Processor Schedule

GU/Jianhua
School of Computer Science
NorthWestern Polytechnical University
谷建华
西北工业大学计算机学院

Processes Schedule

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Real-Time Scheduling

Basic Concepts (1)

- **The objective of multiprogramming is to maximum CPU utilization**
- **For a uni-processor system, there will never be more than one running process.**
- **CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait.**
- **When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process.**

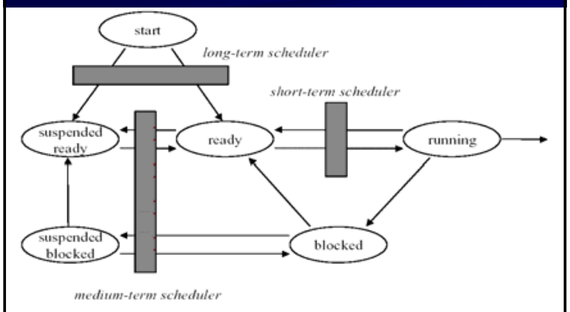
Basic Concepts (2)

- *I/O-bound program*
  - Spends more of its time doing I/O than spends doing computations
  - Example
- *CPU-bound program*
  - Uses more of its time doing computation than an I/O-bound process uses
  - Example

Schedulers

- *Long-term scheduler(Job scheduler)*
  - Selects processes(job) from job pool and loads them into memory for execution.
- *Short-term scheduler*
  - Selects from among the processes that ready to execute, and allocates the CPU to one of them.
- *Medium-term scheduler*
  - The process is swapped out, and later swapped in by Medium-term scheduler.

Schedulers (2)

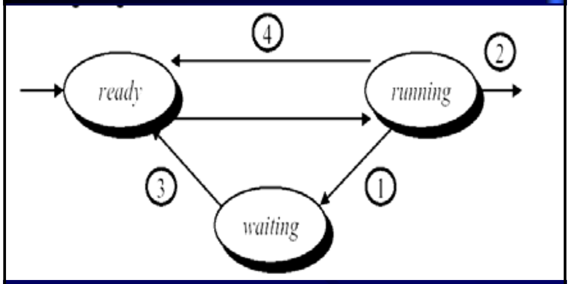


*Relationship among three of these schedulers

CPU Scheduler (1)

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state.
  2. Terminates.
  3. Switches from waiting to ready.
  4. Switches from running to ready state
- Scheduling under 1 and 2 is *non-preemptive*.
- All other scheduling is *preemptive.*

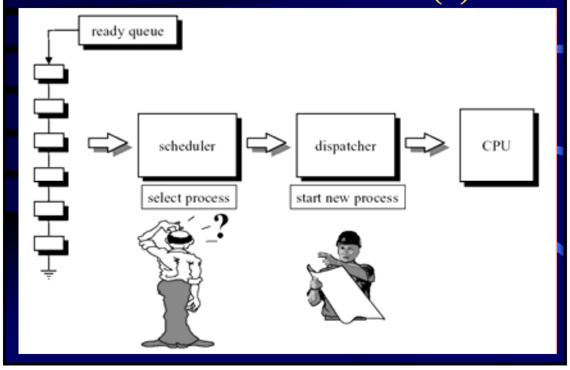
CPU Scheduler (2)



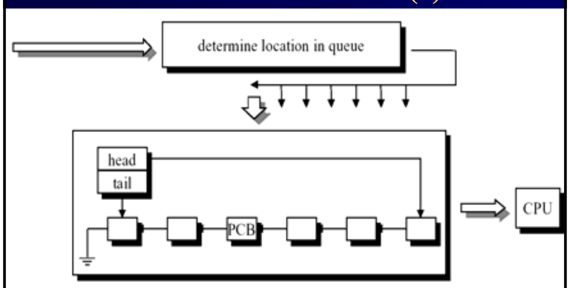
Dispatcher

- 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.

Structure of a Scheduler (1)



Structure of a Scheduler (2)



- Incoming process is put into right location in ready queue.
- Dispatcher always picks first element in ready queue.

