CSCI315 – Operating Systems Design

Department of Computer Science
Bucknell University

Introduction to CPU Scheduling

Ch 5.1-5.2

This set of notes is based on notes from the textbook authors, as well as L. Felipe Perrone, Joshua Stough, and other instructors.
Xiannong Meng, Fall 2021.

What is CPU scheduling?

- CPU scheduling is a mechanism by the operating system to manage processes[1] to maximize CPU utilization and to minimize user waiting time.
- The goals of scheduling may be in conflict, e.g., CPU utilization and user waiting time. Comprises may be needed.
- CPU scheduling involves algorithm, implementation, and evaluation criteria.

1. Again, here we use processes, threads, or tasks interchangeably.

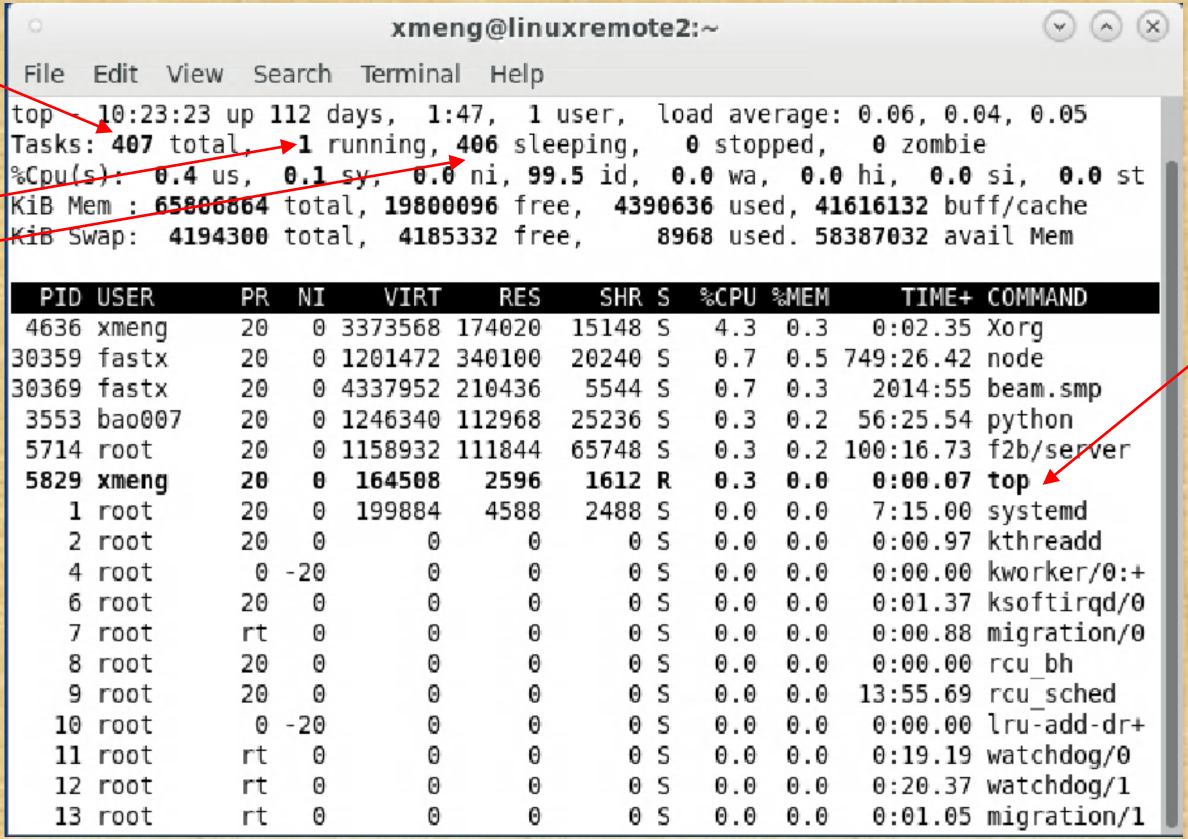
Why CPU scheduling?

