LASSONDE SCHOOL OF ENGINEERING | YORK UNIVERSITÉ UNIVERSITY

# EECS 3221:
# OPERATING SYSTEM FUNDAMENTALS

Hamzeh Khazaei

Department of Electrical Engineering and Computer Science

Week 5, Module 1:

**CPU Scheduling**

Assignment Project Exam Help

Feb 8, 2021



1

https://powcoder.com

Add WeChat powcoder

## Chapter 5: CPU Scheduling

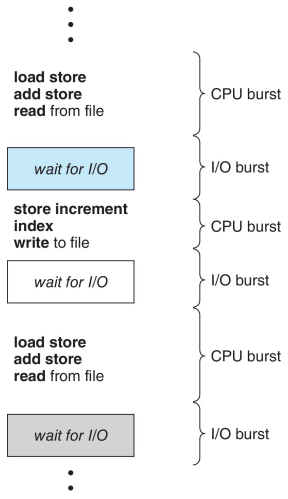
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multi-Processor Scheduling
- Real-Time CPU Scheduling
- Operating Systems Examples
- Algorithm Evaluation



2

## Basic Concepts

- We already know, kernel threads are the one being scheduled.
- However, the terms "process scheduling" and "thread scheduling" are often used interchangeably.
- By CPU we mainly refer to a CPU core, as the computation unit.
- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait
- **CPU burst** followed by **I/O burst**
- CPU burst distribution is of main concern




Assignment Project Exam Help



3

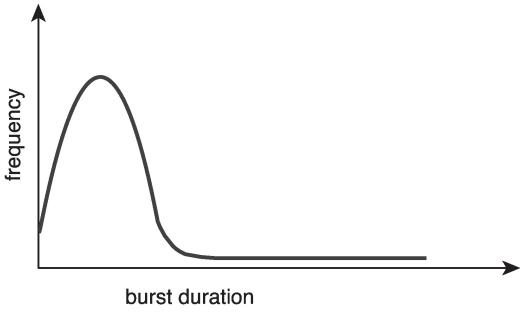
https://powcoder.com

Add WeChat powcoder

## Histogram of CPU-burst Times

Large number of short bursts

Small number of longer bursts






4

## CPU Scheduler

- The **CPU scheduler** selects from among the processes in ready queue, and allocates a CPU core to one of them
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state (I/O or child process)
  2. Switches from running to ready state (interrupts)
  3. Switches from waiting to ready (I/O completion or child finish)
  4. Terminates (last instructions)
- Scheduling under 1 and 4 is **nonpreemptive**
- All other scheduling is **preemptive** (all modern OSs use this)
  - Consider access to shared data (may cause race condition)
  - Consider preemption while in kernel mode (interrupting a system call)
  - Consider interrupts occurring during crucial OS activities

Assignment Project Exam Help



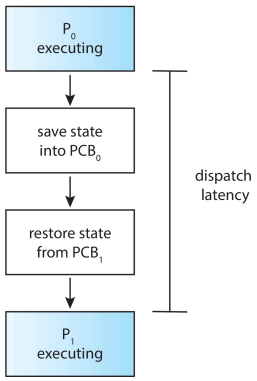
5

https://powcoder.com

Add WeChat powcoder

## Dispatcher

- Dispatcher module 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 user program to restart that program
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running






6

## Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – aggregated amount of time a process has been waiting in the *ready queue*
  - *This is the time that will be affected by scheduling algorithm.*
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)
  - Interactive systems

Assignment Project Exam Help



7

https://powcoder.com

Add WeChat powcoder

## Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

- Now let's investigate important CPU-Scheduling algorithms.
  - We assume we have only one CPU. (for now)



8

## First- Come, First-Served (FCFS) Scheduling

| Process | Burst Time |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

- Suppose that the processes arrive in the order: $P_1$ , $P_2$ , $P_3$
  The **Gantt Chart** for the schedule is:


- Waiting time for $P_1$ = 0; $P_2$ = 24; $P_3$ = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

- FCFS can be easily implemented with a FIFO queue.
- Q: what is it that we put into the ready queue?

Assignment Project Exam Help



9

https://powcoder.com

Add WeChat powcoder

## FCFS Scheduling (Cont.)

- A nonpreemptive algorithm
- Suppose that the processes arrive in the order:

  $P_2$ , $P_3$ , $P_1$

- The Gantt chart for the schedule is:


- Waiting time for $P_1$ = 6; $P_2$ = 0; $P_3$ = 3
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes
- FCFS is not good for interactive systems.



