CPU Scheduling, Part II

CSD2180 Operating Systems
BSc in Computer Science (IMGD / RTIS)
Singapore Institute of Technology / DigiPen Institute of Technology
September 2021

Attendance Taking

https://forms.office.com/r/C1GhHu6Vnf

- Log in to your SIT account to submit the form
- You can only submit the form once
- The codeword is **microphone**

Timeline of Activities for Weeks 1 to 6

So far covered

- Operating system concepts
- Operating structure
- Processes
- CPU scheduling

Remaining topics

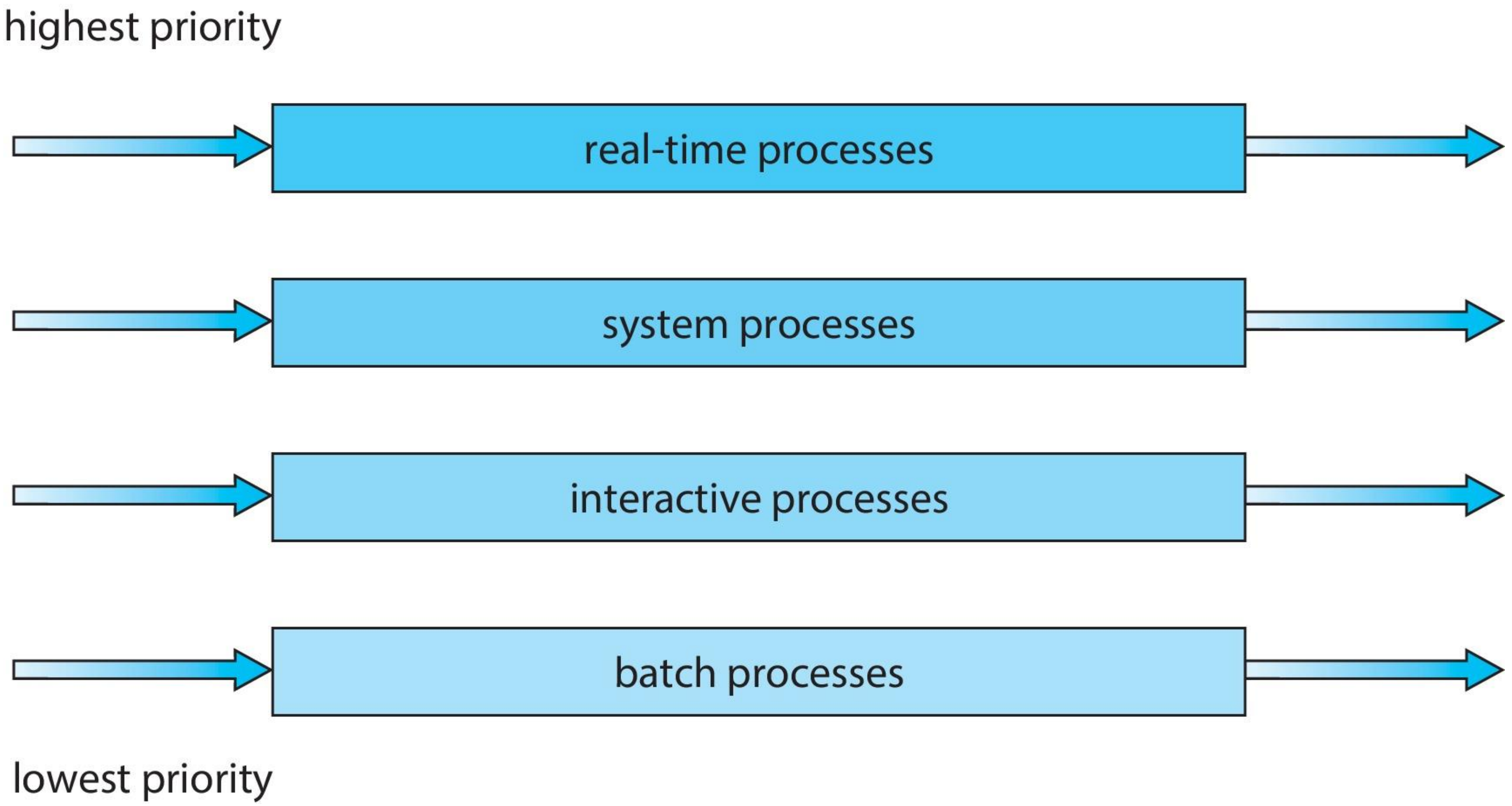
- Threads and concurrency
- Synchronization
- Deadlocks (maybe)