- There are many processes active at any moment on a computer. The operating system has to decide to which process to assign CPU, for how long, and how to arrange competing processes.
- Try the “top” command at the Linux command line. (See next slide.)

Example of showing live processes

407 live processes

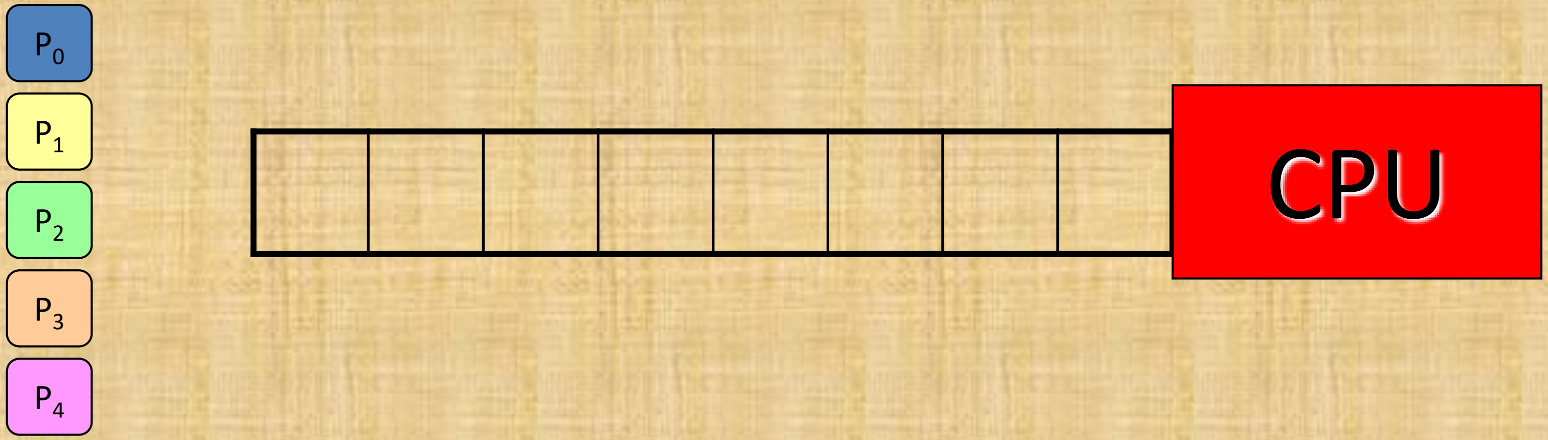
1 running, 406 sleeping



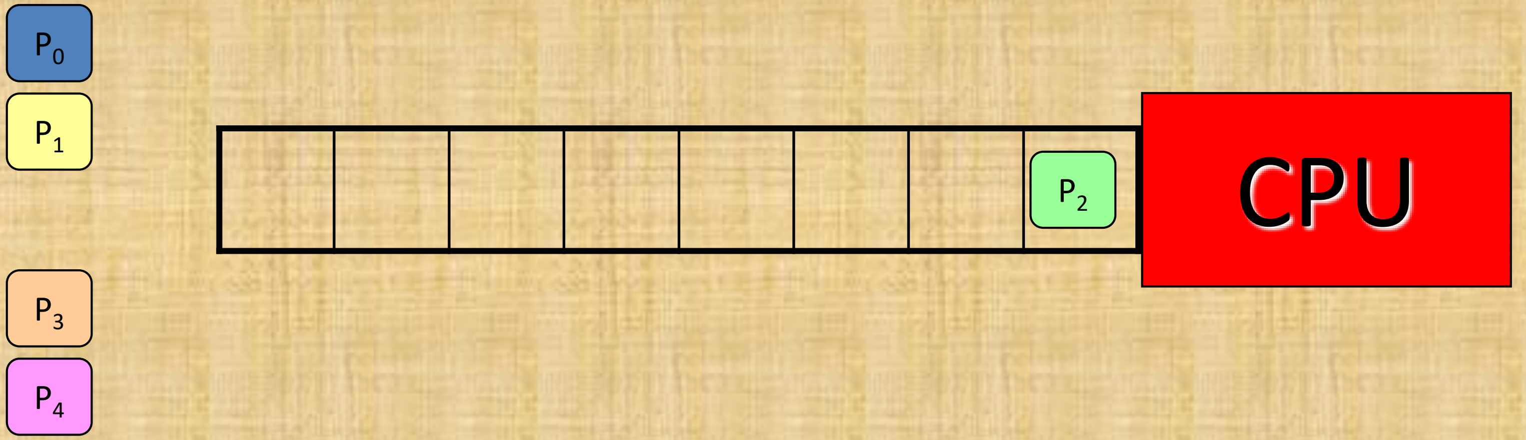
```
xmeng@linuxremote2:~
File  Edit  View  Search  Terminal  Help
top - 10:23:23 up 112 days,  1:47,  1 user,  load average: 0.06, 0.04, 0.05
Tasks: 407 total,   1 running, 406 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.4 us,  0.1 sy,  0.0 ni, 99.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 65806864 total, 19800096 free,  4390636 used, 41616132 buff/cache
KiB Swap:  4194300 total,  4185332 free,     8968 used. 58387032 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 4636 xmeng     20   0 3373568 174020  15148 S   4.3  0.3   0:02.35 Xorg
30359 fastx     20   0 1201472 340100  20240 S   0.7  0.5 749:26.42 node
30369 fastx     20   0 4337952 210436   5544 S   0.7  0.3   2014:55 beam.smp
 3553 bao007    20   0 1246340 112968  25236 S   0.3  0.2  56:25.54 python
 5714 root      20   0 1158932 111844  65748 S   0.3  0.2 100:16.73 f2b/server
 5829 xmeng     20   0  164508   2596   1612 R   0.3  0.0   0:00.07 top
    1 root      20   0  199884   4588   2488 S   0.0  0.0   7:15.00 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.97 kthreadd
    4 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:+
    6 root      20   0       0      0      0 S   0.0  0.0   0:01.37 ksoftirqd/0
    7 root      rt   0       0      0      0 S   0.0  0.0   0:00.88 migration/0
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
    9 root      20   0       0      0      0 S   0.0  0.0  13:55.69 rcu_sched
   10 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 lru-add-dr+
   11 root      rt   0       0      0      0 S   0.0  0.0   0:19.19 watchdog/0
   12 root      rt   0       0      0      0 S   0.0  0.0   0:20.37 watchdog/1
   13 root      rt   0       0      0      0 S   0.0  0.0   0:01.05 migration/1
```

