Goal:

Don’t try to optimize turnaround or response time. Rather, try to guarantee each job a proportional share of the CPU.

**Might randomness help scheduling?**

- Hold a lottery to determine who gets to run
- More important jobs get more tickets (chances) to win
- A ticket represents a process’s share of the CPU

**Side Note: How well do you understand randomness?**

- Randomness tip on Page 88 is groovy
- Most are pseudo-random
- A pseudo-random number generator is an equation
- Generally, \textit{uniformly at random}
- It must pass a set of statistical tests of randomness
- The built-in one in Java does NOT pass these tests – don’t ever use it
- Mersenne Twister is a good one, they have one for many languages (is built in to some, like Python)
- Random.org provides truly random using atmospheric noise

**Basic Approach:**

Draw a ticket randomly after each time slice

**Example:**

Let’s say we have 2 jobs, A and B. We want A to run 75% of the time and B to run 25% of the time. What’s an easy way to accomplish this goal?

- A gets 75 tickets
- B gets 25 tickets

Which of the following are true?

1. They will always cycle: A will run for 3 then B for 1, then start over.
2. It will work out perfectly to get 75% and 25% every time.
3. In the limit it works out to 75% / 25%.

**Assigning Tickets**

**Mechanism 1: Ticket Currency**

Users can be assigned a set of tickets for all of their processes
A Deterministic Solution (new algorithm)

With Stride scheduling each process has a “stride” and a counter called “pass”. Stride is how fast it walks. Pass is its current progress.

Lowest pass runs

Example:

A has stride 100, B stride 200, C stride 40

<table>
<thead>
<tr>
<th>Pass(A)</th>
<th>Pass(B)</th>
<th>Pass(C)</th>
<th>Who Runs?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>100</td>
<td>201</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>100</td>
<td>201</td>
<td>42</td>
<td>C</td>
</tr>
<tr>
<td>100</td>
<td>201</td>
<td>82</td>
<td>C</td>
</tr>
<tr>
<td>100</td>
<td>201</td>
<td>122</td>
<td>A</td>
</tr>
</tbody>
</table>

Do you want your process to have a small or a large stride?

Small!

Why don’t we use either of these algorithms?

- Don’t have a great way to assign tickets
- Stride challenge: what “pass” value is assigned a new process (e.g., 0 → monopolizes CPU) (with lottery we simply add the process and its tickets to the pool)
- Neither deals with I/O well