Schedulers

Tyler Harter
Overview

Review processes

Workloads, schedulers, and metrics (Chapter 7)

A general purpose scheduler, MLFQ (Chapter 8)

Lottery scheduling (Chapter 9)
Review: Processes
Process Creation

CPU

Memory

code
static data

Program
Process Creation
Process Creation

CPU

PC

Memory

code
static data
heap

stack

Process

code
static data
Program
State Transitions

- Running
- Ready
- Blocked

Transitions:
- Running → Ready
  - Scheduled
- Ready → Running
  - Descheduled
- Blocked
  - I/O: initiate
  - I/O: done
How to transition? ("mechanism")
When to transition? ("policy")

- **Running**
  - Transition to **Blocked** via I/O: initiate
  - Transition to **Ready** via Descheduled

- **Blocked**
  - Transition to **Running** via Scheduled
  - Transition to **Ready** via I/O: done

- **Ready**
  - Transition to **Blocked** via I/O: done
  - Transition to **Running** via Scheduled
/ Per-process state

struct proc {
    uint sz;    // Size of process memory (bytes)
    pde_t* pgdir; // Page table
    char *kstack; // Bottom of kern stack for this proc
    enum procstate state; // Process state
    volatile int pid; // Process ID
    struct proc *parent; // Parent process
    struct trapframe *tf; // Trap frame for current syscall
    struct context *context; // swtch() here to run process
    void *chan; // If non-zero, sleeping on chan
    int killed; // If non-zero, have been killed
    struct file *ofile[NOFILE]; // Open files
    struct inode *cwd; // Current directory
    char name[16]; // Process name (debugging)
};
// Per-process state
struct proc {
    uint sz;  // Size of process memory (bytes)
pde_t* pgdir;  // Page table
char *kstack;  // Bottom of kern stack for this proc
enum procstate state;  // Process state
volatile int pid;  // Process ID
struct proc *parent;  // Parent process
struct trapframe *tf;  // Trap frame for current syscall
struct context *context;  // swtch() here to run process
void *chan;  // If non-zero, sleeping on chan
int killed;  // If non-zero, have been killed
struct file *ofile[NOFILE];  // Open files
struct inode *cwd;  // Current directory
char name[16];  // Process name (debugging)
};
<table>
<thead>
<tr>
<th>Operating System</th>
<th>Hardware</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Process A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Operating System</td>
<td>Hardware</td>
<td>Program</td>
</tr>
<tr>
<td>------------------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td></td>
<td>timer interrupt&lt;br&gt;save regs(A) to k-stack(A)&lt;br&gt;move to kernel mode&lt;br&gt;jump to trap handler</td>
<td>Process A&lt;br&gt;...</td>
</tr>
</tbody>
</table>
## Handle the trap

Call **switch()** routine
- save regs(A) to proc-struct(A)
- restore regs(B) from proc-struct(B)
- switch to k-stack
- return-from-trap (into B)

## Hardware

- timer interrupt
- save regs(A) to k-stack(A)
- move to kernel mode
- jump to trap handler

## Program

Process A

...
<table>
<thead>
<tr>
<th>Operating System</th>
<th>Hardware</th>
<th>Program</th>
</tr>
</thead>
</table>
| Handle the trap  | timer interrupt save regs(A) to k-stack(A) move to kernel mode jump to trap handler | Process A ...
| Call `switch()` routine restore regs(B) from proc-struct(B) switch to k-stack return-from-trap (into B) | | |
| save regs(A) to proc-struct(A) | | |
| restore regs(B) from proc-struct(B) | | |
| switch to k-stack | | |
| return-from-trap (into B) | | |
Handle the trap
Call `switch()` routine
  save regs(A) to proc-struct(A)
  restore regs(B) from proc-struct(B)
  switch to k-stack
  return-from-trap (into B)

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Hardware</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer interrupt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>save regs(A) to</td>
<td></td>
<td></td>
</tr>
<tr>
<td>k-stack(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>move to kernel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jump to trap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>handler</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>restore regs(B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>from k-stack(B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>move to user</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jump to B’s IP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Process A ...