Scheduling Criteria

- *System oriented:*
  - ***CPU utilization****:* keep the CPU as busy as possible
  - ***Throughput:*** a number of processes that complete their execution per time unit
- *User oriented:*
  - ***Turnaround time:*** amount of time to execute a particular process
  - ***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 produced

Other Criteria

- ***Deadline***: When process completion deadlines can be specified, the scheduling discipline should subordinate other goals to that of maximizing the percentage of deadlines met.
- ***Predictability***: A given job should run in about same amount of time and at about the same cost regardless of load on the system.
- ***Fairness***: Process should be treated the same, and no process should suffer from starvation.
- ***Balancing Resource:*** The scheduling policy should keep the resources of the system busy.

Optimization Criteria

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

- ***Note:*** maximum/minimum values *vs.* average values *vs.* variance(方差）

Scheduling Algorithms

- **FCFS** : First-come-first-served
- **SPN:** Shortest Process Next(SJN)
- **SRT**: Shortest Remaining Time
- **Priority Scheduling**
- **RR** : Round-Robin
- Multilevel feedback queue scheduling
- *Multiprocessor scheduling*

Schedule Decision Mode

- ***Decision Mode***: specifies the instants in time at which the selection function is exercised.
- There are Two general categories:
  - ***Non-preemptive:*** once a process is in the Running state, it continues to execute until it terminates or blocks itself to wait for I/O or by requesting some operating system service.
  - ***Preemptive:*** The currently running process may be interrupted and moved to Ready state by OS when there is one more appropriate process becomes Ready.

Non-preemptive & Preemptive

- **Non-preemptive**:
  - Simple
  - Lower overhead
- **Preemptive**:
  - More complicated
  - More overhead
  - Provide better service to the total population of processes

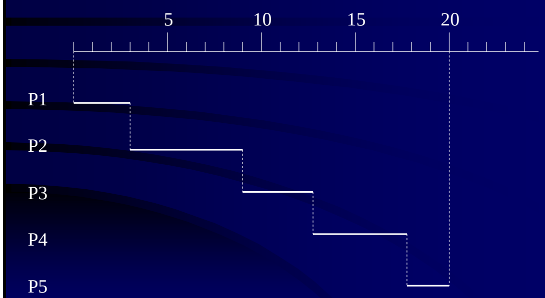
FCFS : First-come-first-served (1)

- As each process becomes ready, it joins the ready queue. When the currently running process ceases to execute, the oldest process in the ready queue is selected for running – *according to the order which the process enter the ready queue.*
- Also named FIFO(First-In-First-Out)
- ***Advantages***:
  - very simple
- ***Disadvantages***:
  - long average and worst-case waiting times
  - poor dynamic behavior

FCFS : First-come-first-served(2)

- FCFS is not an attractive alternative on its own for a single-processor system.
- However, FCFS is often combined with other schedule algorithms to provide an effective scheduler.
- ***Implementation***: Using the queue.

例：先来先服务调度算法



FCFS Scheduling

- Example:

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:

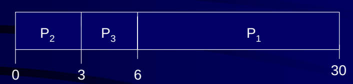
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order

P_2 , P_3 , P_1 .

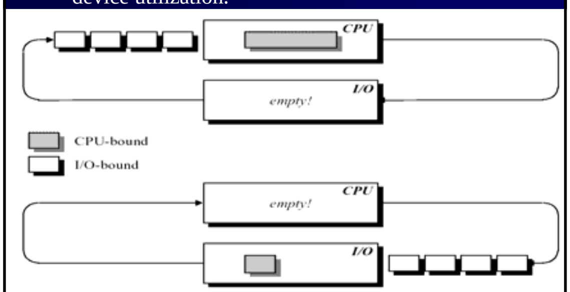
- The Gantt chart for the schedule is:



- 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

Convoy Effects

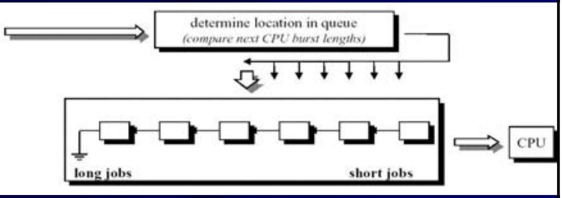
- All the other processes wait for the one big process to get off the CPU. This results in lower CPU and device utilization.



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.
- ***Two schemes***:
  - **Non-preemptive** – once CPU given to the process it cannot be preempted until completes its CPU burst.
  - **Preemptive** – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF).
- SJF is optimal – gives minimum average turnaround time for a given set of processes.

