The dispatcher

A process dispatcher gives control of the CPU to a process selected by the short term scheduler.

Rewind back to the introduction where we said that your boss was the scheduler and the pat-man was the dispatcher. Note that the boss selected you but it was the pat-man who actually opened the door for you and gave you the resources to operate. Similarly, dispatchers can allocate threads to processes and CPU to the threads.

A dispatcher has the following responsibilities:

- **Switching to user mode**: All of the low level operating system processes run on the kernel level security access, but all of the application code and user issued processes run in the application space or the user permission mode. Dispatcher switches the processes to the user mode.

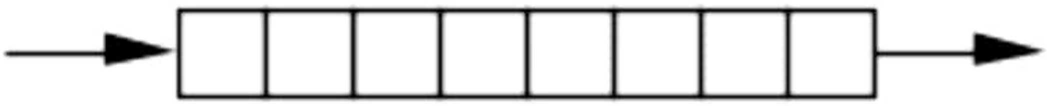
- **Addressing:** The program counter (PC) register points towards the next process that is to be executed. The dispatcher is responsible for addressing that address.

- **Initiation of context switch:** A context switch is when a currently running process is halted and all of its data and its process control block (PCB) are stored in main memory, and another process is loaded in its place for execution.
- **Managing dispatch latency:** Dispatch latency is calculated as the time it takes to stop one process and start another. The lower the dispatch latency, the more efficient the software for the same hardware configuration.

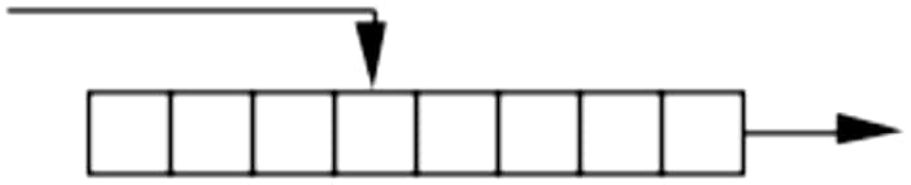
> Note that a dispatcher is NOT a thread. The dispatcher runs on each core, runs a thread for a while, saves its state, loads the state of another thread and runs it.

Selection of processes for dispatch

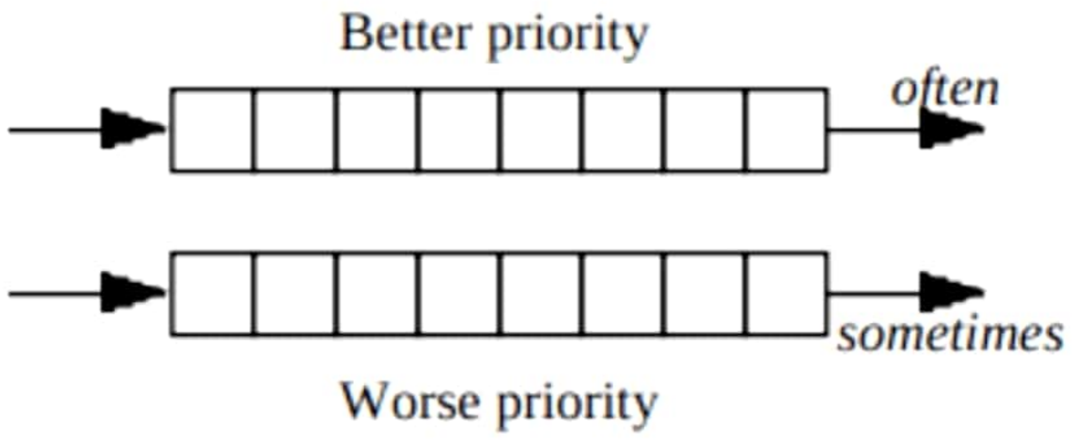
The dispatcher can select processes in the following ways:



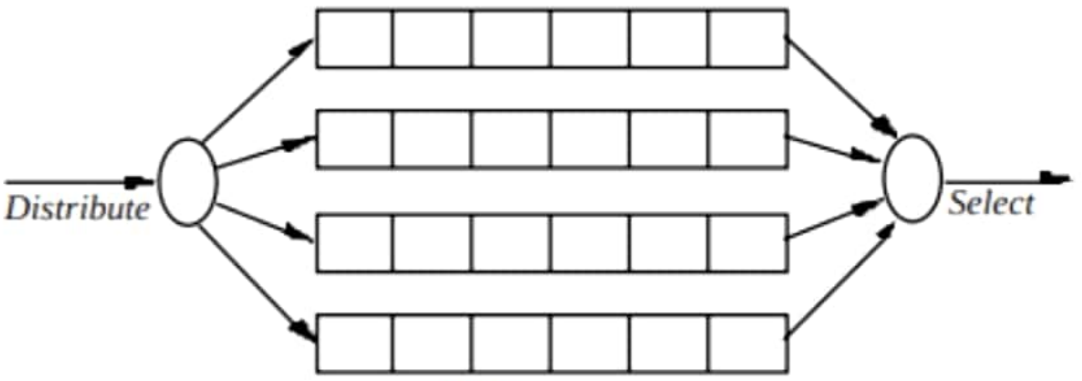
- Search the table from the front and run the first ready thread.



- Dispatch one thread from the queue and run it. If the process is not completed, insert it at the back of the queue.



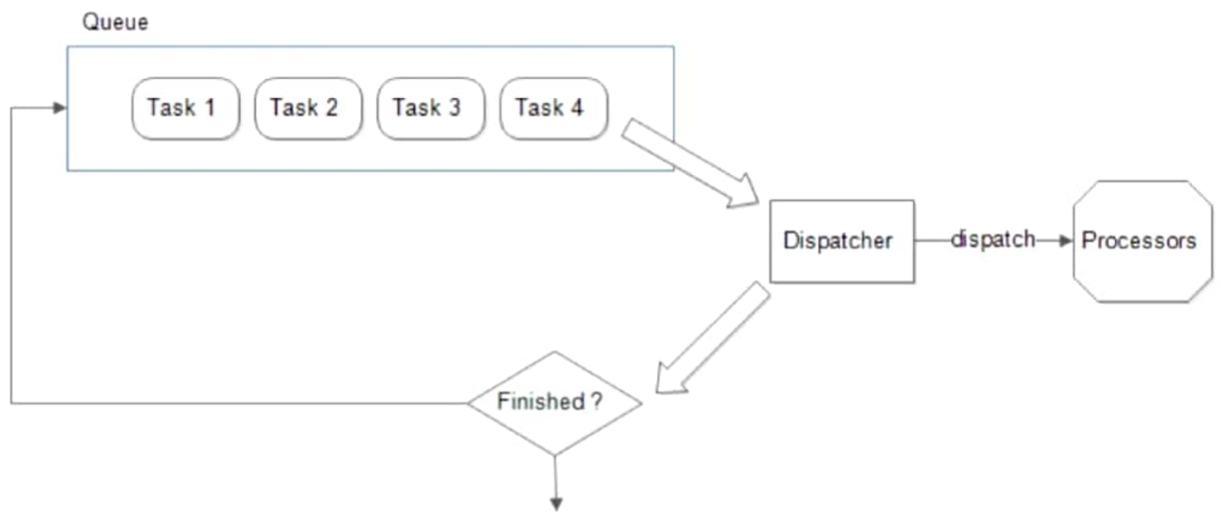
- Give each thread a priority and organize the priority queues accordingly.



FIFO queues

priority based multiple queues

- Have multiple queues for each priority class. Whenever some threads are ready for execution chose the one with the highest priority and run it.



Stopping processes and managing dispatcher failure

If a thread is executing and the dispatcher isn't, it means that the operating system has lost control. In that case the following recovery mechanisms can be employed:

Traps: Traps are essentially events in the operating system which cause a state switch into the operating system. Traps allow execution of a program or task to be continued without loss of program continuity. The return address for the trap handler points to the instruction to be executed after the trapping instruction. They can be used to catch arithmetic errors.

- `System calls`: System calls is an application programming interface through which processes running in user mode can communicate with the kernel mode to leverage operating system specific functionalities like signalling or killing processes.

- `Page Fault`: A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system.

Interrupts: Events occurring outside the current thread that causes a state switch in the operating system. eg: timers, completion of disk operations etc. They can be used to stop running processes.

> Interrupts are hardware interrupts like the completion of I/O events while traps are software-invoked interrupts like division by 0. Both traps and interrupts are asynchronous and are used as signalling or recovery mechanisms by the OS in case of a dispatch failure.

Conclusion

A dispatcher is an integral part of an operating system. It is responsible for maintaining the ready queue and making sure each ready process gets dispatched to utilize the CPU.

Dispatcher is also being used in other systems like sophisticated small-talk messaging systems, and in object oriented programming as dynamic dispatchers. It is time someone patted the pat-man on the back!