High Performance Computing
Lecture 18

Matthew Jacob

Indian Institute of Science

Preemptive Scheduling Policies

1. Round robin
   - Maintain a FCFS ReadyQ
   - When the currently running process is preempted, schedule the process from the front of the ReadyQ
   - Insert the previously running process at the end of the ReadyQ
   - This is much fairer than any of the non-preemptive scheduling policies

Preemptive Scheduling Policies

1. Round robin
2. Priority based
   - The readyQ need not be ordered on FCFS basis
   - It could be ordered on any other priority instead
   - For example: The process that has not run for the most time could get the highest priority
   - The scheduler could even assign a longer CPU timeslice for certain processes

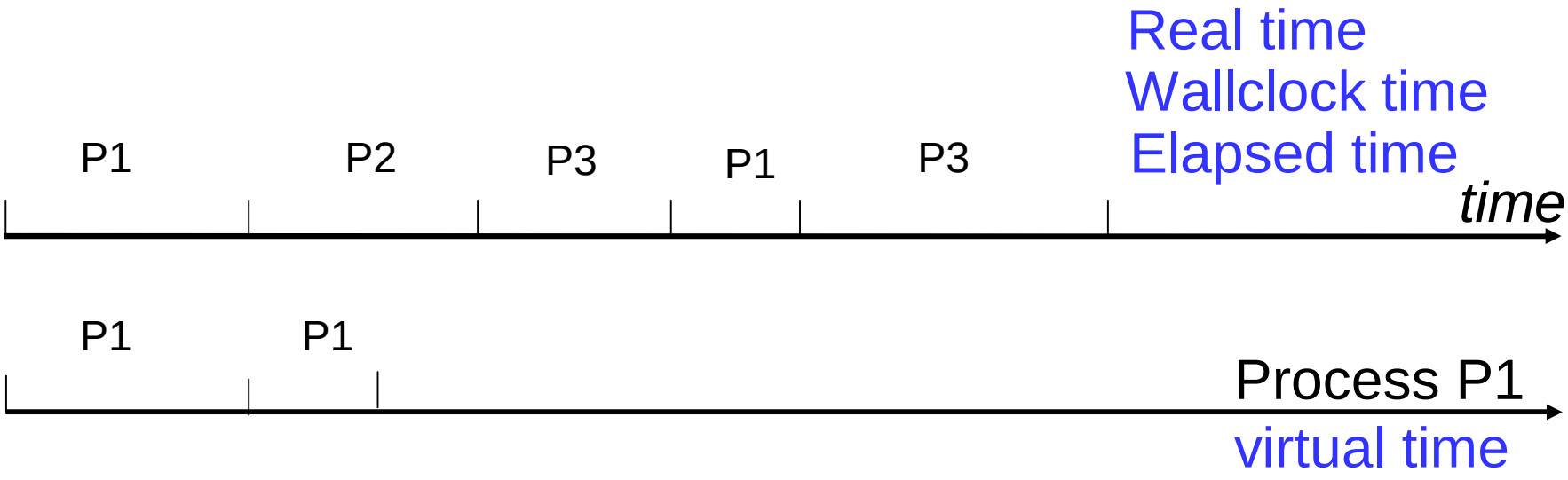
Example: Multilevel Feedback

- Used in some kinds of UNIX
- A Preemptive, Priority-based policy
- Multilevel: OS maintains one readyQ per priority level
  - It schedules the process from the front of the highest priority non-empty queue
- Feedback: Priorities are not fixed
  - A process could be moved to a lower/higher priority queue for fairness

Recall: Process Lifetime

- Process Lifetime: Time between fork() that created the process and exit() that causes its termination

