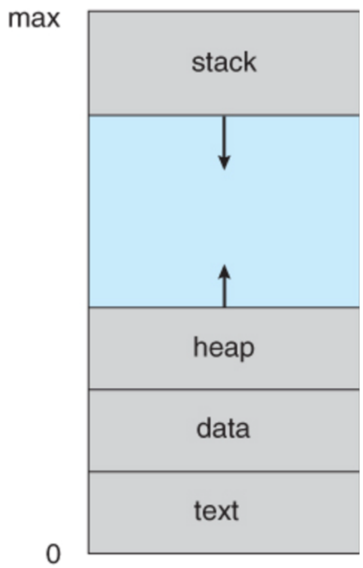
Processes

Processes

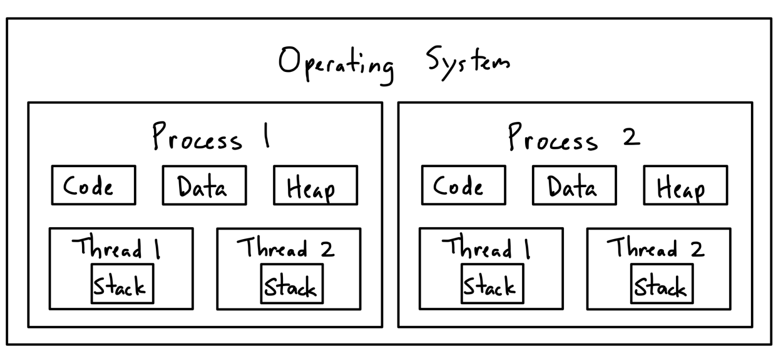
- A **process** is a program loaded into main memory, ready for execution.
- The OS assigns **isolated memory** to each process, ensuring no process sneaks into another's memory like a nosy neighbor. This memory is divided into four segments:



1. **Stack**: Contains local variables, parameters, and return addresses.
2. **Heap**: Used for dynamic memory allocation. (The program's storage unit.)
3. **Data**: Stores global variables, arrays, and structures. It's split into:
   - **BSS (Block Started by Symbol)**: For uninitialized variables.
   - **Data**: Contains initialized variables.
4. **Text**: The program's compiled machine code.

- **Memory Growth**:
  - The stack and heap grow toward each other.
    - **Stack Overflow**: When the stack crosses the line into the heap.
    - **Heap Overflow**: When the heap pushes into the stack.

Threads



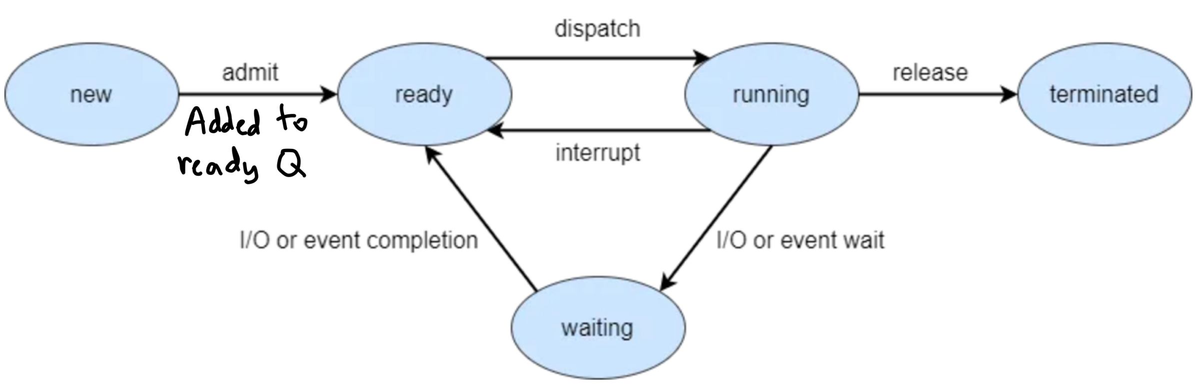
- A **process** contains at least one thread.
- A **thread** is the smallest unit of execution within a process and shares the process's memory space.
- Each thread has its own stack within the allocated memory of the process.
- **Thread Levels**:
    - Threads operate at both **user-level** and **kernel-level**.
    - In a multi-threaded environment, thread management can follow these models:
        - **Many-to-One**
        - **One-to-One**
        - **Many-to-Many**
          (Honestly, I skipped reviewing these because I know I won't remember during an interview.)

