README

Προγραμματισμός Συστήματος
Άσκηση 4

Αλέξανδρος Συμπέθερος
AM: 1115200900261
Γαβριήλ Τζωρτζάκης
AM: 1115200900259

Run on personal computer: Ubuntu Linux (10.10,12.04) and on a few Linux machines in the Unix Lab(linux02, linux03)
Code is 100% our own,
Helpful sources for coding problems: manpages, reference pages for standard library functions online(cplusplus.com for example), and various unknown blogs for valgrind error and ideas and of course the course lessons.
A makefile exists, compiles everything into .o first(allowing following compiles to skip unchanged files), and has separate compilation for main.c A clean function is also available.

Program has been tested with multiple wget requests but also with Firefox with 10,20,30+ tabs refreshed in parallel(with add-on), allowing requests to complete but also stopping(control c) during transmission or even closing tabs in order to see server response.
Also valgrind has been used in most circumstances, with errors and leaks not existent.

About the code:
-Inline Arguments:
The program follows the exercise criteria.
User must give all arguments requested, otherwise error will appear.
If any arguments are written incorrectly error will appear.

Arguments:
-f input_file
-p port

Structure of program:
-Main Server:
Basic functionality is to create necessary structs, semaphores, conditional variables, sockets, pipes and everything else the threads will need to accomplish incoming requests. After initializing everything, socket waits for either termination signal or any requests to the port/ip it hosts. In the case of a request, a separate thread is created to deal with the client. In the case of a signal, conditional variable wait function is activated, and main sleeps until all threads have finished serving their given requests(if no threads currently open, main closes at once).

-Threads:
The first action of a thread is to detach itself. Afterwards, thread attempts to read http request of given client, and based on its type, necessary actions are taken. After completing actions and sending result to client, thread closes socket, decreases total and awakes main processes if necessary and exists.

-HayStackFile:
HayStackFile is composed of a SuperBlock containing 2 ints(a magic number and total number of needles in file) and from zero to as many as possible(Disk wise) Needles. Each needle contains an

id, status,offset and data. In order to quickly work with Haystack file, an index is created to quickly access the offset of a given needle. Index is a simple table(with each id a cell of the table) and a list containing any id's recently deleted from table(In this way, table contains minimum number of unused spaces). If table fills up, reallocation is called, creating double the current size. The table was chosen for its fast access speed for normal size data.

Compaction:
When main process is run, if given file already exists, main checks if file if of type haystack. If not, a Haystack file with given name is created and a Superblock is added to the start(also matrix,queue is initialized), otherwise the given file is searched for non delete needles and a new haystack file is created with them, also a index is built upon the new haystack(matrix and queue respectively).

Notes about code:
Semaphores are used to ensure atomic access to
-file(ensure read/write without error)
-queue(retrieve an id or add another)
-Matrix(retrieve offset for give id, see if id exists, or remove an id)
-total number of threads (add and subtract, if number reaches 0 and exit flag is set, threads must wake main process)
-id (to ensure unique id is given every time)
-conditional variable(C.V. Always needs a mutex)

When upload is requested, all data sent is written to a temporary file which only after entire request has been received will write needle and picture data to the haystack file.

When delete is requested(if id exists in index), the needle in the haystack file is changed from status 1 to status 0, and the cell in the matrix is set to zero. Also, the id of removed needle is added to queue to allow the same id to be used again. The temp file is removed before thread exists. The reason this happens is to maintain the integrity of the haystack file in the case of a corrupted upload.

When download is requested(if it exists), data is written at once to the socket. No extra file is necessary because if anything goes wrong, no damage is done to server, and client can simply resend an request.

If a request is sent wrong, incomplete or not recognized. Likewise, if a request for download/delete ask for an id not available, a 404 is sent. Also, if for any reason something goes wrong in a thread(for even main process for certain steps), a server error message is sent.

About termination:
Main process creates a signal set and blocks the termination signal until just before poll is called(blocks in the start, unblocks before poll and blocks right after). Poll works on 2 filedescriptors, one is the socket waiting for connections, and the other is a temporary pipe that is created only for this purpose. This temp pipe is only written from the signal handler when a termination signal is sent. When poll unblocks, first we check if any connections have happened and if so create a thread to deal with the client. After checking connections, main checks if the temp pipe has any data, and if so main waits until all threads have completed and then exists.