Preemptive real-time operating systems

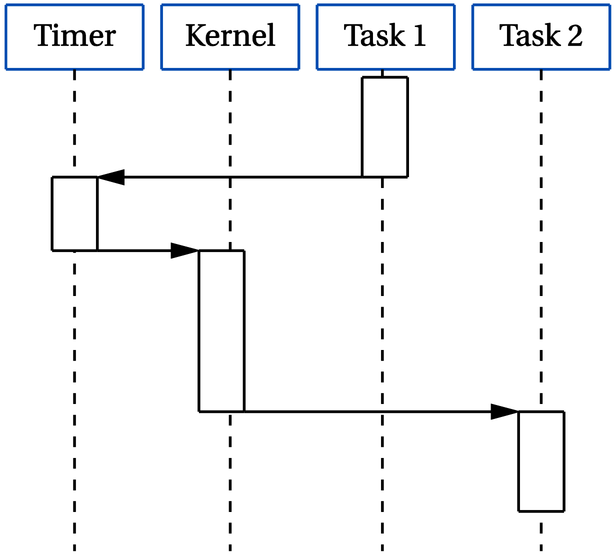
- A preemptive real-time operating system solves the fundamental problems of a cooperative multitasking system.

- It executes processes based upon timing requirements provided by the system designer.

- The most reliable way to meet timing requirements accurately is to build a preemptive operating system and to use priorities to control what process runs at any given time.

Preemptive scheduling

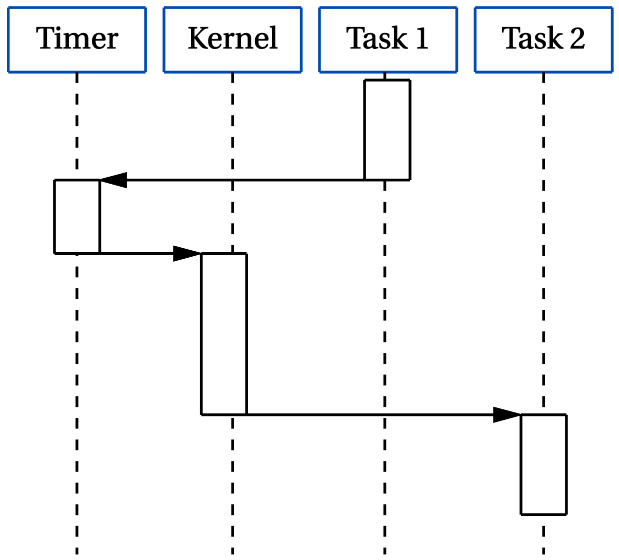
- Fig. shows an example of **preemptive execution** of an operating system.
- We want to share the CPU across two processes. The **kernel** is the part of the operating system that determines what process is running.
- The kernel is activated periodically by the timer.
- The length of the timer period is known as the time **quantum** because it is the **smallest increment** in which we can control CPU activity.
- The **kernel determines** what process will run next and causes that process to run.
- On the next **timer interrupt**, the kernel may pick the same process or another process to run.



Sequence diagram for preemptive execution

Preemptive scheduling

- Timer interrupt gives CPU to kernel.
  - Time quantum is smallest increment of CPU scheduling time.
- Kernel decides what task runs next.
- Kernel performs context switch to new context.



