Computer Science 451: Operating Systems
How to write a Design Document

CSE 451 Course Staff
Adapted from an earlier version authored by Harvard OS/161 course staff

Introduction

Assignments 2 and 3 require that you write and submit a design document. The design documents are, in the opinions of the course staff, the single most important piece of work you will do in this class. It will be a large factor determining whether your code works, or whether you become endlessly frustrated chasing problems that you could have anticipated and therefore avoided.

You may have taken other CS classes that required you to submit a "design document." Most likely, you did this "post-hoc": you hacked away at the assignment until it worked, then went back at the end to write the design. This method, however, will not work in CSE 451 (or in real life). The motto here is "measure twice, cut once." We give you a week to complete design documents for a reason; we want you to spend a significant amount of time thinking your way through the assignment (what the requirements are and what code you might need), before you start hacking. We give you two weeks after the design document due date to code, so there should be no need to rush this process.

You are likely to write many design documents in your professional careers, particularly as you rise into technical leadership. They help organizations verify that your code will solve the problem it is intended to solve, in a clean and elegant fashion. Best to start practicing now.

Peer Review:
In industry, design documents are almost always peer reviewed, pre-hoc, for every significant feature. If you find yourself working for a group that does not do this, you might want to start polishing your resume, since they are likely to be out of business soon.

In section, your group will pair up with another group and you will spend half a quiz section reviewing their document and the other half of section with them reviewing your document. Peer review participation has two benefits. In addition to getting feedback on your design, you will learn a great deal from doing a careful review of another design. Ideally, you will be exposed to alternative thought processes and conclusions. Ask questions! Poke holes. Give constructive feedback. You are welcome to adopt ideas and mechanisms that you learn about through the peer review. To get the most out of the peer review, you will need to come to section with a mostly complete design document. If you wait until the peer review to start your design document, you will already be behind.

Grading:
The design document is worth 30% of your final grade for the assignment. If your group did not receive full points on the design document you may fix the errors, turn it in with the final project, and the TAs will award up to half of the lost points.

Further, the design document is an insurance policy. If your code ends up not working (and realistically, that happens even to the best students), but your design shows a clear understanding of how to solve all of the required elements of the assignment, then we are more likely to think that the issue is just a bug and not a fatal design flaw.

How to write a Design Document:
The audience for the design document is your TA, or a peer. It should convince them you have a solid understanding of how to solve all aspects of the problem. Thus, a good design document is as valuable, or perhaps more valuable, than the resulting code. In other words, a design doc should reflect all the research and brainstorming you did before starting to code. Do not shy away from details!

We next present guidelines for writing a good design document. You do not have to follow these guidelines exactly, but they give a sense for what information we are looking for.

Suggested Design Document Layout:

The approach we take is iterative deepening: start with the high level goals, for example to add system calls to OS/161 while protecting the kernel from user bugs. Then explain how you will achieve those goals.

We recommend you to format your documents into three sections.
1. Overview
2. In-depth Analysis and Implementation
3. Risk Analysis
4. Assignment Questions

1. Overview
Keep the audience in mind. Start with a few (2-3) sentences on the goals of the assignment. Then decompose the assignment into the major parts that you need to complete. For example, in assignment 2, a good topic breakdown might be:

- Identifying processes
- File descriptors
- fork
- execv
- waitpid/exit
- Other system calls
- Synchronization issues

For each section, again, start with a few sentences describing goals. For example, the goal of the system call implementation might be to prevent bugs from user code from corrupting the kernel, and to allow the core system call implementation to be called from within the kernel without protection checks. List the key challenges to accomplishing these goals, e.g., the need to translate addresses or to release locks correctly on error conditions.

In addition, you should describe (where applicable):

- How the different parts of the design interact together.
- Major data structures.
- Synchronization.
- Major algorithms.

2. In depth Analysis and Implementation

Here is where you need to be specific. Break up the major parts of your design into (i) the functions you have to implement, (ii) existing functions you need to modify, (iii) corner cases you will need to handle (e.g., list all possible error conditions returned from each system call), and (iv) test plan. Normally you should include pseudo-code for key algorithms, but for this class its ok to have zero code in the design document. However, it is essential for you to figure out where the subtleties and hidden problems await, and write those down. Every bug you find during design will save you hours of debugging.

Sometimes it might be useful to write down which files you will need to modify for each part. This will force you to think about the actual implementation and how it fits in the system. You are not designing in isolation; you are designing in the context of os/161, and you need to make sure that your designs make sense in that environment. Spend time reading through the os/161 code to see where these pieces should be placed.

The goal of this section is to convince your TA/peer that the design will accomplish the goals and challenges laid out in the overview. Using assignment 2 as an example once again, in the section on fork you will undoubtedly want to explain how you will manage file descriptors across the fork system call.

3. Risk Analysis

Much of real-life engineering is managing risk. Some risk is part of life; the key is reducing avoidable risk. When you are done with the design document, you should look at it and convince yourself that you have explained every difficult detail. Often students write something that translates into "I don't really know how I'm going to solve it, but somehow I will" or "We will implement read. We will also implement write." If you find yourself writing things like this, then the risk is you will not understand what is required until it is too late.

So in this section, write down everything you have not figured out yet. Initially, this might be everything! Then as you refine the document, you can move items out of the risk category. Anything left you should talk to the TAs or your peers about. "I DON'T KNOW HOW TO DO THIS, HELP!!" is a completely ok thing to have here.

We would also like you to estimate how long the project will take you, in person-hours, as three numbers: the time it will take you in the best case, if everything goes well; the time it will take you in the worst case, if things turn out badly; and the time you expect it to take on average. These should be for the assignment as a whole. We don't expect these to be accurate, but the exercise is good practice. The natural tendency is to be optimistic. In industry, realistic estimates are essential; optimistic estimates lead to failed software projects and failed companies.

The final part of risk analysis is a few sentences on how you plan to stage the work, so that if your estimates turn out to be closer to the worst case, you will be able to jettison some lower priority features. For example, if exec does not work in assignment 2, it will be difficult to do assignment 3, but if argument passing does not work for exec, you may be able to work around that. Similarly, you should get system calls to work first assuming that the user program behaves as expected. Only then add error checking.

4. Assignment Questions
Lastly, include any embedded questions from the assignment. Note that we grade these separately from the rest of the assignment.

5. Final Turn-In
Please turn in a revised design document along with your code. We expect that there will be some changes to the design itself, so we plan to re-read the design from scratch; it should reflect reality rather than your plan. If the changes are minor, please write a few sentences here to outline those changes; otherwise, just point us at the main document. We do not plan to re-read the assignment questions unless you tell us explicitly that there were changes to the answers you gave earlier.

At final turn-in, tell us how many hours it actually took you to do the assignment. If this is outside the range you provided in the design, give us a few sentences on why.