10

## Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
  - Shortest-next-CPU-burst to be more exact.
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

Assignment Project Exam Help



11

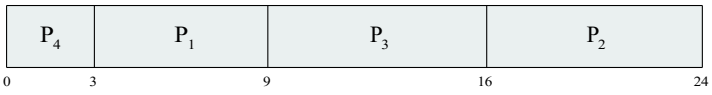
https://powcoder.com

Add WeChat powcoder

## Example of SJF

| Process | Burst Time |
|---|---|
| $P_1$ | 6 |
| $P_2$ | 8 |
| $P_3$ | 7 |
| $P_4$ | 3 |

- SJF scheduling chart




- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7



12

## Determining Length of Next CPU Burst

- Can only estimate the length – should be similar to the previous one
  - Then pick process with shortest predicted next CPU burst
- Can be done by using the length of previous CPU bursts, using **exponential averaging**

1. $t_n$ = actual length of $n^{th}$ CPU burst
2. $\tau_{n+1}$ = predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define :

- Commonly, α set to ½

Assignment Project Exam Help



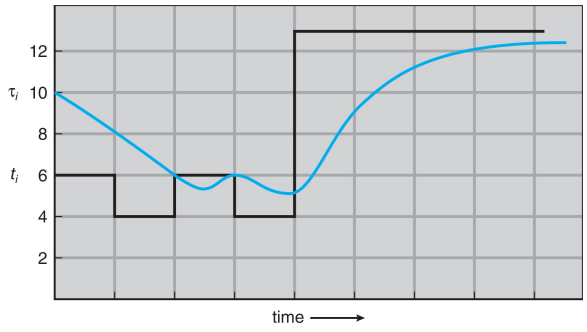
13

https://powcoder.com

Add WeChat powcoder

## Prediction of the Length of the Next CPU Burst

An exponential average with α =1/2 and $\tau_0$ = 10




| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| CPU burst ($t_i$) | 6 | 4 | 6 | 4 | 13 | 13 | 13 | ... |
| "guess" ($\tau_i$) | 10 | 8 | 6 | 6 | 5 | 9 | 11 | 12 | ... |



14

## Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = \alpha\, t_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get:

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\alpha\, t_{n-1} + \ldots + (1 - \alpha)^j \alpha\, t_{n-j} + \ldots + (1 - \alpha)^{n+1} \tau_0$$

- Since both $\alpha$ and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

Assignment Project Exam Help



15

https://powcoder.com

Add WeChat powcoder

## Preemptive vs non-preemptive

- The SJF algorithm can be either preemptive or nonpreemptive.
- The choice arises when a new process arrives at the ready queue while a previous process is still executing.
- The next CPU burst of the newly arrived process may be shorter than what is left of the currently executing process.
- A preemptive SJF algorithm will preempt the currently executing process, whereas a non-preemptive SJF algorithm will allow the currently running process to finish its CPU burst.
- Preemptive SJF scheduling is sometimes called shortest-remaining-time-first scheduling.



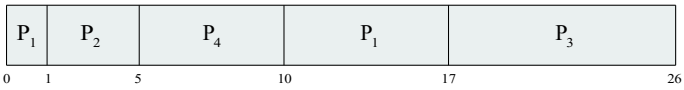
16

## Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis

| Process | *Arrival* Time | Burst Time |
|---|---|---|
| $P_1$ | 0 | 8 |
| $P_2$ | 1 | 4 |
| $P_3$ | 2 | 9 |
| $P_4$ | 3 | 5 |

- *Preemptive* SJF Gantt Chart




- Average waiting time = [(10-1)+(1-1)+(17-2)+(5-3)]/4 = 26/4 = 6.5 msec

Assignment Project Exam Help



17

https://powcoder.com

Add WeChat powcoder

## Round Robin (RR)

- Each process gets a small unit of CPU time (**time quantum** *q*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets 1/*n* of the CPU time in chunks of at most *q* time units at once. No process waits more than (*n*-1)*q* time units.
- Timer interrupts every quantum to schedule next process
- Performance
  - *q* large ⇒ FCFS
  - *q* small ⇒ *q* must be large with respect to context switch, otherwise overhead is too high
- The average waiting time under the RR policy is often long.

