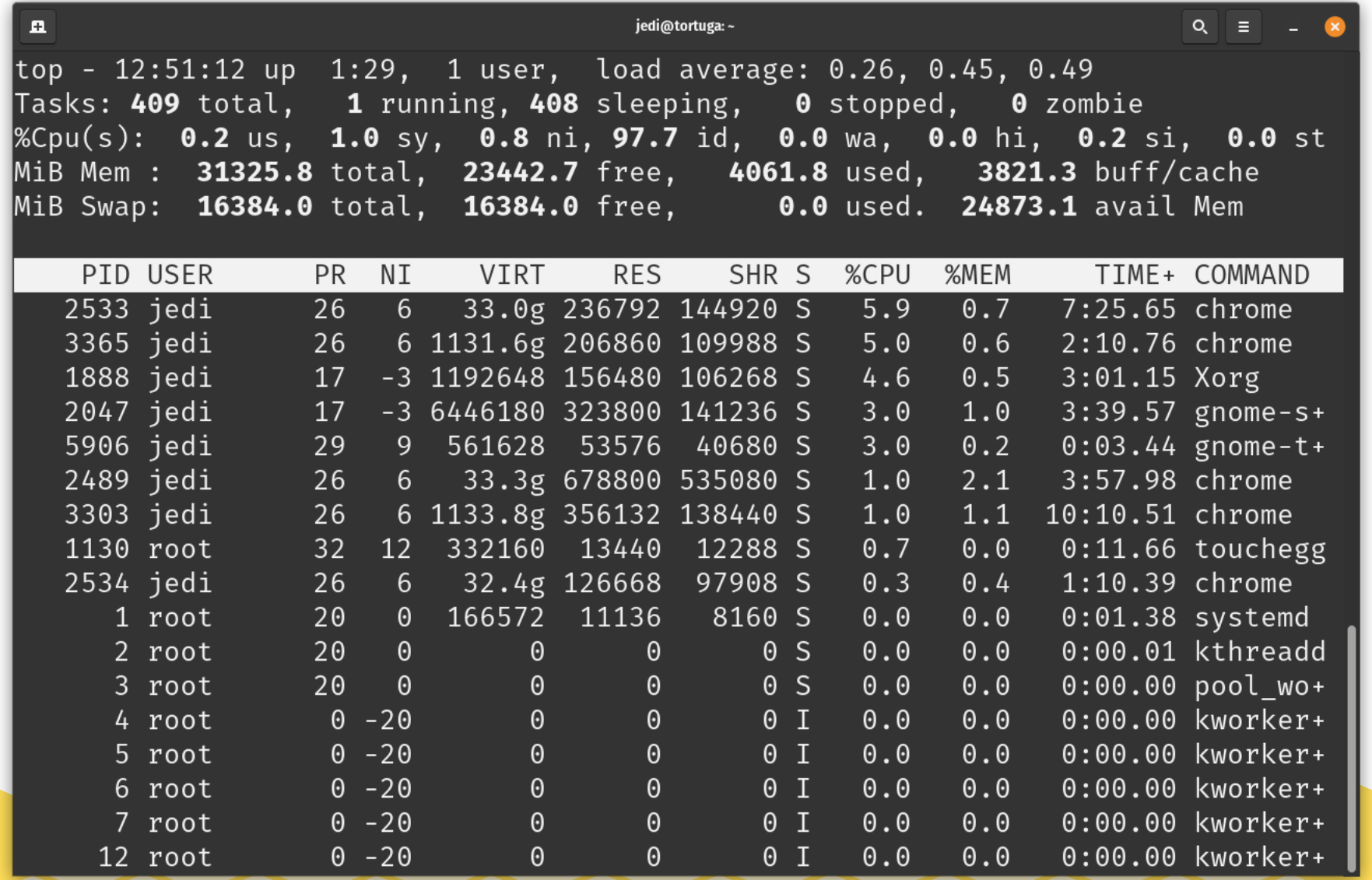
Scheduling, wait queues

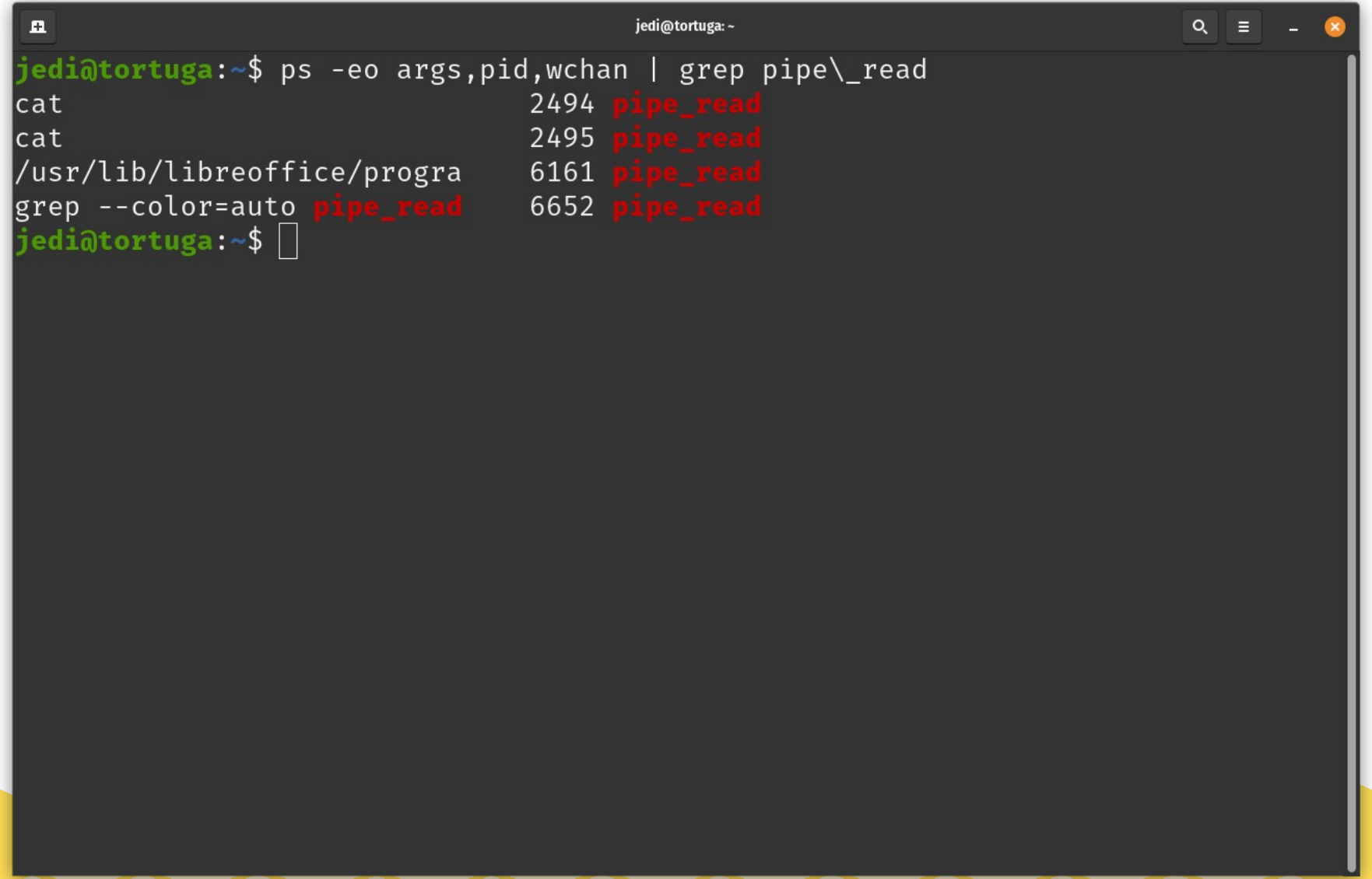
CSE 536 Spring 2024
jedimaestro@asu.edu

Don’t panic because of my teaching philosophy. Systems is a combination of very abstract and very concrete ideas. Like my Chinese teacher told me, you just have to jump into it and then get used to it...

Outline

- Let’s look at processes some more
  - Signals
- Terminology of scheduling
  - Wait states
- Textbook scheduling algorithms
- Actual scheduling algorithms
- Input/Output (I/O)


jedi@tortuga: ~
top - 12:51:12 up 1:29, 1 user, load average: 0.26, 0.45, 0.49
Tasks: 409 total, 1 running, 408 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.2 us, 1.0 sy, 0.8 ni, 97.7 id, 0.0 wa, 0.0 hi, 0.2 si, 0.0 st
MiB Mem : 31325.8 total, 23442.7 free, 4061.8 used, 3821.3 buff/cache
MiB Swap: 16384.0 total, 16384.0 free, 0.0 used. 24873.1 avail Mem
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
2533 jedi 26 6 33.0g 236792 144920 S 5.9 0.7 7:25.65 chrome
3365 jedi 26 6 1131.6g 206860 109988 S 5.0 0.6 2:10.76 chrome
