**Process Burst Cycle**

A **process burst cycle** refers to the alternating phases of **CPU execution** and **I/O waiting** during a process's execution. A typical process execution consists of **two main bursts**:

**CPU Burst Time**

* The time a process spends executing CPU instructions without needing I/O.
* It performs **computations, updates variables, and makes decisions**.
* Duration varies based on process type (e.g., CPU-bound processes have longer CPU bursts).

**I/O Burst Time**

* The time a process spends waiting for or **performing I/O** **operations**.
* Examples: reading from a file, waiting for user input, writing data to disk.
* The process **cannot use the CPU during I/O** and is put into the **waiting state**.

**CPU-bound vs. I/O-bound Processes**

| **Type of Process** | **CPU Burst Time** | **I/O Burst Time** | **Example** |
| --- | --- | --- | --- |
| **CPU-bound** | Long | Short | Scientific computations, 3D rendering |
| **I/O-bound** | Short | Long | Web browsing, File transfer |

* **CPU-bound processes** require more **CPU bursts** (e.g., running complex calculations).
* **I/O-bound processes** spend more time in **I/O bursts** (e.g., waiting for disk read/write operations).

A diagram of a computer program

AI-generated content may be incorrect.

**Scheduling Decision Modes**

The scheduler runs when it needs to select a process to run on the processor.

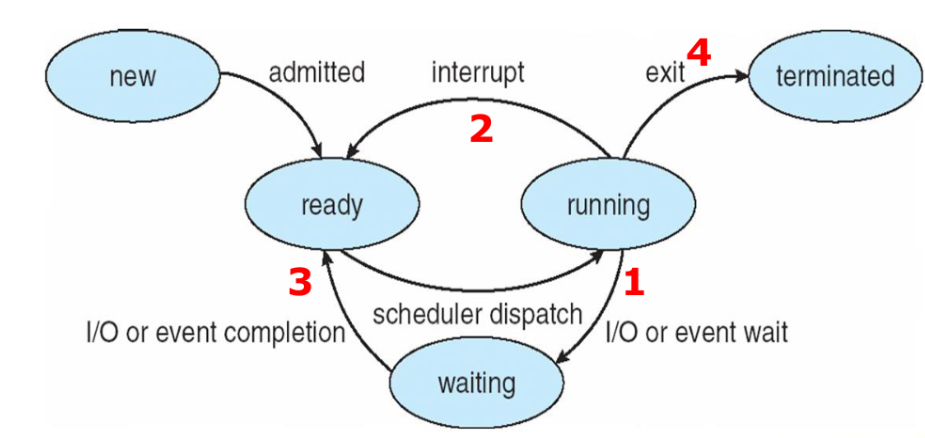
The **scheduling decision mode** specifies the times at which the selection of a process to run is made. This decision can take place at the following times:

1. Switches from running to waiting state; ex: as result of I/O request or wait()

2. Switches from running to ready state; ex: when an interrupt occurs

3. Switches from waiting to ready; at completion of I/O. or because a new process was created and was placed in the queue.

4. Terminates



Scheduling taking place only under circumstances **1 and 4** is nonpreemptive

* Process keeps the CPU until it either terminates or switches to waiting state
* No choice in terms of scheduling; new process must be selected for CPU

In circumstances **2 and 3**, the OS scheduler has a choice:

* it can either allow the current process to continue running, or
* it could step in and put the current process to sleep and select a different process to run.

The latter operation is called "preempting" the current process and scheduling a new one to run.

**CPU Scheduler (Short-Term Scheduler)**

**Preemptive scheduling -** Running process may be interrupted and moved to the Ready queue.

Preemptive scheduling can result in **race conditions** when data is shared among several processes.

Consider the case of two processes that share data. While one process is updating the data, it is preempted so that the second process can run. The second process then tries to read the data, which is in an inconsistent state.

**When does it occur?**

* **Higher-priority process arrives** (Priority Scheduling).
* **Time slice expires** (Round Robin).
* **I/O or system call completes** (Shorter job resumes execution).

**Non-preemptive scheduling: -** once a process is running, it continues to execute until it terminates or blocks for I/O.

**When does it occur?**

* **Process terminates**.
* **Process performs an I/O operation**.

**🔹 When to Use Each?**

* **Use Preemptive Scheduling** when responsiveness is critical (e.g., interactive OS, real-time systems).
* **Use Non-Preemptive Scheduling** when processes are predictable and context switching overhead is a concern.