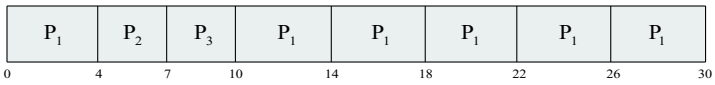


18

## Example of RR with Time Quantum = 4

| Process | Burst Time |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

- The Gantt chart is:




- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

Assignment Project Exam Help

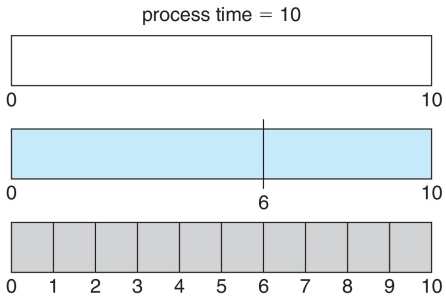


19

https://powcoder.com

Add WeChat powcoder

## Time Quantum and Context Switch Time

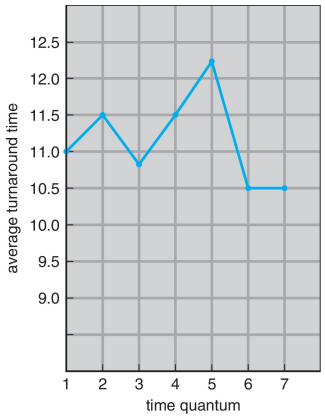



| quantum | context switches |
|---|---|
| 12 | 0 |
| 6 | 1 |
| 1 | 9 |



20

## Turnaround Time Varies With The Time Quantum




| process | time |
|---|---|
| $P_1$ | 6 |
| $P_2$ | 3 |
| $P_3$ | 1 |
| $P_4$ | 7 |

80% of CPU bursts
should be shorter than q

Assignment Project Exam Help



21

https://powcoder.com

Add WeChat powcoder

End of Part 1

## ANY Q?



22

## Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (biggest value ≡ highest priority)
  - Preemptive
  - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

Assignment Project Exam Help



23

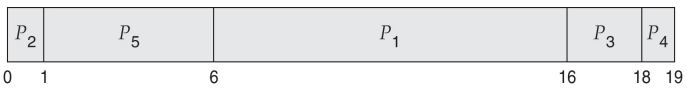
https://powcoder.com

Add WeChat powcoder

## Example of Priority Scheduling

| Process | Burst Time | Priority |
|---|---|---|
| $P_1$ | 10 | 3 |
| $P_2$ | 1 | 1 |
| $P_3$ | 2 | 4 |
| $P_4$ | 1 | 5 |
| $P_5$ | 5 | 2 |

- Priority scheduling Gantt Chart




- Average waiting time = 8.2 msec



24

## Priority Scheduling w/ Round-Robin

| Process | Burst Time | Priority |
|---|---|---|
| $P_1$ | 4 | 3 |
| $P_2$ | 5 | 2 |
| $P_3$ | 8 | 2 |
| $P_4$ | 7 | 1 |
| $P_5$ | 3 | 3 |

- Run the process with the highest priority. Processes with the same priority run round-robin
- Gantt Chart wit 2 ms time quantum


Assignment Project Exam Help



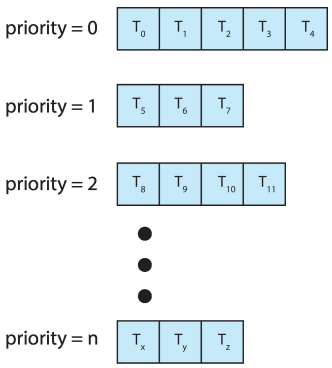
25

https://powcoder.com

Add WeChat powcoder

## Multilevel Queue ?

- With priority scheduling, have separate queues for each priority.
- Schedule the process in the highest-priority queue!



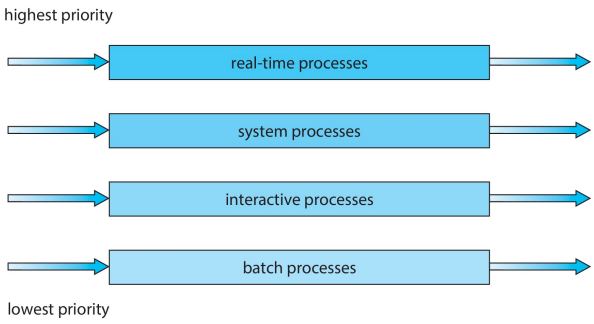

Any idea how we can implement aging? To avoid starvation?



26

## Multilevel Queue

- Prioritization based upon process type




Assignment Project Exam Help



27

https://powcoder.com

Add WeChat powcoder

## Multilevel Feedback Queue

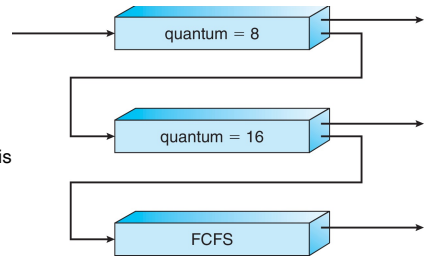
- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service



28

## Example of Multilevel Feedback Queue

- Three queues:
  - $Q_0$ – RR with time quantum 8 milliseconds
  - $Q_1$ – RR time quantum 16 milliseconds
  - $Q_2$ – FCFS
- Scheduling
  - A new job enters queue $Q_0$
    - When it gains CPU, job receives 8 milliseconds
    - If it does not finish in 8 milliseconds, job is moved to queue $Q_1$
  - At $Q_1$ receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue $Q_2$
  - A process that arrives from $Q_1$ will preempt a process in $Q_2$. A process in $Q_1$ will in turn be preempted by a process arriving for $Q_0$.




