COMP 3430

Operating Systems

June 5th, 2019

Goals

By the end of today's lecture, you should be able to:

- Describe scheduling policies
- Describe preemptive multitasking
- Evaluate scheduling policies
- Write a UNIX program that implements scheduling policies

This week *and* next week

Let's take a look at the schedule.

Keep, start, stop.

Your opinion is very important to us. We appreciate your feedback and will use it to evaluate changes and make improvements in our ~~site~~ COURSE.

The tags for this image are
“Excited Person Happy Young Woman Joy Happiness”
(Pixabay License)

Scheduling

- Processes have a lifecycle.
  - They have *states*.
- We've got all these processes/threads talking to each other.
  - Scheduling: decide which one should use resources *next*.

How are we supposed to schedule *that*? (Pixabay License)

Scheduling goals

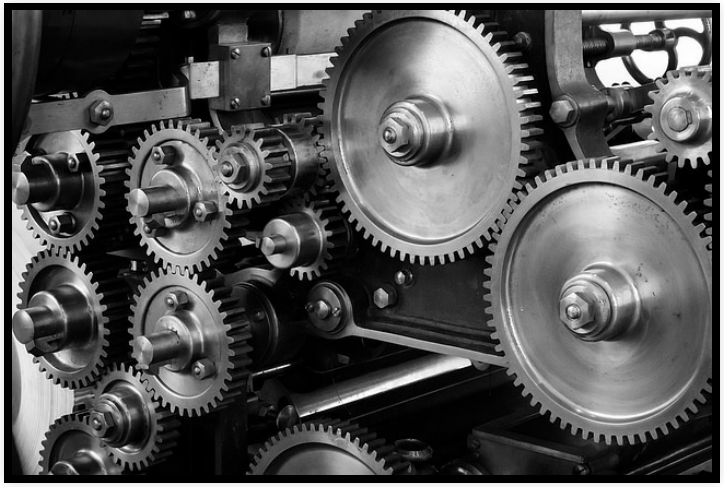
- Schedulers have different (often competing) goals:
  - Minimizing turnaround time ($T_{\text{completion}} - T_{\text{arrival}}$),
  - Maximizing throughput (get as much work done as possible),
  - Being fair to all work that needs to be done (nobody is starved of access to resources),
  - Minimizing response time ($T_{\text{firstrun}} - T_{\text{arrival}}$).

Machinery

Two main parts to the scheduling parts of an OS:

1. The **scheduler** (an oracle that decides which job goes next)
2. The **dispatcher** (the machine that actually *switches* to that job)

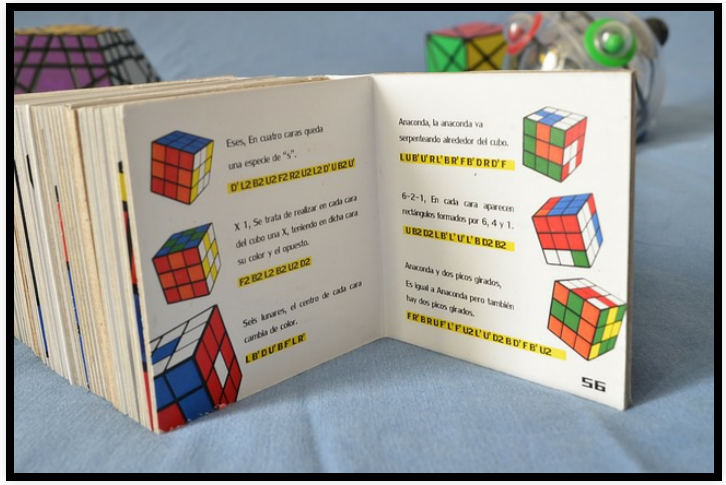
Our focus is on the **scheduler** and the algorithms it might use.



Algorithms

With the person beside you (or thinking about it by yourself, don't be pushy):

Remind yoursel{f,ves}: which scheduling algorithms did we look at in the first class?