Linux command showing this result

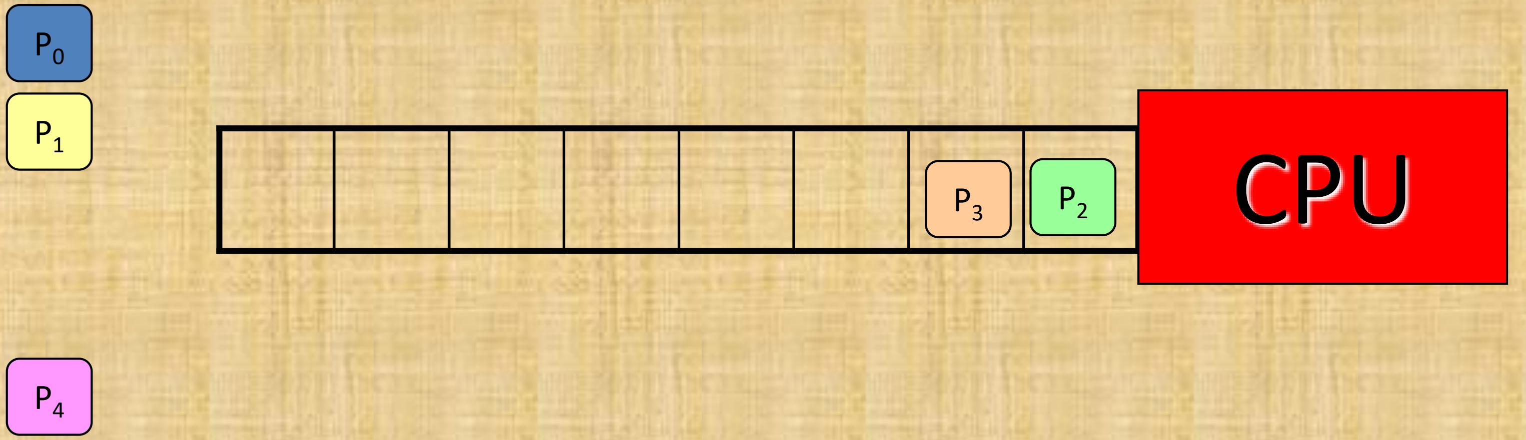
Basic Concepts



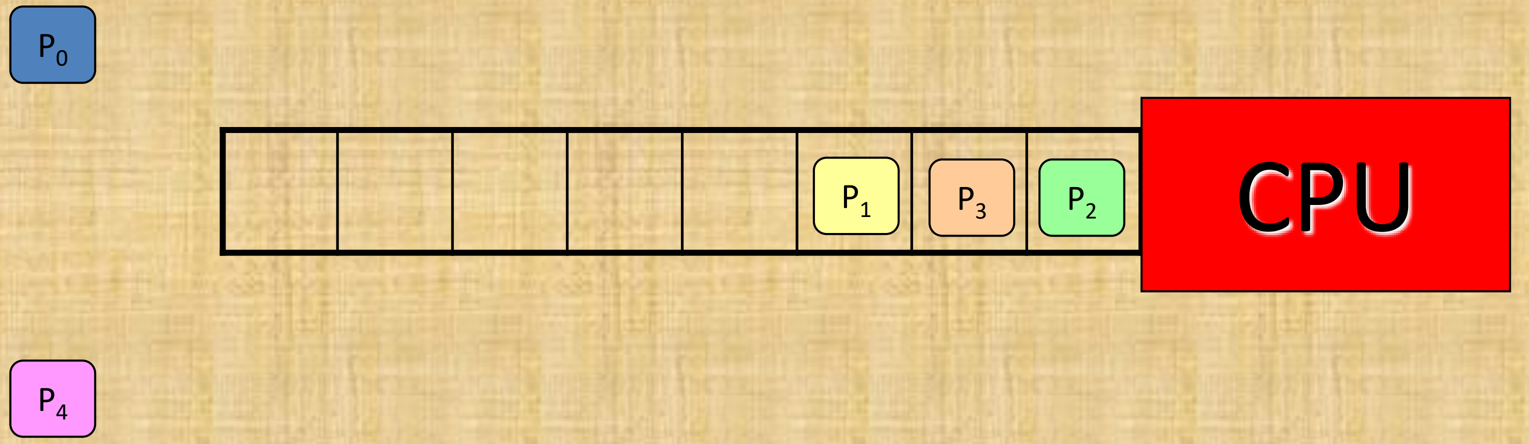
Basic Concepts



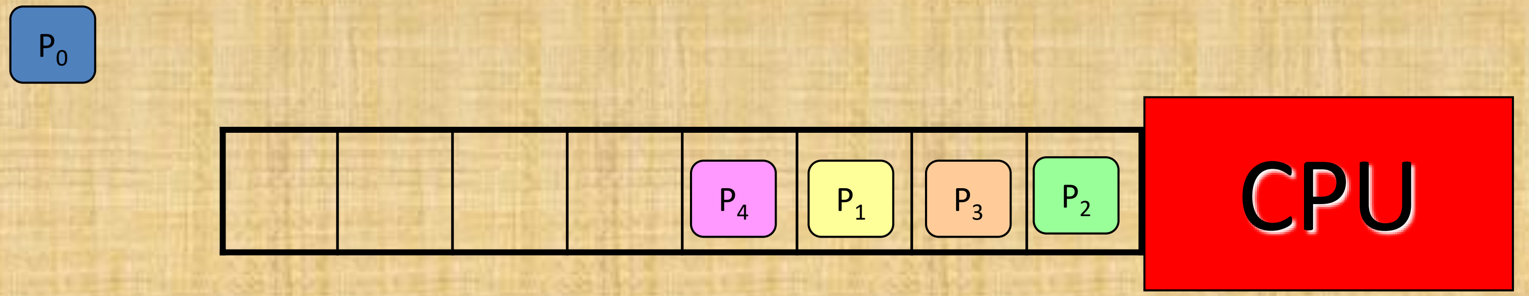
Basic Concepts



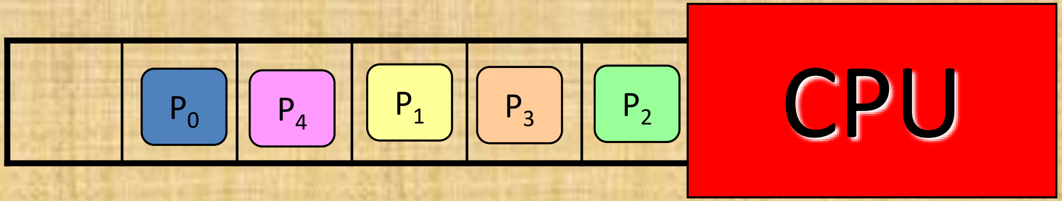
Basic Concepts



Basic Concepts



Basic Concepts



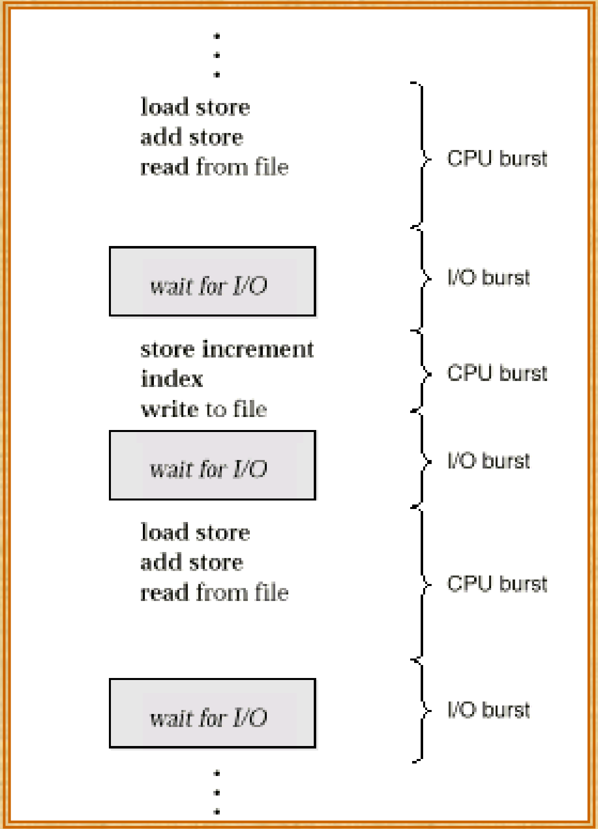
Questions:

- When does a process start competing for the CPU?
- How is the queue of ready processes organized?
- How much time does the system allow a process to use the CPU?
- Does the system allow for priorities and preemption?
- What does it mean to maximize the system's performance?

Basic Concepts

- You want to maximize **CPU utilization** through the use of multiprogramming.
  - **utilization**: percentage of time the CPU is busy
  - **multiprogramming**: allowing multiple processes to be live in the system at the same time
- Each process repeatedly goes through cycles that alternate CPU execution (a CPU burst) and I/O wait (an I/O wait).
  - See the notes on process life cycle in Chapter 3
- Empirical evidence indicates that CPU-burst lengths have a distribution such that there is a large number of short bursts and a small number of long bursts.

