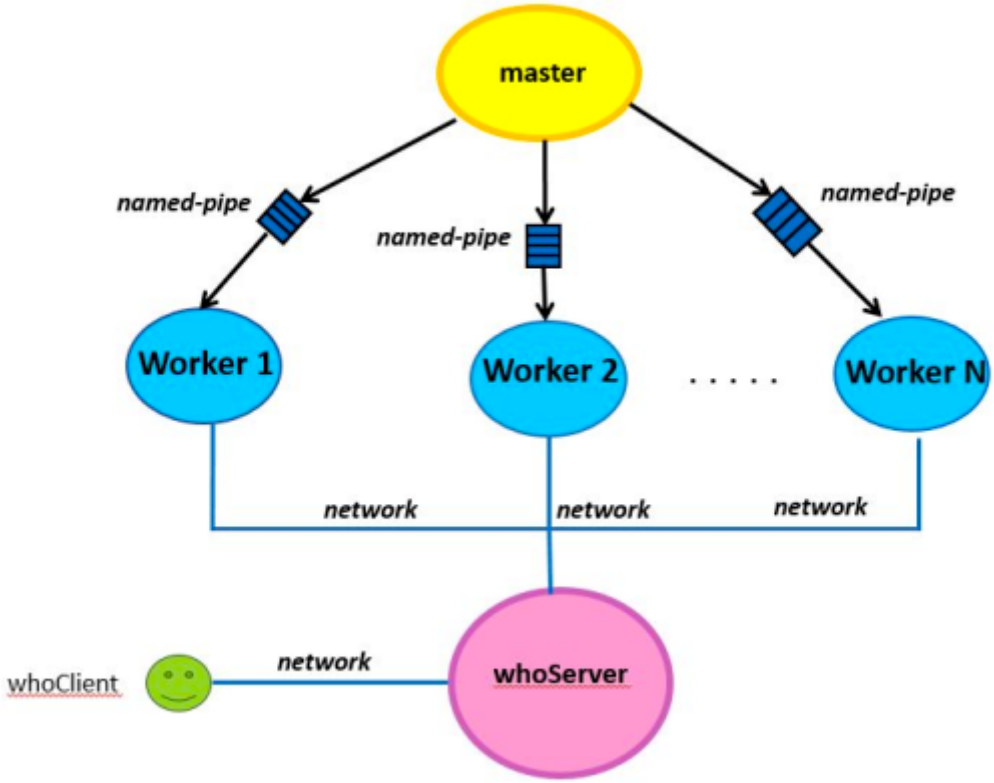
Introduction

The goal of this exercise is to get accustomed with multi-threaded programming and network communication. In this exercise you will implement a distributed system which will offer the same functionality of the process diseaseAggregator of the second exercise. Specifically you will implement three processes. 1) One master process which will create many worker processes (like the parent process of the second exercise), 2) One multi-threaded whoServer which will collect from the network summary statistics from the worker processes and will answer queries from clients, and 3) one multi-threaded client process whoClient which will create many threads, where each thread plays the role of a client which will sent queries to whoServer.



A) The process master (10%)

./master –w numWorkers -b bufferSize –s serverIP –p serverPort -i input_dir

where

- The parameter numWorkers denote the number of Worker process which the process master will create.
- The parameter bufferSize denote the size of the buffer for reading through named pipes.
- The parameter serverIP is the IP address of whoServer to which the worker processes will be connected to send the summary statistics.
- The parameter serverPort is the port number which whoServer listens.
- The parameter input_dir is a directory that contain subdirectories with the files that workers will process. Like the second exercise, each directory will contain the name of the country and will contain filenames of the kind DD-MM-YYYY. Each file DD-MM-YYYY will have exactly the same format as the second exercise and will contain a series of records of patients where each line will describe one patient who entered/left from the hospital the specific day, the recordID of the patient, his/her name, the type of disease and his/her age.

Upon the start of the process master it needs to start numWorkers Workers child processes and distribute uniformonly the subdirectories with the countries that are contained in input_dir to the Workers, like the second exercise. It will initiate the Workers and and will need to inform each Worker through a named pipe the directories that it needs to process. The master process will send also through a named pipe and the IP address and the port number of the whoServer. When it finish creating the Worker processes, the parent process will remain active in order to fork a new Work process if for some reason a Worker terminates.

Each worker process, for each directory will be assigned, will read all the files in chronological order according to the filenames and will fill a series of data structures which will use to answer to queries that whoServer will forward from the client process. It will connect to whoServer and will send the following information:
1) One port number that the worker process will listen for queries that whoServer will forward, and 2) the summary statistics (same as exercise 2). When the Worker process terminates the transfer of information to whoServer, it will listen to the port number that it selected and will want for connections from whoServer for queries that involve the countries it process. Signal handling remain the same as the second exercise.

