**The Scheduling Algorithm**

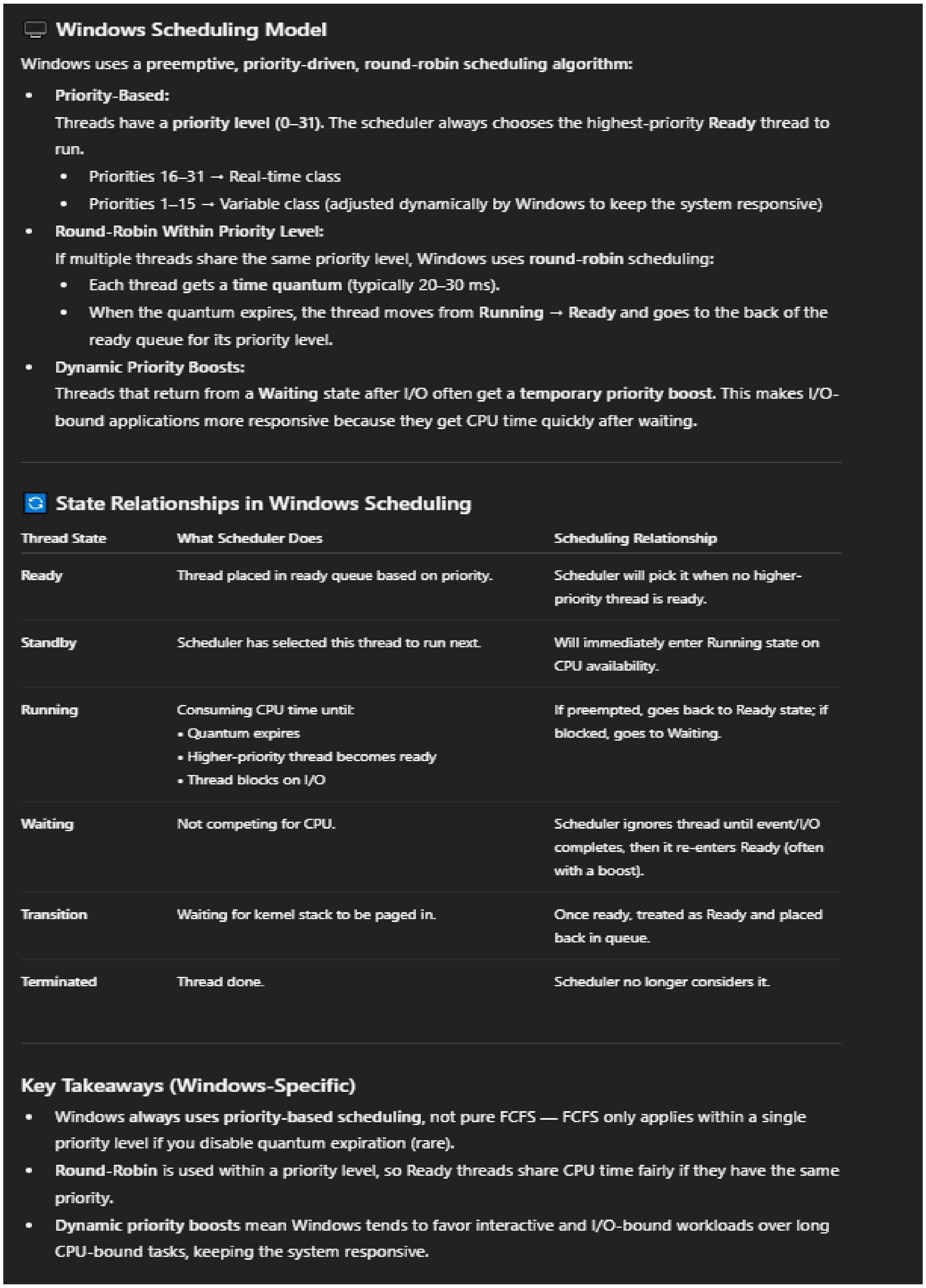
Windows 10 uses a priority-driven preemptive CPU scheduling algorithm with multiple priority levels, often described as a multilevel feedback queue (MLFQ) with a round-robin element. This approach allows for dynamic adjustment of process priorities, giving preference to short jobs and I/O-bound processes while preventing starvation by moving long-waiting processes to higher priority queues.

Key features of the MLFQ scheduling in Windows 10 include:

- Priority-Driven: Every thread is assigned a priority level, and higher-priority threads are given preference for the CPU.
- Preemptive: A higher-priority thread that becomes ready to run can preempt a lower-priority thread that is currently executing, immediately giving the CPU to the more important task.
- Multilevel Feedback Queue (MLFQ):
- Processes are placed into different queues based on their behavior.
- Processes that consume a lot of CPU time (CPU-bound) might be moved to lower-priority queues, while those that wait too long or are I/O-bound are moved to higher-priority queues.
- This dynamic movement ensures that processes don't become "starved" and can respond to changing demands.
- Round-Robin within Priorities: Within each priority level, time slices are given to threads in a round-robin fashion, ensuring fair access among threads of the same priority.

