CS 111 - Project 2

# Design Document

4.30.14

Greg Arnheiter
Ian Hamilton
Justin Kwok
Benjamin Lieu
Yugraj Singh

**Purpose -** This project overviews the design and implementation of a lottery scheduler used in MINIX operating system. Topics covered include systems programming, their kernels, viewing and modifying important OS code used in MINIX, and analysis of varying scheduling methods that can be used in an OS.

**Part 1 - Lottery Scheduler (Static)**

| Queues Used for Lottery Scheduler | |
|---|---|
| **12** | Winner |
| **13** | Past Winners |
| **14** | Losers (Lottery Group) |

This method holds a lottery out of processes available which start in queue 14, with equal tickets for each process. The winner of the lottery is moved to queue 12; If it runs out of quantum then it will be de-prioritized to queue 14 and another lottery will be held. If the winning process becomes blocked, then another process with lower priority will be executed by the kernel. When that new process finished its quantum, any blocking winners will be moved to a lower priority (12 to 13, or 13 to 14) and a lottery will be held again.

| Functions Modified to Implement Static Lottery Scheduler | |
|---|---|
| **`do_no_quantum`** | Apply lottery schedule algorithm |
| **`structproc.h`** | Add ticket field to processes struct. |
| **`do_start & do_stop_scheduling`** | Initialize priority, quantum, and tickets |
| **`do_nice`** | Add adjust_tickets() to modify processes tickets. |

**`do_no_quantum`** – Algorithm for Lottery Selection (Static):

If ( `do_no_quantum` message received from queue 12)
move (process in 12 to 14)
hold lottery in queue 14
If ( `do_no_quantum` message received from queue 13)
if( Queue 12 is NOT empty ) move (process 12 to 13)
move (process in 13 to 14)
hold lottery in queue 14
If ( `do_no_quantum` message received from queue 14)
if( Queue 12 is NOT empty ) move (process 12 to 13)
move (process in 13 to 14)
hold lottery in queue 14

The algorithm for static lottery selection was put into the `do_no_quantum` function. In order to hold a lottery, each processes' struct was modified in `structproc.h` to include a field for tickets. The tickets, as well as a processes' quantum and priority area initialized when `do_start_scheduling` is called, therefore the `do_start` and `do_stop_scheduling` were modified to reflect the desired default values where each process contains 20 tickets and starts in queue 14. Lastly, the `adjust_tickets()` function was added to `do_nice()` to handle computation and adding tickets to processes; this function was also added to dynamically change ticket values if the parameter is turned ON.

**Part 2 - Lottery Scheduler (Dynamic)**

After successful implementation of the static lottery scheduler, the only remaining changes were to add functionality to `adjust_tickets()` to remove two tickets from any process that has used its quantum, and add five tickets to any process that finishes before using its quantum. `Adjust_tickets` is called in `do_nice`, and will use the dynamic lottery feature while the parameter `DYNAMIC` equals 1. If `DYNAMIC` is changed to 0, the regular static scheduler will be used.

**Part 3 - Building and Running**

| Files to be replaced in MINIX 3.1.8 source | |
|---|---|
| **sched/schedule.c** | /usr/src/servers/sched |
| **sched/schedproc.h** | /usr/src/servers/sched |
| **pm/schedule.c** | /usr/src/servers/pm |

To compile:
```
$ cd /usr/src/tools
$ make install
```

**Part 4 - Testing**

| Tests - Static | Results - Static |
|---|---|
| **Two equal CPU bound tasks** | Both ran with approximately equal time. |
| **One process has double tickets** | Process with more tickets ran approximately twice as fast. |
| **Processes with 25%, 50%, 100% ratio tickets** | Processes finished in appropriate respective ratio to their given tickets. |
| **Process with 1 ticket doesn't starve** | Process with 100 tickets didn't starve proc with 1. |

| Tests - Dynamic | Results - Dynamic |
| --- | --- |
| **Scheduler adjusts tickets Dynamically** | CPU bound processes decreased tickets over time, as IO processes increased tickets over time. Observed by printing ticket values to output file and file redirection from the command `top.` |

One of the tests used was a CPU intensive test doing arbitrary computations in large `for` loops, while the I/O intensive test reads and write large files back and forth. CPU bound tests blocked less frequently, and were thus de-prioritized by removing tickets while the I/O tasks were blocked more often yielding more tickets and a higher priority.