Note: You need to think a way to assign a unique port number to each Worker process. The most simple way is to use port zero as input and let the system assign a unique port number by itself.

B) The process whoServer(60%)

The process whoServer will be used as follows:
./whoServer –q queryPortNum -s statisticsPortNum –w numThreads –b bufferSize
where:

- The parameter queryPortNum is the port number that whoServer will listen for connections from the client asking queries.
- The parameter statisticsPortNum is a port number that whoServer will listen from the worker process sending summary statistics.
- The parameter numThreads is the number of threads that whoServer will use to serve incomming connections from the network. The threads should be created only one time in the beginning when whoServer starts.
- The parameter bufferSize denote the size of the **cyclic** buffer and can be accessed be every thread that handle connections. The bufferSize represents the number of file/sockets descriptors that can be stored in him (e.g. 10, means 10 descriptors).

In your implementation, when whoServer starts the main thread will create numThreads threads. The main thread (main process) will listen to queryPortNum and statisticsPortNum, it will accept connections with the accept() system call and will place the file/sockets descriptors for each connection in a cyclic buffer with size as bufferSize. The main thread will not read/write from the connections it receives. Simple, when it receives a connection it will put the descriptor that accept() returns in the cyclic buffer and will continue receiving new connections. The responsibility of the numThreads threads is to serve the connections that the sockets represent and have been placed in the cyclic buffer. Each of the numThreads awakes when there is at least one descriptor in the cyclic buffer.

More specifically, the main thread listens to statisticsPortNum for connections from the Worker processes to receive the summary statistics and the port number that each Worker listens and will listen from queryPortNum for connections from whoClient to receive queries for diseases that have been recorded in the distributed system.

WhoServer can receive and serve the following queries that will originate from whoClient.

- /diseaseFrequency virusName date1 date2 [country]
  If the parameter country is not given whoServer will find for each disease virusName the number of diseases that have been recorded in the system in the range [date1...date2]/ If country is given, whoServer will find for the disease virusName the number of diseases in the specific country in the range [date1...date2]. Date1 and date2 will have the form DD-MM-YYYY. In the protocol of communication between the client and whoServer the system should recognize that country is an optional argument.
- /topk-AgeRanges k country disease date1 date2
  whoServer will find, for country and disease the top k age ranges that have acquired the disease in the specific country and the percentage of the diseases. Date1 and date2 will also have the form DD-MM-YYYY.
- /searchPatientRecord recordID
  whoServer will forward to all Workers the query and will wait to receive from a Worker the patient with the specific recordID.
- /numPatientAdmissions disease date1 date2 [country]
  If country is given whoServer needs to forward the request to the workers so that to find the total number of patients that entered the hospital with the specific disease in the specific country in the range [date1, date2]. If not given, it will find the number of patients with the specific disease that entered the in [date1, date2]. Date1 and date2 also follow the DD-MM-YYYY format.
- /numPatientDischarges disease date1 date2 [country]
  If country is given, whoServer will find the total number of patients admitted with the specific disease that have left the hospital in the time range [date1, date2]. If the country is not given, whoServer will find the total number of patient that had the specific disease and left the hospital in the range [date1, date2]. Date1 and date2 also follow the DD-MM-YYYY format.

When whoServer accepts a query, it forwards it to the worker processes through a socket and it will wait from an answer from the workers. The query that has been forwarded to a worker process with the answers that it receives it prints it in stdout. Also whoServer forwards the answer to the whoClient thread that made the query.

Note: According to your implementation you need to mind when a lot of threads write on stdout.

C) The process whoClient(30%)

The process whoClient will be used as follows:

./whoClient –q queryFile -w numThreads –sp servPort –sip servIP

- The parameter queryFile is the file that contains all the queries that need to be send to whoServer
- The parameter numThreads denote the number of threads that whoClient will create to send the queries to whoServer.
- The parameter servPort is the port number that whoServer listens so as whoClient can connect to it.
- The parameter servIP is the IP address of whoServer that whoClient will be connected.

The functionality of multithreaded whoClient is as follows. It will start and it will read the queryFile line by line. In each line there should be one command of the kind that whoServer can process. For each command a thread will be created which will handle to send the command to whoServer. The thread will be created but will NOT connect immediately to whoServer. When all threads receive the command that need to be processed then all the threads need to wake up and start alltogether an attempt to connect to the server. When it receives the answer it prints it to stdout.

Note:

You need to mind for shared resources with mutexes. It should be stressed that busy-waiting in threads is NOT allowed. You can use C++ but STL is not permitted to be used.