Task Scheduling

(slides are based on Prof. Dr. Jian-Jia Chen and http://www.freertos.org)

Christian Hakert, Dr. Kuan-Hsun Chen

LS 12, TU Dortmund

December 12, 2019

Outline

Outline

Fundamentals (Recall)

- Problem:
  - A quest described by a set of input parameters and a set of constraints.
  - A **feasible solution**: finding a solution that meets all the constraints.
  - An **optimal solution** is a solution with the "best" objective function (defined by the problem) among all feasible solutions.

Fundamentals (Recall)

- Problem:
  - A quest described by a set of input parameters and a set of constraints.
  - A **feasible solution**: finding a solution that meets all the constraints.
  - An **optimal solution** is a solution with the "best" objective function (defined by the problem) among all feasible solutions.
- Algorithm:
  - Logical procedure to solve a certain problem
  - Informally specified a sequence of elementary steps that an "execution machine" must follow to solve the problem
  - Implemented in a formal programming language (Program)

Recurrent Tasks (Recall)

- When jobs (usually with the same computation requirement) are released recurrently, these jobs can be modeled by a recurrent task
- Periodic Task τ_i:
  - A job is released exactly and periodically by a period T_i
  - A phase ϕ_i indicates when the first job is released
  - A relative deadline D_i for each job from task τ_i
  - (ϕ_i, C_i, T_i, D_i) is the specification of periodic task τ_i, where C_i is the worst-case execution time

Recurrent Tasks (Recall)

- When jobs (usually with the same computation requirement) are released recurrently, these jobs can be modeled by a recurrent task
- Periodic Task τ_i:
  - A job is released exactly and periodically by a period T_i
  - A phase ϕ_i indicates when the first job is released
  - A relative deadline D_i for each job from task τ_i
  - (ϕ_i, C_i, T_i, D_i) is the specification of periodic task τ_i, where C_i is the worst-case execution time
- Sporadic Task τ_i:
  - T_i is the minimal time between any two consecutive job releases
  - A relative deadline D_i for each job from task τ_i
  - (C_i, T_i, D_i) is the specification of sporadic task τ_i, where C_i is the worst-case execution time

Scheduling Algorithms

Definition: Schedulability

A schedule of real-time tasks is **feasible** if all the tasks meet their deadlines.
$\rightarrow$ A task set **T** is schedulable, if there exists a schedule, such that every job of every task meets its deadline.

Scheduling Algorithms

Definition: Schedulability

A schedule of real-time tasks is **feasible** if all the tasks meet their deadlines.
$\rightarrow$ A task set **T** is schedulable, if there exists a schedule, such that every job of every task meets its deadline.

- Fixed-Priority Scheduling
  - Different jobs of a task are assigned the same priority

Scheduling Algorithms

Definition: Schedulability

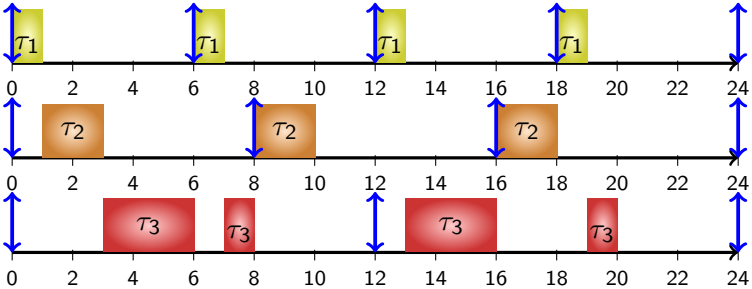
A schedule of real-time tasks is **feasible** if all the tasks meet their deadlines.
$\rightarrow$ A task set **T** is schedulable, if there exists a schedule, such that every job of every task meets its deadline.

- Fixed-Priority Scheduling
  - Different jobs of a task are assigned the same priority
- Dynamic-priority Scheduling
  - Different jobs of the same task may be assigned with different priorities

Rate-Monotonic (RM) Scheduling (Liu and Layland, 1973)

Priority Definition: A task with a smaller period has higher priority, in which ties are broken arbitrarily.

