated in the data application, the total number of applications received for entry into the countries, and the total number of applications approved and rejected. Before it terminates, it will properly release all the free memory.

Log file format (as above):

```
Italy
China
Germany
TOTAL TRAVEL REQUESTS 883832
ACCEPTED 881818
REJECTED 2014
```



The communication between the parent process and each Monitor takes place via named pipes (see image).

The default behavior of named pipes is to enter the standby process by opening one end until the piping opens at the other end. Of course, we can avoid the above behavior if we set the O_NONBLOCK flag to the second argument of the open system call (). For example, if we want to open a named pipe for reading without having to wait, we call open (pipe_name, O_RDONLY | O_NONBLOCK). You are free to choose which method of piping operation you want.

Tips & Warnings The most difficult part of the job is managing named pipes. As you plan your work, think about issues such as: 1) What happens when the -b bufferSize parameter is much smaller than the data that a process wants to send over a named pipe? 2) What happens when the Monitor receives a SIGUSR1 signal while in the middle of sending data to

in the parent process?

3) How will you design the parent process so that it does not block waiting for a slow Monitor while there is another Monitor that has sent data and is waiting to be read in the named pipe? (You may find the select () call useful.)

4) How will a Monitor be informed that the parent process intends to transmit an order through the named pipe so that the latter can go to read it? (This, for example, could

achieved by sending a pre-agreed signal (signal-software interrupt) or by using select () call or by using the named_pipe that you will design).

5) How will a process know how many bytes are a message from another process? That is, how will he interpret the bytes sent to him by the other process? (You will need to design a "communication protocol" between the processes).

Whatever design choices you make, you should describe them in a README file that you will submit with your code.

B) The script create_infiles.sh (10%)

You will write a bash script that creates test subdirectories and input files that you will use to debug your program. Of course, during the development of your program you can use small and small files to debug. The script create_infiles.sh works as follows:

```
./create_infiles.sh inputFile input_dir numFilesPerDirectory
```

- inputFile: a file that has the same format as the citizenRecordsFile input file of the first job. Recall the sample example we gave:

```
889 John Papadopoulos Greece 52 COVID-19 YES 27-12-2020
889 John Papadopoulos Greece 52 ÿ1ÿ1 ÿÿ
776 Maria Tortellini Italy 36 SARS-1 NO
125 Jon Dupont USA 76 H1N1 YES 30-10-2020
```

- input_dir: the name of a directory where the subdirectories and input files will be placed - numFilesPerDirectory: the number of files to be created in each subdirectory

The script does the following: 1. Checks for input numbers 2. Creates a directory with a name given in the second input_dir argument. If the list there is an error message printed and terminated. 3. Reads the inputFile file 4. Inside the input_dir directory creates subdirectories, one for each country name that locates in the inputFile
In each subdirectory, it creates numFilesPerDirectory files named country-n.txt where n is a positive integer. As the inputFile is processed it will locate all the lines corresponding to a country and distribute the round robin lines to the files named country-n.txt

Remarks - This task requires a lot of thought and good planning regarding allocated resources, forks, blocking / non-blocking I / O etc. The exercise does not describe all the details and structures for the simple reason that the design choices are exclusively yours (make sure of course that you describe them in detail in README). If you have different options for a point in the exercise, think about the pros and cons, document them in README, choose what you think is right and logical, and describe why you chose it in README.

Deliverable

- A brief and comprehensive explanation of the choices you have made in designing your program. 1-2 pages of ASCII text are enough. Include the explanation and instructions for compiling and running your program in a README file along with the code you will submit.

- The code you will submit should be yours. It is forbidden to use code that has not been written by you **(this includes code from the Internet!) .** - All your work (source code, Makefile and README) in a tar.gz file named OnomaEponymoProject2.tar.gz. Be careful to submit only code, Makefile, README and not binaries.

- It would be good to have a .tar backup of your exercise just as it was submitted to an easily accessible machine (department server, private github repository, private cloud). - The correct submission of a correct tar.gz that contains the code of your exercise and whatever files are needed is your sole responsibility. **Licenses tar / tar.gz or tar / tar.gz that are wrong and are not extracted are not graded.**

Procedural -

For additional announcements, follow the course forum at piazza.com. The full address is https://piazza.com/uoa.gr/spring2021/k24/home. Attendance at the Piazza Forum is a must. - Your program should be written in C (or C ++). If you use C ++ you can not use the ready-made structures of the Standard Template Library (STL). In any case, your program should run on the Linux workstations of the Department.

- Your program should compile the executable (travelMonitor) and (Monitor) have the same names for the parameters (-m, -b, -s, -i) exactly as described in the pronunciation. - Your password should consist of at least two (and preferably more) different files. Using separate compilation is imperative and your password should have a Makefile.

- Make sure you follow good software engineering practices when implementing the exercise. The organization, readability and existence of comments in the code are part of your rating.

- The submission will be done via eclass.

Other important remarks - The tasks are individual. - Anyone who submits / presents code that has not been written by him / her is reset to lesson.

- Although you are expected to discuss with friends and colleagues how you will try to solve the problem, copying code (of any kind) is not allowed. Anyone involved in copying code simply gets zero in the lesson. This applies to those involved regardless of who gave / received, etc. We emphasize that you should take the appropriate steps to ensure that your password is protected and not stored somewhere accessible to another user (eg, the excuse " I had put it in a github repo and he probably took it from there ", it is not accepted.)

- Programming exercises can be given with a delay of up to 3 days and with a penalty of 5% for every day of delay. Except for these 3 days, no exercises can be submitted.