Assignment Project Exam Help



29

https://powcoder.com

Add WeChat powcoder

## Multiple-Processor Scheduling

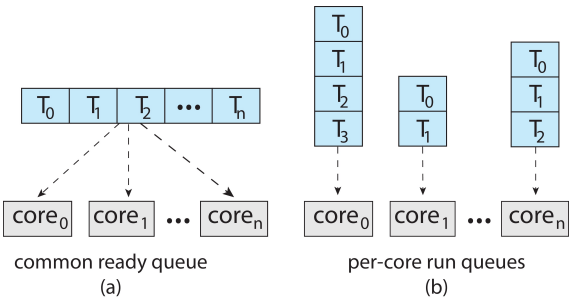
- CPU scheduling more complex when multiple CPUs are available
- Multiprocess may be any one of the following architectures:
  - Multicore CPUs
  - Multithreaded cores
  - NUMA systems
  - Heterogeneous multiprocessing



30

## Multiple-Processor Scheduling

- Symmetric multiprocessing (SMP) is where each processor is self scheduling.
- All threads may be in a common ready queue (a)
- Each processor may have its own private queue of threads (b)




Assignment Project Exam Help



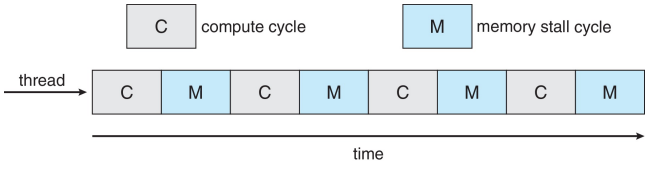
31

https://powcoder.com

Add WeChat powcoder

## Multicore Processors?

- Recent trend to place multiple processor cores on same physical chip
- Faster and consumes less power
- Multiple threads per core also growing
  - Takes advantage of memory stall to make progress on another thread while memory retrieve happens



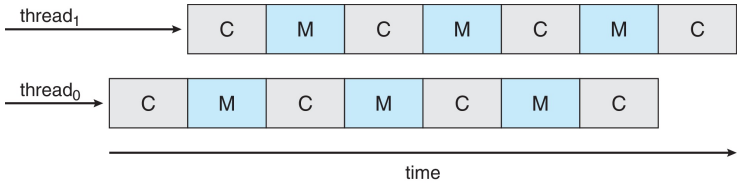



32

## Multithreaded Multicore System

Each core has > 1 hardware threads.

If one thread has a memory stall, switch to another thread!




Assignment Project Exam Help



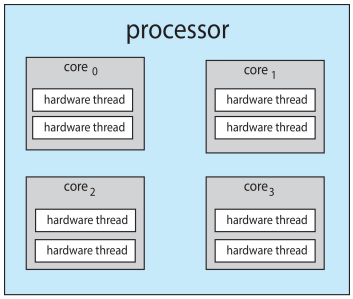
33

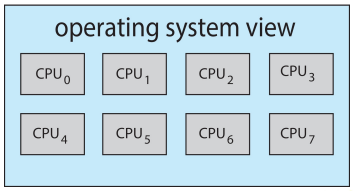
https://powcoder.com

Add WeChat powcoder

## Multithreaded Multicore System

- **Chip-multithreading** (CMT) assigns each core multiple hardware threads. (Intel refers to this as **hyperthreading**.)
- On a quad-core system with 2 hardware threads per core, the operating system sees 8 logical processors.



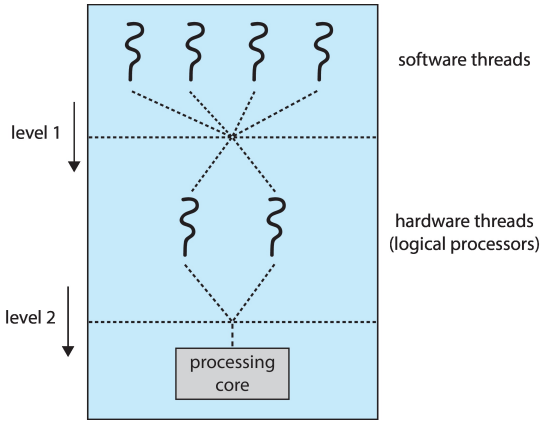



34

## Multithreaded Multicore System

- Two levels of scheduling:

1. The operating system deciding which software thread to run on a logical CPU
2. How each core decides which hardware thread to run on the physical core.




