REAL TIME OPERATING SYSTEMS

Lesson-12:
Preemptive Scheduling Model

1. Common scheduling models

Common scheduling models

- Preemptive Scheduling
- Cyclic Scheduling of periodic tasks and Round Robin Time Slicing Scheduling of equal priority tasks
- Cooperative Scheduling of ready tasks in a circular queue. It closely relates to function queue scheduling.
- Cooperative Scheduling with Precedence Constraints
- Scheduling using 'Earliest Deadline First' (EDF) precedence.

Common scheduling models

- Rate Monotonic Scheduling using ‘higher rate of events occurrence First’ precedence
- Fixed Times Scheduling
- Scheduling of Periodic, sporadic and aperiodic Tasks
- Advanced scheduling algorithms using the probabilistic Timed Petri nets (Stochastic) or Multi Thread Graph for the multiprocessors and complex distributed systems.

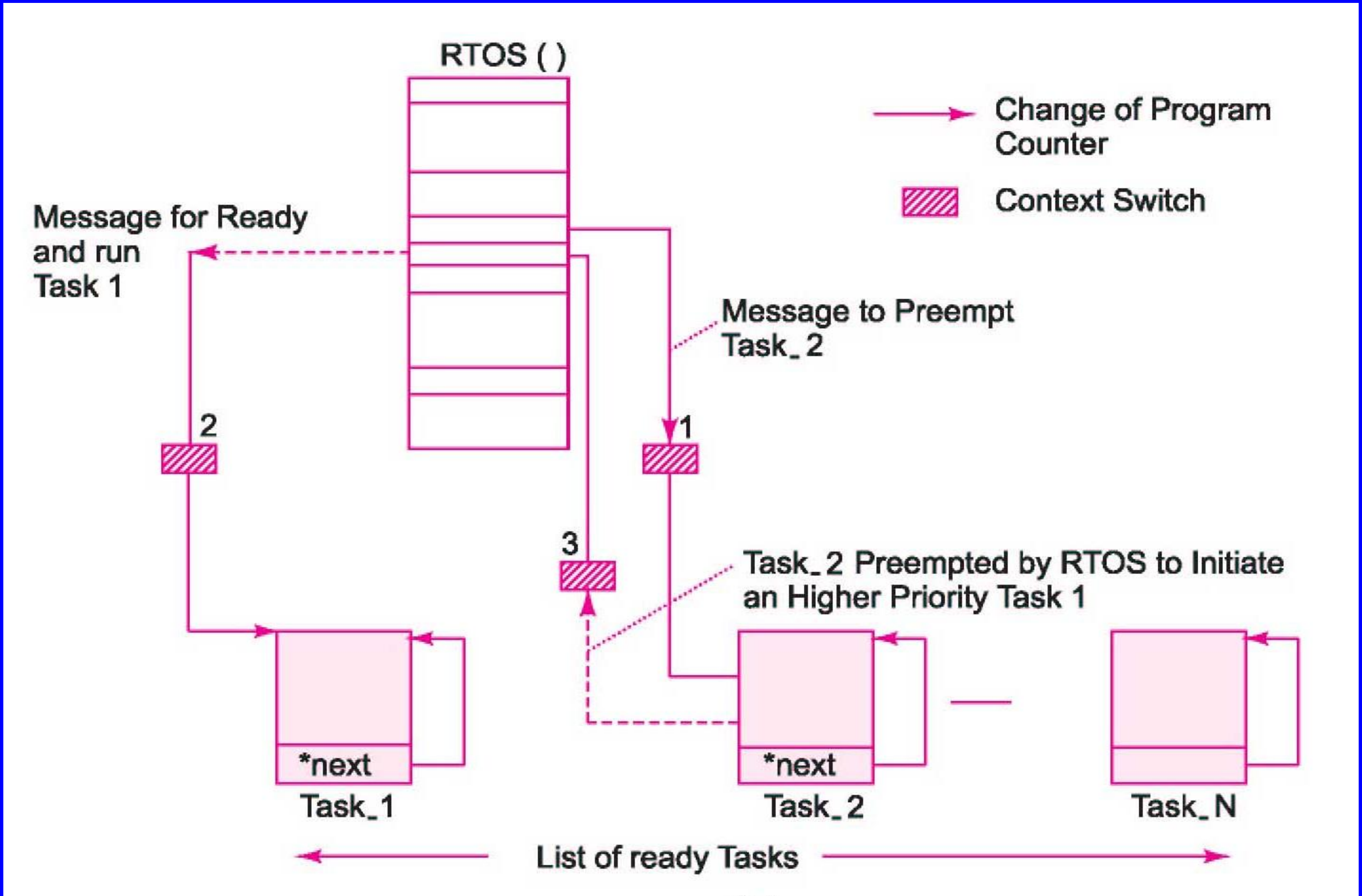
2. Preemptive Scheduling of tasks

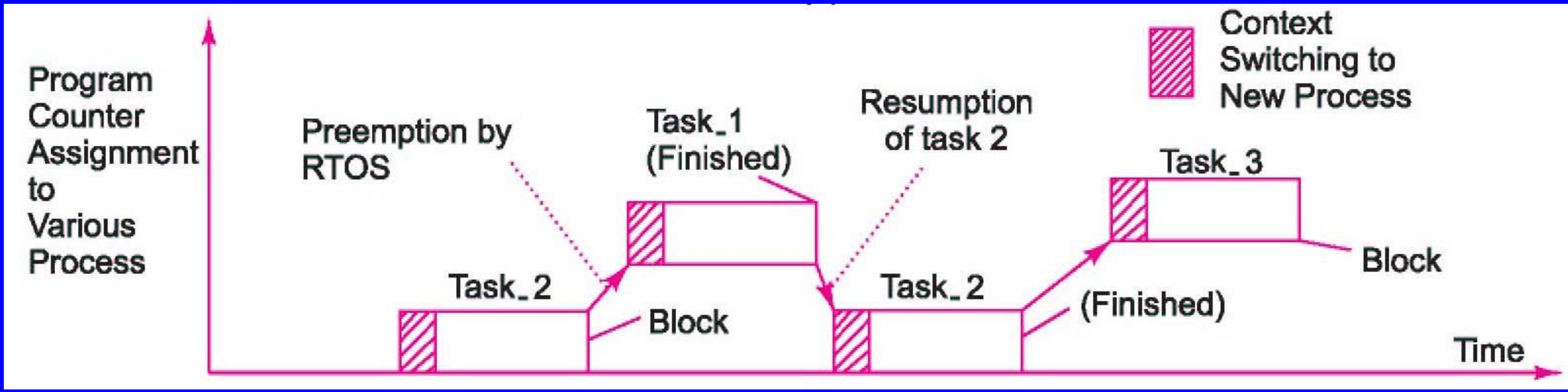
Preemptive Scheduling of tasks

- OS schedules such that higher priority task, when ready (on receiving the IPC for which waiting or signal for start), preempts a lower priority by blocking running task

RTOS Preemptive Scheduling

- Processes execute such that scheduler provides for preemption of lower priority process by higher priority process.
- Assume priority of task_1 > task_2> task_3 > task_4…. > task N
- Each task has an infinite loop from start (Idle state) up to finish.
- Task 1 last instruction points to the next pointed address, *next. In case of the infinite loop, *next points to the same task 1 start.


RTOS ( )
Change of Program Counter
Context Switch
Message for Ready and run Task 1
Message to Preempt Task_2
2
1
3
Task_2 Preempted by RTOS to Initiate an Higher Priority Task 1
*next
*next
Task_1
Task_2
Task_N
List of ready Tasks


Program Counter Assignment to Various Process
Preemption by RTOS
Task_1 (Finished)
Resumption of task 2
Context Switching to New Process
Task_3
Task_2
Block
Task_2
(Finished)
Block
Time

Worst-case latency

- Not Same for every task
- Highest priority task latency smallest
- Lowest priority task latency highest

Worst-case latency

- Different for different tasks in the ready list
- Tworst = $\{(dt_i + st_i + et_i)_1 + (dt_i + st_i + et_i)_2 + \ldots + (dt_i + st_i + et_i)_{p-1} + (dt_i + st_i + et_i)_p\} + t_{ISR}$.
- t_{ISR} is the sum of all execution times for the ISRs
- For an i-th task, let the event detection time with when an event is brought into a list be is dt_i , switching time from one task to another be is st_i and task execution time be is et_i
- i = 1, 2, …, p –1 when number of higher priority tasks = p –1 for the p^{th} task.

ISRs Handling

- Hardware polls to determine whether an ISR with a higher priority ISR than the present one needs the service at the end of an instruction during execution of an ISR, if yes, then the higher priority ISR is executed.

