Remote File Memory Mapping

This is the description of the assignment of unit CPS2003, Systems Programming, for the year 2013/2014. This assignment is worth 60% of the total mark. The deadline of this project is 30th May 2014. You may do this assignment in a group of maximum 2 people. Groups consisting of one individual person is also allowed. A single mark will be allocated to the whole group, so choose your partner carefully and make sure that work gets distributed evenly. Code should not be shared outside your group, although you are free to discuss ideas amongst each other. Each group might be asked to present their implementation during which your program will be executed and the design explained. Also, please read carefully the plagiarism guidelines on the ICT website.

1 Your Task

The aim of this assignment is to implement an API which provides a number of calls allowing processes to memory map sections of a file located on a remote machine. Processes can then read and write to this memory area using the provided calls, with changes being propagated to the remote file by the library. Multiple processes from multiple client machines should be able to access and modify the same file on a remote machine.

A server process is required on the remote server which accepts incoming connections from client machines and executes all client requests, such as reading from and writing to sections of a file. This process must also handle conflicting requests from multiple machines, although conflict resolution from multiple processes on the same client machine can be handled internally by the library.

Server and client machines must of course communicate via a network connection. It's up to you to design and implement an appropriate communication protocol. This protocol must be capable of handling all client/server request and reply functionality as implemented by your library, including data transfer, async notifications, re-connection mechanisms in case of lost connections and any required optimizations (such as aggregating requests from multiple processes through a single connection).

You are expected to implement the API described in `fmmap.h` and allow it to be compiled as a static library. All network parameters are expected to be in network byte ordering. You will write a test program that makes use of the API. Your submission will be tested through sample client programs which will be linked to your library. `fmmap.h` will be used to include the functionality of your library, so make sure that it is not changed for your implementation. You cannot use in-built or third party libraries to implement the mapping or communication protocol.

You can assume that the remote server on which the file is hosted is accessible over TCP/IP. Your server process should be executable through the following command-line format: `fmmap_server <portno>`. You can assume that the server can be killed at any

point and that clients might be started prior to the start of servers, so make sure that you implement proper error handling routines.

2 Things to Consider

Marks will be allocated equally to a functional systems that allows multiple clients to access and modify files as well as to good and efficient design. Several concurrent clients will be run on several files on different machines. Having a system which is free of race conditions, deadlocks and still considerably efficient it ideal. Make sure that you spend enough time designing you system well before starting to code. Try to think of several scenarios which your library might encounter and design appropriate handling mechanism. The following is a non-exhaustive list of things you might want to consider:

- How are the memory mappings represented internally, and what data structures do I require?
- How do memory mappings grow internally? How are large files handled?
- Are data structure instances unique to each calling process, or shared amongst multiple ones? Which one would be faster? What overheads are required to share data structures?
- What unit size should I use for mappings (bytes, pages, dynamic segments)? How will this decision affect conflict resolution and data sharing between local processes?
- What happens when two local processes attempt to modify the same file?
- Are conflicts (two writes, overlapped reads/writes) handled locally or remotely by the server process?
- How are operations from multiple processes sent to the server?
- Is local file caching required (for read-intensive processes using the same file segments)?
- Is the library thread-safe?
- Is asynchronous functionality required?
- Are requests queued and serialized, or is some form of parallelism implemented in the server? Does the latter option provide any benefits?
- What type of conflict resolution is required on the server? How are simultaneous reads and writes on overlapping memory segments handled?
- If each client caches some data, how are file changes propagated to each client?
- What happens when the connection between the remote server and local machine is lost?
- Is the library efficient enough?

- Are proper byte-ordering mechanism implemented on both the client and server?
- How are large reads and writes (greater than the maximum IP packet size) handled?

You have to make sure that your system works on Linux-based systems running on i386 and x86-64 architectures. Both the client and server must be fault-tolerant, and can both be terminated with a CTRL-C or using the kill command. Ensure that proper clean-up is performed in each case. Thus each server must terminate all its child processes (if any) and close all open sockets. The child must close all its connections properly and make sure that any system-wide data structures are properly updated.

3 Deliverables

You must upload your source code (C files together with any accompanying headers), including any test clients and additional utilities, through VLE by the specified deadline. Use .zip or .tar.gz for archives. Include makefiles with you submission which can compile your system. These makefiles must generate the static library and as well as the server. Make sure that your code is properly commented and easily readable. Be careful with naming conventions and visual formatting. Marks may be deducted if I cannot understand what's going on in your code. Every system call output should be validated for errors and appropriate error messages should be reported.

You must also hand-in a report describing:

- The design of your system, including any structure, behavior and interaction diagrams which might help.
- Description of any implemented mechanisms which you think deserve special mention.
- Details of the network protocols between the interacting processes, together with a diagrams visually describing these protocols.
- Your approach to testing the system, listing test cases and results.
- Library and server timings for above test cases, if you so desire.
- A list of bugs and issues of your implementation.

You do not need to describe in great detail your work, just make sure that anyone reading your report can understand the overall design concepts and how these were implemented. Please do not include any code listing in your report. Also, please use double-sided printing. You may submit your report, including the plagiarism form, to myself or the departmental secretary by the specified deadline.