Assignment Project Exam Help



35

https://powcoder.com

Add WeChat powcoder

## Multiple-Processor Scheduling – Load Balancing

- If SMP, need to keep all CPUs loaded for efficiency
- **Load balancing** attempts to keep workload evenly distributed
- **Push migration** – periodic task checks load on each processor, and if found pushes task from overloaded CPU to other CPUs
- **Pull migration** – idle processors pulls waiting task from busy processor



36

## Multiple-Processor Scheduling – Processor Affinity

- When a thread has been running on one processor, the cache contents of that processor stores the memory accesses by that thread.
- We refer to this as a thread having affinity for a processor (i.e. “processor affinity”)
- Load balancing may affect processor affinity as a thread may be moved from one processor to another to balance loads, yet that thread loses the contents of what it had in the cache of the processor it was moved off.
- **Soft affinity** – the operating system attempts to keep a thread running on the same processor, but no guarantees.
- **Hard affinity** – allows a process to specify a set of processors it may run on.

Assignment Project Exam Help



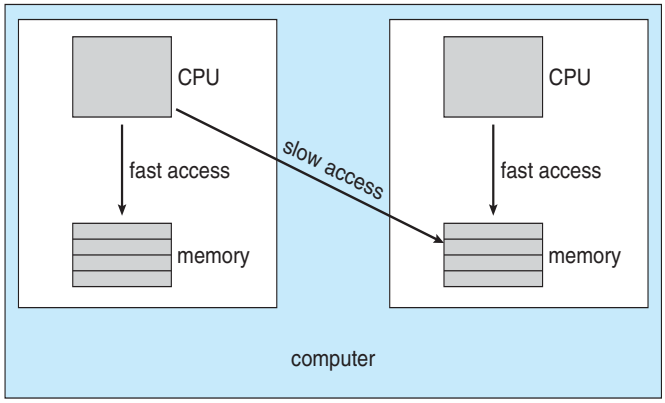
37

https://powcoder.com

Add WeChat powcoder

## NUMA and CPU Scheduling

If the operating system is **NUMA-aware**, it will assign memory closes to the CPU the thread is running on.






38

## 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

Assignment Project Exam Help



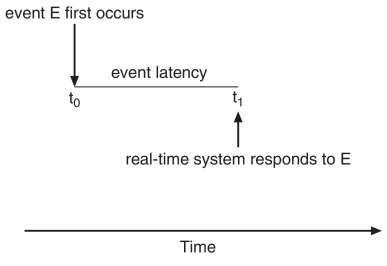
39

https://powcoder.com

Add WeChat powcoder

## Real-Time CPU Scheduling

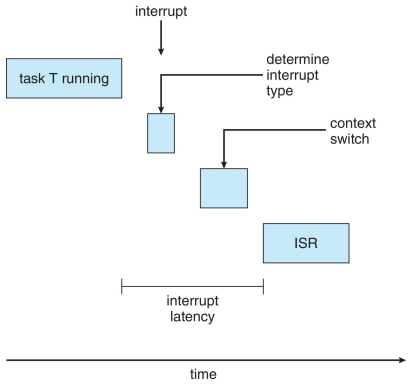
- **Event latency** – the amount of time that elapses from when an event occurs to when it is serviced.
- Two types of latencies affect performance
  1. **Interrupt latency** – time from arrival of interrupt to start of routine that services interrupt
  2. **Dispatch latency** – time for schedule to take current process off CPU and switch to another






40

## Interrupt Latency




Assignment Project Exam Help



41

https://powcoder.com

Add WeChat powcoder

End of Part 2.

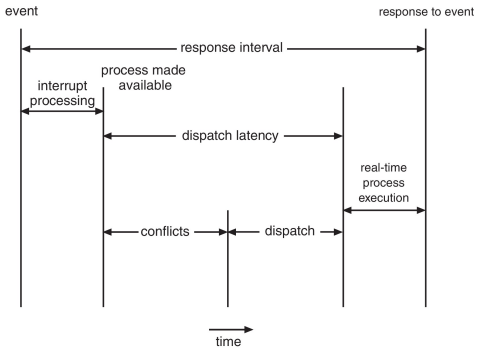
**ANY Q?**



42

## Dispatch Latency

- Conflict phase of dispatch latency:
  1. Preemption of any process running in kernel mode
  2. Release by low-priority process of resources needed by high-priority processes