3. RTOS method for Preemptive Scheduling of tasks

An infinite loop in each task

- Each task design is like as an independent program, in an infinite loop between the task ready place and the finish place.
- The task does not return to the scheduler, as a function does.
- Within the loop, the actions and transitions are according to the events or flags or tokens.

When priority of task_1 > task_2 > task_3

- (1) At RTOS start, scheduler sends a message (Task_Switch_Flag) to task 1 to go to un-blocked state and run, and thus highest priority task 1 runs at start.
- (2) When task 1 blocks due to need of some input or wait for IPC or delay for certain period, a message (Task_Switch_Flag) will be sent to RTOS, task 1 context saves and the RTOS now sends a message (Task_Switch_Flag) to task 2 to go to un-blocked state and run.

When priority of task_1 > task_2 > task_3

- (3) Task 2 now runs on retrieving the context of task 2. When it blocks due to need of some input or wait for IPC or delay for certain period, a message (Task_Switch_Flag) will be sent to RTOS, task 2 context saves and an RTOS message (Task_Switch_Flag) makes the task 3 in un-blocked state. Task 3 will run now after retrieving the context of task 3.

When priority of task_1 > task_2 > task_3

- (4) If during running of task 3, either task 2 or task 1 becomes ready with the required input or IPC or delay period is over, task 3 is preempted, a message (Task_Switch_Flag) will be sent to RTOS, task 3 context saves, and task 1, and if task 1 not ready, then task 2 runs after retrieving the context of task 2.

When priority of task_1 > task_2 > task_3

- (5) A message (Task_Switch_Flag) is sent to RTOS after task 2 blocks due to wait of IPC or need of sum input and task 2 context saves and task 1 if ready then task 1 runs on retrieving the context of task 1
- (6) task 1 if not ready then task 3 runs on retrieving the context of task 3
- (7) Task 1 when ready to run preempts tasks 2 and 3, and Task 2 when ready to run preempts task 3

Specifying timeout for waiting for the token or event

- Specify timeout for waiting for the token or event.
- An advantage of using timeout intervals while designing task codes is that worst-case latency estimation is possible.
- There is deterministic latency of each tasks

Specifying timeout for waiting for the token or event

- Another advantage of using the timeouts is the error reporting and handling
- Timeouts provide a way to let the RTOS run even the preempted lowest priority task in needed instances and necessary cases.

Summary

We learnt

- Preemptive Scheduling
- Scheduler functioning is such that high Priority task when ready to run preempts low priority task
- Task of highest priority has lowest latency

End of Lesson 12 of Chapter 10
on
Preemptive Scheduling Model