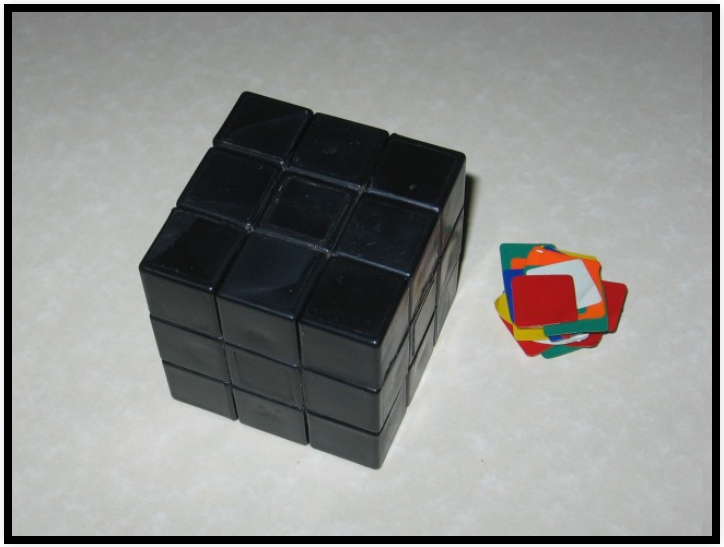


Pixabay License

Weaknesses

Discussion: What kind of *weaknesses* do these algorithms have? Why are they **bad**?

Think about how each of these schedulers might perform with respect to the goals of scheduling.



© Zebeth

1. First come first serve/first in first out(FIFO):
-Minimizing turnaround time: not good with turnaround time.
-Maximizing throughput: good
-Fair to all work: bad(if the first job never get finished, last job cant get access to resources.)
-Minimizing response time: not good.(same reason as last one.)

2. Shortest remaining time first:
pick the smallest amount of remaining resource utilization.
-Minimizing turnaround time: not good with turnaround time.
-Maximizing throughput: good
-Fair to all work: bad(not fair at all)
-Minimizing response time: not good.(same reason as last one.)

3. priority scheduling:
who gets to decide what priorities are?
-this will have the same problems with SRTF.

None them are really good.

Preemption

- **Main** problem: have to wait for a job to complete before scheduling next.
- Idea: Let the OS **stop** a running job *during* its *execution*.
  - When the OS regains control of the processor it can *schedule* then *dispatch* a new job.

When?

Think about it: When might the OS have an opportunity to regain control of the processor?

Now, that's when

- The OS can regain control at any time (…with *hardware assistance*).
  - The hardware can (partially) suspend a job and call code in the kernel.
  - This *interruption* usually happens at specified time intervals (literally a clock).

Round-robin

- Idea: Let each job have some set period of time on the processor (n milliseconds)
  - After that time period has elapsed, *switch* to a different job.
- Let's tie some shoes .

Pixabay License

Problems?

Discuss (or think): What problems might round-robin have? Consider each of the metrics that we defined earlier:

- Turnaround time ($T_{\text{completion}} - T_{\text{arrival}}$),
- Maximizing throughput (get as much work done as possible),
- Being fair to all work that needs to be done (nobody is starved of access to resources),
- Minimizing response time ($T_{\text{firstrun}} - T_{\text{arrival}}$).

Round-Robin Algorithm:
- Turnaround time: Bad
- Maximizing throughput: good
- Being fair: excellent
- Minimizing response time: good,
(better than FIFO)

What's the kernel doing?

Let's (briefly) take a guided tour through the Linux kernel source code.

Start with https://elixir.bootlin.com/linux/latest/source/kernel/sched/core.c, go to line 3555.

```
asmlinkage __visible void __sched schedule(void)
```

- What is `asmlinkage`?
- What is `__visible`?

Goals

By the end of today's lecture, you should be able to:

- Describe scheduling policies (we have seen several)
- Describe preemptive multitasking (how does the OS *do* this?)
- Evaluate scheduling policies (optimizing for *metrics*)
- Write a UNIX program that implements scheduling policies ?


EAT
MY
SHORTS.


EAT
PANT