CPU Scheduling

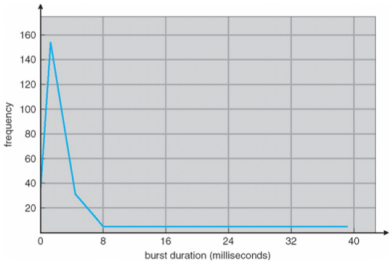
- Scheduling is important when multiple processes wish to run on a single CPU
  - CPU scheduler decides which process to run next
- Two types of processes
  - CPU bound and I/O bound

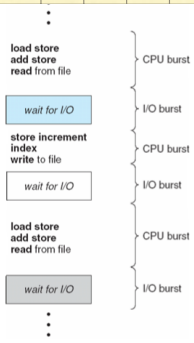
CPU-bound Process	I/O-bound process
Spends most of its running time on the CPU, i.e., **user-time > sys-time**	Spends most of its running time on I/O, i.e., **sys-time > user-time**
Examples - AI course assignments.	**Examples** - **/bin/ls**, networking programs.

进程有两类：CPU-bound（计算）（usr > sys）和 I/O-bound（I/O）（sys > usr）

CPU Burst

- Process execution consists of a *cycle* of CPU execution and I/O wait
- **CPU burst** distribution





CPU Scheduler

- CPU scheduler selects one of the processes that are ready to execute and allocates the CPU to it
- CPU scheduling decisions may take place when a process:
  - 1. Switches from running to waiting state
  - 2. Switches from running to ready state
  - 3. Switches from waiting to ready
  - 4. Terminates
- A scheduling algorithm takes place **only** under circumstances 1 and 4 is **non-preemptive**
- All other scheduling algorithms are **preemptive**

若调度算法只在1和4这两种情况下才调用的话，该调度算法是non-preemptive.（因为是自然中断）
若在2.3其一中调用的话则是preemptive

Scheduling Algorithm Optimization Criteria

- Given a set of processes, with
  - **Arrival time**: the time they arrive in the CPU ready queue (from waiting state or from new state)
  - **CPU requirement**: their expected CPU burst time
- Minimize average turnaround time
  - **Turnaround time**: The time between the arrival of the task and the time it is blocked or terminated.
- Minimize average waiting time
  - **Waiting time**: The accumulated time that a task has waited in the ready queue.
- Reduce the number of context switches

最小化完成时间：从到达到结束/阻塞的时间

最小化阻塞时间：任务在队列中等待时间

减少上下文切换次数

Different Algorithms

- **Shortest-job-first (SJF)**
- **Round-robin (RR)**
- **Priority scheduling**

SJF: Preemptive or Not?

	Non-preemptive SJF	Preemptive SJF
Average waiting time	4	3 (smallest)
Average turnaround time	8	7 (smallest)
# of context switching	3	5 (largest)

The waiting time and the turnaround time decrease at the expense of the **increased number of context switches**.

Task	Arrival Time	CPU Req.
P1	0	7
P2	2	4
P3	4	1
P4	5	4

SJF：每次进程调度时选工作时长最短的.

Non-preemptive：不会抢占.仅在空闲时发生调度

Preemptive：当有新任务到达时立刻调度.

Round Robin (RR)

- Round-Robin (RR) scheduling is preemptive.
  - Every process is given a **quantum** (the amount of time allowed to execute).
  - Whenever the quantum of a process is used up (i.e., 0), the process is preempted, placed at the end of the queue, with its quantum re-charged
  - Then, the scheduler steps in and it chooses the next process which has a non-zero quantum to run.
  - Processes are therefore running one-by-one as a circular queue
- New processes are added to the tail of the ready queue
  - New process's arrival won't trigger a new selection decision

每个进程都分配一段固定时间.

当固定时间用完时，进程会被抢占.置于队列末尾，换下一任务.

新来的进程置于队列末尾

RR是preemptive的.

RR v.s. SJF

	Non-preemptive SJF	Preemptive SJF	RR
Average waiting time	4	3	7 (largest)
Average turnaround time	8	7	11 (largest)
# of context switching	3	5	7 (largest)

So, the RR algorithm gets all the bad! Why do we still need it?

The responsiveness of the processes is great under the RR algorithm. E.g., you won't feel a job is "frozen" because every job gets the CPU from time to time!

在quantum比较小的时候不会注意到进程停止了.

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Nonpreemptive: newly arrived process simply put into the queue
  - Preemptive: if the priority of the newly arrived process is higher than priority of the currently running process---preempt the CPU
- Static priority and dynamic priority
  - static priority: fixed priority throughout its lifetime
  - dynamic priority: priority changes over time
- SJF is a priority scheduling where priority is the next CPU burst time

每个进程都有优先编号

priority可能随时间变化.

SJF也是优先调度.优先级是下一个任务的CPU burst time.

- Problem ≡ **Starvation** – low priority processes may never execute
  - Rumors has it that when they shut down the IBM 7094 at MIT in 1973, they found a low priority process that had been submitted in 1967 and had not yet been run.
- Solution ≡ **Aging** – as time progresses increase the priority of the process
  - Example: priority range from 127 (low) to 0 (high)
  - Increase priority of a waiting process by 1 every 15 minutes
  - 32 hours to reach priority 0 from 127

低优先级的进程可能永不会被执行

随时间优先级变大.

Linux Scheduling

- Before Linux kernel version 2.5, traditional UNIX scheduling, not adequately support SMP
- Linux kernel version 2.5, O(1) scheduler
  - Constant scheduling time regardless number of tasks
  - Better support for SMP
  - Poor response time for interactive processes
- After Linux kernel version 2.6.23, CFS-completely fair scheduler
  - Default scheduler now

Completely Fair Scheduler

- Scheduling class
  - Standard Linux kernel implements two scheduling classes
  - (1) Default scheduling class: CFS
  - (2) Real-time scheduling class
- Varying length scheduling quantum
  - Traditional UNIX scheduling uses 90ms fixed scheduling quantum
  - CFS assigns a proposition of CPU processing time to each task
- Nice value
  - -20 to +19, default nice is 0
  - Lower nice value indicates a higher relative priority
  - Higher value is "being nice"
  - Task with lower nice value receives higher proportion of CPU time

调度quantum是会变的.

nice value越低越高优先度.

- Virtual run time
  - Each task has a per-task variable **vruntime**
  - Decay factor
    - Lower priority has higher rate of decay
    - nice = 0 virtual run time is identical to actual physical run time
    - A task with nice > 0 runs for 200 milliseconds, its **vruntime** will be higher than 200 milliseconds
    - A task with nice < 0 runs for 200 milliseconds, its **vruntime** will be lower than 200 milliseconds
- Lower virtual run time, higher priority
  - To decide which task to run next, scheduler chooses the task that has the smallest **vruntime** value
  - Higher priority can preempt lower priority
- Example: Two tasks have the same nice value
- One task is I/O bound and the other is CPU bound
- **vruntime** of I/O bound will be shorter than **vruntime** of CPU bound
- I/O bound task will eventually have higher priority and preempt CPU-bound tasks whenever it is ready to run

每个任务都有一个虚拟运行时间.

nice = 0, vruntime = 实际物理运行时间.
nice > 0, vruntime > ...
nice < 0, vruntime < ...

越低的vruntime, 越高优先度.

I/O bound的vruntime小于CPU bound的.