## Write and explain the algorithm for allocating a region

- 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:
  - o A pointer to the inode of the file whose contents were originally loaded into the region
  - o The region type (text, shared memory, private data or stack)
  - o The size of the region
  - o The location of the region in physical memory.
  - o The reference count.
  - o The status of a region, which may be a combination of
    - — in the process of being loaded into memory
    - — valid, loaded into memory.
- The kernel allocates a new region (algorithm allocreg) during fork, exec, and shmget (shared memory) system calls.
- The kernel contains a region table whose entries appear either on a free linked list or on an active linked list
- When it allocates a region table entry, the kernel removes the first available entry from the free list, places it on the active list, locks the region, and marks its type (shared or private).
- These regions can be shared among processes based on the associated inode.
- When allocating a region, the kernel performs the following steps:
  - o Removes the first available entry from the free list.
  - o Places it on the active list.
  - o Locks the region to prevent concurrent modifications.
  - o Marks the region as either shared or private.
  - o Associates the region with an inode (explained below).

*Algorithm allocreg /*allocate a region data structure*/*
*Input: (1) inode pointer*
*(2) Region type*
*Output: locked region*
*{*
*remove region from linked list of free regions;*
*assign region type;*
*assign region inode pointer;*

```
if (inode pointer not null)
increment inode reference count;
place region on linked list of active regions;
return(locked region);
}
```

## Explain with a diagram the context of a process in detail.

- The kernel contains a process table with an entry that describes the state of every active process in the system. The u area contains additional information that controls the operation of a process. The process table entry and the u area are part of the context of a process.
- The context of a process consists of the contents of its (user) address space and the contents of hardware registers and kernel data structures that relate to the process.
- Formally, the context of a process is the union of its user-level context, register context, and system-level context
- The user-level context consists of the process text, data, user stack, and shared memory that occupy the virtual address space of the process.
- Parts of the virtual address space of a process that periodically do not reside in main memory because of swapping or paging still constitute a part of the user-level context.

The register context consists of the following components:

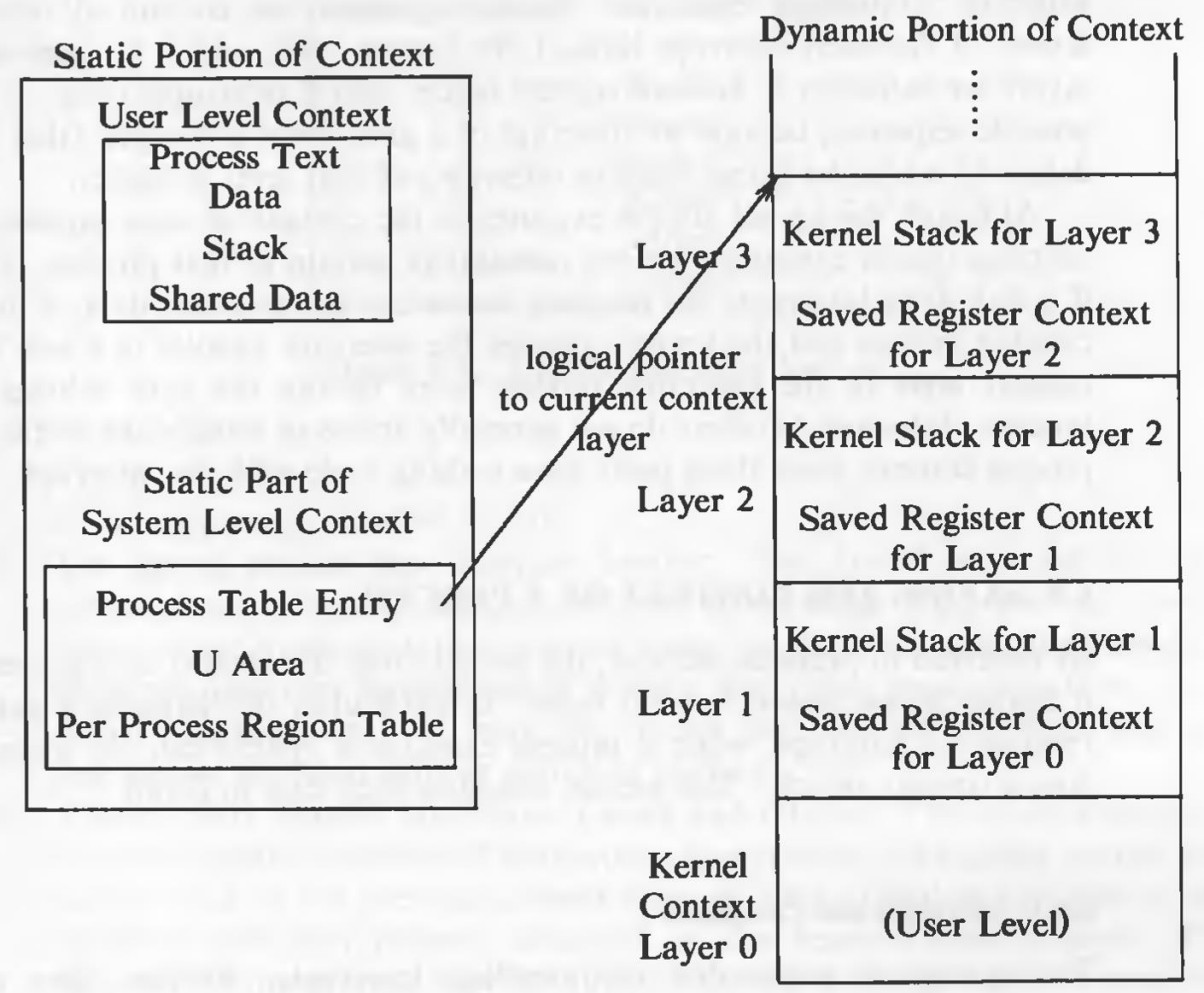
- Program counter specifies the next instruction to be executed. It is an address in kernel or in user address space.
- The processor status register (PS) specifies hardware status relating the process. It has subfields which specify if last instruction overflowed, or resulted in 0, positive or negative value, etc. It also specifies the current processor execution level and current and most recent modes of execution (such as kernel, user).
- The stack pointer points to the current address of the next entry in the kernel or user stack. If it will point to next free entry or last used entry it dependent on the machine architecture. The direction of the growth of stack (toward numerically higher or lower addresses) also depend on machine architecture.
- The general purpose registers contain data generated by the process during its execution.
- The system level context has a "static part" and a "dynamic part". A process has one static part throughout its lifetime. But it can have a variable number of dynamic parts.

The static part consists of the following components:

- The process table entry

- The u-area
- Pregion entries, region tables and page tables.

**The following figure shows the components that form the context of a process:**



## What is a region? Discuss mapping between the per-process region table and page table.

- The UNIX system divides its virtual address space in logically separated regions. The regions are contiguous area of virtual address space of a process that can be treated as a distinct object to be shared or protected.
- The region table entry contains the information necessary to describe a region. In particular, it contains the following entries:

o A pointer to the inode of the file whose contents were originally loaded into the region

o The region type (text, shared memory, private data or stack)

o The size of the region

o The location of the region in physical memory.

o The reference count.