Sequence diagram for preemptive execution

Context switching

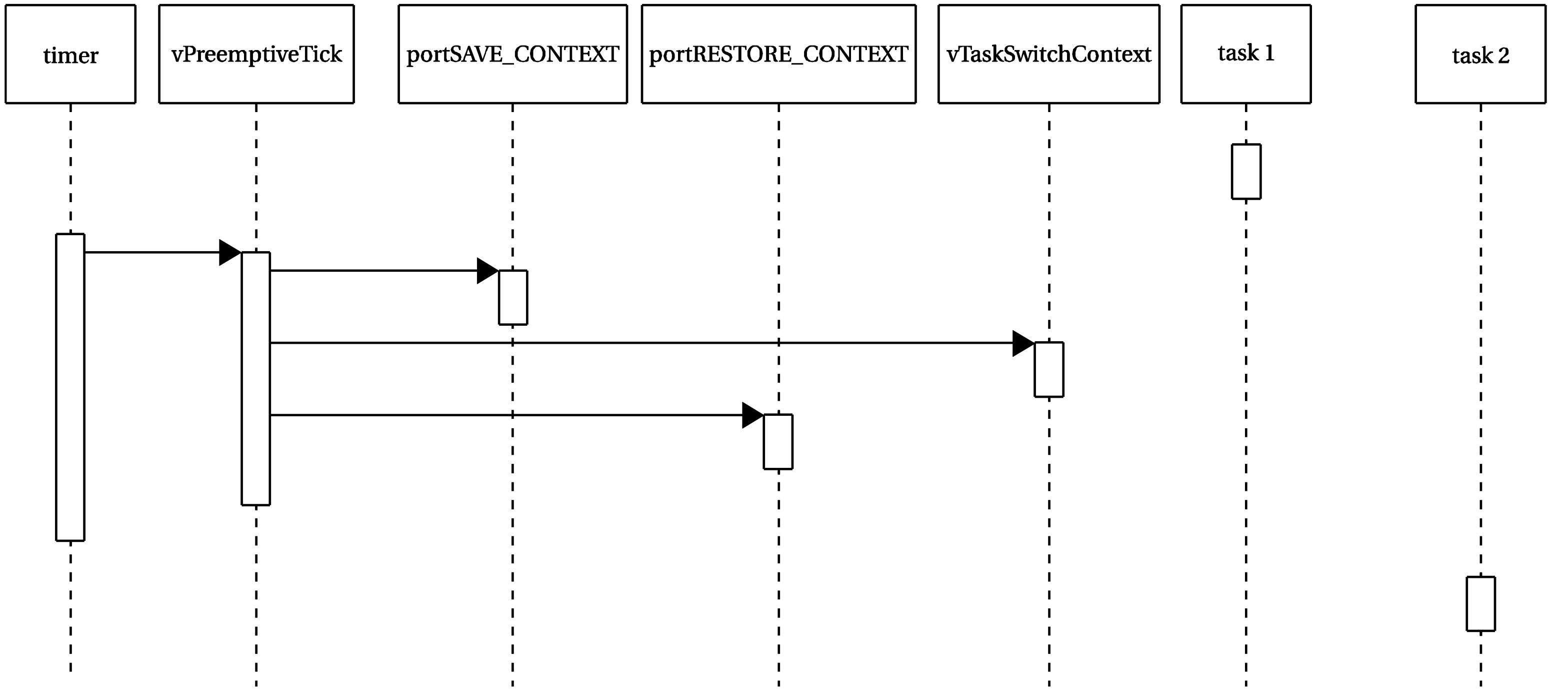
- Set of registers that define a process's state is its context.
  - Stored in a record.
- Context switch moves the CPU from one process's context to another.
- Context switching code is usually assembly code.
  - Restoring context is particularly tricky.

freeRTOS.org context switch

This diagram shows the application tasks, the hardware timer, and all the functions in the kernel that are involved in the context switch:

- PreemptiveTick() is called when the timer ticks.
- SIG_OUTPUT_COMPARE1A responds to the timer interrupt and uses portSAVE_CONTEXT() to swap out the current task context.
- TaskIncrementTick() updates the time and vTaskSwitchContext chooses a new task.
- portRESTORE_CONTEXT () swaps in the new context.