This design makes the Windows scheduler very flexible and effective at managing a wide variety of workloads, from background tasks to interactive applications.

## The State Values



### Windows Scheduling Model

Windows uses a preemptive, priority-driven, round-robin scheduling algorithm:

- **Priority-Based:**
  Threads have a **priority level (0–31).** The scheduler always chooses the highest-priority **Ready** thread to run.
  - Priorities 16–31 → Real-time class
  - Priorities 1–15 → Variable class (adjusted dynamically by Windows to keep the system responsive)
- **Round-Robin Within Priority Level:**
  If multiple threads share the same priority level, Windows uses **round-robin** scheduling:
  - Each thread gets a **time quantum** (typically 20–30 ms).
  - When the quantum expires, the thread moves from **Running** → **Ready** and goes to the back of the ready queue for its priority level.
- **Dynamic Priority Boosts:**
  Threads that return from a **Waiting** state after I/O often get a **temporary priority boost.** This makes I/O-bound applications more responsive because they get CPU time quickly after waiting.

---

### State Relationships in Windows Scheduling

| Thread State | What Scheduler Does | Scheduling Relationship |
|---|---|---|
| Ready | Thread placed in ready queue based on priority. | Scheduler will pick it when no higher-priority thread is ready. |
| Standby | Scheduler has selected this thread to run next. | Will immediately enter Running state on CPU availability. |
| Running | Consuming CPU time until:<br>• Quantum expires<br>• Higher-priority thread becomes ready<br>• Thread blocks on I/O | If preempted, goes back to Ready state; if blocked, goes to Waiting. |
| Waiting | Not competing for CPU. | Scheduler ignores thread until event/I/O completes, then it re-enters Ready (often with a boost). |
| Transition | Waiting for kernel stack to be paged in. | Once ready, treated as Ready and placed back in queue. |
| Terminated | Thread done. | Scheduler no longer considers it. |

