PRINCIPLES OF COMPUTER SYSTEMS DESIGN

CSE130

Winter 2020

CPU Scheduling IV

Baskin Engineering
UC SANTA CRUZ

Notices

- **Lab 2** due 23:59 **Sunday February 9**
- **Lab 1** Grades Released
  - Come see me if you got an unexpected grade
  - Take care copying lab installations
    - There's a hidden directory (.pintos) that needs be copied too
- **MIDTERM CANCELLED**
  - **Assignment 3** will now be written
  - Questions similar to final
  - Individual work, no conferring
  - Consider it an open-book midterm

Today's Lecture

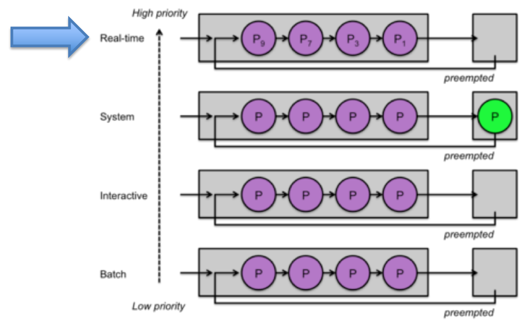
- Real-Time Scheduling
- CPU Scheduling Summary
- More Lab 2 Secret Sauce

Real-Time Scheduling

- **Hard Real-Time**: System guarantees to complete a critical task within a specified time period
- **Soft Real-Time**: Critical processes receive priority over less important ones, but no guarantees are given
- **Near Real-Time**: Marketing speak so manufactures don't get sued if their Hard Real-Time OS were ever to not quite deliver ☺
  - IBM's Transaction Processing Facility (TPF) for their zSeries mainframes is a good example https://en.wikipedia.org/wiki/Transaction_Processing_Facility
  - *"The depth of the CPU ready list is measured as any incoming transaction is received, and queued for the I-stream (processor) with the lowest demand, thus maintaining continuous load balancing among available processors."*

https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems

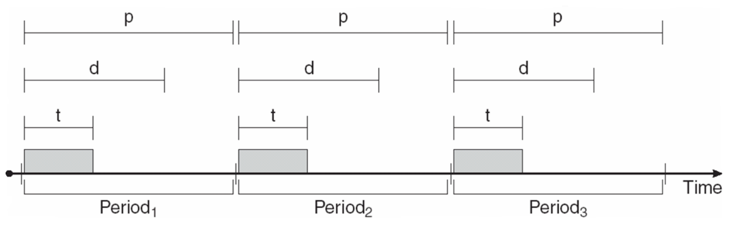
Soft Real-Time Scheduling



Only one CPU - Diagram is not the best ☹

Real-Time Scheduling

- Periodic processes require the CPU for discrete periods
  - **p** = duration of the **period**
  - **d** = **deadline** by when the process must be serviced
  - **t** = the processing **time**



- **In most cases p == d**

Can a Schedule be Found?

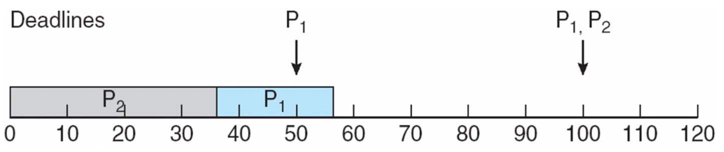
- If there are **m** periodic events and event **i** occurs with period **P_i** and requires **C_i** time units of CPU time to handle each event, then the load can be handled only if:

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

- **However**:
  - Satisfying this equation does ***not*** mean a schedule exists, it means one ***might*** exist if you can find a suitable scheduling algorithm
  - But ***failing*** to satisfy this equation absolutely means ***no schedule exists*** regardless of scheduling algorithm

Deadline Unaware Scheduling

- We have 2 processes:
  - **P_1** with $p_1 = 50$, $t_1 = 20$
  - **P_2** with $p_2 = 100$, $t_2 = 35$



