Stack-based Resource Access Protocol

Osman, Hamid
Li, Guanpeng
Ton, Hason
Wong, Cary

October 26, 2012

1 Current Scheduler

Changes in task 2 extended the scheduler to EDF scheduling and prepared us to extend the scheduler to support stack-based resource policy.

2 Modified Scheduler

Stack-based Resource Policy: Resource sharing protocol for dynamic priority scheduling

Preemption Level: Determines the ability for a task to preempt a running task. For EDF, if a task has a smaller absolute deadline, then it gains a higher preemption level.

Resource Ceiling: The highest preemption level among all tasks that share a resource R_k. System Ceiling: The highest preemption level among all R_k that is currently locked.

Required outcome from this task: extend the scheduler to support SRP

Based on the requirements, we need calculate and store a new variable *system-ceiling* to keep track of the system ceiling if SRP is being used. We also need to modify *nrk_add_to_readyQ* to satisfy the preemption requirement:

A task can preempt another running task if

1. The task has a higher preemption level than the running task AND
2. The task has a higher preemption level than the system ceiling

This is achieved by adding an extra condition to the existing conditions to decide whether a task is added before or after a running task.

We also add mechanisms to calculate the system ceiling. The system ceiling can change whenever a resource is acquired or released. In the code space, this is done by acquiring and releasing of semaphores, managed by *nrk_sem_pend* and *nrk_sem_post*. From now on, we will assume semaphore and resource are use interchangeably.

nrk_sem_pend is responsible for helping a task acquire a resource. We decrement the resource flag to 0 to indicate that the resource is not available in the case another task would like to access the resource. If the resource is unavailable, the requesting task will wait until the resource is released. If successful, we calculate the system ceiling by iterating through $ARRAY$ which holds all the resources in the system. Recall that system ceiling is the maximum resource ceiling of all resources that are currently **locked**. In order to determine whether a resource is currently locked, we look at the availability. A 0 availability means

it is currently in use, so we evaluate the resource ceiling of this resource. (Availability only gets the value 0 or 1 in this assignment; it is a binary semaphore). In the case that the resource ceiling is higher than the system ceiling, the system ceiling gets the resource ceiling level; otherwise, it stays the same.

nrk_sem_post is responsible for unlocking a R_k when a task has completed using the resource. We increment the availability so that other tasks may use the resource. We also need to recalculate the system ceiling by using the same logic in *nrk_sem_pend* If we skip this process, no other task may run, since the system ceiling may be held at a higher preemption level than any other task.

With 3 additional mechanisms, we have successfully implemented SRP with EDP. The following sections outline our tests and test results.

3 Testing

The main function as part of main.c, is responsible for setting up and initialization of the system. It also holds the task set required to verify the accuracy of the scheduler. The task set used is equivalent to the task set used in Task 2.

Table 1: Task Set for EDF with SRP

Task	e_i	P_i	R
1	2	4	A
2	3	7	A

4 Expected Results

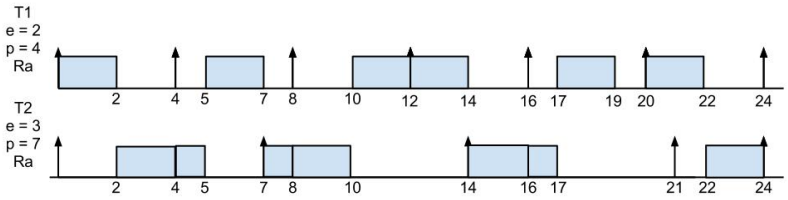

Figure 1: EDP with SRP

T1 → T2 → T1 → T2 → T1 → T2 → T1 → T1 → T1 → T2

Note: the expected results is **different** than the results in Task 2. This occurrence is due to the changes in protocols.

5 Results

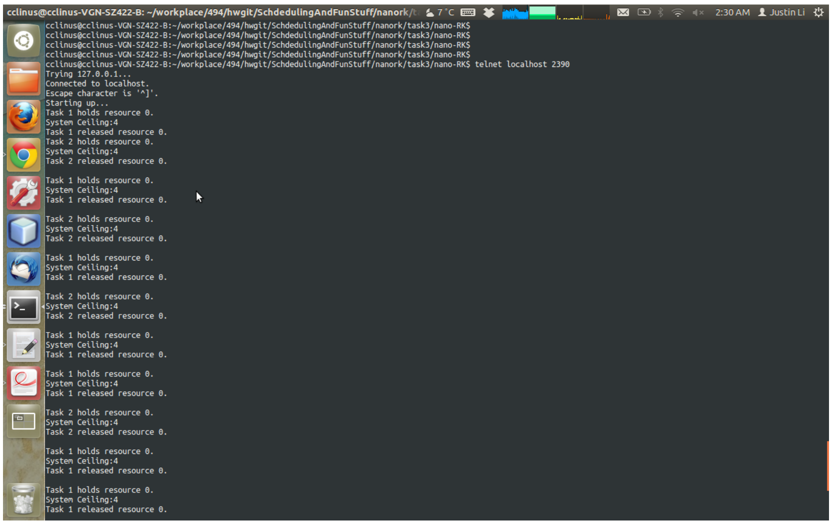

Figure 2: EDP with SRP screencap

T1 $\rightarrow$ T2 $\rightarrow$ T1 $\rightarrow$ T2 $\rightarrow$ T1 $\rightarrow$ T2 $\rightarrow$ T1 $\rightarrow$ T1 $\rightarrow$ T1 $\rightarrow$ T2