**🔹 Key Differences**

| **Feature** | **Preemptive Scheduling** | **Non-Preemptive Scheduling** |
| --- | --- | --- |
| **Interruptions** | A process **can** be interrupted | A process **cannot** be interrupted |
| **CPU Utilization** | High (ensures fairness) | Can be inefficient (one long process may block others) |
| **Implementation Complexity** | More complex due to context switching | Simpler, no forced process switching |
| **Example Algorithm** | Round Robin, SRTF | FCFS, SJF (non-preemptive) |
| **Best For** | Multitasking, real-time systems | Batch processing, simple systems |

A diagram of a scheduler and dispatcher

AI-generated content may be incorrect.**Dispatcher**

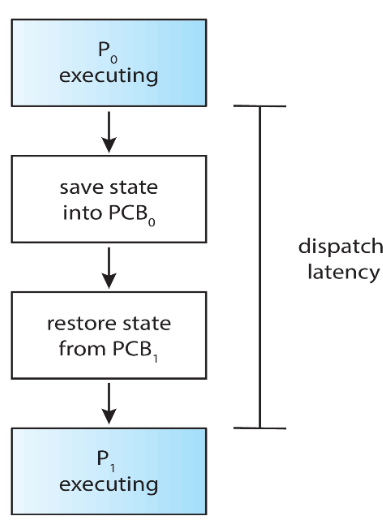
The dispatcher is the kernel routine that performs the context switch.

Dispatcher gives control of the CPU to the process selected by the **Short-Term Scheduler**.

**Role/Responsibilities of Dispatcher**

The dispatcher is invoked by the **CPU scheduler** and does the following:

1. **Context Switching:** Saves the state of the current process and loads the state of the next selected/scheduled process.
2. **Switching to User Mode:** It then changes the CPU mode to user mode, since it runs in kernel mode.
3. **Jumping to the Right Location:** It transfers control to the new/next process. It moves execution to the next instruction of the newly loaded process.



**Dispatch latency –** time it takes for the dispatcher to stop one process and start another running. High dispatch latency can slow down system responsiveness.

**Dispatcher vs. CPU Scheduler**

| **Feature** | **CPU Scheduler** | **Dispatcher** |
| --- | --- | --- |
| **Function** | Selects which process runs next | Performs the switch between processes |
| **Speed Concern?** | No (decision-making) | Yes (must be fast to avoid delays) |
| **Includes Context Switching?** | No | Yes |

**Scheduling Criteria**

**CPU utilization –** keep the CPU as busy as possible

**Throughput –** # of processes that complete their execution per time unit

* TP = No. Of jobs / Last Job Completion Time

**Turnaround time (TAT) –** amount of time to execute a particular process

* Sum of times spent in **job pool + ready queue + CPU execution + doing I/O**
* TAT = Waiting Time + Burst Time **OR**
* TAT = Completion Time - Arrival Time **(Recommended)**

**Waiting time (WT)–** amount of time a process has been waiting in the ready queue

* **Non-pre-emptive:** WT = Start Time – Arrival Time
* **Pre-emptive:** WT = Total Waiting Time – No. Of Seconds Process Executed – Arrival Time
* **For Both:** WT = Turnaround Time - Burst Time **(Recommended)**

**Response time –** amount of time it takes from when a request was submitted until the first response is produced.

* RT = Waiting Time

**Burst time –** total time a process requires for its execution on the CPU, **excluding** I/O time.

**Scheduling Algorithm Optimization Criteria**

1. Max CPU utilization
2. Max throughput
3. Min turnaround time
4. Min waiting time
5. Min response time

**🔹 1. CPU Scheduler (Short-Term Scheduler)**

✅ **Runs frequently** (every few milliseconds).  
✅ **Selects which process** should run next from the ready queue.  
✅ **Operates in milliseconds** to minimize CPU idle time.  
✅ Uses **preemptive** (e.g., Round Robin) or **non-preemptive** (e.g., FCFS) scheduling.

**Example:** If **Process P1** is running and a higher-priority **Process P2** arrives, the CPU scheduler may preempt **P1** and switch to **P2**.

**🔹 2. Long-Term Scheduler (Job Scheduler)**

✅ **Controls process admission** into the system (decides which jobs enter the Ready queue).  
✅ Runs **infrequently** (seconds or minutes).  
✅ Maintains a balance between **CPU-bound** and **I/O-bound** processes.