Assignment Project Exam Help



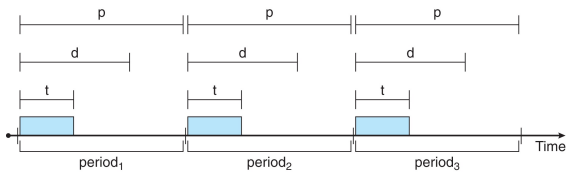
43

https://powcoder.com

Add WeChat powcoder

## Priority-based Scheduling

- For real-time scheduling, scheduler must support preemptive, priority-based scheduling
  - But only guarantees soft real-time
- For hard real-time must also provide ability to meet deadlines
- Processes have new characteristics: **periodic** ones require CPU at constant intervals
  - Has processing time *t*, deadline *d*, period *p*
  - $0 \le t \le d \le p$
  - **Rate** (aka **frequency**) of periodic task is $1/p$
  - **Utilization (u)** $= t/p$

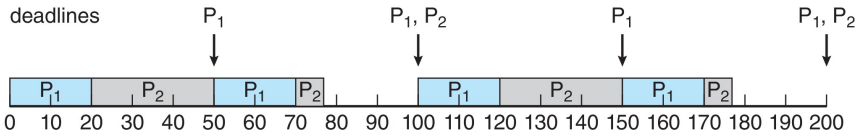





44

## Rate Monotonic Scheduling (RMS)

- A priority is assigned based on the inverse of its period
- Shorter periods = higher priority;
- Longer periods = lower priority
- $P_1$ is assigned a higher priority than $P_2$.
- $P_1$=50, $t_1$=20, $P_2$=100, $t_2$=35
- $u_1$=20/40 = 0.4, $u_2$=35/100 = 0.35 → $u_t$ = 0.75




Assignment Project Exam Help



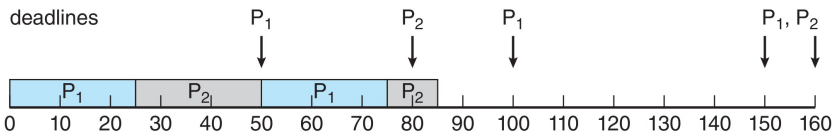
45

https://powcoder.com

Add WeChat powcoder

## Missed Deadlines with Rate Monotonic Scheduling

- $P_1$=50, $t_1$=25, $P_2$=80, $t_2$=35
- $u_1$=25/50 = 0.5, $u_2$=35/80 = 0.44 → **$u_t$ = 0.94**
- Process P2 misses finishing its deadline at time 80




- Max utilization when having N processes in the system:

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

- With two processes, CPU utilization is bounded at about 83%.
- With one process in the system, CPU utilization is 100%, but it falls to approximately 69% as the number of processes approaches infinity.

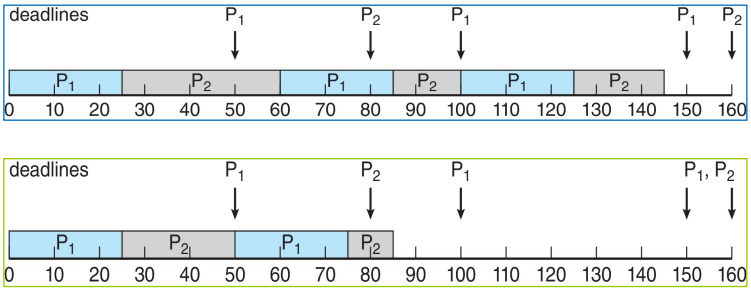


46

## Earliest Deadline First Scheduling (EDF)

- Priorities are assigned according to deadlines:

  the earlier the deadline, the higher the priority;
  the later the deadline, the lower the priority







Rate Monotonic Scheduling

Assignment Project Exam Help



47

https://powcoder.com

Add WeChat powcoder

## POSIX Real-Time Scheduling

- The POSIX.1b standard
- API provides functions for managing real-time threads
- Defines two scheduling classes for real-time threads:
  1. SCHED_FIFO - threads are scheduled using a FCFS strategy with a FIFO queue. There is no time-slicing for threads of equal priority
  2. SCHED_RR - similar to SCHED_FIFO except time-slicing occurs for threads of equal priority
- Defines two functions for getting and setting scheduling policy:

1. `pthread_attr_getsched_policy(pthread_attr_t *attr, int *policy)`
2. `pthread_attr_setsched_policy(pthread_attr_t *attr, int policy)`



49

## POSIX Real-Time Scheduling API

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5
int main(int argc, char *argv[])
{
   int i, policy;
   pthread_t tid[NUM_THREADS];
   pthread_attr_t attr;
   /* get the default attributes */
   pthread_attr_init(&attr);
   /* get the current scheduling policy */
   if (pthread_attr_getschedpolicy(&attr, &policy) != 0)
      fprintf(stderr, "Unable to get policy.\n");
   else {
      if (policy == SCHED_OTHER) printf("SCHED_OTHER\n");
      else if (policy == SCHED_RR) printf("SCHED_RR\n");
      else if (policy == SCHED_FIFO) printf("SCHED_FIFO\n");
   }
```

Assignment Project Exam Help



50

https://powcoder.com

Add WeChat powcoder

## POSIX Real-Time Scheduling API (Cont.)