1888 jedi 17 -3 1192648 156480 106268 S 4.6 0.5 3:01.15 Xorg
2047 jedi 17 -3 6446180 323800 141236 S 3.0 1.0 3:39.57 gnome-s+
5906 jedi 29 9 561628 53576 40680 S 3.0 0.2 0:03.44 gnome-t+
2489 jedi 26 6 33.3g 678800 535080 S 1.0 2.1 3:57.98 chrome
3303 jedi 26 6 1133.8g 356132 138440 S 1.0 1.1 10:10.51 chrome
1130 root 32 12 332160 13440 12288 S 0.7 0.0 0:11.66 touchegg
2534 jedi 26 6 32.4g 126668 97908 S 0.3 0.4 1:10.39 chrome
1 root 20 0 166572 11136 8160 S 0.0 0.0 0:01.38 systemd
2 root 20 0 0 0 0 S 0.0 0.0 0:00.01 kthreadd
3 root 20 0 0 0 0 S 0.0 0.0 0:00.00 pool_wo+
4 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 kworker+
5 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 kworker+
6 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 kworker+
7 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 kworker+
12 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 kworker+


jedi@tortuga: ~
jedi@tortuga:~$ ps -eo args,pid,wchan | grep pipe_read
cat 2494 pipe_read
cat 2495 pipe_read
/usr/lib/libreoffice/progra 6161 pipe_read
grep --color=auto pipe_read 6652 pipe_read
jedi@tortuga:~$