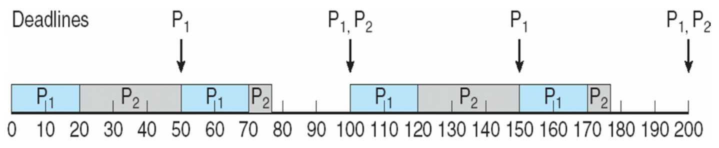
- If **P_2** runs before **P_1**, then **P_1** will miss its 1st deadline ☹
  - (assuming deadlines are at period ends)

Rate Monotonic

- **Monotonic**: A given order of process execution is preserved
- Processes assigned **priority** ***inversely*** based on their period
  - Shorter period = higher priority
  - Longer period = lower priority
- **Preemptive!**
  - If a long process does not finish within shorter processes period, longer process is kicked off the CPU
- Assumes:
  - **CPU bursts are consistent**
  - **A deadline hitting schedule can be found**

Deadline Aware Scheduling

- With **Rate Monotonic Scheduling**, **P_1** has a higher priority because its period is shorter, again:
  - **P_1** with **p_1** = 50, **t_1**= 20
  - **P_2** with **p_2** = 100, **t_2**= 35
- Now **P_1** will preempt **P_2** at **T** = 50



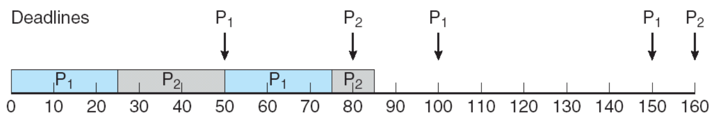
- And deadlines for both **P_1** and **P_2** are met ☺

Rate Monotonic

- **Considered optimal:**
  - If a set of processes cannot be scheduled by Rate Monotonic, the set cannot be scheduled by any other algorithm that assigns 'static' priorities
- As more processes need to be scheduled, attainable system utilisation falls

Missed Deadlines with Rate Monotonic

- If **P_1**'s processing time is increased, **t_1** = 25 (was 20)
- And **P_2**'s period is decreased, **p_1** = 80 (was 100)
- **P_1** still has a higher priority because its period is less than **P_2**



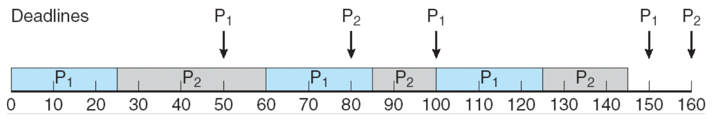
- **P_1** will preempt **P_2** at T = 50 and **P_2** will miss first deadline ☹
  - If **P_2** had not been preempted by **P_1**, both deadlines would be met
  - i.e. **RM** *unable to find a schedule*, even though CPU demand is <100%

Earliest Deadline First

- Priorities are assigned dynamically according to deadlines
  - Earlier deadline => higher priority
  - Later deadline => lower priority
  - **Equivalent to preemptive SJF**
- When put on ready queue, process states its deadline
  - Unlike SJF, we know the deadline, no guesses needed ☺
- The process with the earliest deadline is run first
- Dynamic, so can cope with:
  - **Non periodic processes**
  - **Variable processing times**
    - Processes can change their mind about their deadline each time they enter ready state (i.e. each time they get onto the Real-Time Ready Queue)

Earliest Deadline First: Example

- Same schedule that Rate Monotonic failed:
  - **P1** $\mathbf{p_1}$ = 50
  - $\mathbf{P_1}$ runs and finishes, $\mathbf{P_1}$'s first deadline met
  - $\mathbf{P_2}$ runs, $\mathbf{P_1}$ does not preempt as $\mathbf{P_2}$'s first deadline earlier than $\mathbf{P_1}$'s second deadline
  - $\mathbf{P_2}$ runs and finishes, $\mathbf{P_2}$'s first deadline met
  - and so on…