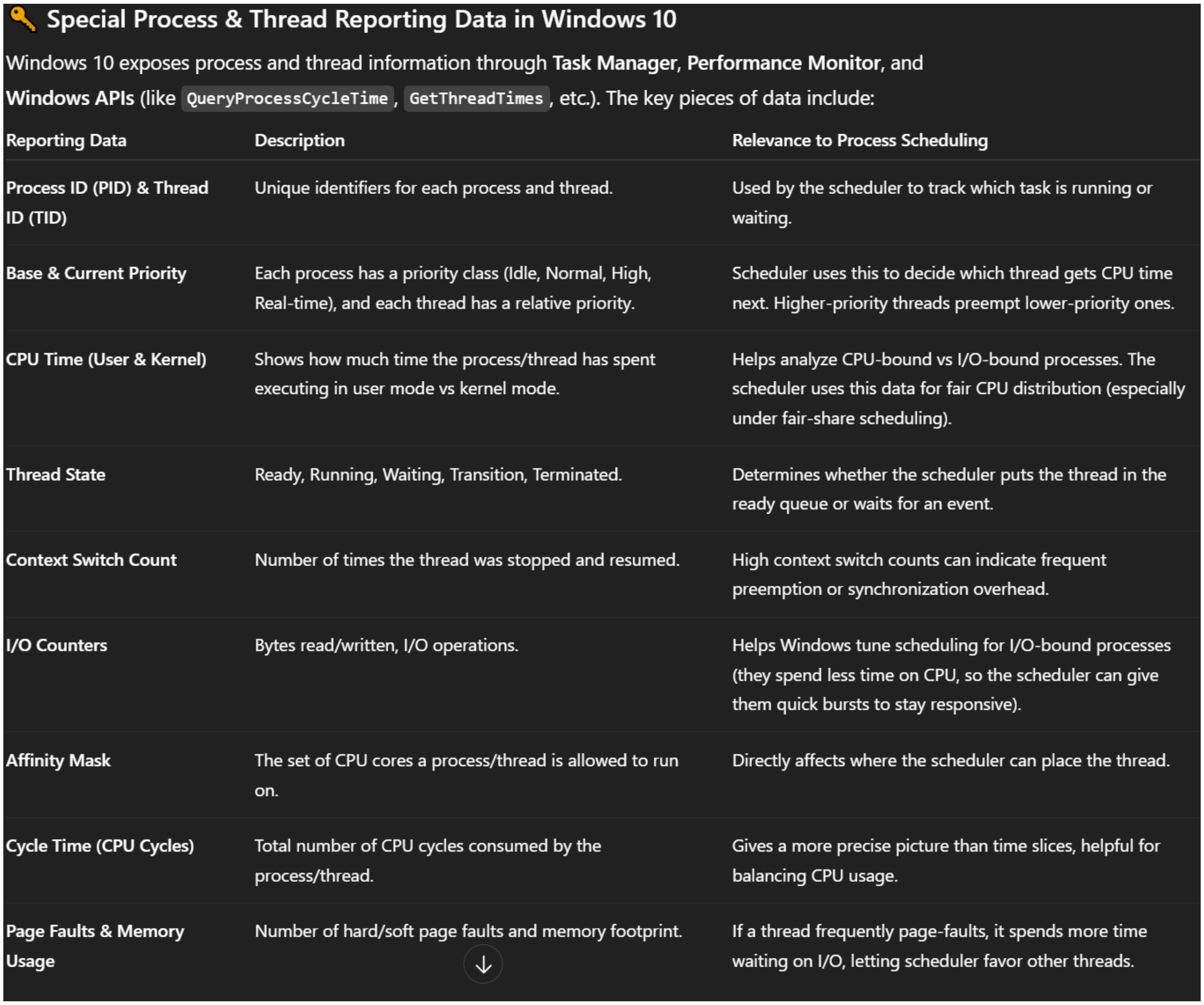
---

### Key Takeaways (Windows-Specific)

- **Windows always uses priority-based scheduling**, not pure FCFS — FCFS only applies within a single priority level if you disable quantum expiration (rare).
- **Round-Robin** is used within a priority level, so Ready threads share CPU time fairly if they have the same priority.
- **Dynamic priority boosts** mean Windows tends to favor interactive and I/O-bound workloads over long CPU-bound tasks, keeping the system responsive.

**Special Process & Thread Reporting Data in Windows 10**

Windows 10 exposes process and thread information through Task Manager, Performance Monitor, and Windows APIs (like QueryProcessCycleTime, GetThreadTimes, etc.). The key pieces of data include:



🔑 **Special Process & Thread Reporting Data in Windows 10**

Windows 10 exposes process and thread information through **Task Manager**, **Performance Monitor**, and **Windows APIs** (like `QueryProcessCycleTime`, `GetThreadTimes`, etc.). The key pieces of data include:

| Reporting Data | Description | Relevance to Process Scheduling |
|---|---|---|
| **Process ID (PID) & Thread ID (TID)** | Unique identifiers for each process and thread. | Used by the scheduler to track which task is running or waiting. |
| **Base & Current Priority** | Each process has a priority class (Idle, Normal, High, Real-time), and each thread has a relative priority. | Scheduler uses this to decide which thread gets CPU time next. Higher-priority threads preempt lower-priority ones. |
| **CPU Time (User & Kernel)** | Shows how much time the process/thread has spent executing in user mode vs kernel mode. | Helps analyze CPU-bound vs I/O-bound processes. The scheduler uses this data for fair CPU distribution (especially under fair-share scheduling). |
| **Thread State** | Ready, Running, Waiting, Transition, Terminated. | Determines whether the scheduler puts the thread in the ready queue or waits for an event. |
| **Context Switch Count** | Number of times the thread was stopped and resumed. | High context switch counts can indicate frequent preemption or synchronization overhead. |
| **I/O Counters** | Bytes read/written, I/O operations. | Helps Windows tune scheduling for I/O-bound processes (they spend less time on CPU, so the scheduler can give them quick bursts to stay responsive). |
| **Affinity Mask** | The set of CPU cores a process/thread is allowed to run on. | Directly affects where the scheduler can place the thread. |
| **Cycle Time (CPU Cycles)** | Total number of CPU cycles consumed by the process/thread. | Gives a more precise picture than time slices, helpful for balancing CPU usage. |
| **Page Faults & Memory Usage** | Number of hard/soft page faults and memory footprint. | If a thread frequently page-faults, it spends more time waiting on I/O, letting scheduler favor other threads. |

## References

Microsoft. (2023, October 26). Processes and threads. Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/procthread/processes-and-threads

Microsoft. (2023, October 26). Process and thread functions. Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/procthread/process-and-thread-functions

Microsoft. (2023, October 26). Scheduling. Microsoft Learn. https://learn.microsoft.com/en-us/windows/win32/procthread/scheduling

Russinovich, M. E., Solomon, D. A., & Ionescu, A. (2017). Windows Internals, Part 1: System architecture, processes, threads, memory management, and more (7th ed.). Microsoft Press.

Which type of scheduling algorithm is used by Windows 10?. Quora. (n.d.). https://www.quora.com/Which-type-of-scheduling-algorithm-is-used-by-windows-10