Process B ...
Basic Schedulers
Vocabulary

**Workload**: set of job descriptions

**Scheduler**: logic that decides when jobs run

**Metric**: measurement of scheduling quality
Vocabulary

**Workload**: set of job descriptions

**Scheduler**: logic that decides when jobs run

**Metric**: measurement of scheduling quality

Scheduler “algebra”, given 2 variables, find the 3rd:

\[ f(W, S) = M \]
Workload Assumptions

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. The run-time of each job is known
Scheduling Basics

**Workloads:**
- arrival_time
- run_time

**Schedulers:**
- FIFO
- SJF
- STCF
- RR

**Metrics:**
- turnaround_time
- response_time
Scheduling Basics

**Workloads:**
- arrival_time
- run_time

**Schedulers:**
- FIFO
- SJF
- STCF
- RR

**Metrics:**
- turnaround_time
- response_time
Example: workload, scheduler, metric

<table>
<thead>
<tr>
<th>JOB</th>
<th>arrival_time (s)</th>
<th>run_time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.0001</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>0.0002</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>0.0003</td>
<td>10</td>
</tr>
</tbody>
</table>

**FIFO**: First In, First Out (run jobs in `arrival_time` order)

**What is our turnaround?**: `completion_time - arrival_time`
Example: workload, scheduler, metric

<table>
<thead>
<tr>
<th>JOB</th>
<th>arrival_time (s)</th>
<th>run_time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>~0</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>~0</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>~0</td>
<td>10</td>
</tr>
</tbody>
</table>

**FIFO**: First In, First Out (run jobs in `arrival_time` order)

**What is our turnaround?**: `completion_time - arrival_time`
## Event Trace

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A arrives</td>
</tr>
<tr>
<td>0</td>
<td>B arrives</td>
</tr>
<tr>
<td>0</td>
<td>C arrives</td>
</tr>
<tr>
<td>0</td>
<td>run A</td>
</tr>
<tr>
<td>10</td>
<td>complete A</td>
</tr>
<tr>
<td>10</td>
<td>run B</td>
</tr>
<tr>
<td>20</td>
<td>complete B</td>
</tr>
<tr>
<td>20</td>
<td>run C</td>
</tr>
<tr>
<td>30</td>
<td>complete C</td>
</tr>
</tbody>
</table>
Trace Visualization

A | B | C

0  20  40
What is the average turnaround time? (Q1)

Def: \( \text{turnaround\_time} = \text{completion\_time} - \text{arrival\_time} \)
What is the average turnaround time? (Q1)

Def: \( \text{turnaround\_time} = \text{completion\_time} - \text{arrival\_time} \)
What is the average turnaround time? (Q1)

Def: \textit{turnaround\_time} = \textit{completion\_time} - \textit{arrival\_time}
What is the average turnaround time? (Q1)

\[
(10 + 20 + 30) / 3 = 20s
\]
Scheduling Basics

**Workloads:**
- arrival_time
- run_time

**Schedulers:**
- FIFO
- SJF
- STCF
- RR

**Metrics:**
- turnaround_time
- response_time
<table>
<thead>
<tr>
<th>Workloads:</th>
<th>Schedulers:</th>
<th>Metrics:</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrival_time</td>
<td>FIFO</td>
<td>turnaround_time</td>
</tr>
<tr>
<td>run_time</td>
<td>SJF</td>
<td>response_time</td>
</tr>
<tr>
<td></td>
<td>STCF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RR</td>
<td></td>
</tr>
</tbody>
</table>
Workload Assumptions

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. The run-time of each job is known
Workload Assumptions

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. The run-time of each job is known
“Solve” for $W$

$$f(W, S) = M$$

**Workload**: ?

**Scheduler**: FIFO

**Metric**: turnaround is high
Example: Big First Job

<table>
<thead>
<tr>
<th>JOB</th>
<th>arrival_time (s)</th>
<th>run_time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>~0</td>
<td>60</td>
</tr>
<tr>
<td>B</td>
<td>~0</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>~0</td>
<td>10</td>
</tr>
</tbody>
</table>

What is the average turnaround time? (Q2)
Example: Big First Job

<table>
<thead>
<tr>
<th>JOB</th>
<th>arrival_time (s)</th>
<th>run_time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>~0</td>
<td>60</td>
</tr>
<tr>
<td>B</td>
<td>~0</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>~0</td>
<td>10</td>
</tr>
</tbody>
</table>

