**Basic Concepts**

CPU or I/O Burst Cycles: CPU cycle, then I/O wait, then back to CPU cycle.

CPU Burst: Process is running, using CPU

I/O Burst: Blocks process, uses I/O

In a processes life cycle, it spends **most** of its time waiting for I/O, with a short CPU burst.

**CPU Scheduler (FOR TESTS, know when preemtive and non must take place, NB)**

Short term scheduler takes processes from ready to running queue.

CPU scheduler decisions take place when a process switches from: (**Non Preemptive**)

1. Running to Waiting
2. Running to Ready
3. Waiting to Ready
4. Terminates

All other scheduling is **Preemptive**

Non Preemtive: Cannot stop it, unless process gives up CPU itself.

**Dispatcher**

Takes selected process, and makes sure it will run. Includes:  
Switching contect  
Switching to user mode  
Loading proper locations for program restarts

We want the **dispatcher** to be as **fast/short** as **possible**

**Dispatch latency:** Time it takes for a context switch to take place.

**Scheduling Criteria**

Decisions we make when designing scheduler to.

**CPU Uitilization:** CPU must be as busy as possible

**Throughput:** We want as many processes to be executed as possible per time unit

**Turnaround:** Amount of time to execute a particular process. Want to make this as short as possible.

**Waiting time:** Amount of time a process takes from waiting to ready.

**Response Time:** Amount of time from request submitted, to first response.

**BELOW CAN BE FOUND IN SCHEDULING ALGORITM SLIDES (NB for ST)**

**First-come, first-served**   
This algorithm is used in **non pre-emptive** environments, like batch systems. Three processes A, B and C arrive in the system during 0,1 and 2 msec. They all sit in a queue and the process that arrived first gets executed until finished

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| First come, first served | | |  |  |  |  |  |  |
| Process | Arrival Time | Burst Time (Number of CPU cycles before finishing CPU burst) |  |  | Time Spent | 24 | 3 | 3 |
| A | 0 | 24 |  |  | Implemented as | A | B | C |
| B | 1 | 3 |  |  |  |  |  |  |
| C | 2 | 3 |  |  |  |  |  |  |

B waits 23 till running, and C waits 25… so can have huge bottlenecks

23 as A is 24, but B arrived at 1 (24-1)  
and C is (24+3-2)

**Shortest Job First**   
This algorithm is used in **non pre-emptive** environments, like batch systems. Four jobs are scheduled. The job with the shortest burst time gets added first to the queue then the next shortest etc.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Shortest Job First | | |  |  |  |  |  |  |  |
| Process | Arrival Time | Burst Time |  |  | Time Spent | 3 | 6 | 7 | 8 |
| A | 0 | 6 |  |  | Implemented as | D | A | C | B |
| B | 0 | 8 |  |  |  |  |  |  |  |
| C | 0 | 7 |  |  |  |  |  |  |  |
| D | 0 | 3 |  |  |  |  |  |  |  |

**Shortest Remaining Time Next and Shortest Process Next**

Both the batch and interactive algorithm is discussed. In interactive systems, the burst time is determined by past behaviour. This technique is called aging

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Shortest Remaining Time Next and Shortest Process Next** | | |  |  |  |  |  |  |  |  |
| Process | Arrival Time | Burst Time |  |  | Time Spent | 1 | 4 | 5 | 7 | 9 |
| A | 0 | 8 |  |  | Implemented as | A | B | D | A | C |
| B | 1 | 4 |  |  |  |  |  |  |  |  |
| C | 2 | 9 |  |  |  |  |  |  |  |  |
| D | 3 | 5 |  |  |  |  |  |  |  |  |

**Round-Robin (5 Quanta) – The unit of time it runs till switching**

Assumes one quanta is 5 msec long. Processes A, B, C and D all arrive within the first quanta, that is why the immediately go into the ready queue.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Round-Robin (5 Quanta)** | | |  |  |  |  |  |  |  |  |  |  |  |  |
| Process | Arrival Time | Burst Time |  |  | Time Spent | 5 | 4 | 5 | 5 | 5 | 4 | 5 | 5 | 4 |
| A | 0 | 24 |  |  | Implemented as | A | B | C | D | A | C | A | A | A |
| B | 1 | 4 |  |  |  |  |  |  |  |  |  |  |  |  |
| C | 2 | 9 |  |  |  |  |  |  |  |  |  |  |  |  |
| D | 3 | 5 |  |  |  |  |  |  |  |  |  |  |  |  |

**Priority Scheduling**

he textbook describe different ways in which priority scheduling can be implemented. Other textbooks assume priority scheduling is only used in non pre-emptive system, and multiple queue.

