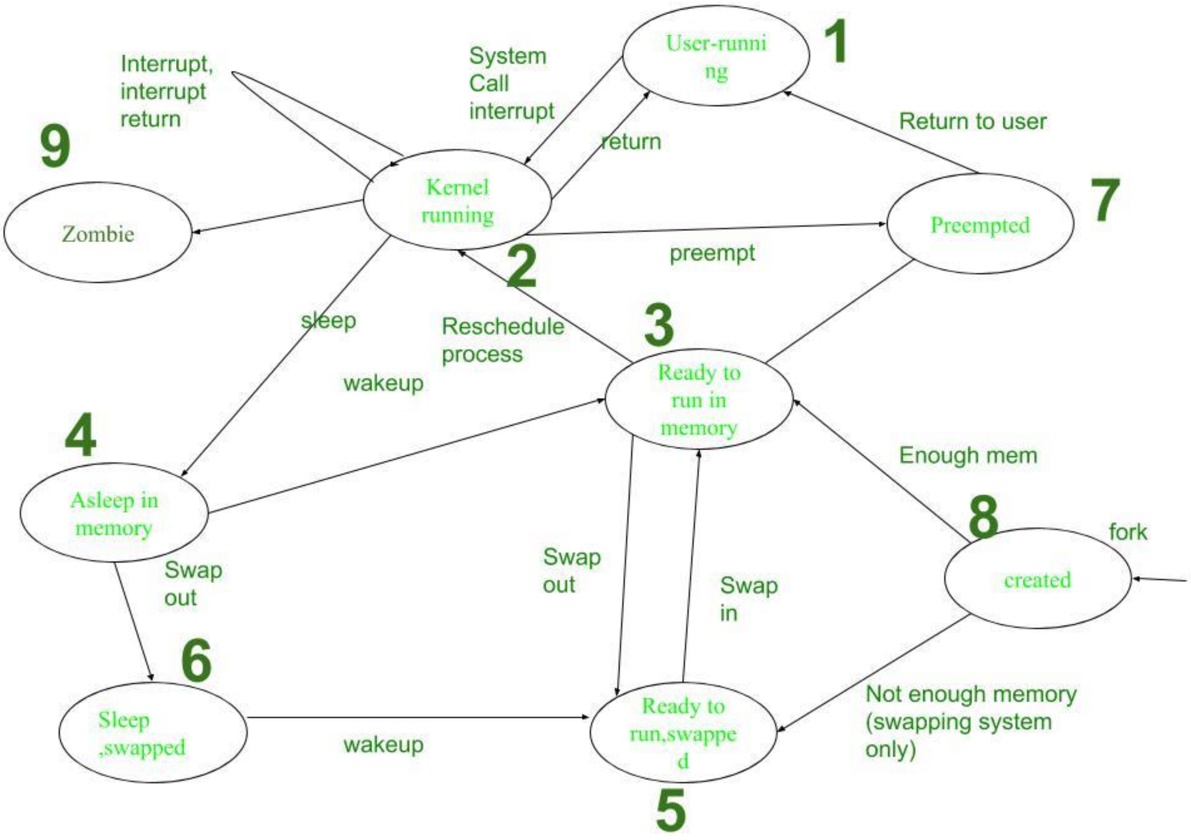
Explain in detail the process state transition in an Unix OS, and draw a neat diagram illustrating the various states a process can go through during its execution.

- Process is an instance of a program in execution. A set of processes combined together make a complete program. There are two categories of processes in Unix, namely
    1. User processes: They are operated in user mode.
    2. Kernel processes: They are operated in kernel mode.

Process States :

- The lifetime of a process can be divided into a set of states, each with certain characteristics that describe the process.



Process Transitions

The working of Process is explained in following steps:

1. **User-running:** Process is in user-running.
2. **Kernel-running:** Process is allocated to kernel and hence, is in kernel mode.
3. **Ready to run in memory:** Further, after processing in main memory process is rescheduled to the Kernel.i.e.The process is not executing but is ready to run as soon as the kernel schedules it.
4. **Asleep in memory:** Process is sleeping but resides in main memory. It is waiting for the task to begin.

5. **Ready to run, swapped:** Process is ready to run and be swapped by the processor into main memory, thereby allowing kernel to schedule it for execution.

6. **Sleep, Swapped:** Process is in sleep state in secondary memory, making space for execution of other processes in main memory. It may resume once the task is fulfilled.

7. **Pre-empted:** Kernel preempts an on-going process for allocation of another process, while the first process is moving from kernel to user mode.

8. **Created:** Process is newly created but not running. This is the start state for all processes.