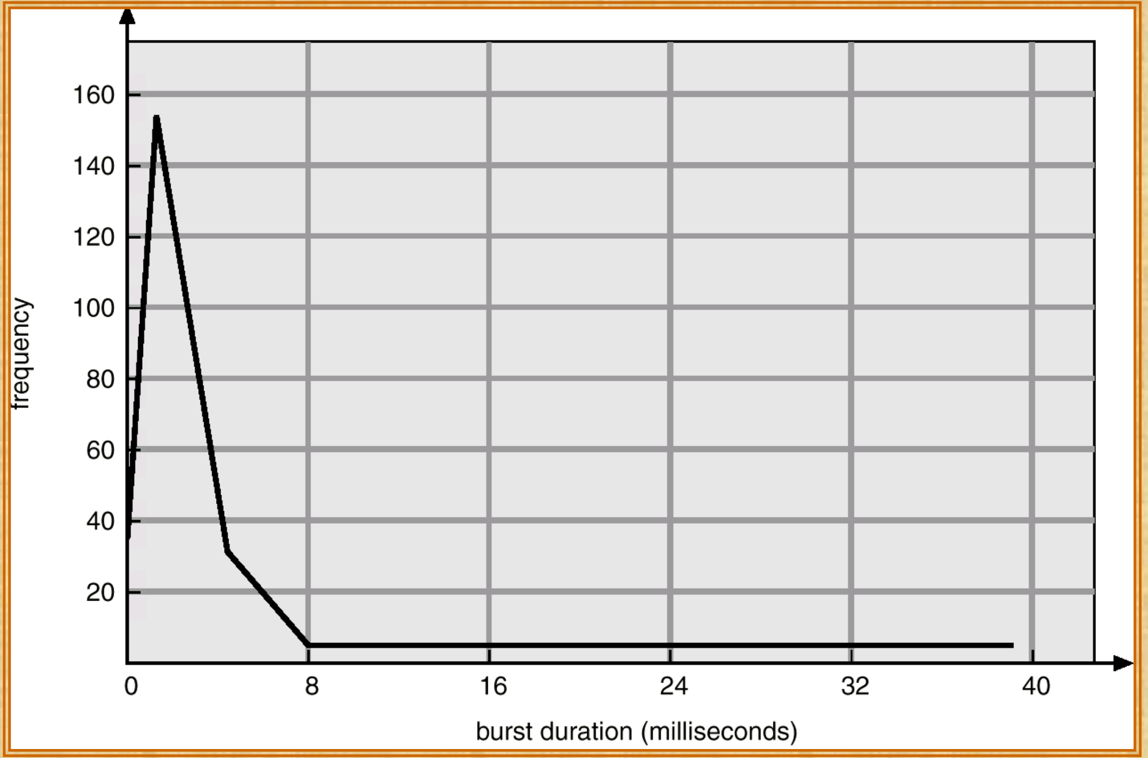
Alternating Sequence of CPU And I/O Bursts



Sequence of instructions of a sample process.

Histogram of CPU-burst Times

This diagram indicates that this process has large number of small CPU bursts of length less than 8 ms, relatively few long CPU bursts that are greater than 8 ms.



CPU Scheduler

- A.K.A. *short-term scheduler*.
- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

Question: Where does the system keep the processes that are ready to execute?

- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state (e.g., request I/O)
  2. Switches from running to ready state (e.g., time slice expires)
  3. Switches from waiting to ready (e.g., completed I/O)
  4. Terminates.

Preemptive Scheduling

- In **cooperative** or **non-preemptive** scheduling, when taking over the CPU, the process keeps it until the process either enters waiting state or terminates.