```
   /* set the scheduling policy - FIFO, RR, or OTHER */
   if (pthread_attr_setschedpolicy(&attr, SCHED_FIFO) != 0)
      fprintf(stderr, "Unable to set policy.\n");
   /* create the threads */
   for (i = 0; i < NUM_THREADS; i++)
      pthread_create(&tid[i],&attr,runner,NULL);
   /* now join on each thread */
   for (i = 0; i < NUM_THREADS; i++)
      pthread_join(tid[i], NULL);
}

/* Each thread will begin control in this function */
void *runner(void *param)
{
   /* do some work ... */
   pthread_exit(0);
}
```



51

## Operating System Examples

- Linux scheduling
- Windows scheduling (not included in exams)
- Solaris scheduling (not included in exams)

Assignment Project Exam Help



52

https://powcoder.com

Add WeChat powcoder

## Linux Scheduling Through Version 2.5

- Prior to kernel version 2.5, ran variation of standard UNIX scheduling algorithm
- Version 2.5 moved to constant order ***O*(1)** scheduling time
  - Preemptive, priority based
  - Two priority ranges: time-sharing and real-time
  - **Real-time** range from 0 to 99 and normal range **nice** values from 100 to 140
  - Map into global priority with numerically lower values indicating higher priority
  - Higher priority gets larger q (quantum or time slice)
  - Task run-able as long as time left in time slice (**active**)
  - If no time left (**expired**), not run-able until all other tasks use their slices
  - Worked well, but poor response times for interactive processes



53

## Linux Scheduling in Version 2.6.23 +

- ***Completely Fair Scheduler*** (CFS)
- **Scheduling classes**
  - Each has specific priority
  - Scheduler picks highest priority task in highest scheduling class
  - Rather than quantum based on fixed time allotments, based on proportion of CPU time
  - 2 scheduling classes included, others can be added
    1. default
    2. real-time
- Quantum calculated based on **nice value** from -20 to +19
  - Lower value is higher priority
  - Calculates **target latency** – interval of time during which task should run at least once
  - Target latency can increase if say number of active tasks increases
- CFS scheduler maintains per task **virtual run time** in variable **`vruntime`**
  - Associated with decay factor based on priority of task – lower priority is higher decay rate
  - Normal default priority yields virtual run time = actual run time
- To decide next task to run, scheduler picks task with lowest virtual run time

Assignment Project Exam Help



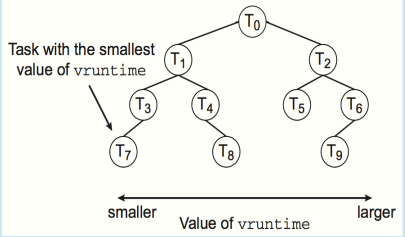
54

https://powcoder.com

Add WeChat powcoder

## CFS Performance

The Linux CFS scheduler provides an efficient algorithm for selecting which task to run next. Each runnable task is placed in a red-black tree—a balanced binary search tree whose key is based on the value of `vruntime`. This tree is shown below:



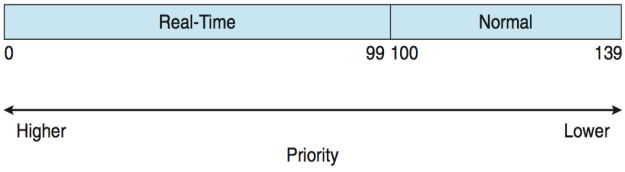

When a task becomes runnable, it is added to the tree. If a task on the tree is not runnable (for example, if it is blocked while waiting for I/O), it is removed. Generally speaking, tasks that have been given less processing time (smaller values of `vruntime`) are toward the left side of the tree, and tasks that have been given more processing time are on the right side. According to the properties of a binary search tree, the leftmost node has the smallest key value, which for the sake of the CFS scheduler means that it is the task with the highest priority. Because the red-black tree is balanced, navigating it to discover the leftmost node will require $O(\lg N)$ operations (where $N$ is the number of nodes in the tree). However, for efficiency reasons, the Linux scheduler caches this value in the variable `rb_leftmost`, and thus determining which task to run next requires only retrieving the cached value.