Activity timeline

- **26 / 09:** Release of Assignment One
- **26 / 09:** Release of practice question sets
- **13 / 10:** Midterms (on-campus)
- **31 / 10:** Deadline for Assignment One

More Advanced Scheduling Algorithms

"All Processes Are Equal, but Some Processes Are More Equal Than Others"

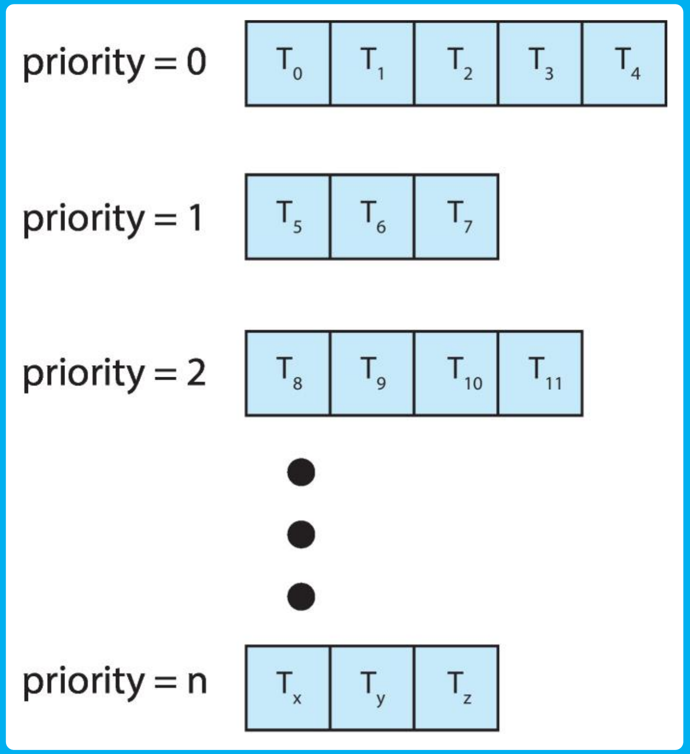


More Advanced Scheduling Algorithms

Possible to combine characteristics of different scheduling algorithms into one

Example: Priority scheduling with round robin

- Allow high-priority process to run first
- Prevent starvation for processes of the same priority level



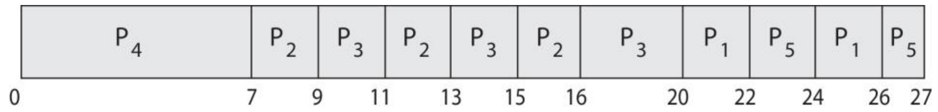
Priority Scheduling with Round Robin Example (Time Quantum = 2)

Process	Burst Time	Priority
P1	4	3
P2	5	2
P3	8	2
P4	7	1
P5	3	3

P4 runs first

Followed by P2 and P3

Then followed by P1 and P5



Average waiting time: $\frac{(22+11+12+24)}{5} = 13.8$

Real-Time CPU Scheduling

Real-Time CPU Scheduling

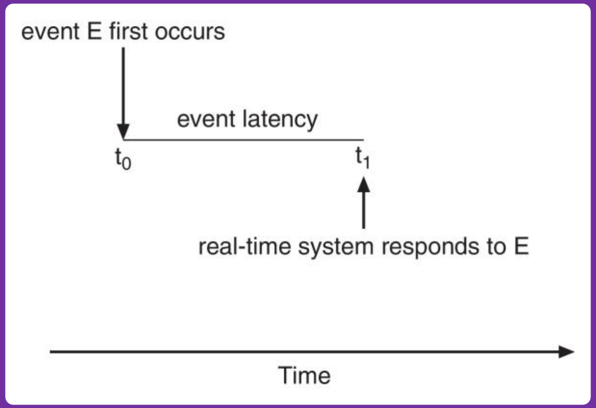
Real-time CPU scheduling can present obvious challenges

Soft real-time systems: Critical real-time tasks have the highest priority, but no guarantee as to when tasks will be scheduled

Hard real-time systems: Task must be serviced by its deadline; service after the deadline has expired is the same as no service at all