Process Control Block (PCB)

- The **Process Control Block (PCB)** is a data structure used by the OS to manage process metadata. It's like the process's personal file folder, filled with everything the OS needs to know.
- Key elements of the PCB include:
    - **PID (Process ID):** A unique identifier for the process.
    - **Parent PID:** The ID of the parent process.
    - **Child PID:** The ID of any child processes.
    - **PC (Program Counter):** The location of the next instruction to execute.
    - **Memory Limits:** Defines the boundaries of the process's memory.
    - **Other Metadata:** Includes process state, scheduling info, and I/O status.

Process Creation

- A new process (**child**) is created by **forking** (using the fork() system call) from an existing process (**parent**).
  - **fork()** Return Values:
    - The parent process receives the **child's PID**.
    - The child process receives **0**.
- All processes are created by the CPU and can exist in one of the following states:
  1. **New:** Just created, waiting to be admitted.
  2. **Ready:** Prepared to run when the CPU is available.
  3. **Running:** Actively being executed by the CPU.
  4. **Waiting:** Paused, waiting for an event (e.g., I/O operation).
  5. **Terminated:** Finished execution, now ready to rest in peace.



- **State Transitions**:
  - A process in the **ready** state moves to **running** if it has the highest priority.
  - If it times out, it returns to the **ready** state.
    - *(This timeout involves an interrupt, essentially a polite "hey CPU, your attention please!")*

Multiprocessing and Multithreading

Concurrency vs. Parallelism

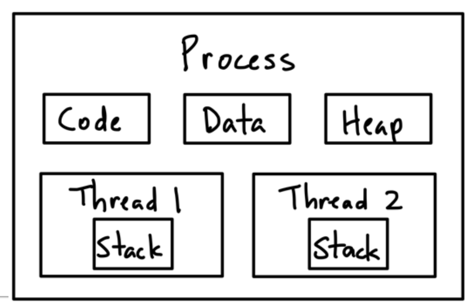
- **Concurrency**: A single core CPU switches between tasks (context switching).
- **Parallelism**: A multicore CPU executes multiple tasks simultaneously.

Multiprocessing

- **Multiprocessing** involves splitting a task into multiple processes.
  - It's **more stable** because processes operate independently, but comes with **overhead**.
  - The overhead arises from **context switching**, where the CPU pauses the current job, stores its state, and loads the new job's state. This costs both **time** and **memory**.
- Each process is allocated its own memory space, ensuring isolation.
- Since processes cannot directly access each other's memory, they communicate via **IPC (Inter-Process Communication)**.

Multithreading

- Multiple threads exist within a process, each performing different tasks.
- Threads share the **heap, data, and text** segments but have separate stacks.
- **Pros**:
  - Less overhead in context switching.
  - No need for IPC.
- **Cons**:
  - Shared stack requires synchronization.
  - A problem in one thread may affect others.



Context Switching

- **Interrupts**: Requests made to the CPU for attention due to events like:
  - I/O operations.
  - CPU usage time expiration.

    - Creation of child processes.
- **Mechanism**:
    - The CPU can only process one task at a time.
    - An interrupt, often caused by the CPU scheduler, switches the CPU to another process.
    - Overhead occurs during context switching because the CPU idles while loading a process's state into the registers.
- **Program Counter (PC) and Stack Pointer**:
    - **PC:** Holds the address of the next instruction to execute.
    - **Stack Pointer:** Points to the largest address in the stack.