Shortest-Job-First (SJF) Scheduling



- Whenever CPU is idle, picks process with ***shortest next CPU burst.***
- **Advantages:** minimizes average waiting times.
- **Problem**: How to determine length of ***next*** CPU burst?
- **Problem**: Starvation of jobs with long CPU bursts.

Example of SJF

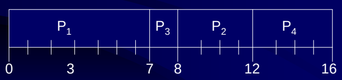
Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- Non-Preemptive
- Preemptive
- Please give the Waiting Time of each process

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

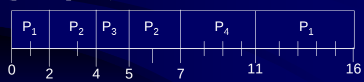


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)



- Average waiting time = (9 + 1 + 0 +2)/4 = 3

Determining Length of Next CPU Burst

- Can only estimate the length(user give).
- Can be done by using the length of previous
- CPU bursts, using exponential averaging.

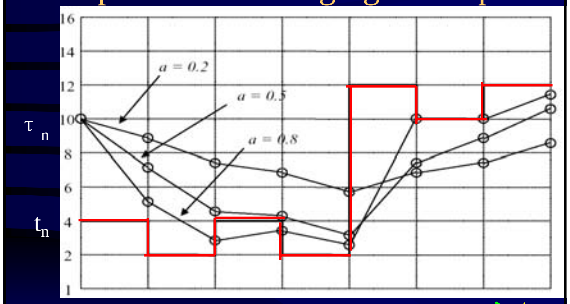
1. t_n = actual lenght of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define : $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$.

Examples of Exponential Averaging

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

  $\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\, \alpha\, t_{n-1} + \dots + (1 - \alpha)^j \alpha\, t_{n-j} + \dots + (1 - \alpha)^{n+1} \tau_0$
- Since both α and (1 - α) are less than or equal to 1, each successive term has less weight than its predecessor.

Exponential Averaging: Example



$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n ; \tau_0 = 10 ; t_0 = 4;$$

Example

进程	创建时刻	运行时间（毫秒）	优先数
P1	0	3	3
P2	2	6	5
P3	4	4	1
P4	6	5	2
P5	8	2	4

Features of SJN

- 可能造成长作业的饿死。
- 没有考虑进程的重要程度。
- 作业的运行时间不易给出，一般由用户给出，但不甚合理。

Highest Response Ratio Next

- Select the highest ***Response Ratio***

RR = (w+s) /s = w/s + 1

Where

w = time spent waiting for the processor

s = expected service time (running time)

Priority Scheduling

- Non-preemptive Priority Scheduling
- Preemptive Priority Scheduling
- Static Priority Scheduling
- Dynamic Priority Scheduling

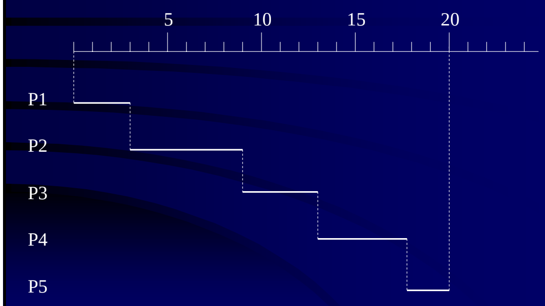
Non-preemptive Priority Scheduling (1)

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority
- 优先级法的基本思想是：系统为每个进程设置一个优先数（对应一个优先级），把所有的就绪进程按优先级从大到小排序，调度时从就绪队列中选择优先级最高的进程投入运行，仅当占用CPU的进程运行结束或因某种原因不能继续运行时，系统才进行重新调度 。

Non-preemptive Priority Scheduling (2)

- ***Lower overhead:*** the number of switching from process to another is lower
- ***Problems:***
  - *Starvation:* low priority processes may never execute.
    - Solution: *Aging* – as time progresses increase the priority of the process.
  - *Priority inversion:* The higher-priority process which is ready has to wait for a lower-priority process to finish.
    - Solution: *priority-inheritance protocol* – lower-priority processes inherit the priority of higher-priority process when they are accessing the resource which higher-priority process needs until they are done with the resource.

