Real-time Operating System

Syllabus

- RTOS concepts
  - Definition
  - RTOS vs GPOS
  - Types of RTOS
    - Traditional/Embedded RTOS
    - Real-time Linux
    - Linux based RTOS
  - Task management
  - Task scheduling
  - Inter-task communication
  - Synchronization
  - Performance metrics
  - RTOS porting
  - RTOS practicals
  - FreeRTOS - Embedded RTOS

Evaluation

- Theory: 40 marks - MCQ (CCEE)
- Lab: 40 marks - FreeRTOS
- Internals: 20 marks

Pre-requisites

- Operating System
- Linux
- C Programming
- Computer fundamentals

RTOS

- RTOS is OS in which accuracy of result is not only dependent on "correctness of the calculation", but also depeneds on "time duration" in which results are produced.
- Based on timing requirements there are three types of RTOS
  - Hard realtime OS
  - Soft realtime OS
  - Firm realtime OS

Characteristics of RTSO

- Consistentency
- Reliability
- Predictability

- Performance
- Scalability

Hard realtime OS

- If interrupt/tasks deadline miss -- catastrophic effect.
- Usually doesn't have secondary storage.
- Timing: 10 to 100 "usec"
- E.g. uC-OS, FreeRTOS, Xenomai, RTAI, ...

Soft realtime OS

- Less time critical - If deadline miss, affect product quality (not catastrophic).
- May have secondary storage
- Timing: 1 ms to 10 ms
- E.g. Linux (PREEMP_RT), ...

Firm realtime OS

- Like hard realtime.
- Rare miss of deadline is acceptable (not catastrophic).

Functions of RTOS

- Task Management
- Scheduling
- Resource Allocation
- Interrupt Handling

GPOS vs RTOS

GPOS vs RTOS -- Similiraties

```
1. Both provides hardware abstraction.
2. Both provides some level of multi-tasking.
3. Both does resource management.
4. Both provides system services (syscalls) for applications.
```

GPOS vs RTOS -- Differences

1. Customization Support:
    - GPOS is little or no customizable (Linux is exception).
    - RTOS is fully customizable - so that it can be used with minimal memory.
2. Interrupt Latencies:
    - GPOS have higher interrupt latencies i.e. in msec.
    - RTOS handle interrupts in deterministic time and with lower latencies i.e. in usec.
3. IPC Latencies:

   - In GPOS, IPC buffers are allocated at runtime (which may block).
   - In GPOS, task awakening is not real-time.
   - In GPOS, signal handling is not real-time.
   - In RTOS, IPC buffers are preallocated (system heap).
   - In RTOS, task awakening is done in deterministic time.
   - In RTOS, signals are processed in real-time.
4. Memory Requirements:
   - GPOS uses more memory for IPC mechanisms, tasks, etc.
   - RTOS should have minimal memory footprints as compared to GPOS, so that it can smoothly work on low end embedded devices.
5. Disk Support:
   - GPOS has full support for secondary storage devices.
   - RTOS may have little or no support for secondary storage devices, as they are slowest devices.
6. CPU arch support:
   - GPOS usually supports many different architectures (e.g. x86, x86_64, ARM, ...).
   - Most of the RTOSes has limited number of supported arch, because RTOS is dependent on most of hardware features.
7. Scheduling Policies:
   - GPOS supports soft-real time policies (FIFO,RR) as well as non-real time policies (NORMAL, BATCH & IDLE).
   - RTOS supports only real-time policies (FIFO,RR).
8. Timer Management:
   - In GPOS, timer hardware can be fine-grained (1ms) or coarse-grained (10ms). Also timer frequency is set during booting and cannot be reprogrammed.
   - In RTOS, timer must be fine-grained (1ms). Timer is re-programmable by special APIs proivided in RTOS. Typically timer can be programmed in one-shot mode or periodic mode.
9. Interrupt Management:
   - In GPOS, interrupts are processed in two steps i.e. top half (non-blocking code) and bottom half (blocking code) to ensure minimal interrupt latency.
   - In RTOS, ISRs are minimal & non-blocking. Also interrpt handlers are executed as highest priority tasks, so that they will be executed before any other task.
10. IO Subsystem & Device Drivers:
    - In GPOS, drivers have higher latencies, because they deal with rest of the OS through multiple layers.
    - In RTOS, drivers and tasks are present in same address space (i.e. kernel space), so that minimal latecies are ensured.
11. Task Management:
    - In GPOS, processes & threads are heavy-weight i.e. have higher memory requirements.
    - The RTOS tasks are light-weight i.e. with minimum memory requirements.