Process Control Block

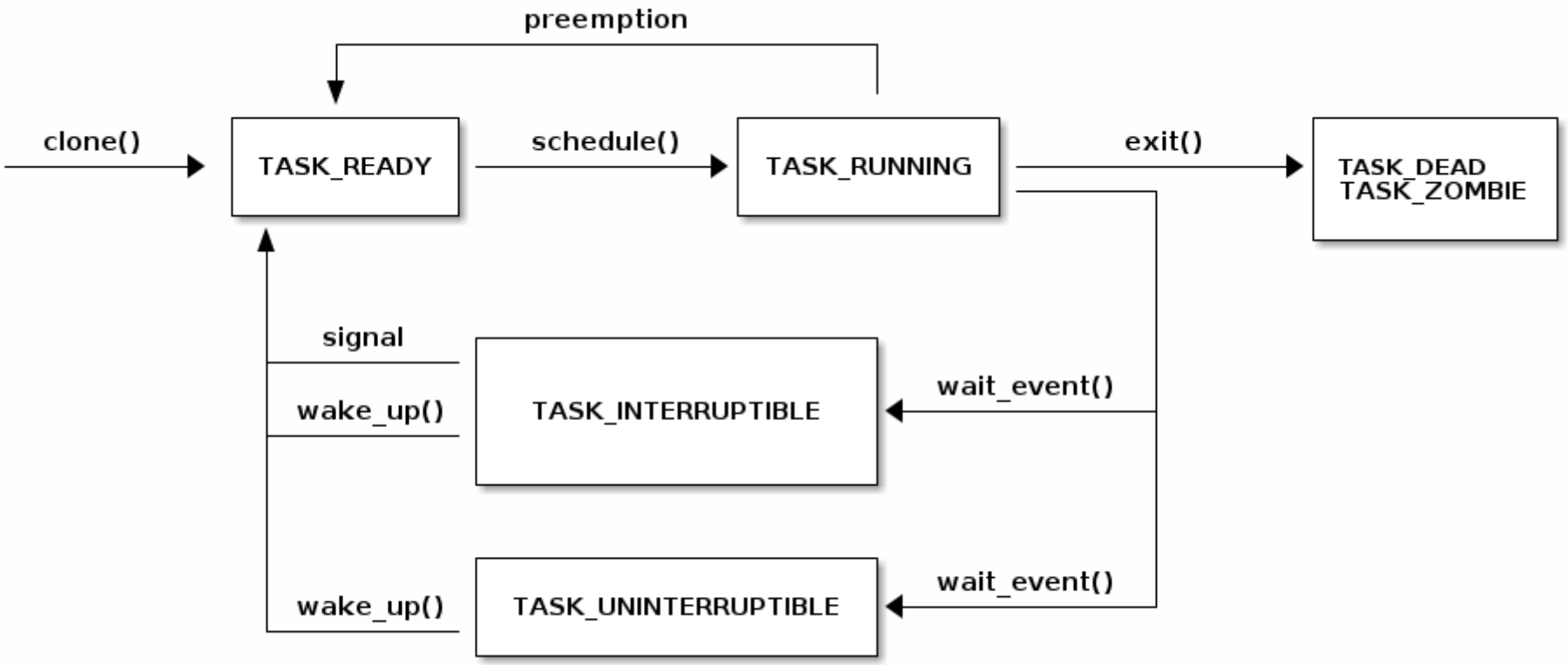
- State
- Saved registers
- Address space
- File descriptor table
- Signal information
- Much more…

https://www.baeldung.com/linux/pcb

```
/* Simplified representation of the task_struct structure in Linux kernel */

struct task_struct {
    volatile long state;            // Process state (e.g., TASK_RUNNING,
TASK_STOPPED)
    struct thread_info *thread_info;
    struct exec_domain *exec_domain; // Execution domain information
(deprecated)
    struct mm_struct *mm;           // Memory management information (address
space)
    struct fs_struct *fs;           // Filesystem information
    struct files_struct *files;     // File descriptor table
    struct signal_struct *signal;   // Signal handlers and signals pending
    struct sighand_struct *sighand; // Signal handling information
    ...
    /* Various other fields */
    ...
};
```

https://linux-kernel-labs.github.io/refs/heads/master/lectures/processes.html



- TASK_INTERRUPTIBLE … Can be woken up by a signal.
- TASK_UNINTERRUPTIBLE … Can’t be woken up by a signal, *e.g.*, is waiting for some special event
  - Probably a kernel thread

fork() and exec()

```
int pid = fork()
if (pid == 0) {
    exec(“/bin/ls”);
} else {
    waitpid(pid, &status, options);
}
```

```
man fork
man clone3
```

Illusions

- Create the illusion each process has its own CPU
  - Context switch
- Create the illusion each process has its own memory
  - Virtual memory
    - Physical memory is divided into different virtual memory spaces
    - We'll discuss this more later in the semester
- Create the illusion each OS has its own physical memory and CPU
  - Virtualization

Context switches

- Reasons a CPU stops executing a process and starts executing code inside the kernel (*e.g.*, interrupt handler or scheduler)
  - Exceptions, *e.g.* …
    - Divide by zero
    - System call (could also be placed under yield)
  - Interrupts, *e.g.* …
    - I/O event
  - Yield
    - I/O request or placed on some wait queue

Simple schedulers

- FIFO, *a.k.a.*, FCFS (First In First Out, or First Come First Serve)
  - 11111111112223333
- Turnaround time
  - How long a process takes to complete
  - Assume all processes are ready in sequence 1, 2, 3 at the beginning
    - Average turnaround time = average(10, 13, 19) = 14

Simple schedulers (continued...)

- Shortest Job First
  - 22233333311111111111
  - Turnaround time improved
    - Average(3, 9, 19) = 10.333... (less than 14)

So, why not use Shortest Job First?

Reason #1

- We can’t see into the future

Reason #2

- Without *preemption*, it's hard to get a good response time
  - Response time: How long it takes the CPU to respond to a request made by a process
    - *E.g.*, you press a key, you'd like to see that letter on the screen

Add a timer interrupt...

- …that, *e.g.*, goes off every 10ms or 1ms
- Gives the scheduler a chance to schedule a new process, *e.g.*, if there's been some input and they're out of the wait queue
- Round Robin scheduler
  - Divide CPU time into slices
  - E.g., each process gets two time slices
    - 11223311233113311111

Can do even better...