Event Latency

The amount of time that elapses from when an event occurs to when it is serviced



Two types of latencies affect performance

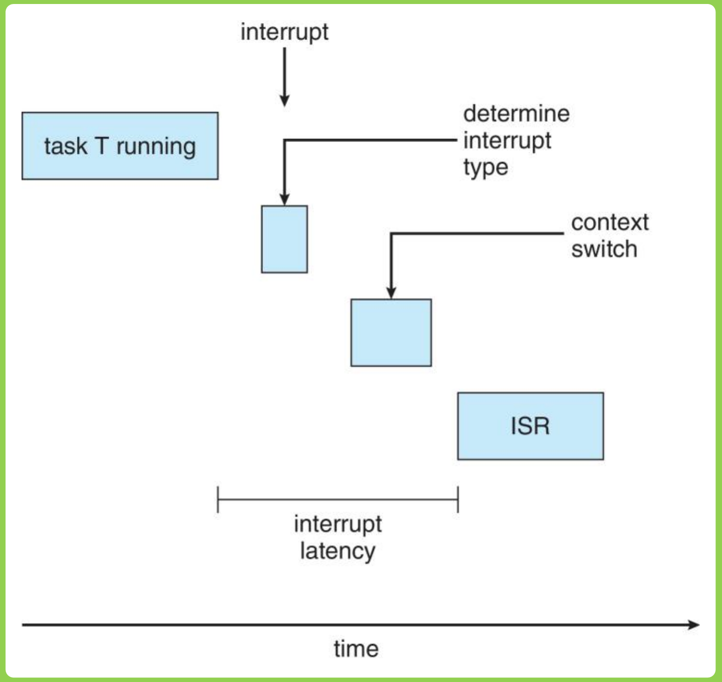
- **Interrupt latency:** Time from arrival of interrupt to start of routine that services the interrupt
- **Dispatch latency:** Time for dispatcher to take current process off CPU and switch to another

Interrupt Latency

Time from arrival of interrupt to start of routine that services the interrupt

Important to *minimize* interrupt latency to ensure that real-time tasks receive immediate attention

For hard real-time systems, interrupt latency must be **bounded to strict requirements of system**

