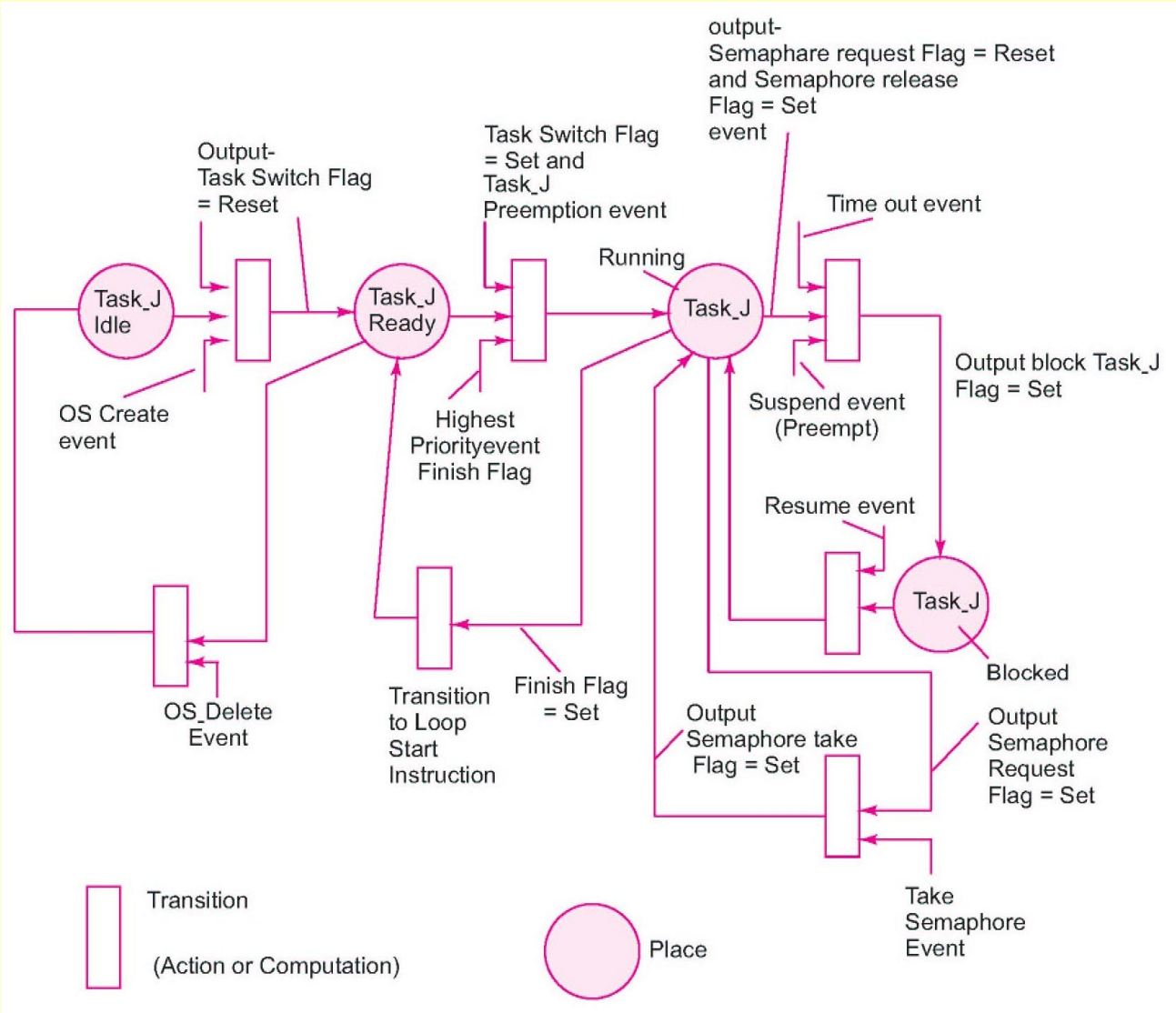
REAL TIME OPERATING SYSTEMS

Lesson-20:
Model for Preemptive Scheduling

Petri net concept based model

- Petri net concept based model which models and helps in designing the codes for a task
- The model shows places by the circles and transitions by the rectangles.

Petri net model for the task with a preemptive scheduler and semaphore taking and releasing



Petri net model in Figure

- (*i*) Each task is in idle state (or at idleTaskPlace) to start with, and a token to the RTOS is *taskSwitchFlag* = reset.
- (*ii*) Let us consider the task_J_Idle place, which is currently has of highest priority among the ready tasks.
- When the RTOS creates task, task_J, the place, task_J_Idle undergoes a transition to the ready state (or to readyTaskPlace), task_J_Ready place.

Petri net model in Figure

- The RTOS initiates *idle* to *ready* transition by executing a function, task create ( ).
- For the present case it is done by executing a function, task_J_create ( ).
- A transition from the idle state of the task is fired as follows.
- RTOS sends two tokens, RTOS_CREATE Event and *taskJSwitchFlag*. The output token from the transition is *taskSwitchFlag = true*.

Petri net model in Figure

- (*iii*) When after task J finishes, the RTOS sends a RTOS_DELETE event (a token) the task, it returns to the task_J_Idle place and its corresponding *taskJSwitchFlag* resets.

Petri net model in Figure

- (*iv*) At task_J_Ready place, the scheduler takes the priority parameter into account. If the current task current happens to be of highest priority, the scheduler sets two tokens, *taskJSwitchFlag* = true (sends a token) and *highest Priority Event* = true, for the transition to the running task J place, task_J_Running. The scheduler also resets and sends the tokens, task switch flags, for all other tasks that are of lesser priority. This is because the system has only one CPU to process at an instant

Petri net model in Figure

- (*v*) From the task_J_Running place, the transition to the task_J_Ready place will be fired when the task finish flag sets

Petri net model in Figure

- (*vi*) At task_J_Running place, the codes of the switched task J are executed. [Refer to the top-right most transition in the figure.]

Petri net model in Figure

- (*vii*) At the runningTaskPlace, the transition for preempting will be fired when RTOS sends a token, *suspendEvent*. Another enabling token if present, is *time_out_event* will also fire the transition. An enabling token for both situations is the semaphore release flag, which must be set. Semaphore release flag is sets on finishing the codes of task J critical-sections.

Petri net model in Figure

- On firing, the next place is task_J_Blocked. Blocking is in two situations., oOne situation is of preemption. It happensis when the *suspendEvent* occurs on a call at the runningTaskPlace asking the RTOS to suspend the running. Another situation is a time-out of an SWT, which that associates with the running task place

Petri net model in Figure

- (viii) On a *resumeEvent* (a token from RTOS) the transition to task_J_Running place occurs

Petri net model in Figure

- (*ix*) At the task_J_Running place, there is another transition that fires so that the task J is at back at to the to task_J_Running place when the RTOS sends a token, take_Semaphore_Event for to asking the task J to take the semaphore

Petri net model in Figure

- (x) There can be none or one or several sections taking and releasing semaphore or message.
- RTOS during the execution of a section, the RTOS resets the semaphore release flag and sets the take semaphore event token.

Summary

We learnt

- Petri net model helps in programming the scheduler actions

End of Lesson 20 of Chapter 8