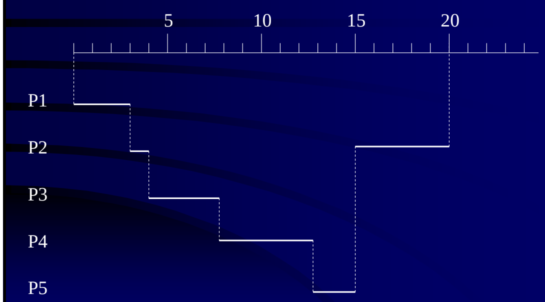
例：非剥夺式优先级调度算法



Preemptive Priority Scheduling (1)

- 系统为每个进程设置一个优先数（对应一个优先级），把所有的就绪进程按优先级从大到小排序，调度时从就绪队列中选择优先级最高的进程投入运行，当系统中有另一优先级更高的进程变成就绪态时，系统应立即剥夺现运行进程占用处理机的权力，使该优先级更高的进程投入运行。
- 反映了进程的优先级特征，使系统能及时处理紧急事件，但系统开销较大。

例：剥夺式优先级调度算法



优先级分组法

- 保留非剥夺式优先级和剥夺式优先级各自的优点，克服其缺点。
- 方法：组间可剥夺，组内不可剥夺（组内相同优先级则按FCFS处理）

Static Priority Scheduling

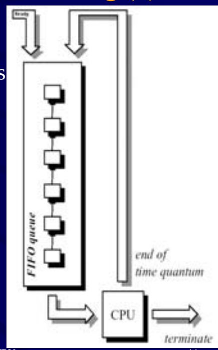
- 按进程的重要程度给每个进程分配一个优先级，该优先级在进程的整个运行过程中不再改变。该方法中优先级的设定是一次性的，故一定要慎重、合理。
- 一般用于系统中每个进程的重要程度事先知道的系统。
- 简单

Dynamic Priority Scheduling

- Process Priority Change
- How to change ??
- Example: to increase the priority of process as the length of waiting time of process

Round Robin (RR) Scheduling (1)

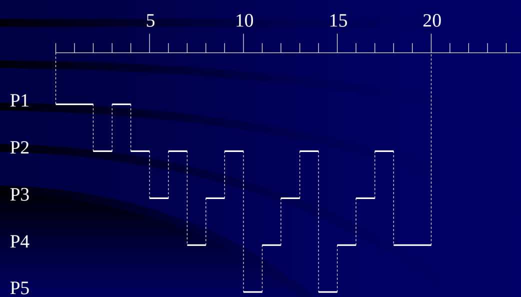
- Basic thought: Each process gets a small unit of CPU time (*time quantum* or *time slice*). After this time has elapsed, the process is preempted and added to the end of the ready queue.
- Implement: We keep the ready queue as a FIFO queue of processes. New processes are added to the tail of ready queue. CPU scheduler picks the first process from the ready queue、, sets a timer to interrupt after 1 time slice.



Round Robin (RR) Scheduling (2)

- Round Robin scheduling is designed especially for time sharing system.
- Categories:
  - Static RR
  - Dynamic RR

例：时间片轮转调度算法



Example: RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 37 57 77 97 117 121 134 154 162

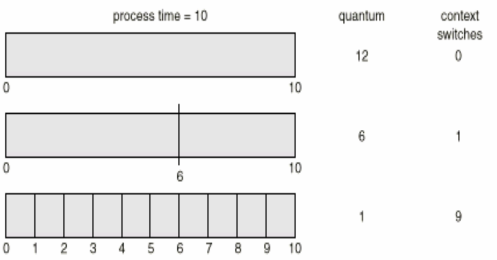
- Typically, *higher average turnaround* than SJF, but *better response*.

About Time Quantum of RR Scheduling

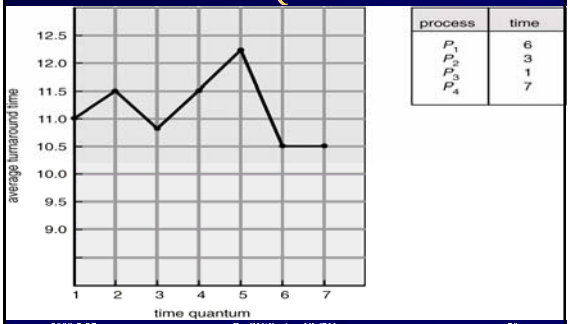
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.
- Performance
  - q large ⇒ FIFO
  - q small ⇒ Processor sharing, overhead is too high,

  q must be large with respect to context switch, otherwise. (See Next)

