1 Priority Queues

Jobs can have **different priority levels** that are **fixed**.
Jobs of the **same priority** are run in **round robin** fashion
Priority queues are usually implemented by using **multiple queues**, on for each priority level

2 Multi-level feedback queues

Different **scheduling algorithms** can be used on **individual queues**
Feedback queues allow **priorities to change dinamically**, i.e., jobs can **move between queues**

- Move to **lower priority queue** if too much CPU time is used (prioritise I/O and interactive processes)
- Move to **higher proprity queue** to prevent **starvation** and avoid **inversion of control**

Feedback queues are highly **configurable** and offer significant flexibility. Defining characteristics of feedback queues include:

- The **number of queues**
- The **scheduling algorithms** used for individual queues
- **Migration policy** between queues
- Initial **access** to the queues

2.1 Windows7

An interactive system using a *preemptive scheduler* with **dynamic** priority levels. **Two** priority classes with **16** different priority levels exist

- **Real time** processes/threads hava a **fixed** priority level
- **Variable** processes/threads can have their priorities **boosted** temporarily

A **round robin** algorithm is used within the queues.
Priorities arebased on the **process** (0-15) and **thread** (+-2) base priorities.
A thread's priority **dinamically** changes during execution between its base priority and the **maximum** priority whithin its class

- Interactive **I/O** bound processes (e.g. keyboard) receive a **larger boost**
- Boosting priorities prevents *priority inversion*

3 Scheduling in Linux

3.1 The completely fair scheduler

Process scheduling has evolved over different versions of Linux to account for **multiple processors/cores**, processor **affinity**, and load balancing between cores.
Linux distinguishes between two types of tasks for scheduling:

- **Real time tasks** (to be POSIX compliant), divided into
  - Real time FIFO tasks
  - Real time Round Robin tasks
- **Time sharing tasks** using a *preemptive* approach (simillar to variable in windows)

The most recent scheduling algorithm in Linux for **time sharing tasks** is *completely fair scheduler* (CFS, before the 2.6 kernel, this was O(1) scheduler)

3.2 Real-Time tasks

Real time FIFO taks have the **highest priority** and are scheduled using *FCFS* approach, using **preemption if a higher priority** job shows up.
Real time Round Robin tasks are preemptable by **clock interupts** and have a **time sclice** associated with them.
Both approaches **cannot** guarantee hard deadlines.

3.3 Time Sharing Tasks (equal priority)

The CFS **divides** the CPU time between all processes. The lenght of the **time slice** and the available CPU time are based on the *targeted latency* If number of tasks is very large, the context switch time will be dominant, hence a lower bound on the time slice is imposed by the *minimum granularity*

3.4 Time sharing tasks (different priority)

A *weighting scheme* is used to take different priorities into account. If processes have **different** priorities

- Each process i is allocated a weight w_i that reflects its priority
- The "time slice" allocated to process i is the **proportional to** (math equation here)

The tasks with the **lowest** proportional amount of used CPU time are selected first

4 Multi-processor scheduling

4.1 Scheduling decisions

Single processor machine: **which process (thread)** to run next (one dimensional).
Multiprocessor/Core machine:

- Which process (thread) to run **where**, which CPU ?
- Which process (thread) to run **when**

4.2 Shared queues

A single or multi-level queue **shared** between all CPUs.
Advantage: automatic *load balancing*
Disadvantage:

- *Contention* for the queues (locking is needed)
- Processor affinity:
  - Cache becomes **invalid** when moving to different CPU
  - Translation look aside buffers (TLBs - part of MMU) become invalid

Windows will allocate **highest priority** threads to the individual CPUs/cores

4.3 Private queues

Each CPU has a **private** set of queues
Advantages:

- CPU affinity is automatically satisfied
- Contention fro shared queue is minimised

Disadvantages: less *load balancing*
Push and **pull** migration between CPUs is possible

5 Related vs unrelated threads

Related: multiple threads that **communicate** with one another and **ideally run** together
Unrelated: e.g. processes/threads that are **independent**, possibly started by different users, running different programs

6 Scheduling Related Threads

6.1 Working together

The aim is to get threads **running**, as much as possible, at the **same time** across **multiple CPUs**
Approches include

- Space scheduling
- Gang scheduling

6.2 Space Scheduling

Approach:

- N threads are allocated to N **dedicated** CPUs.
- N threads are kept waiting until N CPUs are available.
- **Non-preemptive**, i.e. blocking calls result in **idle CPUs** (less context switching overhead but results in CPU idle time)

The number N can be **dynamically** adjusted to match processor capacity

6.3 Gang Scheduling

Time slices are **synchronised** and the scheduler **groups** threads together to run simultaneosly (as much as possible). A *preemptive* algorithm. **Blocking** threads result in idle CPUs. A scheduling algorithm for parallel systems that schedules related threads or processes to run simultaneously on different processors. Usually these will be threads all belonging to the same process, but they may also be from different processes. For example, when the processes have a producer-consumer relationship, or when they all come from the same MPI(Message Passing Interface) program.

Reference section

priority inversion
: problematic scenario in scheduling in which a high priority task is indirectly preempted by a lower priority task effectively "inverting" the relative priorities of the two tasks.

preemptive scheduler
: Tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.

targeted latency
: CFS sets a target for its approximation of the infinitely small scheduling duration in perfect multitasking. This target is called the targeted latency. every process should run **atleast once** during this interval

minimum granularity
: CFS imposed a floor on the timeslice assigned to each process.This floor is called the minimum granularity. Minimum time a task will be be allowed to run on CPU before being pre-empted out.

contention
: describes conflicts between different virtual machines in a virtualized hardware system where they are competing for the same resources. The term 'CPU contention might describe an event or series of events, or it might generically refer to situations where these kinds of conflicts occur.