- If we could see into the future?
  - Could improve response time by prioritizing (*i.e.*, letting them skip in line) processes that are very likely to yield the CPU quickly
- Can predict future behavior based on past behavior

Multilevel Feedback Queue

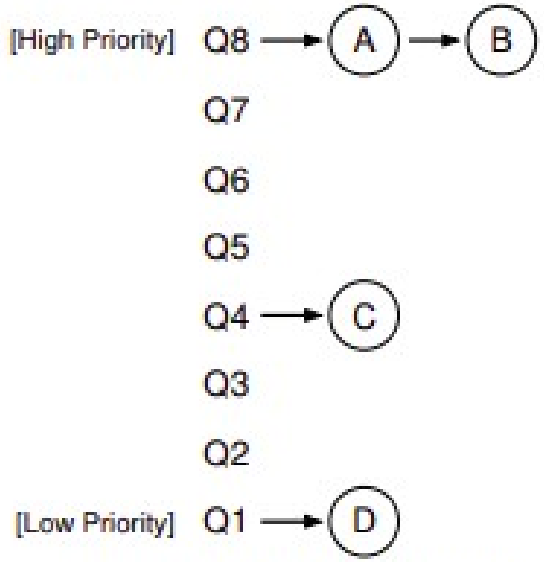


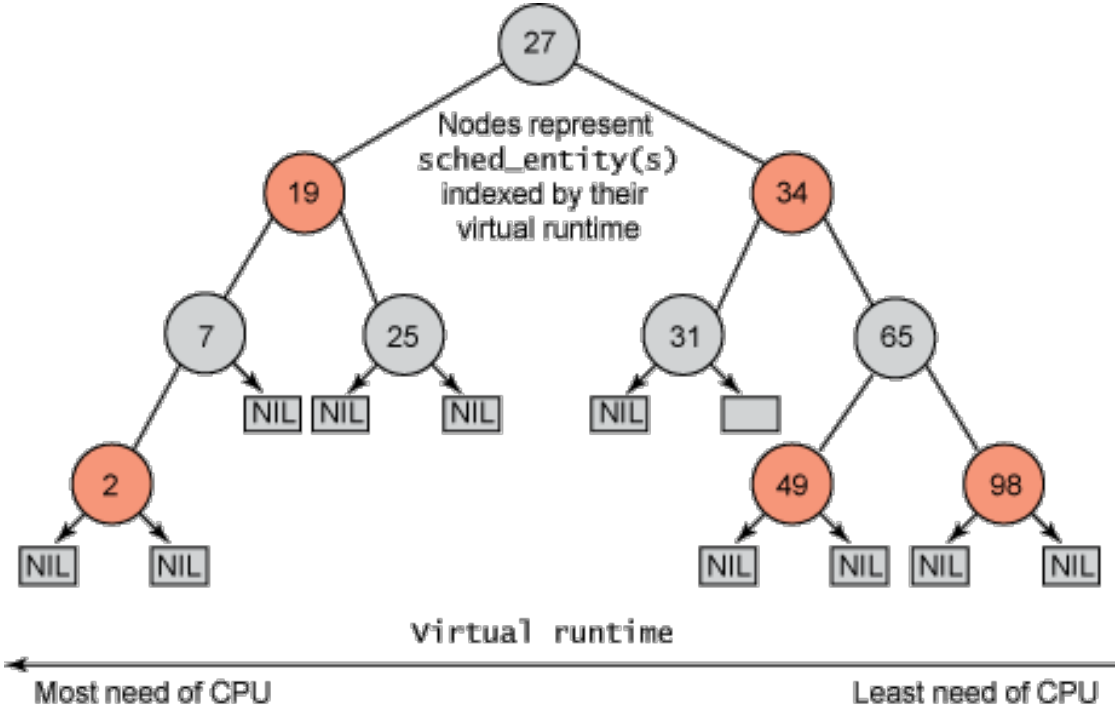
Figure 8.1: **MLFQ Example**

https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched-mlfq.pdf

MLFQ

- Fernando Corbato’s Turing award is based on this
- Famously used in Solaris
- Modern schedulers are not always MLFQ’s but are based on the same ideas
- Priorities can also have a static element to them, in general
  - `man nice`
- What about starvation?