How a Smaller Time Quantum Increases Context Switches



Turnaround Time Varies With The Time Quantum



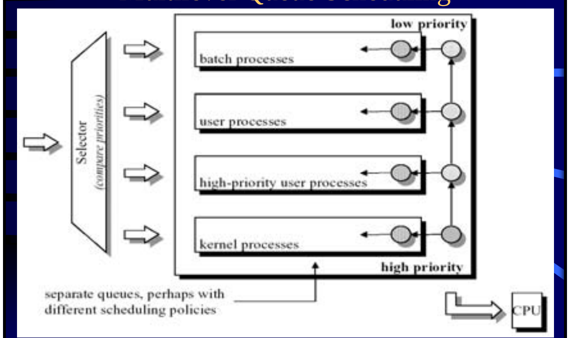
A Solution of Time Slice Problem

- Multi-Time-Slice queue:
- 0.02 seconds → 0.2 seconds → 2 seconds

Multilevel Queue

- Ready queue is partitioned into separate queues:
  foreground (interactive)
  background (batch)
- Each queue has its own scheduling algorithm,
  foreground – RR
  background – FCFS
- Scheduling must be done between the queues.
  - Fixed priority preemptive scheduling; i.e., the foreground queue may have priority over background. Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e.,
    80% to foreground in RR
    20% to background in FCFS

Multilevel Queue Scheduling



Processes are permanently assigned to a queue on entry to system.

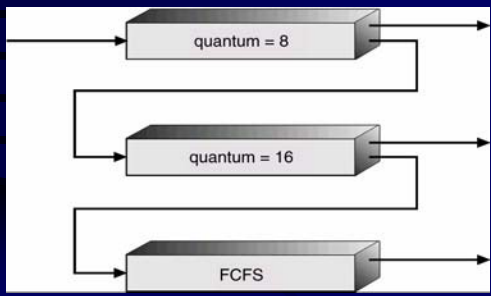
Multilevel Feedback Queue

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

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – time quantum 8 milliseconds
  - Q_1 – time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A new job enters queue Q_0 which is served FCFS. 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 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2.

Multilevel Feedback Queues

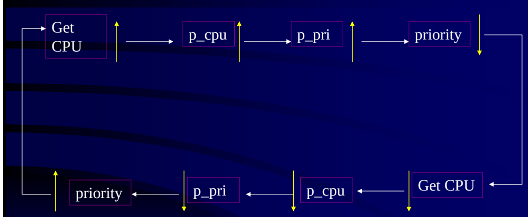


Real-Time Scheduling

- ***Hard real-time systems*** – required to complete a critical task within a guaranteed amount of time.
- ***Soft real-time computing*** – is less restrictive, requires that critical processes receive priority over less fortunate ones.
- ***Earliest (Shortest) Deadline First : EDF***
- ***Rate Monotonic Scheduling: RMS***

Unix Scheduling

- p_cpu in *proc*



Note: This is a simplified view.

Example on UNIX System V (1)

- Clock handler generates 60 clock ticks per second.
- Each PCB contains a field **CPU** ("recent CPU usage"), which is incremented on every clock tick while process is running.
- Every 60 ticks(1 second) scheduler is awakened and
  - adjusts recent CPU usage according to a decay function:
    decay(**CPU***)* = **CPU***/2*
  - recalculates priorities according to following formula (higher priorities have lower priority values!):
    priority = **CPU***/2 + base_priority*
- Decay rate controls aging.
- Priority recalculation controls demotion

Example on UNIX System V (2)

- 3 processes, each with base priority 60:

time	Process A priority	Process A CPU count	Process B priority	Process B CPU count	Process C priority	Process C CPU count
	60	0 1 ... 60	60	0	60	0
1	75	30	60	0 1 ...	60	0
2	67	15	75	30	60	0 1 ...
3	63	7 8 ... 67	67	15	75	30
4	76	33	63	7 8 ... 67	67	15
5	68	16	76	33	63	7

Thread Scheduling

- ***Local Scheduling*** – How the threads library decides which thread to put onto an available LWP.
- ***Global Scheduling*** – How the kernel decides which kernel thread to run next.

Dispatch Latency