COMPUTER SCIENCES DEPARTMENT
UNIVERSITY OF WISCONSIN-MADISON

CS 736 **Bart Miller**
Fall 1988

Project List

(Brief Description Due: Wednesday, October 26)
(Midway Interview: Friday, November 18)
(Final Report Due: Thursday, December 15)

General Comments

The projects are intended to give you an opportunity to study a particular area related to operating systems. Your project may require a test implementation, measurement study, simulation, literature search, paper design, or some combination of these.

The project suggestions below are briefly stated. They are intended to guide you into particular areas and you are expected to expand these suggestions into a full project descriptions. This gives you more freedom in selecting an area and more burden in defining your own project. There may be more issues listed for a project than you can cover. If you have a topic of your own that is not listed below, you should come and talk with me so we can work out a reasonable project description.

You will write a paper that reports on your project. This paper will structured as if you were going to submit it to a conference. I will provide more details on the project report later in the semester.

You can work in teams of two people on the project and report.

Projects

(1) *Operating System Utility Program Reliability – The Fuzz Generator:* The goal of this project is to evaluate the robustness of various UNIX utility programs, given an unpredictable input stream. This project has two parts. First, you will build a *fuzz* generator. This is a program that will output a random character stream. Second, you will take the fuzz generator and use it to attack as many UNIX utilities as possible, with the goal of trying to break them. For the utilities that break, you will try to determine what type of input cause the break.

The fuzz generator will generate an output stream of random characters. It will need several options to give you flexibility to test different programs. Below is the start for a list of options for features that *fuzz* will support. It is important when writing this program to use good C and UNIX style, and good structure, as we hope to distribute this program to others.

-p only the printable ASCII characters

-a all ASCII characters

-0 include the null (0 byte) character

-l generate random length lines (\n terminated strings)

-f name record characters in file ‘‘name’’

-d nnn delay nnn seconds following each character

-r name replay characters in file ‘‘name’’ to output

The *fuzz* program should be used to test various UNIX utilities. These utilities include programs like vi, mail, cc, make, sed, awk, sort, etc. The goal is to first see if the program will break and second to understand what type of input is responsible for the break.

(2) *Security in a Workstation CPU Server:* The Condor system allows users to automatically run their programs on any of a collection of idle workstations. It automatically chooses an idle workstation, checkpoints the program, and can move the program to another workstation when the owner returns. This system was developed in our Computer Sciences Department.

While Condor is an effective system for load distribution and providing free cycles to the user community, it has only briefly addressed security issues. If you own a workstation, you would like to be confident that the guest programs that are running cannot access or damage your resources (processes, files, devices, etc.). The goal of this project is to study the existing Condor features, evaluate the UNIX security facilities, and then design and implement security modifications to Condor.

(3) *Visual Shell:* Personal computers, like the Mac, are easy to use because of their simple and visual user interfaces. The goal of this project is to build a visual shell for UNIX. This shell uses graphic display and mouse input to list files, delete files, start programs, build pipes, redirect input and output, etc. Keyboard input will also be need to be smoothly integrated in the design.

This shell should also allow for more advanced features such as non-linear pipes, editing of commands and program output (and feeding this back into programs), visual aliases, etc.

(4) *A Language for Distributed Games:* DREGS is a system for helping design and build distributed, multi-player games. DREGS runs on our local uVax, Bobcat, and IBM RT/PC workstations. This project involves the interaction of distributed systems and programming languages.

The DREGS design includes a language call GDL (game description language) that simplifies the programming of the games. Currently, games are hand-coded according to a GDL-like coding style. The goal of this project would be to build a simple GDL compiler. There are several games that currently run under DREGS, and part of this project would be to convert at least one of these to directly use GDL. This project is especially suited to a person with a strong interest in compilers.

An alternative version of this project would be to investigate the use of an object-oriented programming language (probably C++) to support a set of standard classes to support GDL. You would then go on to implement these classes and test them by converting one of the simpler games.

The current DREGS system is described in two papers (I can supply copies of these):

A. Bricker, M. Clark, T. Lebeck, B.P. Miller, and P. Wu, ‘‘Experience with DREGS’’, *Proc. of the 1987 Summer USENIX Conf.*, Phoenix, June 1987.

A. Bricker, T. Lebeck and B.P. Miller, ‘‘DREGS: A Distributed Runtime Environment for Game Support’’, *Proc. of the EUUG 1986 Conf.*, Manchester, England, September 1986.

(5) *A Resource Scheduler for a Distributed Environment:* The Computer Sciences Dept. has a resource scheduling problem: that of scheduling conference rooms. This problem has several interesting characteristics.

First, the service must be accessible from throughout the campus network, but not from outside the campus (or department). Second, we need some level of protection. If one person makes a reservation, then another person should not be able to delete it. Third, data consistency must be preserved. This means that if two people are making reservations, you must serialize requests and avoid race conditions. Fourth, you will need a decent user interface. The users should be able to gracefully examine the schedule and make reservations. A graphic interface (using X windows) would be best.

(6) *Distributed Simulation Algorithm:* Chandy has developed a simulation algorithm that is supposed to provide good parallel execution in a distributed environment. The goal of this project is to take the description of Chandy's algorithm, build an implementation of a simulation, and evaluate its performance (as compared to a sequential version).

(7) *Expert Systems in Operating Systems:* Expert systems and programming languages such as Prolog are receiving increasing attention. How might this affect the design of an operating system? Will an operating system that supports expert systems look any different from current systems?

From a different perspective, how might the techniques from expert systems be applied to the design and implementation of operating systems? What parts of an operating system could benefit from these techniques.