# Definitions

Resource Ceiling: Highest preemption level among all tasks that can use the shared resource (statically defined)  
System Ceiling: Largest resource ceiling among all currently locked shared resources (changes at runtime)

# Phase 2

1. System Ceiling– OS\_SYSTEM\_CEILING
   1. Stack used to store Resource Ceilings
   2. head->data->Resource\_Ceiling->Deadline;
      1. Find minimum deadline function
         1. Min deadline = highest preemption leve
   3. Pushing and popping mutex pointer
      1. Push --> pend
      2. Pop --> post
2. Resource Ceiling 🡪 highest preemption level among all task that can use shared resource
   1. T1 is the resource ceiling for all mutexes
   2. Periods: T1 = 3, T2 = 4, T3 = 5.
   3. Period implicitly the deadline also
   4. Store as TCB pointer
   5. Store inside mutex object
3. Mutex pend
   1. Update system ceiling
4. Mutex post
   1. Update system ceiling
   2. Unblock tasks from RB Tree that have higher preemption (lower deadline) than the newly changed system ceiling.
      1. If not we risk the blocked tasks never being able to run
5. OSSched
   1. If a task that is NOT the current task wants to pre-empt, need check   
      deadline from EDF < deadline (from system ceiling) 🡪 higher priority than system ceiling
      1. If fail check, remove task from ready list, add to RB tree. Run EDF scheduler again for new task to run; run the above check again.
   2. If task picked by EDF is current task, resume current task

To improve:

* Use event flags for synchronous release
* Use memget for memory management

# Flow of Events

1. OSRecTaskCreate() 🡪 push all recursive tasks into heap
2. Revive rec task 🡪 synchronous release of tasks into readylist
   1. Done by when tick counter = 5, iterate through heap then put in readylist
   2. We also insert them into avl tree
      1. Problem here as same deadline not inserted at all
   3. When OSSched called, readylist stuff will be used, then when task is called it will run rectaskdelete to delete the task and remove it from readylist
3. When time’s up according to tick counter when compared to heap[0] (always the lowest deadline)
   1. Initialize every p\_tcb thing
   2. Update its new deadline for future use
      1. Then we need to pop and push new details into heap
      2. Same for avl tree
         1. Problem here too!
      3. Insert into ready list
   3. When OSSched runs, will call accordingly
4. In OSSched, we have rbtree to handle blocked tasks which is blocked when it does not satisfy SRP requirements
   1. Depends on mutex holding
   2. Problem in OSSched because we will check avl tree accordingly
      1. But when the task to run is in readylist, but not here cos same deadline
         1. Problem!!!

Tried a method to increment deadline by 1, but problematic after task3 rectaskdel.

OSSched

472, jump 480 then all the way, jump to tasksempost

# Problems to check

1. When deadline miss
   1. Need remove from avl and readylist in OSTick
      1. Post mutex properly
         1. After posting still got nesting counter
      2. Nesting counter exists
      3. When another task wants to take it, didn’t take properly, due to nesting counter existence
2. **We fixed the above by adding checks in revive rec task**
   1. When a task according to deadline needs to run
   2. We check if it missed its deadline previously
      1. If yes then we need to remove from previous
      2. Release mutex and decrement nesting counter
   3. As the deadline gets updated, it may or may not be the next task to run
      1. But this has no problems because mutex already released properly prior, and anyone that wants them will be able to get them.

Below is the task sets we try to run, assuming they are schedulable.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Task One** | **Mutex Held** | **Task Two** | **Mutex Held** | **Task Three** | **Mutex Held** | **Result** | **Remarks** |
| 5 | 2, 3 | 6 | 2, 3 | 6 | - | ok |  |
| 5 | 2, 3 | 6 | 2, 3 | 7 | - | ok |  |
| 5 | 2, 3 | 6 | 2, 3 | 6 | 2 | ok |  |
| 5 | 1, 2, 3 | 6 | 2, 3 | 6 | 2 | ok |  |
| 5 | 1, 2, 3 | 6 | 2, 3 | 6 | 1, 2 | ok |  |
| 6 | 1, 2, 3 | 5 | 2, 3 | 5 | 1, 2 | ok |  |
| 5 | 1, 2, 3 | 5 | 2, 3 | 5 | 1, 2 | ok |  |
| 5 | 1, 2, 3 | 5 | 1, 2, 3 | 5 | 1, 2, 3 | ok |  |
| 5 | 1, 2, 3 | 8 | 1, 2, 3 | 10 | 1, 2, 3 | ok |  |

# Time Stamps

**Scheduling** (1%)

* OSSched with Sched 🡪 277
* Without Sched 🡪 116

**Periodicity** (1%)

* Revive rec task nothing to revive 🡪 37
* Revive rec task sync release 🡪 1291
* Revive rec task task to revive 🡪 1925
* Heap push 🡪 45
* Heap pop 🡪 120
* Avl insert 🡪 200
* Avl remove 🡪 380

**Mutex acquire/release overhead** (1%)

* Mutex Pend 🡪 225
* Mutex Post 🡪 168
* Stack push 🡪 37
* Stack pop 🡪 29

**Task execution time** (1%) 🡪 as of 21st April each task does printfs only

* Average time for each task to run
* TaskOne
  + Average(102671172, 102675347, 102671463) 🡪 102672660
* TaskTwo
  + Average(51550275, 51551147, 51551659) 🡪 51551027
* TaskThree
  + Average(58420976, 58420976, 58412640) 🡪 58418197

**Interrupt latency** (1%)

* TickTask
  + 26 excluding OS\_revive\_rec\_task() and OS\_TickListUpdate()
* TickListUpdate
  + 190
* Rbtree insert
  + 140
* Rbtree delete
  + 200