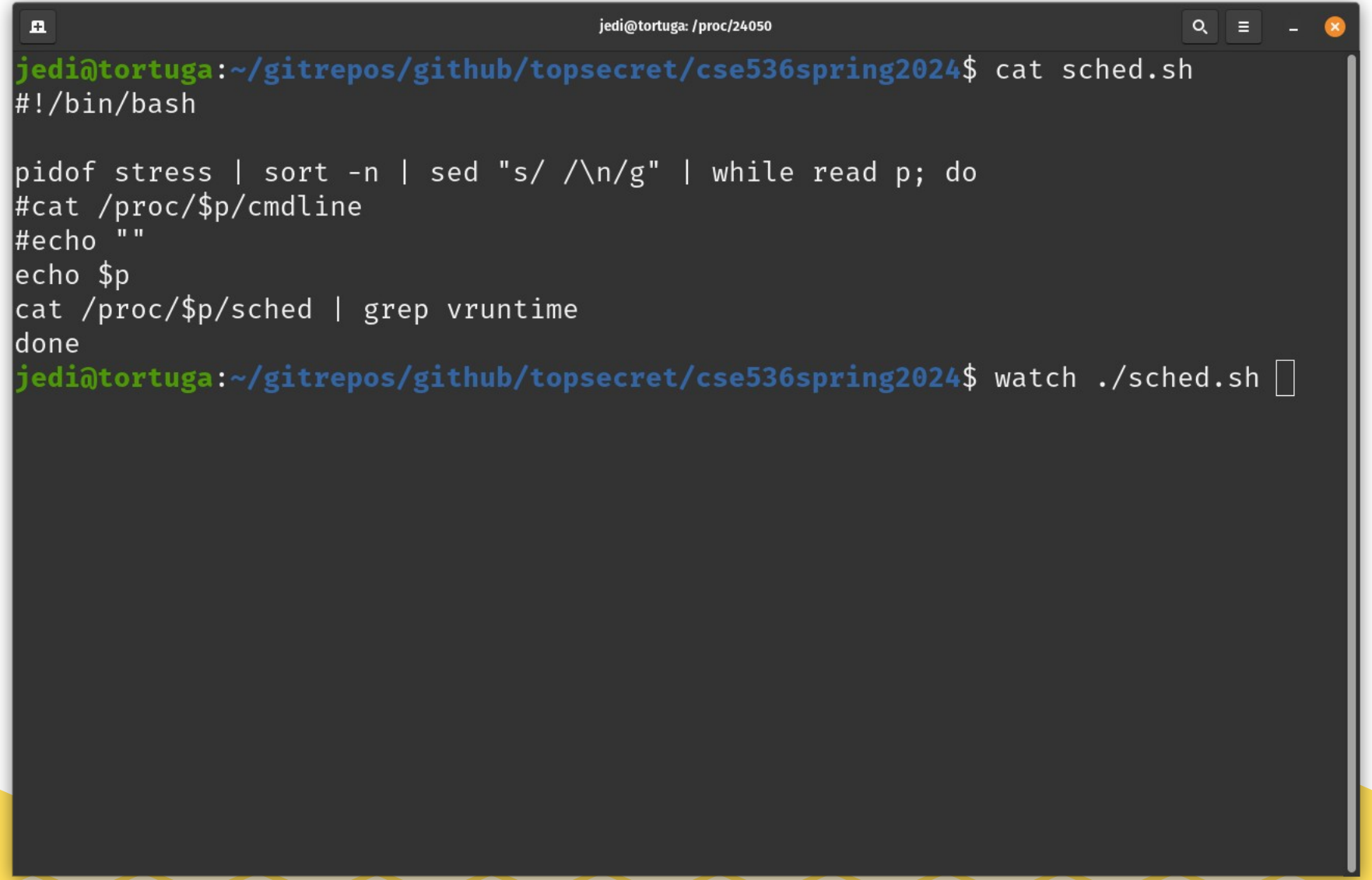
Linux Completely Fair Scheduler

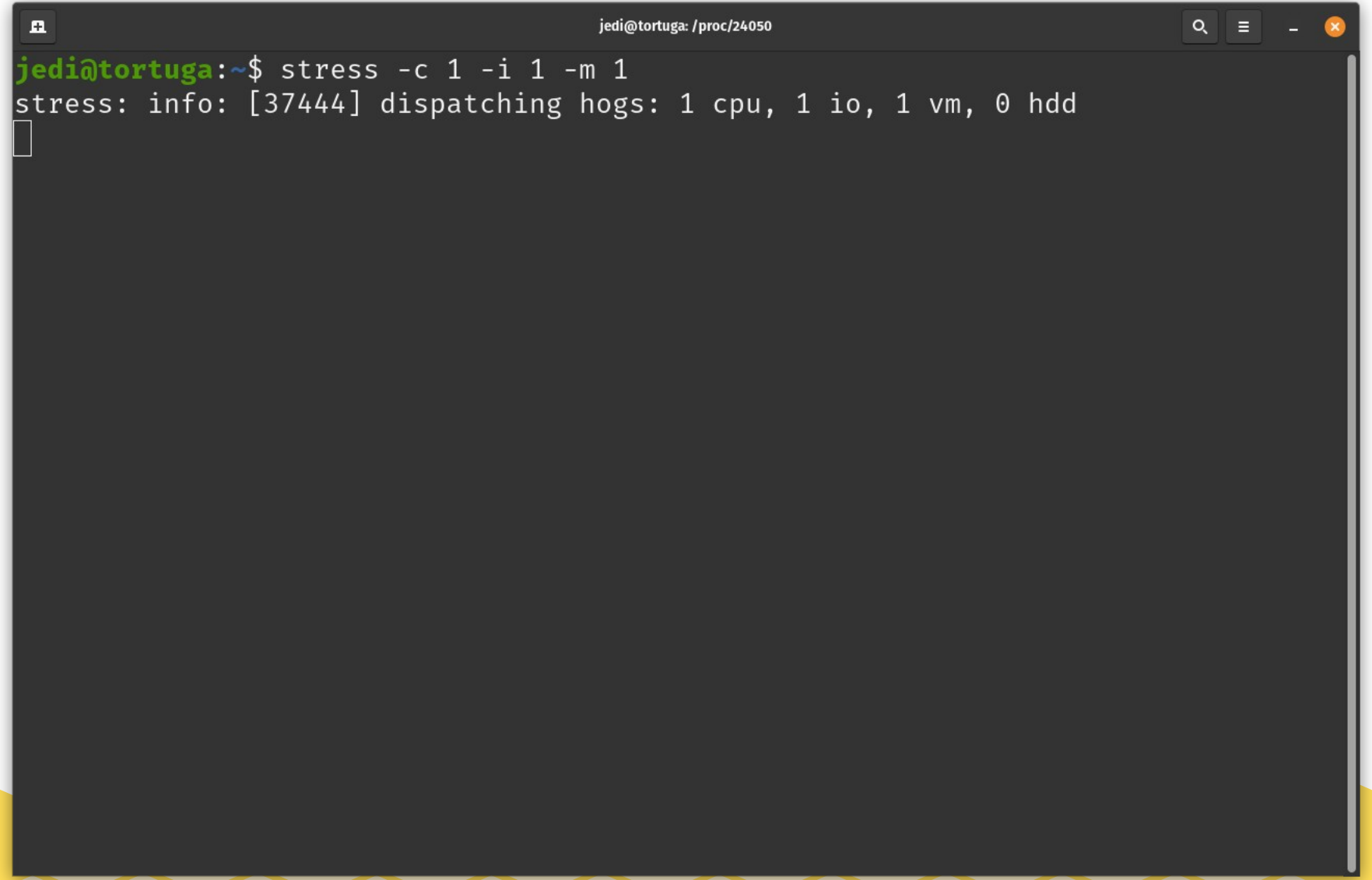