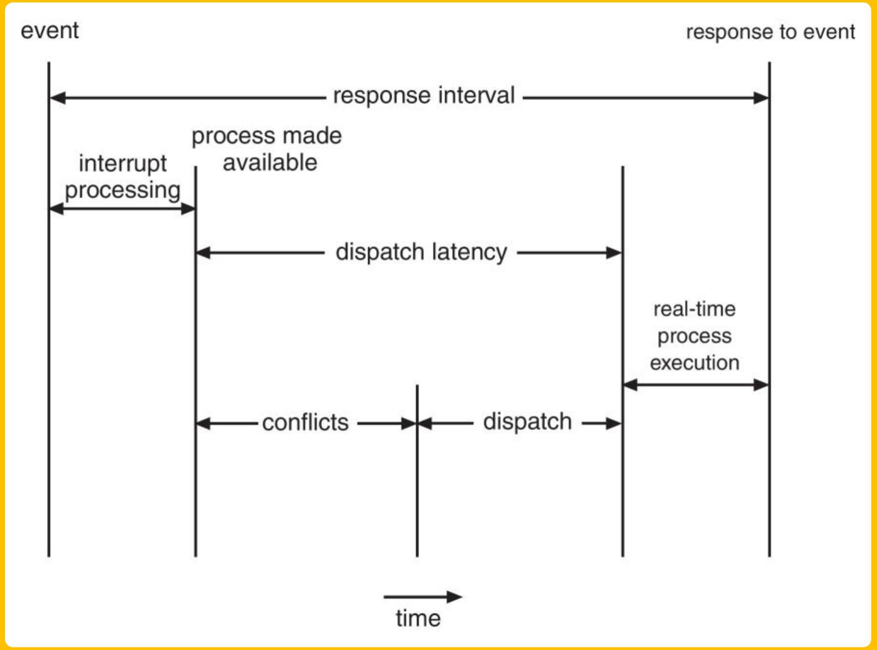


Dispatch Latency

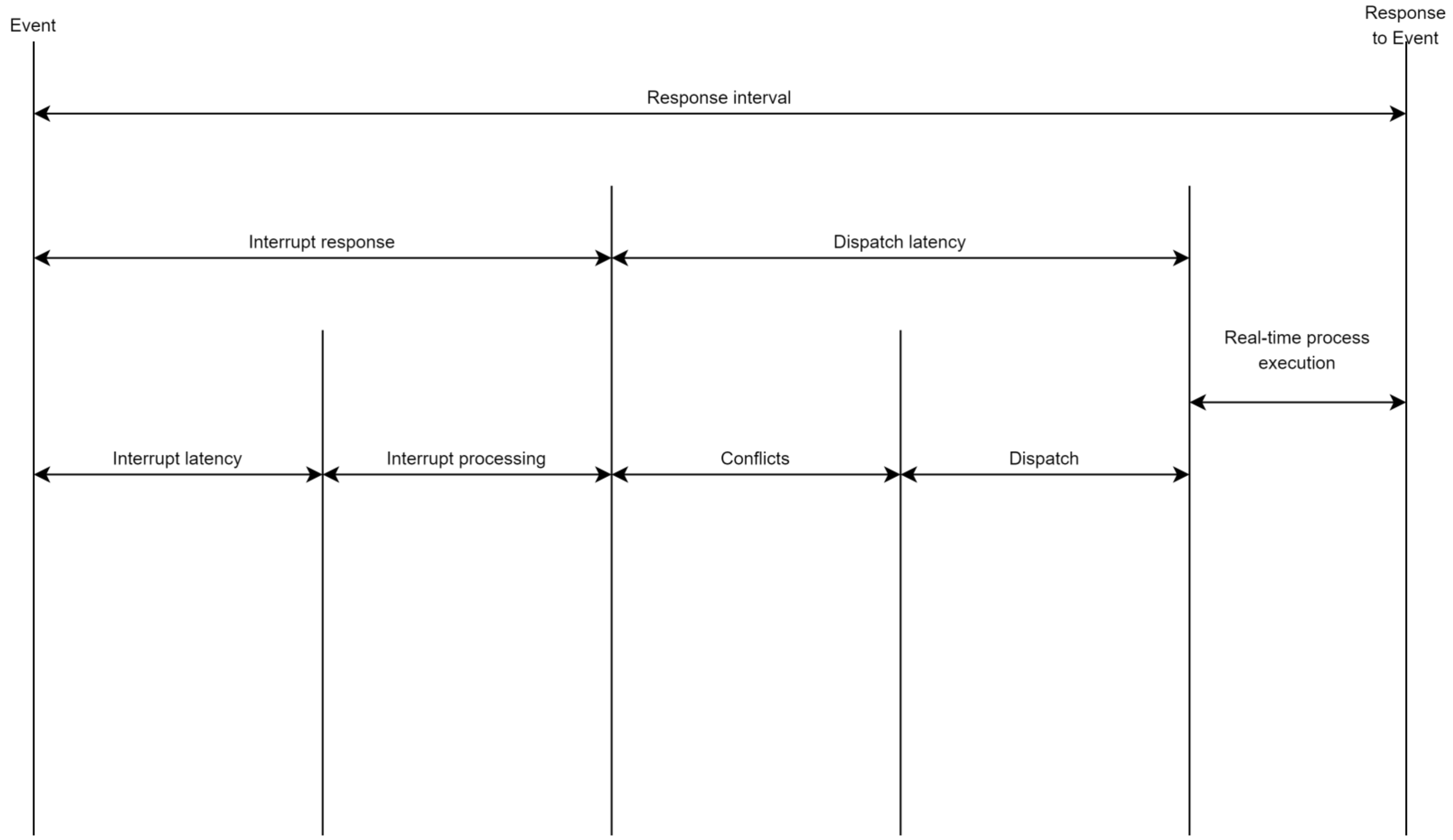
Time required for dispatcher to stop one process and start another

Conflict phase of dispatch latency

- Preemption of any process running in kernel mode
- Low-priority process releases resources needed by high-priority processes



Event Latency Revisited



Priority-Based Scheduling

Real-time operating system need to respond immediately to a real-time process as soon as that process requires the CPU

Scheduler must support a priority-based algorithm with preemption

- Priority-based algorithm assign each process a priority level based on its importance
- With preemption, a process currently running on the CPU will be preempted if a higher-priority process becomes available to run

However, providing a preemptive, priority-based scheduler only guarantees *soft real-time*

Hard real-time systems must further guarantee that real-time tasks will be serviced within their deadline

Making such hard real-time guarantees requires additional scheduling features

Periodic and Aperiodic Tasks

Real-time tasks that are repeated after a certain time interval is known as *periodic real-time tasks*