What is the average turnaround time? (Q2)
Example: Big First Job

A: 60s  
B: 70s  
C: 80s  

Average turnaround time: 70s
Convoy Effect
Better Schedule?
Better Schedule?
New scheduler: SJF (Shortest Job First)

Policy: when deciding what job to run next, choose the one with smallest run_time
Example: Shortest Job First

<table>
<thead>
<tr>
<th>JOB</th>
<th>arrival_time (s)</th>
<th>run_time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>~0</td>
<td>60</td>
</tr>
<tr>
<td>B</td>
<td>~0</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>~0</td>
<td>10</td>
</tr>
</tbody>
</table>

What is the average turnaround time with SJF? (Q3)
Example: Shortest Job First

<table>
<thead>
<tr>
<th>JOB</th>
<th>arrival_time (s)</th>
<th>run_time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>~0</td>
<td>60</td>
</tr>
<tr>
<td>B</td>
<td>~0</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>~0</td>
<td>10</td>
</tr>
</tbody>
</table>

What is the average turnaround time with SJF? (Q3)
What is the average turnaround time with SJF? (Q3)

\[
(80 + 10 + 20) / 3 = \sim 36.7\text{s}
\]
Scheduling Basics

**Workloads:**
- arrival_time
- run_time

**Schedulers:**
- FIFO
- SJF
- STCF
- RR

**Metrics:**
- turnaround_time
- response_time
Scheduling Basics

**Workloads:**
- arrival_time
- run_time

**Schedulers:**
- FIFO
- SJF
- STCF
- RR

**Metrics:**
- turnaround_time
- response_time
Workload Assumptions

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. The run-time of each job is known
Workload Assumptions

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. The run-time of each job is known
### Shortest Job First (Arrival Time)

<table>
<thead>
<tr>
<th>JOB</th>
<th>arrival_time (s)</th>
<th>run_time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>~0</td>
<td>60</td>
</tr>
<tr>
<td>B</td>
<td>~10</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>~10</td>
<td>10</td>
</tr>
</tbody>
</table>

What is the average turnaround time with SJF?
Stuck Behind a Tractor Again

What is the average turnaround time?

(Q4)
What is the average turnaround time?

(Q4)
Stuck Behind a Tractor Again

A: 60s  B: 60s  C: 70s

What is the average turnaround time?

\[
\frac{60 + 60 + 70}{3} = 63.3s
\]
A Preemptive Scheduler

Prev schedulers: FIFO and SJF are non-preemptive

New scheduler: STCF (Shortest Time-to-Completion First)

Policy: switch jobs so we always run the one that will complete the quickest
SJF

[B,C arrive]

Average turnaround time: 70s
Average turnaround time: (Q4)
STCF

[B, C arrive]

Average turnaround time: (Q4)
Average turnaround time: 36.6
Scheduling Basics

**Workloads:**
- arrival_time
- run_time

**Schedulers:**
- FIFO
- SJF
- STCF
- RR

**Metrics:**
- turnaround_time
- response_time
Scheduling Basics

Workloads:
- arrival_time
- run_time

Schedulers:
- FIFO
- SJF
- STCF
- RR

Metrics:
- turnaround_time
- response_time
Response Time

Sometimes we care about when a job starts instead of when it finishes.

Why?

\[ \text{response\_time} = \text{first\_run\_time} - \text{arrival\_time} \]
Response vs. Turnaround

B’s turnaround: 20s
B’s response: 10s

[B arrives]
Round-Robin Scheduler

Prev schedulers: FIFO, SJF, and STCF have poor response time

New scheduler: RR (Round Robin)

Policy: alternate between ready processes every fixed-length slice
FIFO vs. RR (Q5) — which is each?
FIFO vs. RR (Q5) — which is each?

Avg Response Time?

Q5

Avg Response Time?

Q5
FIFO vs. RR (Q5) — which is each?

