Linux Interview Essentials – Part IV (Process management)

1. **What is scheduling?**

   The act of determining which process in the ready state should be moved to the running state is known as Process Scheduling.
   The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs. For achieving this, the scheduler must apply appropriate rules for swapping processes IN and OUT of CPU.

2. **Explain context switching in detail.**

   Context switching is the procedure of storing the state of an active process for the CPU when it has to start executing a new one. For example, process A with its address space and stack is currently being executed by the CPU and there is a system call to jump to a higher priority process B; the CPU needs to remember the current state of the process A so that it can suspend its operation, begin executing the new process B and when done, return to its previously executing process A.
   - Suspending the progression of one process and storing the CPU's state (i.e., the context) for that process somewhere in memory.
   - Retrieving the context of the next process from memory and restoring it in the CPU's registers.
   - Returning to the location indicated by the program counter (i.e., returning to the line of code at which the process was interrupted) in order to resume the process.

3. **During which process states CPU scheduling decisions are made? Explain**

   CPU scheduling decisions take place under one of four conditions:
   - When a process switches from the running state to the waiting state, such as for an I/O request or invocation of the wait() system call.
   - When a process switches from the running state to the ready state, for example in response to an interrupt.
   - When a process switches from the waiting state to the ready state, say at completion of I/O or a return from wait().
   - When a process terminates.

4. **What is pre-emptive and non-pre-emptive scheduling algorithms?**

   - **Non pre-emptive scheduling:** When the currently executing process gives up the CPU voluntarily.
   - **Pre-emptive scheduling:** When the operating system decides to favor another process (High priority), pre-empting the currently executing process.

5. **Briefly explain the following algorithms:**

   - FCFS:
     1. Round Robin – Time slice based
        In RR time slices are assigned to each process in equal portions and in circular order, handling all processes without priority (also known as cyclic executive). Round-robin scheduling is simple, easy to implement, and starvation-free.
     2. Round Robin – Priority based
        It works same as RR time slice, but we can also assign priority to process. More priority means large time slice.
   - Priority based:
     1. Rate monotonic
        Rate-monotonic scheduling (RMS) is a scheduling algorithm used in real-time operating systems (RTOS) with a static-priority scheduling class. The static priorities are assigned according to the cycle duration of the job, so a shorter cycle duration results in a higher job priority. Smaller the period, higher the priority.
     2. Earliest deadline first

Earliest deadline first (EDF) or least time to go is a dynamic scheduling algorithm used in real-time operating systems. A priority queue will be maintained and will be searched for the process closest to its deadline. This process is the next to be scheduled for execution.

6. **What is a real-time systems (RTS)? What are the different types of RTS?**

   Real Time Operating System (RTOS) is designed to provide a predictable (normally described as deterministic) execution pattern. This is particularly of interest to embedded systems as embedded systems often have real time requirements. A real time requirement is one that specifies that the embedded system must respond to a certain event within a strictly defined time (the deadline). A guarantee to meet real time requirements can only be made if the behavior of the operating system's scheduler can be predicted.

   Soft-RTOS
   Soft real time systems can miss some deadlines, but eventually performance will degrade if too many are missed. Example – multimedia streaming, computer games
   Hard-RTOS
   Hard real-time means you must absolutely hit every deadline. Very few systems have this requirement. Some examples are nuclear systems, some medical applications such as pacemakers, a large number of defense applications, avionics etc. Where life critical situation we have to use hard real time system.

7. **Explain differences between General Purpose Operating System (GPOS), Real Time Operating System (RTOS) and Embedded Operating System (EOS)**

   - GPOS

   General Purpose Operating System (GPOS) is a software that provides interface to the user to access various applications. These are running in a commonly used systems & run applications like desktop PCs (ex: Windows 10). They run on a general purpose hardware.

   - RTOS

   Real Time Operating System (RTOS) is a software that has certain time constraint. Such OS operate with stringent time expectations to deliver a predictable response to use applications (ex: robotics). This OS can run either on a general or specific hardware that is designed.

   - EOS

   Embedded OS (EOS) is a software that runs on a specific hardware that runs specific set of applications. Along with running in a specific target, EOS will have resource constraints (ex: size) hence it should be customizable and configurable to meet embedded system needs.

8. **Is Linux a RTOS? Explain your justification**

   Linux is not RTOS but RTLinux is a RTOS version of Linux OS. RTLinux is a hard real-time RTOS microkernel that runs the entire Linux operating system as a fully preemptive process.
   Ref Qn 6

9. **What is real time scheduling?**

   Rate monotonic and earliest deadline first are real time scheduling. Refer Qn 5

10. **What specific situations you need to take care while building an RTS?**

    - Reliability
    - Predictability
    - Performance
    - Compactness
    - Scalability
    - User control over OS Policies
    - Responsiveness
      - Fast task switch

- Fast interrupt response