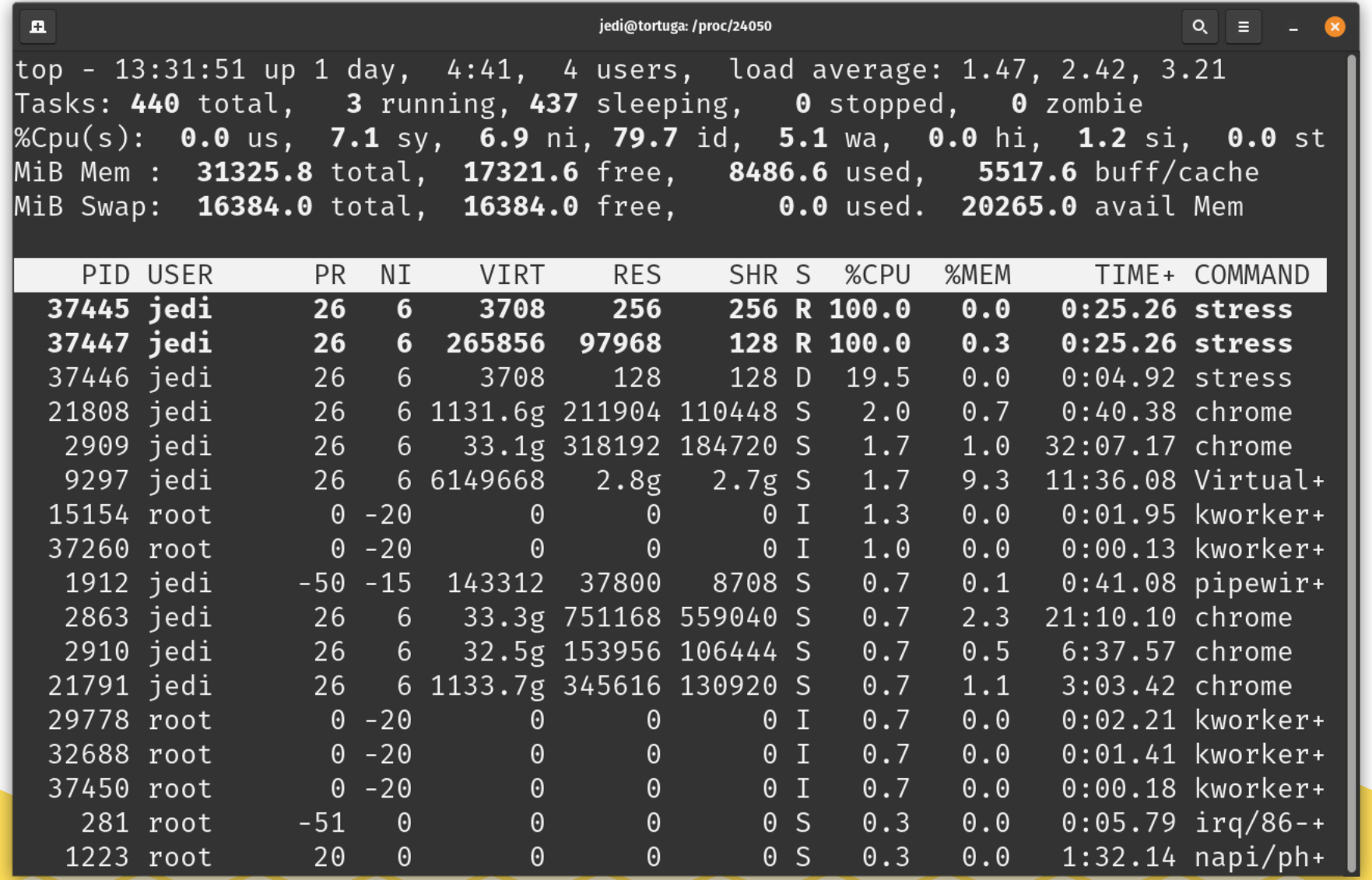
https://svalaks.medium.com/linux-internals-completely-fair-scheduling-cfs-cpu-scheduler-algorithm-7412c08d2e37