55

## Linux Scheduling (Cont.)

- Real-time scheduling according to POSIX.1b
  - Real-time tasks have static priorities
- Real-time plus normal map into global priority scheme
- Nice value of -20 maps to global priority 100
- Nice value of +19 maps to priority 139




Assignment Project Exam Help



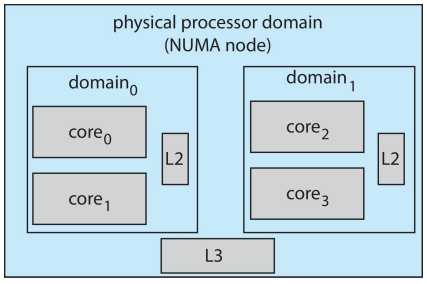
56

https://powcoder.com

Add WeChat powcoder

## Linux Scheduling (Cont.)

- Linux supports load balancing, but is also NUMA-aware.
- **Scheduling domain** is a set of CPU cores that can be balanced against one another.
- Domains are organized by what they share (i.e. cache memory.) Goal is to keep threads from migrating between domains.






57

# Algorithm Evaluation

- How to select CPU-scheduling algorithm for an OS?
- Determine criteria, then evaluate algorithms
- **Deterministic modeling**
  - Type of **analytic evaluation**
  - Takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Consider 5 processes arriving at time 0:

| Process | Burst Time |
|---|---|
| $P_1$ | 10 |
| $P_2$ | 29 |
| $P_3$ | 3 |
| $P_4$ | 7 |
| $P_5$ | 12 |

Assignment Project Exam Help



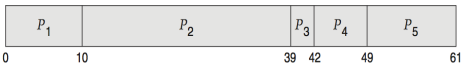
58

https://powcoder.com

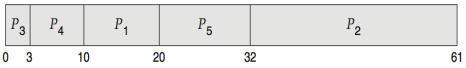
Add WeChat powcoder

# Deterministic Evaluation

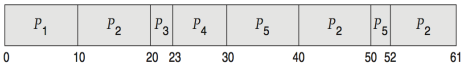
- For each algorithm, calculate minimum average waiting time
- Simple and fast, but requires exact numbers for input, applies only to those inputs
  - FCFS is 28ms:




  - Non-preemptive SFJ is 13ms:




  - RR is 23ms:






59

## Queueing Models

- Describes the arrival of processes, and CPU and I/O bursts probabilistically
  - Commonly exponential, and described by mean
  - Computes average throughput, utilization, waiting time, etc
- Computer system described as network of servers, each with queue of waiting processes
  - Knowing arrival rates and service rates
  - Computes utilization, average queue length, average wait time, etc


Assignment Project Exam Help



60

https://powcoder.com

Add WeChat powcoder

## Little's Law

- $n$ = average queue length
- $W$ = average waiting time in queue
- $\lambda$ = average arrival rate into queue
- Little's law – in steady state, processes leaving queue must equal processes arriving, thus:
  $n = \lambda \times W$
  - Valid for any scheduling algorithm and arrival distribution
- For example, if on average 7 processes arrive per second, and normally 14 processes in queue, then average wait time per process = 2 seconds



61

## Simulations

- Queueing models limited
- **Simulations** more doable:
  - Programmed model of computer system
  - Clock is a variable
  - Gather statistics indicating algorithm performance
  - Data to drive simulation gathered via
    - Random number generator according to probabilities
    - Distributions defined mathematically or empirically
    - Trace files record sequences of real events in real systems

Assignment Project Exam Help

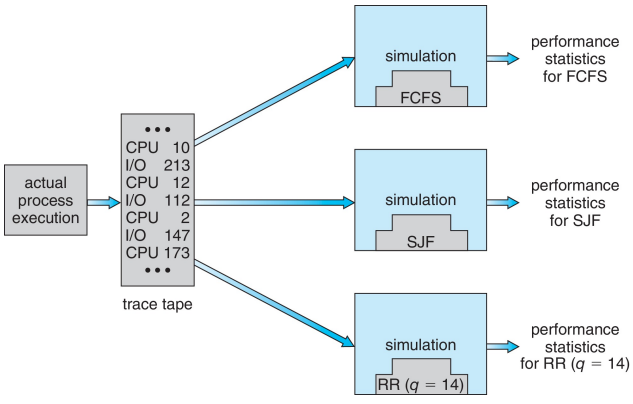


62

https://powcoder.com

Add WeChat powcoder

## Evaluation of CPU Schedulers by Simulation






63

## Real Implementation

- Even simulations have limited accuracy
- Just implement new scheduler and test in real systems
  - High cost, high risk
  - Environments vary
- Most flexible schedulers can be modified per-site or per-system
- Or APIs to modify priorities
- But again environments vary

Assignment Project Exam Help



64

https://powcoder.com

Add WeChat powcoder

End.

**ANY Q?**



65