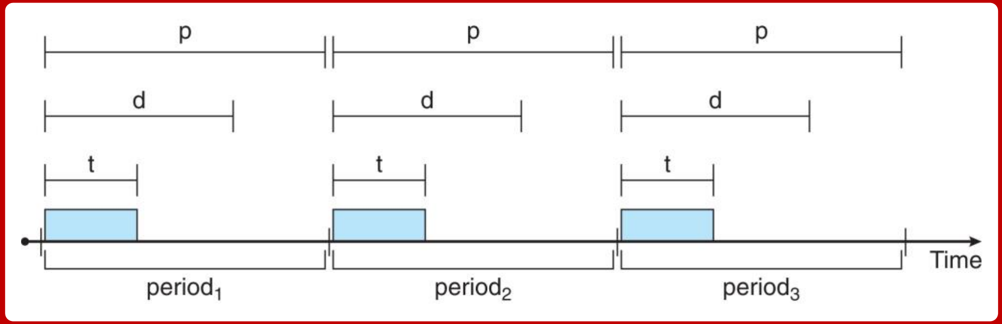
Real-time tasks that occur at any random time is known as *aperiodic real-time tasks*

Periodic Processes

Periodic processes require CPU at constant intervals

Has processing time t, deadline d, period p, where $0 \leq t \leq d \leq p$

Rate of periodic task is $1/p$



Rate Monotonic Scheduling

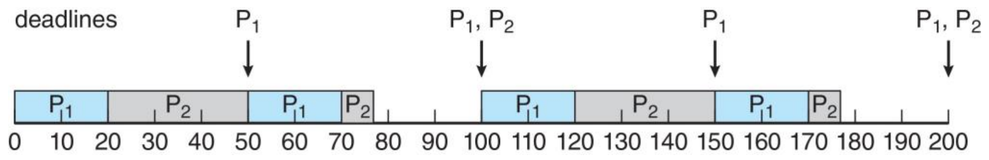
Rate-monotonic scheduling algorithm schedules periodic tasks using a static priority policy with preemption

If lower-priority process is running and a higher-priority process becomes available to run, preempt the lower-priority process

Each periodic task is assigned a priority inversely based on its period

- Shorter periods = higher priority
- Longer periods = lower priority

Rationale: Assign higher priority to tasks that require the CPU more often



Rate Monotonic Scheduling Example

Process	Period, p	Processing Time, t	Deadline, d
P1	50	20	50
P2	100	35	100

Can we meet the deadline?

Measure the CPU utilization of a process P, i.e., t/p

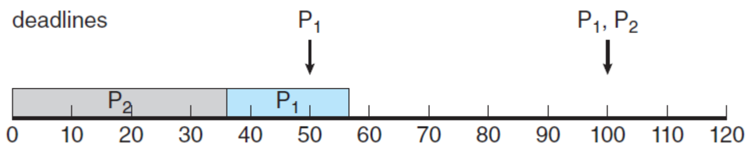
- **P1:** $20/50 = 0.40$
- **P2:** $35/100 = 0.35$
- **Total CPU utilization:** $0.40 + 0.35 = 0.75$

Seems that we can schedule these tasks ☺

Rate Monotonic Scheduling Example

Process	Period, p	Processing Time, t	Deadline, d
P1	50	20	50
P2	100	35	100

Let's give P2 a higher priority than P1



Rate Monotonic Scheduling Example

Process	Period, p	Processing Time, t	Deadline, d
P1	50	20	50
P2	100	35	100

Let's give P2 a higher priority than P1

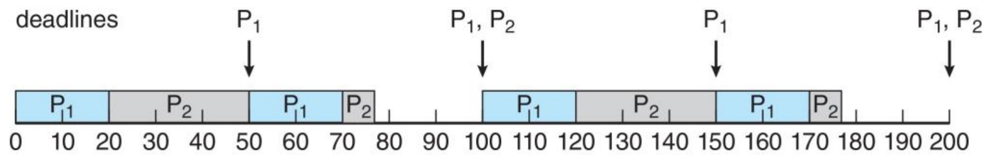
Process	Period, p	Processing Time, t	Deadline, d
P1	50	20	50
P2	100	35	100