CPU Scheduing

Process Life Cycle

Process States

- New

- New process PCB is created and added into job queue. PCB is initialized and process get ready for execution.
- Ready
  - The ready process is added into the ready queue. Scheduler pick a process for scheduling from ready queue and dispatch it on CPU.
- Running
  - The process runs on CPU. If process keeps running on CPU, the timer interrupt is used to forcibly put it into ready state and allocate CPU time to other process.
- Waiting
  - If running process request for IO device, the process waits for completion of the IO. The waiting state is also called as sleeping or blocked state.
- Terminated
  - If running process exits, it is terminated.
- Linux: TASK_RUNNING (R), TASK_INTERRUPTIBLE (S), TASK_UNINTERRUPTIBLE (D), TASK_STOPPED(T), TASK_ZOMBIE (Z), TASK_DEAD (X)

Types of Scheduling

Non-preemptive

- The current process gives up CPU voluntarily (for IO, terminate or yield).
- Then CPU scheduler picks next process for the execution.
- If each process yields CPU so that other process can get CPU for the execution, it is referred as "Co-operative scheduling".

Preemptive

- The current process may give up CPU voluntarily or paused forcibly (for high priority process or upon completion of its time quantum)

Scheduling criteria's

CPU utilization: Ideal - max

```
* On server systems, CPU utilization should be more than 90%.
* On desktop systems, CPU utilization should around 70%.
```

Throughput: Ideal - max

```
* The amount of work done in unit time.
```

```
```

Waiting time: Ideal - min

```
* Time spent by the process in the ready queue to get scheduled on the CPU.
* If waiting time is more (not getting CPU time for execution) -- Starvation.
```

Turn-around time: Ideal - CPU burst + IO burst

```
* Time from arrival of the process till completion of the process.
* CPU burst + IO burst + (CPU) Waiting time + IO Waiting time
```

Response time: Ideal - min

```
* Time from arrival of process (in ready queue) till allocated CPU for first time.
```

Scheduling Algorithms

FCFS

- Process added first in ready queue should be scheduled first.
- Non-preemptive scheduling
- Scheduler is invoked when process is terminated, blocked or gives up CPU is ready for execution.
- Convoy Effect: Larger processes slow down execution of other processes.

SJF

- Process with lowest burst time is scheduled first.
- Non-preemptive scheduling
- Minimum waiting time

SRTF - Shortest Remaining Time First

- Similar to SJF - but Preemptive scheduling
- Minimum waiting time

Priority

- Each process is associated with some priority level. Usually lower the number, higher is the priority.
- Preemptive scheduling or Non Preemptive scheduling
- Starvation
  - Problem may arise in priority scheduling.
  - Process not getting CPU time due to other high priority processes.
  - Process is in ready state (ready queue).

  - May be handled with aging -- dynamically increasing priority of the process.
- Deadlock
  - Problem may arise due to synchronization when four conditions hold true simultaneously (No preemption, Mutual exclusion, Hold & wait, Circular wait).
  - Processes involved in deadlock are blocked (waiting queue of IO device/Sync object).
  - No (peaceful) solution for deadlock.

Round-Robin

- Preemptive scheduling
- Process is assigned a time quantum/slice.
- Once time slice is completed/expired, then process is forcibly preempted and other process is scheduled.
- Min response time.

Fair-share

- CPU time is divided into epoch times.
- Each ready process gets some time share in each epoch time.
- Process is assigned a time share in proportion with its priority.
- In Linux, processes with time-sharing (TS) class have nice value. Range of nice value is -20 (highest priority) to +19 (lowest priority).