

# Scheduling in DLX/OS

Prof. Darrell Long<sup>†</sup>  
Computer Science Department  
Jack Baskin School of Engineering  
University of California, Santa Cruz

Design Due: October 15, 23:59  
Code Due: October 22, 23:59

## 1 Purpose

There are two main goals for this project:

1. To familiarize you with the DLX/OS system-how it works, how to use it, and how to compile code for it.
2. To modify the DLX/OS scheduler to be more flexible. You must implement
  - A two-level round-robin scheduler.
  - A lottery scheduler.

You should also read over the general project information pages before you start this project. In them, you will find information about the DLX/OS system as well as general guidelines and hints for projects in this class.

## 2 Basics

The goal of this assignment is to get everyone up to speed on DLX/OS and to gain some familiarity with scheduling. In this assignment you are to implement a two-level round-robin scheduler and a lottery scheduler. A lottery scheduler assigns each process some number of tickets, then randomly draws a ticket among those allocated to ready processes to decide which process to run. That process is allowed to run for a set time quantum, after which it is interrupted by a timer interrupt and the process is repeated.

The number of tickets assigned to each process determines both the likelihood that it will run at each scheduling decision as well as the relative amount of time that it will get to execute. Processes that are more likely to get chosen each time will get chosen more often, and thus will get more CPU time.

One goal of best-effort scheduling is to give I/O-bound processes both faster service and a larger percentage of the CPU when they are ready to run. Both of these things lead to better responsiveness, which is one subjective measure of computer performance for interactive applications. CPU-bound processes, on the other hand, can get by with slower service and a relatively lower percentage of the CPU when there are I/O-bound processes that want to run. Of course, CPU-bound processes need lots of CPU, but they can get most of it when there are no ready I/O-bound processes. One fairly easy way to accomplish this in a lottery scheduler is to give I/O-bound processes more tickets – when they are ready they will get service relatively fast, and they will get relatively more CPU than other CPU-bound processes.

The key question is how to determine which processes are I/O-bound and which are CPU-bound. One way to do this is to look at whether or not processes block before using up their time quantum. Processes that block before using up their time quantum are doing I/O and are therefore more I/O-bound than those that do not. On the other hand, processes that do not block before using up a time quantum are more CPU-bound than those that do. So, one way

---

<sup>†</sup>Parts of the assignment have been taken from those of Ethan Miller, Clay Shields and Scott Brandt.

to do this is to start with every process with some specified number of tickets. If a process blocks before using up its time quantum, give it another ticket (up to some set maximum, say 10). If it does not block before using up its time quantum, take a ticket away (down to some set minimum, say 1). In this way, processes that tend to do relatively little processing after each I/O completes will have relatively high numbers of tickets and processes that tend to run for a long time will have relatively low numbers of tickets. Those that are in the middle will have medium numbers of tickets.

This system has several important parameters: time quantum, minimum and maximum numbers of tickets, and the speed at which tickets are given and taken away.

## 2.1 Design Document

The assignments in this class do not require you to write large quantities of code. For most assignments, you need only write a few hundred lines of code, usually fewer. However, the concepts covered in class and in the operating system are quite difficult for most people. As a result, deciding which lines of code to write is very difficult. This means that a good design is crucial to getting your code to work, and well-written documentation is necessary to help you and your group to understand what your code is doing.

The most important thing to do is write your design first and do it early! Each assignment is about two weeks long; your design must be complete by the end of the first week. Doing the design first has many advantages:

- You understand the problem and the solution before writing code.
- You can discover issues with your design before wasting time debugging code that you'll never use.
- You can get help with the concepts without getting bogged down in complex code.

Doing your design early is the single most important factor for success in completing your programming projects!

## 3 Details

In this project, you will modify the scheduler for DLX/OS. This should only involve modifying code in `process.c` and `process.h`, though you may want to look at other modules to see how they fit together. To test your code, you can create additional processes in `sysproc.c`.

### 3.1 Dual Round Robin Queues

The first algorithm you need to implement uses two round robin queues. Processes are placed into the first queue when they are created, and move to the second queue after they have had five quanta (that is, after they have been scheduled five times). The scheduler runs all of the processes in the first queue once and then runs a single process from the second queue. This can be implemented in several ways; one possibility is to include a “pseudo-process” in the first queue that, when at the front of the queue, causes a process from the second queue to be run.

Assume the following processes are in the two queues:

- Queue 1:  $P_1, P_2, P_3$
- Queue 2:  $P_4, P_5, P_6, P_7$

The scheduler would run processes in this order:

$$P_1, P_2, P_3, P_4, P_1, P_2, P_3, P_5, P_1, P_2, P_3, P_6 \dots$$

This allows long-running processes to make (slow) progress, but gives high priority to short-running processes.

## 3.2 Lottery Scheduling

For lottery scheduling, you should assign tickets using `ProcessSetPriority()`, where  $n$  corresponds to the number of tickets given to the process. Each time the scheduler is called, it should randomly select a ticket (by number) and run the process holding that ticket. Clearly, the random number must be between 0 and  $n\text{Tickets} - 1$ , where  $n\text{Tickets}$  is the sum of all the outstanding tickets. You may use the `random()` call to generate random numbers and the `srandom()` call to initialize the random number generator (these calls function the same way they do in UNIX).

For dynamic priority assignment, you should modify lottery scheduling to decrease a process's priority by 1 each time it receives a full quantum, and increase its priority by 1 each time it blocks without exhausting its quantum. A process's priority should never drop below 1 and should never exceed its original (desired) priority.

You must implement lottery scheduling as follows:

1. Basic lottery scheduling. Start by implementing a lottery scheduler where every process starts with 5 tickets and the number of tickets each process has does not change.
2. Lottery scheduling with dynamic priorities Modify your scheduler to have dynamic priorities, as discussed above.

## 3.3 Profile the Scheduler

At each timer interrupt, see how many processes are at each priority level, average this information over some large number of interrupts (maybe 100), then print it out. Conclude something about the settings of the various parameters: are they too long, too short, just right. Justify your conclusions.

*Remember:* You must check and correctly handle all return values.

## 4 Deliverables

A compressed `tar` file of your project directory, including your design document. You must do "make clean" before creating the `tar` file. In addition, include a `README` file to explain anything unusual to the TA or grader (for example, testing procedures). Your code and other associated files must be in a single directory so they will build properly in the submit directory.

*Remember:* Do not submit object files, assembler files, or executables. Every file in the submit directory that could be generated automatically by the compiler or assembler will result in a 5 point deduction from your programming assignment grade.