Each process has a priority, and can jump the queue

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Priority Scheduling**  **(0 = highest) NBNB!** | | |  |  |  |  |  |  |  |  |  |  |  |  |
| Process | Priority | Burst Time |  |  | Time Spent | 1 | 5 | 10 | 2 | 1 |  |  |  |  |
| A | 3 | 10 |  |  | Implemented as | B | E | A | C | D |  |  |  |  |
| B | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |
| C | 4 | 2 |  |  |  |  |  |  |  |  |  |  |  |  |
| D | 5 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |
| E | 2 | 5 |  |  |  |  |  |  |  |  |  |  |  |  |

It is common to mix Priority with round robin

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Priority Scheduling**  **0 = highest; Quanta 5 msec; Preemtive implementation, decrease priority at each quanta, round robin** | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Process | Priority | Burst Time |  |  | Time Spent | 5 | 5 | 5 | 5 | 2 | 2 | 5 | 5 | 3 | 5 | 5 | 5 | 5 | 5 |
| A | 3 | 20 |  |  | Implemented as | B | E | B | A | E | B | C | A | C | A | D | A | D | D |
| B | 1 | 12 |  |  | Priority | 1 | 2 | 2 | 3 | 3 | 3 | 4 | 4 | 5 | 5 | 6 | 6 | 7 | 8 |
| C | 4 | 8 |  |  | New Priority | 2 | 3 | 3 | 4 |  |  | 5 | 5 |  | 6 | 7 |  | 8 |  |
| D | 6 | 15 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| E | 2 | 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Can also have this without decrese in priority

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Priority Scheduling**  **0 = highest; Quanta 5 msec; round robin** | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Process | Priority | Burst Time |  |  | Time Spent | 5 | 5 | 5 | 5 | 2 | 5 | 5 | 5 | 5 | 5 | 5 | … |  |  |
| A | 3 | 23 |  |  | Implemented as | B | D | B | D | B | D | D | D | D | C | E | … |  |  |
| B | 1 | 12 |  |  | Priority |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| C | 2 | 8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D | 1 | 30 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| E | 2 | 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**Determining length of next CPU Burst**

-Not asked – sir doesn’t care

**Priority Scheduling**

Problem: **Starvation** – low priority processes never executed

Solution: **Aging**: Increases priority of processes the more they are waiting

**Round Robin**

Quanta time: Shorter has more processes, but less . Larger quanta, more

**Multilevel Queues**

Apps can be in the **Background** or **Foreground.** Different priority can be assigned based on this.

Eg Refresh screen = high priority, then goes back down again.

**Multilevel Feedback Queues**

Processes go in queues, where queues have priority levels eg   
System Process: high priority  
Student Processes: Low Priority

**Multilevel feedback queues**

Method to decide when to upgrade or downgrade a process from a queue

Windows has a base priority (8) that apps start at, and a dynamic one Higher priority is chosen first.

**Ideal Processor:** The processor it will choose to run it if possible

**Thread Scheduling**

System contention scope **(SCS):** Look at all threads In system when scheduling (one to one)

Process-contention scope (**PCS)**: many to one and many to many.

**Multiple processor scheduling**

Each processor has a scheduler. This allows the controller to know the processors best use, as not all processes are equal in strength.

**Homogeneous processor**: within a multiprocessor

**Asymmetric multiprocessor**: Only one processor accesses system data structures

**Symmetric multiprocessor**: Each processor is self scheduling, all have common ready queue or private ready queue

**Processor Affinity:** Processors have an affinity for the processor it is running on.  
-So when a eg context switch occurs for P1, it is better for when P1 is running again to run on the chip it was originally running on. This is due to cache memory, as its instructions can still be in the cache line (not always reset)

If a different processor is free, it may be better to use that one instead:

**-Soft affinity:** Process has ideal processor, but can migrate to another processor  **-Hard affinity:** Process has ideal processor, and cannot migrate to another processor

**NUMA architecture (non uniform memory access):** Have 2 CPUs, and many RAM slots. A CPU should only use the RAM that is close to it.

This is vital in virtual machines, as can run multiple VMs with these high end boards.

**Multiple processor scheduling – load balancing**

**Load balancing:** Trying to balance workload evenly

**1. Push migration:** If processor overburdened, push processes to another processor.

**2. Pull migration:** Idle processors pulls waiting task from busy processor

**Multicore Processors**

Faster and consumers less power.

Each core, is a multiprocessor.

**Multithreaded Multicore system**

You interleave of when threads are computing or accessing memory, you want to interlink (so when processor 1 is computing, processor 2 is accessing memory(?))  
-not sure if accessing is the right word