9. **Zombie:** Process has been executed thoroughly and exit call has been enabled. The process, thereby, no longer exists. But, it stores a statistical record for the process. This is the final state of all processes.

Evaluate the significance of the process address space manipulation in Unix, explaining how it enables memory allocation, deallocation, and sharing among processes.

- In Unix-like operating systems, a process can transition through several states during its execution . These states represent the different stages a process goes through from creation to termination.
- The UNIX system divides its virtual address space in logically separated *regions*. The regions are contiguous area of virtual address space.
- The region table entry contains the information necessary to describe a region. In particular, it contains the following entries:
  - A pointer to the inode of the file whose contents were originally loaded into the region
  - The region type (text, shared memory, private data or stack)
  - The size of the region
  - The location of the region in physical memory.
  - The reference count.
  - The status of a region, which may be a combination of
    — in the process of being loaded into memory
    — valid, loaded into memory.
- Following are the opeartions that manipulate region :
  1. Lock a region.
  2. Unlock a region.
  3. Allocate a region.
  4. Attach a region.
  5. Change the size of a region.
  6. Load a region.
  7. Free a region.
  8. Detach a region.
  9. Duplicate a region.

Memory allocation –

- During certain system calls (such as fork, exec, and shmget for shared memory), the kernel allocates a new memory region for a process.
- The kernel maintains a region table, where each entry represents a memory region. These entries can be either on a free linked list or an active linked list.
- The allocreg algorithm ensures that processes receive locked, allocated memory regions for their execution. These regions can be shared among processes based on the associated inode.
- When allocating a region, the kernel performs the following steps:
  - Removes the first available entry from the free list.
  - Places it on the active list.
  - Locks the region to prevent concurrent modifications.
  - Marks the region as either **shared** or **private**.
  - Associates the region with an **inode** (explained below).

Deallocation / Detaching a region –

- The kernel performs region detachment during specific system calls, such as exec, exit, and shmdt.
- During this process, the kernel:

  - Updates the **pregion entry**, which contains information about the shared memory region.
  - Invalidates the associated **memory management register triple**, effectively cutting the connection to the physical memory.
  - Note that the invalidation applies specifically to the process, not to the entire shared memory region (as seen in the `freereg` algorithm).
  - Decrements the **region reference count**.
- If the region's reference count drops to 0 and there's no reason to keep the region in memory (as determined later), the kernel frees the region using the `freereg` algorithm.
- Otherwise, if the region is still needed by other processes, the kernel only releases the region and inode locks.

Describe the process of system boot and explain the role of the Init process in initializing an operating system

- The system boot process is a critical sequence of events that initializes an operating system and prepares it for user interaction.
- It involves several stages, starting from the execution of firmware instructions stored in the computer's nonvolatile memory to the initialization of essential system processes.
- **Firmware Execution:** The boot process begins with the execution of firmware instructions stored in the computer's nonvolatile memory, such as BIOS or ROM. These instructions are automatically executed when the power is turned on or the system is reset.
- **Boot Program Execution:** The firmware instructions locate and execute the system's boot program, which is typically stored in a standard location on a bootable device, such as block 0 of the root disk or a special partition. The boot program is responsible for loading the Unix kernel into memory and passing control of the system to it.
- **Kernel Initialization:** Once the kernel is loaded into memory, it initializes various system components, including device drivers, file systems, and process management. At this stage, the kernel sets up essential data structures and prepares the system for user interaction.
- **Init Process Start:** The kernel starts the Init process, also known as initialization. Init is the first user-space process started during the booting process. It continues running until the system is shut down and is responsible for managing other processes.

Roll of init process :

- Init is the parent of all processes, executed by the kernel during the booting of a system. Its principle role is to create processes from a script stored in the file **/etc/inittab**.
- Init is a daemon process that continues running until the system is shut down. It is the direct or indirect ancestor of all other processes and automatically adopts all orphaned processes.
- A kernel panic will occur if the kernel is unable to start it, or it should die for any reason. Init is typically assigned process identifier 1.
- The init process also sets the runlevel, which defines the system's mode of operation (e.g., single-user mode, multi-user mode, or graphical mode).

- **Runlevels**:
  - The **runlevel** determines which services are started during boot.
  - Common runlevels include:

- The init process transitions the system to the appropriate runlevel.

Runlevel	Mode	Action
0	Halt	Shuts down system
1	Single-User Mode	Does not configure network interfaces, start daemons, or allow non-root logins
2	Multi-User Mode	Does not configure network interfaces or start daemons.
3	Multi-User Mode with Networking	Starts the system normally.
4	Undefined	Not used/User-definable
5	X11	As runlevel 3 + display manager(X)
6	Reboot	Reboots the system