o The status of a region, which may be a combination of

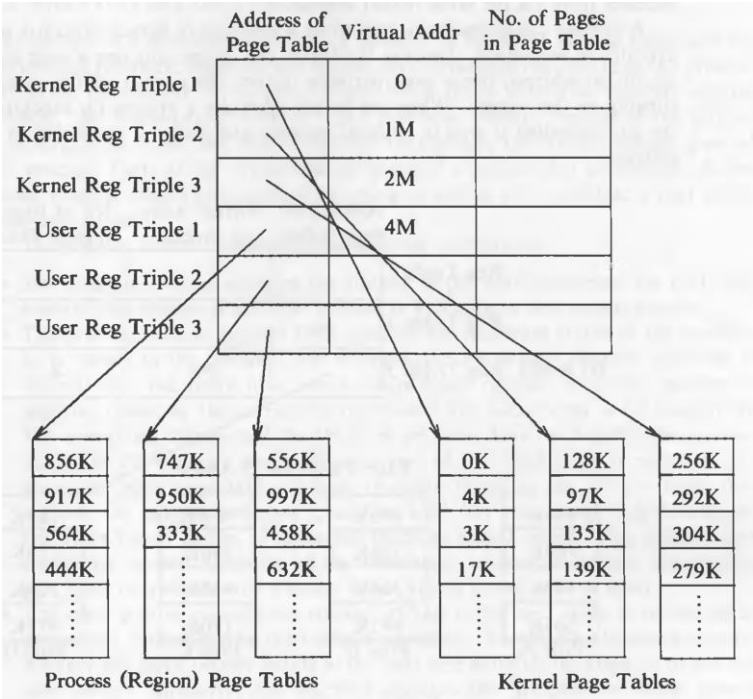
— in the process of being loaded into memory

— valid, loaded into memory.

- Each process contains a private *per process regions table*, called a *pregion*. The pregion entry contains a pointer to an entry in the region table, and contains starting virtual address of the region.
- *pregion* are stored in process table, or u-area, or a separately allocated memory space, according to the implementation.
- The pregion entries contain the access permissions: read-only, read-write, or read-execute. The pregion and the region structure is analogous to file table and the in-core inode table.
- But since, pregions are specific to a process, *pregion* table is private to a process, however the file table is global. Regions can be shared amongst processes.
- In a memory model based a pages, the physical memory is divided into equal sized blocks called *pages*. Page sizes are usually between 512 bytes to 4K bytes, and are defined by the hardware.
- The kernel maintains a mapping of logical to physical page numbers in a table which looks like this:

| Logical Page Number | Physical Page Number |
|---|---|
| 0 | 177 |
| 1 | 54 |
| 2 | 209 |
| 3 | 17 |

- Mapping of logical to physical page numbers. These tables are called page tables.
- When in user mode, access to kernel page tables is prohibited. When the process switches to kernel mode, only then it can access kernel page tables. Some system implementations try to allocate the same physical pages to the kernel, keeping the translation function, an identity function.
- Example:



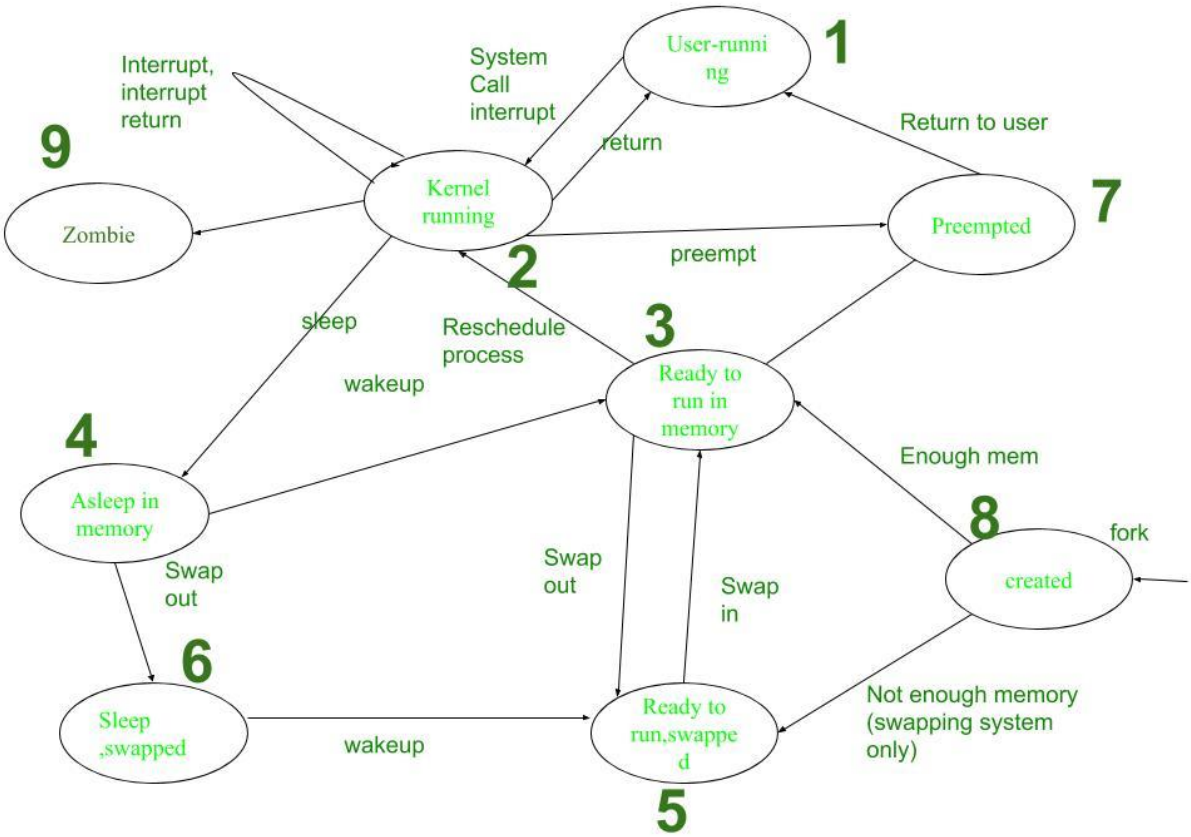
**Explain in detail the process state transition in an Unix OS, and draw a neat diagram illustrating the various states a process can go through during its execution.**

- 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**

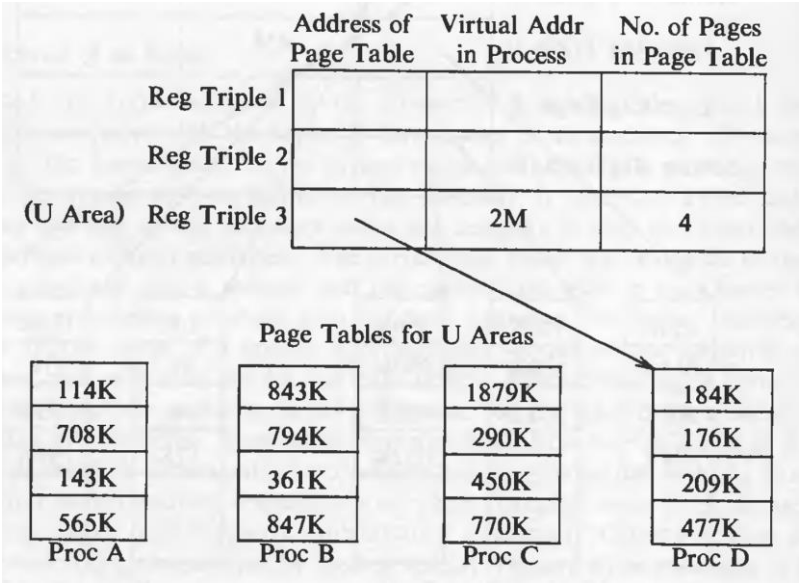
The working of Process is explained in following steps:

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.