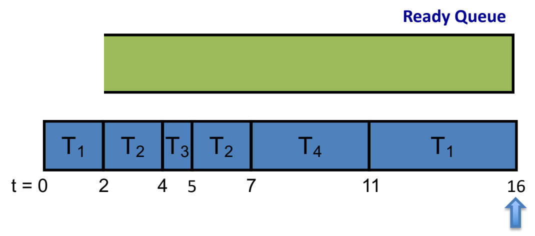


CPU Scheduling Summary

- **Efficient use of the CPU** (fundamentally a preemptible resource)
- **Scheduling Opportunities** & **Scheduler** vs. **Dispatcher**
- **Scheduling Criteria** (Utilisation, Throughput, **Turnaround Time**, **Waiting Time**, Response Time)
- **Scheduling Algorithms** (FCFS, RR, SJF, Priority Based, Preemptive & Non-preemptive)
- **Multi-level Queues** & **Multi-level Feedback Queues**
- Real World Example : **4.4 BSD Unix**
- **Priority Inversion** (the problem) **Priority Donation** (the solution)
- **Multi Core** and **Multi Processor** (L1/L2/L3 Caches, AMP, SMP, Push/Pull, Processor Affinity)
- **Real-Time Scheduling** (Types, Schedulability, RM, EDF)

Preemptive SJF

	burst$_1$	t_{arrive}
T_1	7	0
T_2	4	2
T_3	1	4
T_4	4	5



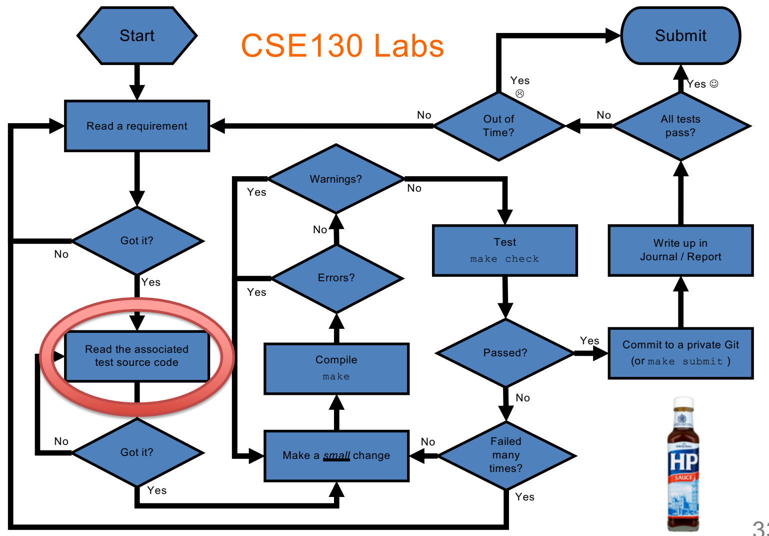
T_1 waiting time = (0-0) + (11-2) = 9
T_2 waiting time = (2-2) + (5-4) = 1
T_3 waiting time = 4-4 = 0
T_4 waiting time = 7-5 = 2

See Class Webcast for Animation

Mean Waiting Time = (9 + 1 + 0 + 2) / 4 = **3**

Mean Turnaround Time = ((16-0) + (7-2) + (5-4) + (11-5)) / 4 = **7**

CSE130 Labs



Lab 2 - Secret Sauce

- Read the tests line-by-line
  - How many threads are there?
  - What state(s) are they in?
  - Which one is on the CPU?
  - What lists are the others in?
- Draw Pictures
  - Use the whiteboards
- Discuss with classmates
  - But do ***NOT*** share code

Lab 2 - Secret Sauce

- Must deal with Priority *and* Preemption
- Pintos' Ready queue is a:
  - List of threads
- Semaphore waiting list is a:
  - List of threads
- Condition Variable waiting list is a:
  - List of semaphores
- Lock holder is a:
  - Thread
- Once you get into the "extreme" requirements
  - The TAs and I will offer only minimal guidance
  - Priority will be given to students stuck on earlier tests

Lab 2 - Secret Sauce

- General Approach
  - Only write code to pass the test you're working on right now
  - Once you've passed the test, `make submit` to "bank" that grade
- Common mistakes
  - `priority-sema`
    - Yielding before incrementing the semaphore value
  - `priority-condvar`
    - Trying to sort the waiting list as if it were a list of threads
  - `priority-donate-single`
    - How many locks can one thread hold?
    - Is the lock being released the one that triggered the donation?
  - All tests
    - Dealing with priority but not preemption

Next Lecture

- Introduction to Operating System Security