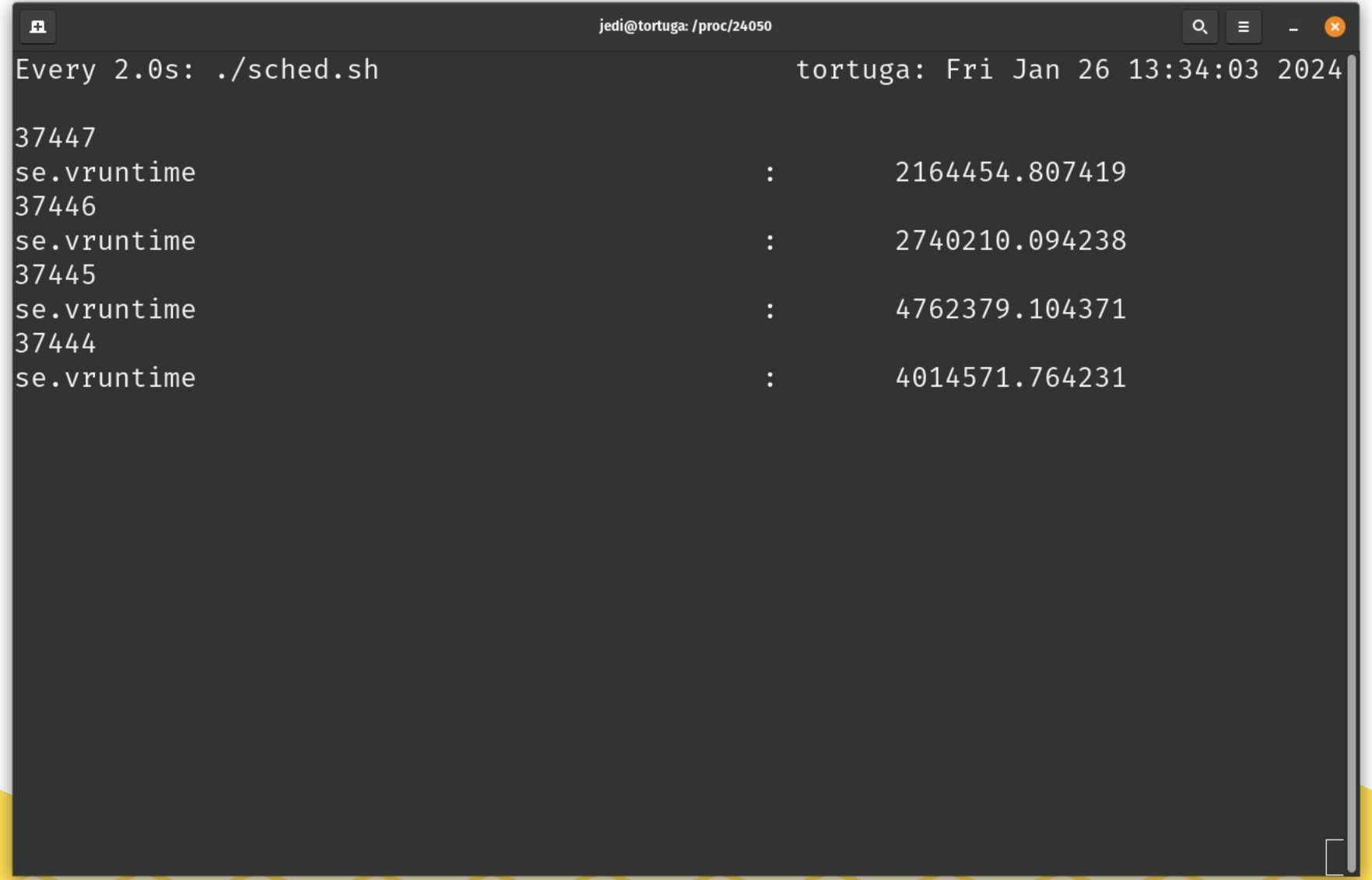
Linux CFS

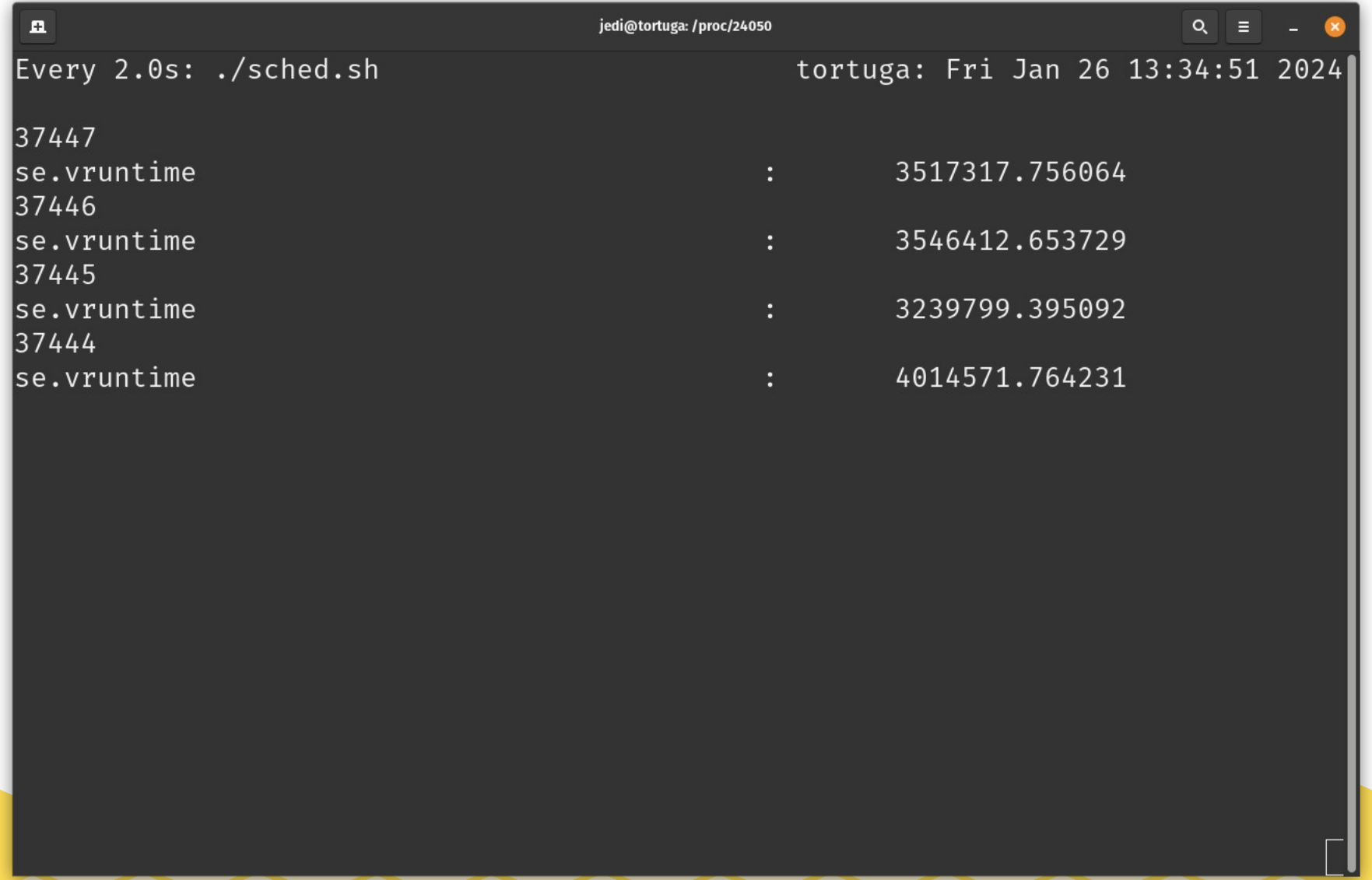
- Picks process from left-most node in O(1) time
- Reinserts when a process is done in O(log(N)) time
- The more you yield the CPU, the more you stay to the left
- The more CPU you hog, the more you move to the right
- Priority is also part of the slice calculation
- Good tradeoff of throughput and responsiveness, no starvation