## What is the U area? List fields from the U area

- Every process has a private u area, yet the kernel accesses it as if there were only one u area in the system, that of the running process.
- The kernel changes its virtual address translation map according to the executing process to access the correct u area.
- When compiling the operating system, the loader assigns the variable u, the name of the u area, a fixed virtual address.
- The value of the u area virtual address is known to other parts of the kernel, in particular, the module that does the context switch.
- The kernel knows where in its memory management tables the virtual address translation for the u area is done, and it can dynamically change the address mapping of the u area to another physical address.
- The two physical addresses represent the u areas of two processes, but the kernel accesses them via the same virtual address.
- A process can access its u area when it executes in kernel mode but not when it executes in user mode.
- Because the kernel can access only one u area at a time by its virtual address, the u area partially defines the context of the process that is running on the system.
- When the kernel schedules a process for execution, it finds the corresponding u area in physical memory and makes it accessible by its virtual address.




- kernel text and data (the addresses and pointers are not shown), and the third triple refers to the u area for process D.

- If the kernel wants to access the u area of process A, it copies the appropriate page table information for the u area into the third register triple.
- At any instant, the third kernel register triple refers to the u area of the currently running process, but the kernel can refer to the u area of another process by overwriting the entries for the u area page table address with a new address.
- The entries for register triples 1 and 2 do not change for the kernel, because all processes share kernel text and data.

## What is a context switch? Explain the steps for the Context switch

- Context switching in an operating system involves saving the context or state of a running process so that it can be restored later, and then loading the context or state of another. process and run it.
- As seen previously, the kernel permits a context switch under 4 situations:
  - When a process sleeps
  - When a process exits
  - When a process returns from a system call to user mode but is not the most eligible process to run.
  - When a process returns from an interrupt handler to user mode but is not the most eligible process to run.
- The procedure for a context switch is similar to handling interrupts and system calls.
- Following are the steps involved in context switching –

Decide whether to do a context switch, and whether a context switch is permissible now:

- Before initiating a context switch, the operating system evaluates whether it's necessary to switch to another process.
- Various factors influence this decision, including time quantum expiration, I/O completion, or a higher-priority process becoming ready to execute.
- Additionally, the system must ensure that a context switch is permissible at the current moment, considering system constraints, priorities, and any critical sections that should not be interrupted.

Save the context of the "old" process:

- Once the decision to switch processes is made, the operating system saves the context of the currently executing or "old" process.
- This involves preserving the state of the process by saving critical data such as the contents of the processor's registers, program counter, stack pointer, and any other relevant execution state.

