

Submission Requirements

If any of these requirements create a problem for you or are unclear, come talk to me.

- Your exercise submission must consist of the writeup of your answers to the questions. You should not include any additional material.
- You must do this assignment by yourself. You should not discuss the questions, code, or solutions with other students. You may discuss them with the TA or me.
- The assignment is due on in two parts, Part 1 on Tuesday Feb 10th, and Part 2 on Thursday Feb. 12th at the beginning of class. You should turn it in in printed form. Also email me an electronic copy.
- If you discover any ambiguities in this assignment, send email to the class email list (`cs184-2004@hermes.gwu.edu`) and I will clarify it.

Goals

The goal of this homework is to gain some experience with highly concurrent network software design and different models of concurrency.

Assignment

Part 1: Due Tuesday Feb. 10

Read the paper *A Design Framework for Highly Concurrent Systems* by Welsh, Gribble, Brewer, and Culler. They propose a particular way to design networked software that will allow it to scale to high loads, handle many connections, and behave in a controlled way under widely varying conditions.

Answer the following questions based on the paper and lecture presentations

1. What are two advantages of event-based designs over threaded designs?
2. What are two advantages of threaded designs over event-based designs?
3. Briefly describe the io multiplexing primitives (`select`, `poll`, and `/dev/poll`) and how much data each one has to copy into and out of the kernel in order for 1000 sockets to be created, monitored, and closed. Where the set is checked 20 times and each time they are checked 50 sockets (out of the 1000) have data available.
4. The Tasks, Queues, and Threads framework proposed in the paper, along with several design patterns can be applied to many different types of network servers. Write a fairly detailed description of how you would design a dynamic web application server using this framework. The web server must have the following characteristics: handles many client connections, has plugin modules that can handle certain types of requests (for example CGIs, ASPs, PHP, file

uploads, ...), a web forum/bulletin board system, and has a database backend running on a separate machine that stores the web accounts for the users and the state of the posts to the web forum. The database itself can be considered to be a standard component, and is not part of the design.

This design should include at least one picture showing how the components of the system relate to each other, and a mapping of some of the design patterns to the system design.

Part 2: Due Thursday Feb. 12

Read the man pages I handed out in class about the `sys_epoll` service in Linux.

Use the `sys_epoll` interface and system calls to write an efficient, light-weight, web server. You may use either the Level Triggered or Edge Triggered model of `epoll`. The level-triggered is often easier to get right, the edge-triggered can be slightly more efficient. At a minimum your webserver should serve static files, some support for CGI or other types of dynamic content is encouraged.

NOTE: The server should be written in a c-like pseudo-code.

It does not need to actually compile or run. However, I want the code to be detailed enough that I can see how you use the interfaces and that the design of the code is clear. For example, you do not need to worry about accurate memory management, strict typing, routine parsing code, etc. What you should include is any IO system calls (although all the parameters are not required) such as file reading or writing, creating, accepting, sending and receiving on network sockets, and the `epoll` calls (obviously). Uninteresting pieces of code, or obvious application handling that does not involve IO can simply be specified with ellipsis and comments as in:

```
/* parse GET request and get file name to serve */
...

fd = open(filename);

while( !(eof(fd) ) {
    len = read(fd, file_buffer, 4096);
    send(client_s, file_buffer, len);
}
```

As you notice in this example, I skip any function parameters that are not critical to understanding what you are doing. In this case I included the buffer length because how you handle buffers can affect the design of the software.

I will be grading the pseudo-code on clean design, clarity of function names and code snippets, accurate use of the `epoll` calls, general correctness of socket calls, code documentation.