Processes

Program: A passive collection of instructions

Process: The abstraction provided by the OS of a running program

Machine State: What a program can read and change when it is running (registers, address spaces, open files, etc.)

Creation of A Process by OS:

- Load data from disk to memory
- Allocate space for the run-time stack and initialize the stack with arguments (i.e. fill in the parameters for argc and argv)
- Allocate memory for program's heap. Initially small, but OS may grow the heap as needed.
- Setup initial file descriptors (stdin, stdout, stderr).
- Transfer control of the CPU to the newly-created process (i.e. $\mathrm{main}()$).

Context Switch: the CPU stops running one process (or thread) and starts running another

- Process A executes
- Hardware: generates timer interrupt, save `regs(A)` to kernel stack, move to kernel mode and jump to trap handler
- OS: Handle the trap, call `switch` routine, save `regs(A)` $\rightarrow$ `proc_t(A)`, restore `regs(B)` $\leftarrow$ `proc_t(B)`, switch to `k-stack(B)`, return-from-trap (into B)
- Hardware: restore `regs(B)` from kernel stack, move to user mode and jump to process B
- Process B executes.

CPU Scheduling

Running Time Metrics:

- $T_{\text{turnaround}} = T_{\text{completion}} - T_{\text{arrival}}$
- $T_{\text{response}} = T_{\text{first run}} - T_{\text{arrival}}$

FIFO/FCFS: First Come First Served, nonpreemptive

SJF: Shortest Job First, nonpreemptive

STCF: Shortest Time to Completion First, preemptive. Always run job that will complete the quickest

MLFQ: Multi-Level Feedback Queue, preemptive

1. If Priority $(A) >$ Priority (B) then A runs
2. If Priority $(A) ==$ Priority (B) then A&B run in RR
3. Processes start at top priority
4. Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced
5. After some time period S, move all the jobs in the system to the topmost queue.

Lottery Scheduler: Randomly selects the next process to run based on ticket probabilities, giving each process CPU time proportional to its number of tickets.

- Ticket Currency: allows a user to allocate tickets among their running processes.
- Ticket Transfer: allows a process to temporarily hand off its tickets to another Process.
- Ticket Inflation: trusted processes can boost tickets to indicate its need for more CPU time.

Unfairness Metric: $U = \frac{T(\text{process1 completion})}{T(\text{process2 completion})}$

Stride Scheduler

- Each process is assigned a stride, which is the inverse proportion to the number of tickets the process has.
- Every time a process runs, its pass value is incremented by its stride.
- Scheduler selects the process with the smallest pass value.

The Linux Completely Fair Scheduler (CFS):

- Divide a time length evenly among n processes.
- Each process has a `vruntime` and a `nice` value.
- The process with the smallest `vruntime` is selected to run.
- Update `vruntime` of the running process by

$$\texttt{vruntime}_i {+}{=} \frac{\texttt{weight}_0}{\texttt{weight}_{\texttt{nice}_i}} \times \texttt{runtime}_i$$

- Ready jobs' `vruntime` are kept in a red-black tree.
- When jobs wake up, their `vruntime` is set to the minimum value in the tree.

The Linux Completely Fair Scheduler (CFS):

Multi-level Paging

A 2-level paging example:

```
      VA = 0x0214  (15 bits -> 5|5|5)
      +-------------+------------+----------+
      |DirIdx=0x00  |PTIdx=0x10  |Off=0x14 |
      +-------------+------------+----------+
             |
PDBR=13  -->  Physical Page #13 (Page Directory)
             |  Read byte 0 = PDE=0x83
             v  (V=1, Page Table PFN=0x03)
          Physical Page #3 (Page Table)
             |  Read byte 16 = PTE=0x8E
             v  (V=1, Data PFN=0x0E)
          Physical Page #14 (Data Page)
             |  Offset 0x14
             v
    Physical Address = (0x0E<<5) | 0x14 = 0x1D4
```

Innovative Innovation
https://github.com/innovativeinnovation