1. Writing the Source Code

When a programmer writes a program, the source code (e.g., `program.c`) is saved in **secondary memory** such as a hard disk or SSD. This is done using a text editor or an IDE. At this point, only the written text (human-readable code) exists on the storage device.

---

2. Compilation Stage

During compilation, the compiler reads the source code and converts it into **machine code**, which is stored in an **executable file** (e.g., `a.out` or `program.exe`) in **secondary memory**. While compiling, the compiler also creates a memory layout that includes:

- **Code Segment** – contains the actual instructions to be executed.
- **Data Segment** – includes initialized and uninitialized global/static variables.

These segments are defined but not yet loaded into RAM. The executable file contains all this layout information and is stored in **secondary memory** using special sections inside the file:

- **.text section** → corresponds to the **code segment**.
- **.data section** → corresponds to the **initialized data segment**.
- **.bss section** → corresponds to the **uninitialized data segment**.
- **.rodata section** → stores read-only data like string literals.

So in secondary memory, the **binary file** is stored using these section headers, but they are not live segments until loaded into RAM.

---

3. Execution Stage (Run Time)

When the user runs the executable file, the **Operating System (OS)** loads it into **RAM (Primary Memory)**. During this process:

- The **code segment** and **data segment** are loaded from secondary memory into RAM.
- The **stack** and **heap** segments are created **at runtime**, but only if needed.

- **Stack** is used for function calls, local variables, and return addresses.
- **Heap** is used for dynamic memory allocation (`malloc`, `calloc`, etc.) and is created only when required.

Thus, the full memory structure in RAM at runtime includes: code, data, stack, and heap segments.

---

4. Role of User Space and Kernel Space

In any operating system, **memory is divided into two main regions**:

- **User Space**:
  - This is where **user applications** (like your program) are executed.
  - The code, data, heap, and stack segments of your running program reside in user space.
  - Each process has its own isolated user space to prevent interference from other programs.
- **Kernel Space**:
  - This area is reserved for the **Operating System kernel**, drivers, and system-level operations.
  - The kernel manages memory, process scheduling, hardware communication, and system calls.
  - User programs cannot access kernel space directly; they interact with it using system calls.

Important Note: When the OS loads your program into RAM, it does so under the control of the kernel, which maps and protects the memory regions in user space. This separation enhances security and stability.

---

5. What Happens After Program Execution

Once the program finishes execution and the output is displayed, the **Operating System deallocates the memory** that was assigned to that program. The **stack** and **heap segments** are removed first, as they are created during runtime and are no longer needed. The **code** and **data segments**, which were loaded from secondary memory, are also released from RAM.

At this point:

- Only the **binary executable file** remains in **secondary memory**, mainly stored using the **.text** (code), **.data**, and **.bss** sections.
- The **source code file** (e.g., `.c` file) remains in secondary memory as well.
- **RAM is cleared** of all segments used by the program, making memory available for other processes.

So after execution, only the **binary (machine code)** and **source code files** are present in secondary memory. These files are not live or active in RAM unless the program is re-run. The **binary file** corresponds to the **code segment structure** in the executable format, and the **source file** is just plain text stored on disk, not part of any segment.

---

6. Summary of the Full Process

1. **Write Source Code** – Stored in secondary memory.
2. **Compile Code** – Creates executable file with defined memory segments and file sections.
3. **Run Program** – OS loads code and data segments into RAM.
4. **Create Stack and Heap** – Only when needed, during runtime.
5. **After Execution** – All segments are removed from RAM; only source and binary files remain in secondary memory.
6. **User Space** – Contains all program segments (code, data, stack, heap).
7. **Kernel Space** – Controls and manages system resources and security.

This structured flow ensures that your program runs efficiently and securely, with proper memory management handled by the operating system.