freeRTOS.org context switch



Sequence diagram for a freeRTOS.org context switch

Priority-driven scheduling

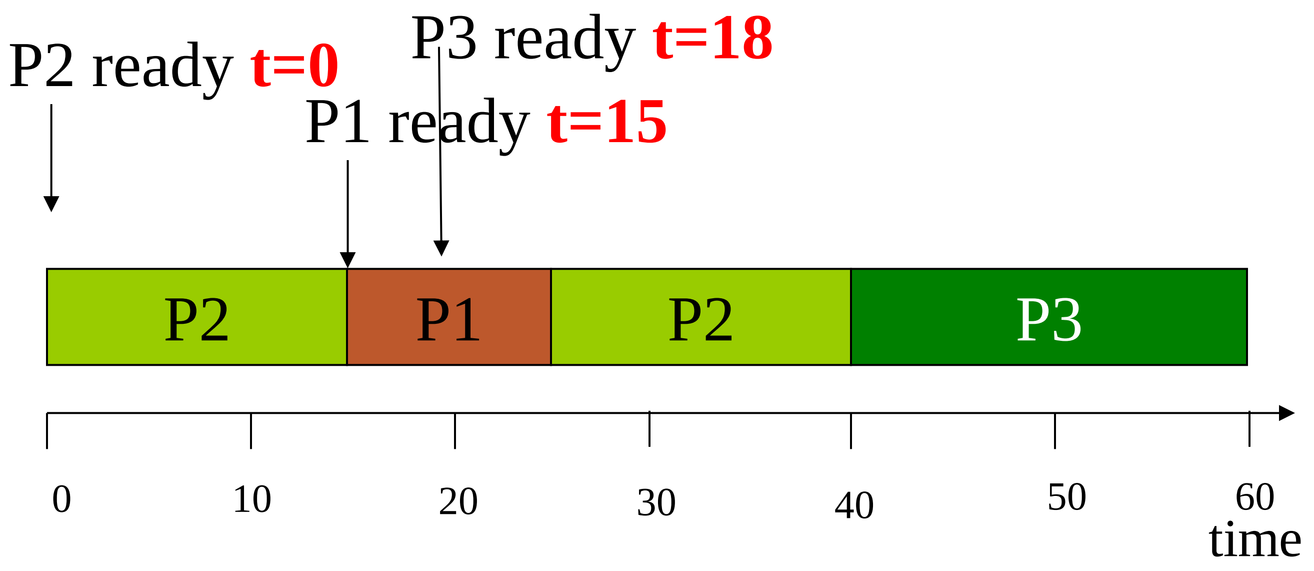
- Each process has a priority.
- CPU goes to highest-priority process that is ready.
- Priorities determine scheduling policy:
  - fixed priority;
  - time-varying priorities.

Priority-driven scheduling example

- Rules:
  - each process has a fixed priority (1 highest);
  - highest-priority ready process gets CPU;
  - process continues until done.

- Processes
  - P1: priority 1, execution time 10
  - P2: priority 2, execution time 30
  - P3: priority 3, execution time 20

Priority-driven scheduling example



When the system begins execution, P2 is the only ready process, so it is selected for execution. At time 15, P1 becomes ready; it preempts P2 and begins execution because it has a higher priority. Because P1 is the highest-priority process in the system, it is guaranteed to execute until it finishes. P3's data arrive at time 18, but it cannot preempt P1. Even when P1 finishes, P3 is not allowed to run. P2 is still ready and has higher priority than P3. Only after both P1 and P2 finish can P3 execute.

The scheduling problem

- Can we meet all deadlines?
  - Must be able to meet deadlines in all cases.
- How much CPU horsepower do we need to meet our deadlines?

Process initiation disciplines

- Periodic process: executes on (almost) every period.
- Aperiodic process: executes on demand.
- Analyzing aperiodic process sets is harder---must consider worst-case combinations of process activations.

Timing requirements on processes

- Period: interval between process activations.
- Initiation interval: reciprocal of period.
- Initiation time: time at which process becomes ready.
- Deadline: time at which process must finish.

Scheduling metrics

- How do we evaluate a scheduling policy:
  - Ability to satisfy all deadlines.
  - CPU utilization---percentage of time devoted to useful work.
  - Scheduling overhead---time required to make scheduling decision.

Rate monotonic scheduling

- RMS (Liu and Layland): widely-used, analyzable scheduling policy.

- Analysis is known as Rate Monotonic Analysis (RMA).

RMA model

- All process run on single CPU.
- Zero context switch time.
- No data dependencies between processes.
- Process execution time is constant.
- Deadline is at end of period.
- Highest-priority ready process runs.

RMS Priorities

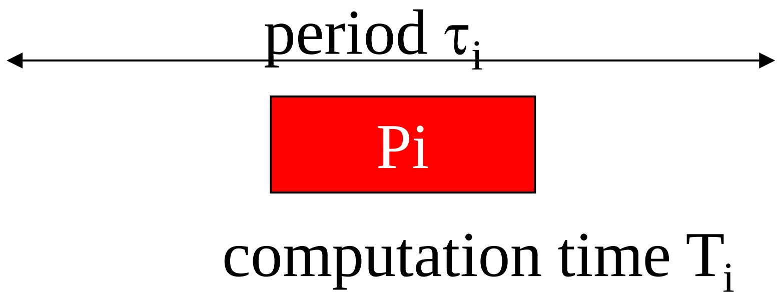
- Optimal (fixed) priority assignment:
  - shortest-period process gets highest priority;
  - priority inversely proportional to period;
  - break ties arbitrarily.
- No fixed-priority scheme does better.