P1 misses deadline ☹

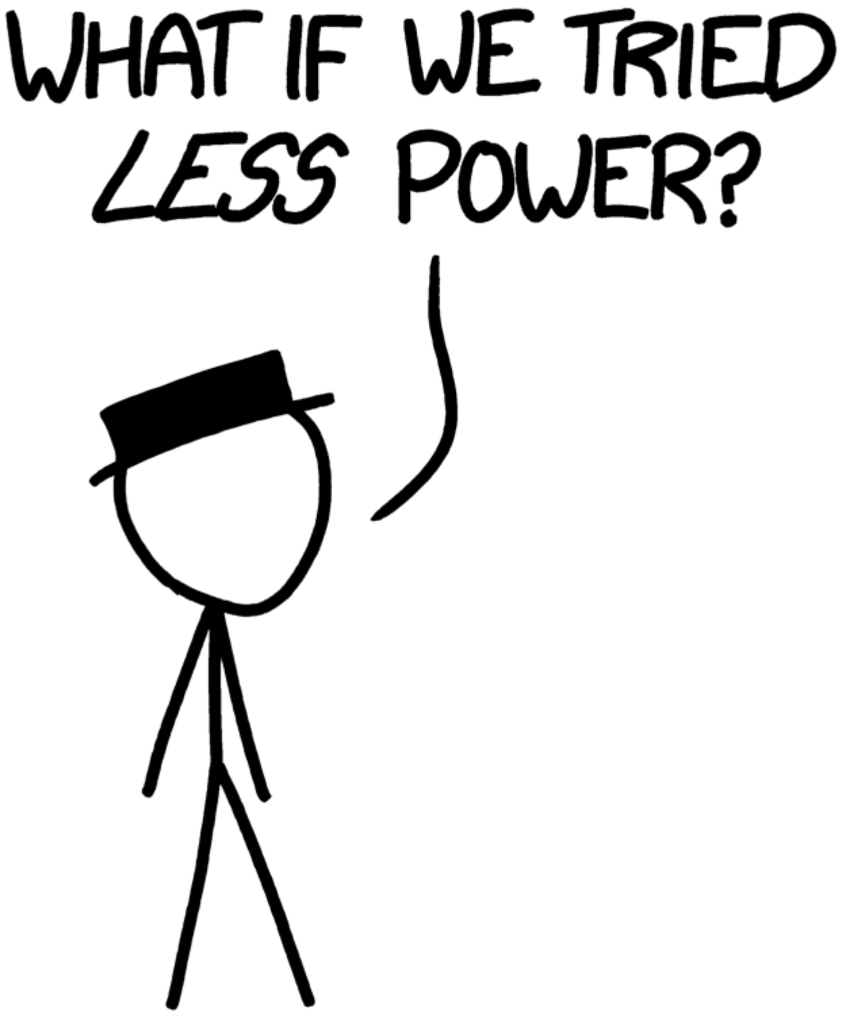
Rate Monotonic Scheduling Example

Process	Period, p	Processing Time, t	Deadline, d
P1	50	20	50
P2	100	35	100

Let's give P1 a higher priority than P2, in accordance to rate monotonic scheduling



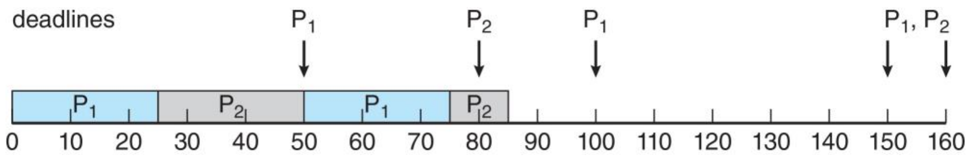
P1 and P2 meets the deadline ☺


WHAT IF WE TRIED LESS POWER?

Rate Monotonic Scheduling Limitations

Process	Period, p	Processing Time, t	Deadline, d
P1	50	25	50
P2	80	35	80

Rate-monotonic scheduling is considered optimal in that if a set of processes cannot be scheduled by this algorithm, it cannot be scheduled by any other algorithm that assigns static priorities



CPU utilization for process P, i.e., t/p

- **P1:** $25/50 = 0.50$
- **P2:** $35/80 = 0.4375$
- **Total CPU utilization:** $0.50 + 0.4375 = 0.9375$

Seems legit, but...