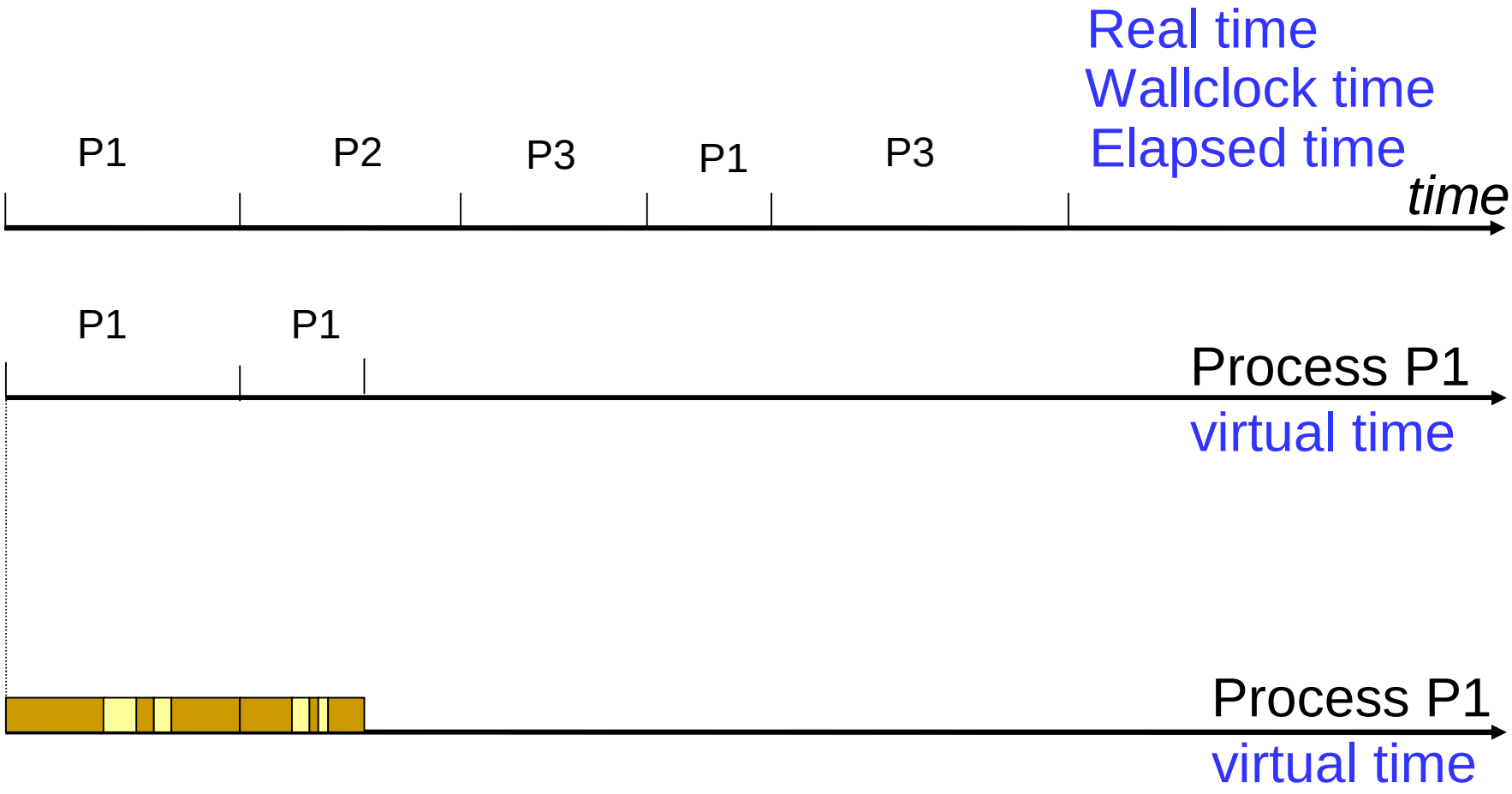
About Time


Real time
Wallclock time
Elapsed time
P1
P2
P3
P1
P3
time
P1
P1
Process P1
virtual time

Recall: Process Lifetime

- Process Lifetime: Time between fork() that created the process and exit() that causes its termination
- At any given point in time, a running process is executing either in user mode or in system mode
- Can find out the total CPU time used by a process, as well as CPU time in user mode, CPU time in system mode

Time: Process virtual and Elapsed



How is a Running Process Preempted?

- OS preemption code must run on the CPU
  - How does OS get control of CPU from running process to run its preemption code?
- Hardware timer interrupt
  - Hardware generated periodic event
  - When it occurs, hardware automatically transfers control to OS code (timer interrupt handler)
  - An interrupt is an example of a more general phenomenon called an exception

Exceptions

- Certain exceptional events that occur during program execution, handled by the processor HW
- There are two kinds of exceptions
  1. Traps: Synchronous, software generated
     - Page fault, Divide by zero, System call
  2. Interrupts: Asynchronous, hardware generated
     - Timer, keyboard, disk

What Happens on an Exception

1. Hardware
   - Saves processor state
   - Transfers control to corresponding piece of OS code, called the exception handler
2. Software (exception handler)
   - Takes care of the situation as appropriate
   - Ends with return from exception instruction
3. Hardware (execution of RFE instruction)
   - Restores the saved processor state
   - Transfers control back to the saved PC value