**Example:** If 100 processes try to enter the system at once, the **long-term scheduler** may select only 20 to avoid overloading.

**🔹 3. Medium-Term Scheduler (Swapper)**

✅ **Suspends and resumes processes** (swaps them in and out of memory).  
✅ Helps manage **RAM usage** by moving inactive processes to disk (swapping).  
✅ Reduces the load on the system when too many processes are running.

**Example:** If **P1** is waiting for I/O or is inactive, the medium-term scheduler **swaps** it to disk to free memory for another process.

**🔹 Key Differences**

| **Feature** | **CPU Scheduler** | **Long-Term Scheduler** | **Medium-Term Scheduler** |
| --- | --- | --- | --- |
| **Purpose** | Selects process for execution | Controls process admission | Manages swapping |
| **Execution Frequency** | Milliseconds (very often) | Seconds or minutes (rarely) | Occasionally |
| **Process State** | From Ready → Running | From New → Ready | From Suspended ↔ Ready |
| **System Type** | Always present | Not always present (e.g., in batch systems) | Only in systems with swapping |

**Multilevel Queue Scheduling (MLQ)**

Multilevel Queue Scheduling is a CPU scheduling technique used when processes are divided into **different categories** based on their characteristics, such as priority, memory size, process type (system/user), or CPU/IO-bound nature.

**1. Why Use Multilevel Queue?**

* Some systems require strict **separation** of process types.
* Some processes need **higher priority** (e.g., system processes over user processes).
* Certain processes have different CPU **response time needs** (e.g., interactive vs. batch processes).

**2. How It Works**

* The **ready queue** is divided into **multiple queues**.
* Each queue has **its own scheduling algorithm** (e.g., FIFO, RR, SJF).
* A **fixed priority order** decides which queue gets CPU time.

**3. Structure of Multilevel Queue**

A typical example of queue classification:

| **Queue Type** | **Process Type** | **Scheduling Algorithm** |
| --- | --- | --- |
| **System Queue** | OS/System Processes | **Highest Priority, FCFS** |
| **Interactive** | Real-time User Tasks | **Round Robin (RR)** |
| **Foreground** | Shorter CPU Jobs (User) | **RR or SJF** |
| **Background** | Long CPU Jobs (Batch) | **FCFS or SJF** |

The **highest-priority queue always executes first**. If it's empty, the next queue is considered.

**4. Scheduling Among Queues**

There are two ways to schedule processes among queues:

1. **Fixed Priority Scheduling** (Non-Preemptive)
   * Higher-priority queues execute first.
   * Lower queues execute **only when higher ones are empty**.
   * Example: System processes always execute before user processes.
2. **Time-Slice Scheduling** (Preemptive)
   * Each queue gets **a fixed CPU time slice**.
   * Example: The foreground queue gets **70% CPU time**, and the background queue gets **30%**.

**5. Example: Multilevel Queue in Action**

Let's assume we have 3 queues:

1. **Foreground (RR, Time Quantum = 5 ms)**
2. **Interactive (FCFS)**
3. **Background (FCFS, Lowest Priority)**

**Process Table:**

| **Process** | **Arrival Time** | **Burst Time** | **Queue Type** |
| --- | --- | --- | --- |
| P1 | 0 ms | 10 ms | Foreground |
| P2 | 2 ms | 6 ms | Foreground |
| P3 | 3 ms | 15 ms | Interactive |
| P4 | 5 ms | 8 ms | Background |

**Execution Order:**

1. **P1 (RR - Foreground)** executes for 5ms.
2. **P2 (RR - Foreground)** executes for 5ms.
3. **P1 (RR - Foreground)** executes remaining 5ms.
4. **P2 (RR - Foreground)** executes remaining 1ms.
5. **P3 (Interactive, FCFS)** executes for **15ms**.
6. **P4 (Background, FCFS)** executes for **8ms**.

**6. Advantages**

✅ **Better System Management** – Separates system and user processes.  
✅ **Efficient Scheduling** – Different scheduling for different process types.  
✅ **Improved Responsiveness** – Interactive processes get CPU quickly.

**7. Disadvantages**

❌ **Queue Starvation** – Lower-priority queues may never get CPU time.  
❌ **Complex Implementation** – Managing multiple scheduling algorithms is harder.