Rate Monotonic Scheduling Limitations

The worst-case CPU utilization for scheduling N processes is

$$N(2^{1/N} - 1)$$

1 process: $1(2 - 1) = 1.0 = 100\%$

2 process: $2\left(2^{1/2} - 1\right) = 0.82843 \approx 83\%$

3 process: $3\left(2^{1/3} - 1\right) = 0.77976 \approx 78\%$

…

In previous example, CPU utilization stands at around 93.75%, which is greater than 83%

Hence rate-monotonic scheduling cannot guarantee that the processes can be scheduled so that they meet their deadlines

Other Real-Time CPU Scheduling

Earliest Deadline First (EDF) Scheduling

EDF scheduling varies priorities dynamically, giving the highest priority to the process with the earliest deadline

Proportional Share Scheduling

Proportional share scheduling works by dividing the total amount of time available up into an equal number of shares, and then each process must request a certain share of the total when it tries to start

POSIX Real Time Scheduling

POSIX defines two scheduling classes for real time threads, **`SCHED_FIFO`** and **`SCHED_RR`**, depending on how threads of equal priority share time

- **`SCHED_FIFO`** schedules tasks in a first in first out order, with no time slicing among threads of equal priority
- **`SCHED_RR`** performs round robin time slicing among threads of equal priority

Algorithm Evaluation

How do we select a CPU-scheduling algorithm for a particular system?

Determine criteria, then evaluate algorithms...

So how do we evaluate CPU-scheduling algorithms?

Algorithm Evaluation

Deterministic modeling	Queuing models
Simulation	Implementation

Deterministic Modeling

- Takes a particular predetermined workload and defines the performance of each algorithm for that workload

- Type of analytic evaluation
- For each algorithm, calculate e.g., average waiting time, turnaround time, etc.
- Simple and fast, but requires exact numbers for input, and results apply only to those inputs

Deterministic Modeling

Process	Burst time
P1	10
P2	29
P3	3
P4	7
P5	12

Which algorithm is best?

Deterministic Modeling

Process	Burst time
P1	10
P2	29
P3	3
P4	7
P5	12

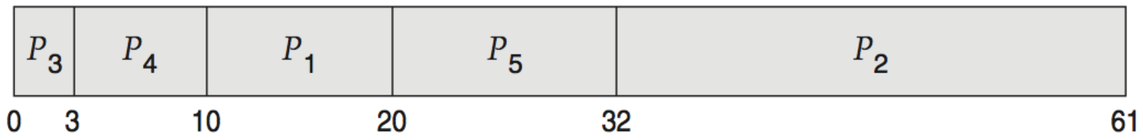
FCFS algorithm

Average waiting time: $\frac{0+10+39+42+49}{5} = 28$

Deterministic Modeling

Process	Burst time
P1	10
P2	29
P3	3
P4	7
P5	12

SJF algorithm

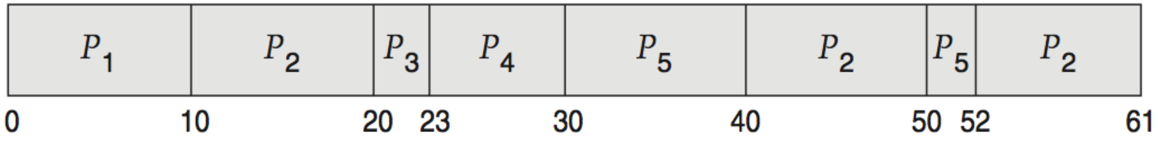


Average waiting time: $\frac{10+32+0+3+20}{5} = 13$

Deterministic Modeling

Process	Burst time
P1	10
P2	29
P3	3
P4	7
P5	12

RR algorithm, with time quantum = 10



Average waiting time: $\frac{0+32+20+23+40}{5} = 23$

Algorithm Evaluation

Simulations

- Run computer simulations of the different proposed algorithms under different load conditions, and to analyze the results to determine the "best" choice of operation for a particular load pattern

Queuing Models

- Specific process data is often not available, particularly for future times
- However, study of historical performance can often produce statistical descriptions of certain important parameters, e.g., arrival rate of new processes, ratio of CPU bursts to I/O times, distribution of CPU burst times and I/O burst times, etc.

Implementation

- The only real way to determine how a proposed scheduling algorithm is going to operate is to implement it on a real system
- Naturally, the most "expensive" approach

SINGAPORE INSTITUTE OF TECHNOLOGY

Questions?

Thank You!

Weihan.Goh {at} Singaporetech.edu.sg

https://www.singaporetech.edu.sg/directory/faculty/weihan-goh

https://sg.linkedin.com/in/weihan-goh