Load Balancing

Both processors are kept running
Statistical analysis will be used to determine which ready queue processes should go into.

Remember the goal: the processor is the most expensive part of the computer. You want to keep it (them) busy, otherwise you have a poor cost/performance ratio.

There are two methods of achieving this:

Common-ready queue

> If a processor becomes idle, its dispatcher simply gets a process from the ready queue and the ready queue shrinks.
>
> The responsibility of load balancing becomes something that is not managed by the OS, but a processor simply keeps itself busy.

Private (separate) ready queues

> If a processor becomes idle, the operating system has to pull a process from a ready queue and place it on the CPU.
>
> The worst case: dispatcher finds no processes in the ready queue. In this case, a process must be taken from another processor's ready queue.

Techniques used to move processes from one queue to another in a private ready queue system.

Push Migration
OS process periodically checks the queues and balances them

Push / Pull –

Pull Migration
The idle processor dispatcher pulls a process from another ready queue.

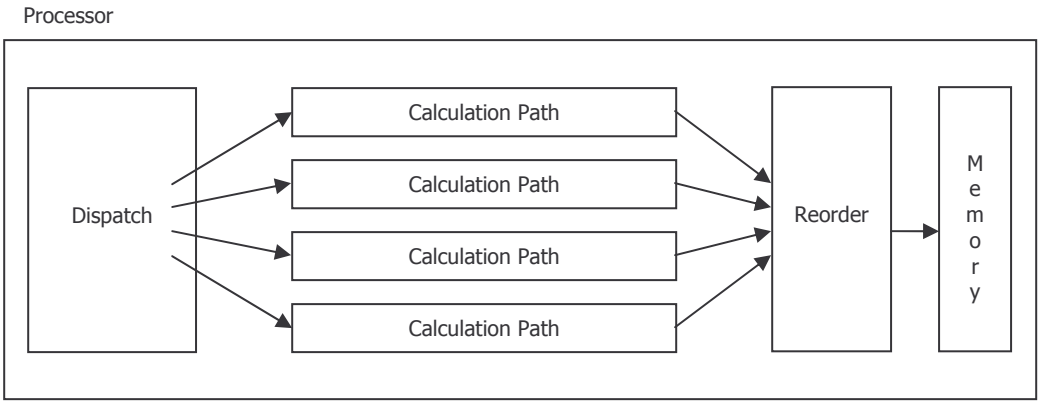
On Exam!

Hyperthreading

Symmetric multithreading

Multithread processors that look the same to us – have the same resources, etc.

Implemented in the Intel Pentium 4 with HT technology



Take a scalar calculation path and split it into several calculation paths, thus making the process superscalar.

Multiple machine state registers

Windows XP Scheduling

Preemptive

- Arrival of a higher priority process to a ready queue
- Time slice expiration
- Process termination
- Process blocks for I/O, child, or signal

32 levels of priority (lowest: 0 to highest: 31)

- Variable priority: 1-15
- Real-time priority: 16-31

Real time processes – must be given the CPU within a given length of time for the system to remain stable.

There is one queue per priority, for a total of 32 queues. The dispatcher looks at every queue, starting at 31, and runs the first it finds.

Priority classing: real time, high, above normal, normal, below normal, idle
Relative classes: time critical, highest, above normal, below normal, lowest, idle

Real-time priorities are static

With variable priority processes, priority is decreased if process execution time takes longer than expected.