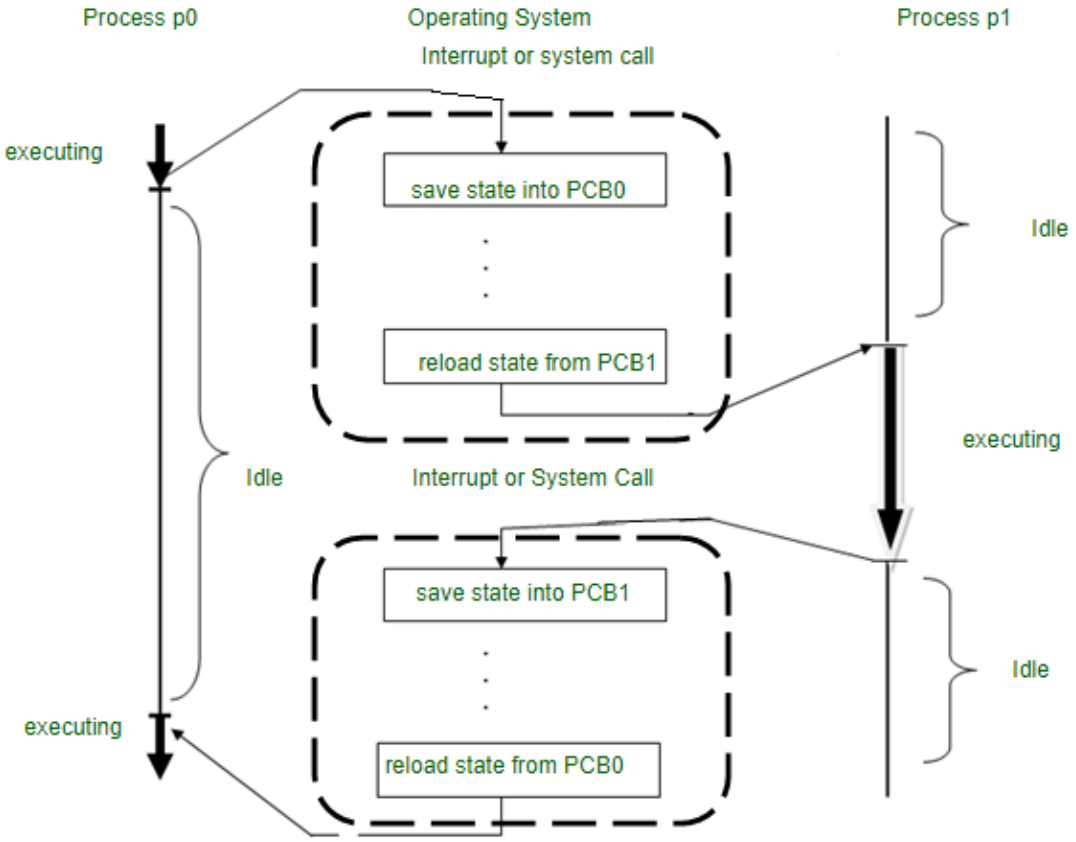
- Furthermore, system-level context, such as memory mappings, file descriptors, and open sockets, may also need to be saved to maintain process integrity.

Find the "best" process to schedule for execution, using the process scheduling algorithm:

- With the context of the old process saved, the operating system selects the next process to execute.
- This selection is based on the process scheduling algorithm, which considers factors such as process priority, CPU burst time, and scheduling policy.
- The goal is to identify the most suitable process for execution, balancing system performance, responsiveness, and fairness.

Restore the context of the selected process:

- Once the new process is chosen, the operating system restores its context from the previously saved state.
- This involves loading the saved data, including the contents of the processor's registers, program counter, stack pointer, and any other relevant execution state.
- Additionally, system-level context, such as memory mappings, file descriptors, and open sockets, are restored to ensure the process can continue its execution seamlessly.


Process p0
Operating System
Process p1
Interrupt or system call
executing
save state into PCB0
reload state from PCB1
Idle
executing
Idle
Interrupt or System Call
save state into PCB1
reload state from PCB0
Idle
executing

## List and explain the fields of process table.

- Two kernel data structures describe the state of a process: the process table entry and the u area.
- The process table contains fields that must always be accessible to the kernel, but the u area contains fields that need to be accessible only to the running process.
- Therefore, the kernel allocates space for the u area only when creating a process: It does not need u areas for process table entries that do not have processes.
- The fields in the process table are the following:

  - **State** - The state field identifies the process state.
  - The process table entry contains fields that allow the kernel to locate the process and its u area in main memory or in secondary storage. The kernel uses the information to do a context switch to the process when the process moves from state "ready to run in memory" to the state "kernel running" or from the state "preempted" to the state "user running."
  - **ID** - Several user identifiers (UIDs) determine various process privileges.
  - **Process identifiers -** Process identifiers (PIDs) specify the relationship of processes to each other. These ID fields are set up when the process enters the state "created" in the fork system call
  - **Event descriptor** - The process table entry contains an event descriptor when the process is in the "sleep" state.
  - **Scheduling parameters** - Scheduling parameters allow the kernel to determine the order in which processes move to the states "kernel running" and "user running".
  - **Signa**l - A signal field enumerates the signals sent to a process but not yet handled.
  - **user-set timer** - Various timers give process execution time and kernel resource utilization, used for process accounting and for the calculation of process scheduling priority. One field is a user-set timer used to send an alarm signal to a process.