Rate-monotonic analysis

- Response time: time required to finish process.
- Critical instant: scheduling state that gives worst response time.
- Critical instant occurs when all higher-priority processes are ready to execute.

Process parameters

T_i is computation time of process i; τ_i is period of process i.



Why shortest period gets highest priority?

- consider two processes P1 and P2.
  - P1 has a period τ_1 and an execution time T_1
  - P2 has a period τ_2 and an execution time T_2
  - $\tau_1 < \tau_2$
- If we give higher priority for P1 then

$$\left\lfloor \frac{\tau_2}{\tau_1} \right\rfloor T_1 + T_2 \leq \tau_2$$

- If we give higher priority for P2 then

$$T_1 + T_2 \leq \tau_1$$

- Clearly the first constraint is easier to satisfy

RMS CPU utilization

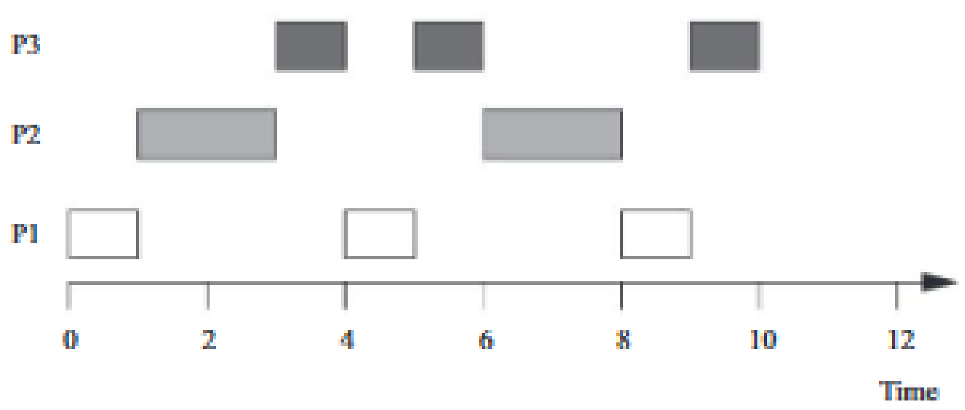
- Utilization for n processes is
  - $U=\sum_i T_i / \tau_i$
- Given *m* tasks and ratio between any two periods less than 2:
  - $U = m\left(2^{1/m} - 1\right)$
- As number of tasks approaches infinity, maximum utilization approaches 69%.

RMS CPU utilization, cont'd

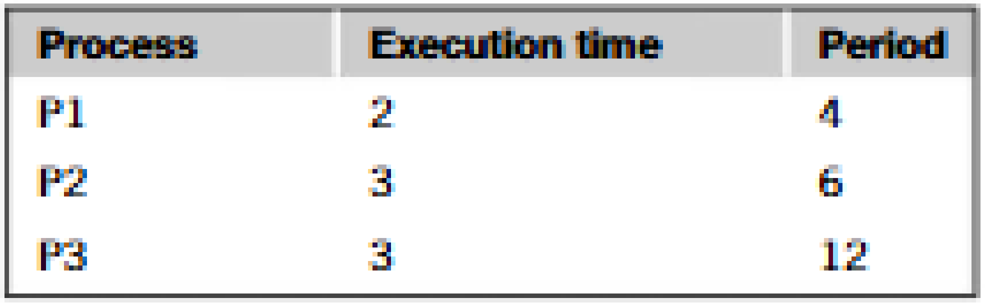
- RMS may not be able to use 100% of CPU, even with zero context switch overhead.
- Must keep idle cycles available to handle worst-case scenario.
- However, RMS guarantees all processes will always meet their deadlines as long as combining all processes do not exceed CPU cycle time.
- Efficient implementation:
  - scan processes;
  - choose highest-priority active process.

Working Example: utilization <100%

Process	Execution time	Period
P1	1	4
P2	2	6
P3	3	12



Non working Example: utilization>100%



Process	Execution time	Period
P1	2	4
P2	3	6
P3	3	12