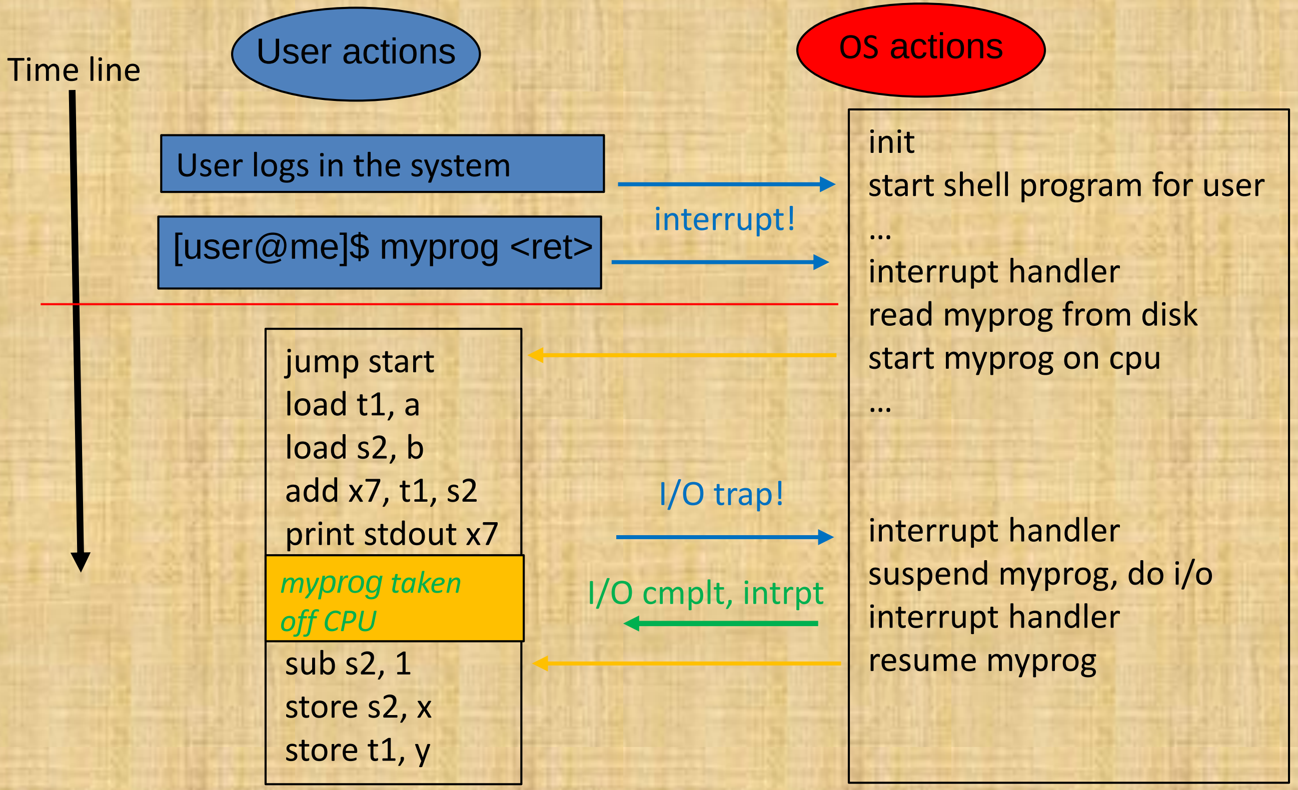
- In **preemptive scheduling**, a process holding the CPU may be forced to give up the CPU. Preemption causes context-switches, which introduce overhead. Preemption also calls for care of data shared with another process or kernel data structures when a process loses the CPU.

Dispatcher

- The **dispatcher** module in OS gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context,
  - switching to user mode,
  - jumping to the proper location in the selected process to restart that program.
- The **dispatch latency** is the time it takes for the dispatcher to stop one process and start another.

How Do They Work Together?

--- A Big Picture



Scheduling Criteria

These are **performance** metrics such as:

- **CPU utilization** – percentage of time the CPU is busy
- **Throughput** – the number of processes that complete their execution per time unit.
- **Turnaround time** – amount of time to complete a particular process, including waiting and execution.
- **Waiting time** – amount of time a process has been waiting in the ready queue.
- **Response time** – amount of time it takes from when a request was submitted until the first response is received, **not** output (for time-sharing environment).

These metrics may conflict with each other. It makes sense to look at averages of these metrics.

Optimizing Performance

- **Maximize** CPU utilization.
- **Maximize** throughput.
- **Minimize** turnaround time.
- **Minimize** waiting time.
- **Minimize** response time.