Avg Response Time?
(0+1+2)/3 = 1

Avg Response Time?
(0+5+10)/3 = 5
Scheduling Basics

**Workloads:**
- arrival_time
- run_time

**Schedulers:**
- FIFO
- SJF
- STCF
- RR

**Metrics:**
- turnaround_time
- response_time
Workload Assumptions

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. The run-time of each job is known
Workload Assumptions

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. The run-time of each job is known
Not I/O Aware

CPU:
A  A  A  B

Disk:
A  A

0  20  40  60  80
I/O Aware (Overlap)

CPU: A B A B A B

Disk: A A
Workload Assumptions

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. The run-time of each job is known
Workload Assumptions

1. Each job runs for the same amount of time
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. The run-time of each job is known (need smarter, fancier scheduler)
MLFQ
MLFQ (Multi-Level Feedback Queue)

Goal: general-purpose scheduling

Must support two job types with distinct goals
- “interactive” programs care about response time
- “batch” programs care about turnaround time

Approach: multiple levels of round-robin
Priorities

Rule 1: If priority(A) > Priority(B), A runs
Rule 2: If priority(A) == Priority(B), A & B run in RR

Q3 → A
Q2 → B
Q1
Q0 → C → D
Priorities

Rule 1: If priority(A) > Priority(B), A runs
Rule 2: If priority(A) == Priority(B), A & B run in RR

How to know process type to set priority?

Approach 1: nice
Approach 2: history
Priorities

Rule 1: If priority(A) > Priority(B), A runs
Rule 2: If priority(A) == Priority(B), A & B run in RR

Q3 → A
Q2 → B
Q1
Q0 → C → D

How to know process type to set priority?

Approach 1: nice
Approach 2: history
Processes alternate between I/O and CPU work

Consider each CPU session its own “job”

Guess what a job will be like based on past jobs from the same process
More MLFQ Rules

Rule 1: If priority(A) > Priority(B), A runs
Rule 2: If priority(A) == Priority(B), A & B run in RR

More rules:
Rule 3: Processes start at top priority
Rule 4: If job uses whole slice, demote process
An Interactive Process Joins

Q3
Q2
Q1
Q0
What are problems?
What are problems?
- unforgiving
- gaming the system
- hard to tune
(read OSTEP)
Lottery
Lottery Scheduling

Goal: proportional share

Approach:
- give processes lottery tickets
- whoever wins runs
- higher priority => more tickets
int counter = 0;
int winner = getrandom(0, totaltickets);
node_t *current = head;
while(current) {
    counter += current->tickets;
    if (counter > winner)
        break;
    current = current->next;
}
// current is the winner
Lottery Scheduler

- Job A: 1
- Job B: 1
- Job C: 100
- Job D: 200
- Job E: 100

Total tickets: 402
winner = random(402)
Job A (1)  Job B (1)  Job C (100)  Job D (200)  Job E (100)

winner = 102

402 total tickets
Job A (1)  Job B (1)  Job C (100)  Job D (200)  Job E (100)

402 total tickets

is $102 < 1$ ?

winner = 102
winner = 101

is 101 < 1 ?

Job A (1)  Job B (1)  Job C (100)  Job D (200)  Job E (100)

402 total tickets
winner = 100

is 100 < 100 ?

Job A (1)  Job B (1)  Job C (100)  Job D (200)  Job E (100)

402 total tickets
Job A (1)  Job B (1)  Job C (100)  Job D (200)  Job E (100)

402 total tickets

winner = 0

is 0 < 200 ?
Run D!

is $0 < 200$ ?

402 total tickets
int counter = 0;
int winner = getrandom(0, totaltickets);
node_t *current = head;
while(current) {
    counter += current->tickets;
    if (counter > winner)
        break;
    current = current->next;
}

// current gets to run

Who runs if winner is:
50 (Q6)
350 (Q7)
0  (Q8)
Other Lottery Ideas

Ticket Transfers

Ticket Currencies

Ticket Inflation

(read more in OSTEP)
Review Basic Policies
**Workload**

<table>
<thead>
<tr>
<th>JOB</th>
<th>arrival</th>
<th>run</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

**Schedulers:**
- FIFO
- SJF
- STCF
- RR

**Timelines**
**Workload**

<table>
<thead>
<tr>
<th>JOB</th>
<th>arrival</th>
<th>run</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

**Schedulers:**

- FIFO
- SJF
- STCF
- RR

**Timelines**

**RR**

```
ABCABCABABAAAA
```

**SJF**

```
B C A
```

**STCF**

```
BC B A
```

**FIFO**

```
A B C
```
Understand your goals (metrics) and workload, then design your scheduler around that.

General purpose schedulers need to support processes with different types of goals.

Random algorithms are often simple to implement, and avoid corner cases.