jedi@tortuga: /proc/24050
jedi@tortuga:~/gitrepos/github/topsecret/cse536spring2024$ cat sched.sh
#!/bin/bash
pidof stress | sort -n | sed "s/ /\n/g" | while read p; do
#cat /proc/$p/cmdline
#echo ""
echo $p
cat /proc/$p/sched | grep vruntime
done
jedi@tortuga:~/gitrepos/github/topsecret/cse536spring2024$ watch ./sched.sh


jedi@tortuga: /proc/24050
jedi@tortuga:~$ stress -c 1 -i 1 -m 1
stress: info: [37444] dispatching hogs: 1 cpu, 1 io, 1 vm, 0 hdd


jedi@tortuga: /proc/24050
top - 13:31:51 up 1 day, 4:41, 4 users, load average: 1.47, 2.42, 3.21
Tasks: 440 total, 3 running, 437 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 7.1 sy, 6.9 ni, 79.7 id, 5.1 wa, 0.0 hi, 1.2 si, 0.0 st
MiB Mem : 31325.8 total, 17321.6 free, 8486.6 used, 5517.6 buff/cache
MiB Swap: 16384.0 total, 16384.0 free, 0.0 used. 20265.0 avail Mem
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
37445 jedi 26 6 3708 256 256 R 100.0 0.0 0:25.26 stress
37447 jedi 26 6 265856 97968 128 R 100.0 0.3 0:25.26 stress
37446 jedi 26 6 3708 128 128 D 19.5 0.0 0:04.92 stress
21808 jedi 26 6 1131.6g 211904 110448 S 2.0 0.7 0:40.38 chrome
2909 jedi 26 6 33.1g 318192 184720 S 1.7 1.0 32:07.17 chrome
9297 jedi 26 6 6149668 2.8g 2.7g S 1.7 9.3 11:36.08 Virtual+
15154 root 0 -20 0 0 0 I 1.3 0.0 0:01.95 kworker+
37260 root 0 -20 0 0 0 I 1.0 0.0 0:00.13 kworker+
1912 jedi -50 -15 143312 37800 8708 S 0.7 0.1 0:41.08 pipewir+
2863 jedi 26 6 33.3g 751168 559040 S 0.7 2.3 21:10.10 chrome
2910 jedi 26 6 32.5g 153956 106444 S 0.7 0.5 6:37.57 chrome
21791 jedi 26 6 1133.7g 345616 130920 S 0.7 1.1 3:03.42 chrome
29778 root 0 -20 0 0 0 I 0.7 0.0 0:02.21 kworker+
32688 root 0 -20 0 0 0 I 0.7 0.0 0:01.41 kworker+
37450 root 0 -20 0 0 0 I 0.7 0.0 0:00.18 kworker+
281 root -51 0 0 0 0 S 0.3 0.0 0:05.79 irq/86-+
1223 root 20 0 0 0 0 S 0.3 0.0 1:32.14 napi/ph+


jedi@tortuga: /proc/24050
Every 2.0s: ./sched.sh tortuga: Fri Jan 26 13:34:03 2024
37447
se.vruntime : 2164454.807419
37446
se.vruntime : 2740210.094238
37445
se.vruntime : 4762379.104371
37444
se.vruntime : 4014571.764231


jedi@tortuga: /proc/24050
Every 2.0s: ./sched.sh tortuga: Fri Jan 26 13:34:51 2024
37447
se.vruntime : 3517317.756064
37446
se.vruntime : 3546412.653729
37445
se.vruntime : 3239799.395092
37444
se.vruntime : 4014571.764231

Demo

- Parent on bottom never changes
  - In a wait state
- CPU intensive (37445) stays pretty high all the time
- Memory intensive (37447) jumps back and forth
- I/O intensive (37446) usually the lowest