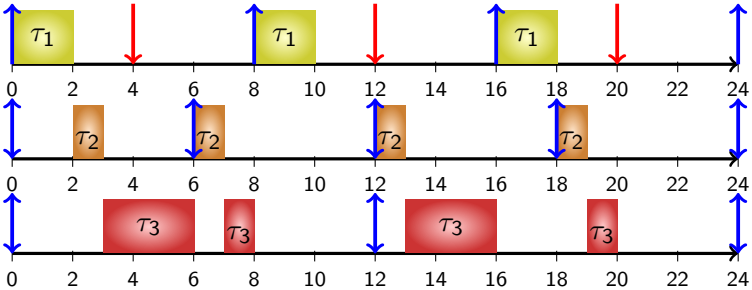
Example Schedule: $\tau_1 = (1, 6, 6)$, $\tau_2 = (2, 8, 8)$, $\tau_3 = (4, 12, 12)$.



Deadline-Monotonic (DM) Scheduling (Leung and Whitehead)

Priority Definition: A task with a smaller relative deadline has higher priority, in which ties are broken arbitrarily.

Example Schedule: $\tau_1 = (2, 8, 4)$, $\tau_2 = (1, 6, 6)$, $\tau_3 = (4, 12, 12)$.



Optimality (or not) of RM and DM

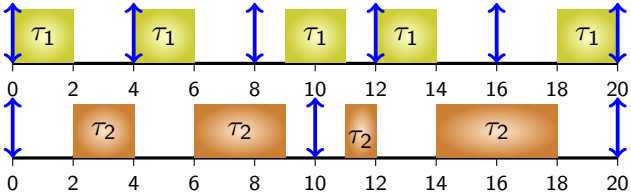
Example tasks: $\tau_1 = (2, 4, 4)$, $\tau_2 = (5, 10, 10)$

- Are they schedulable? Try to schedule them within 20 time units

Optimality (or not) of RM and DM

Example tasks: $\tau_1 = (2, 4, 4)$, $\tau_2 = (5, 10, 10)$

- Are they schedulable? Try to schedule them within 20 time units

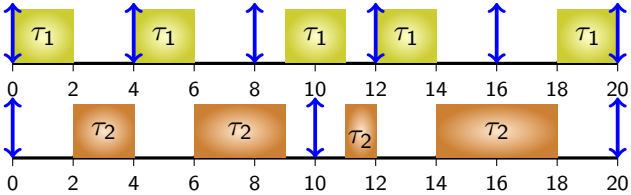


Actually, the above tasks are schedulable.

Optimality (or not) of RM and DM

Example tasks: $\tau_1 = (2, 4, 4)$, $\tau_2 = (5, 10, 10)$

- Are they schedulable? Try to schedule them within 20 time units



Actually, the above tasks are schedulable.
However, a deadline will be missed, regardless of how we choose to (statically) prioritize τ_1 and τ_2. Therefore, no fixed-priority scheme is optimal for scheduling periodic tasks.

Corollary

Neither RM nor DM is optimal.

Utilization-Based Schedulability Test

- Task utilization:

$$u_i := \frac{C_i}{T_i}$$

- System (total) utilization:

$$U(\mathbf{T}) := \sum_{\tau_i \in \mathbf{T}} \frac{C_i}{T_i}$$

A task system $\mathbf{T}$ fully utilizes the processor under scheduling algorithm A if any increase in execution time (of any task) causes A to miss a deadline. In this case, $U(\mathbf{T})$ is an **upper bound** on utilization for A, denoted $U_{ub}(\mathbf{T}, A)$.

$U_{lub}(A)$ is the **least upper bound** for algorithm A:

$$U_{lub}(A) = \min_{\mathbf{T}} U_{ub}(\mathbf{T}, A)$$

Liu and Layland Bound

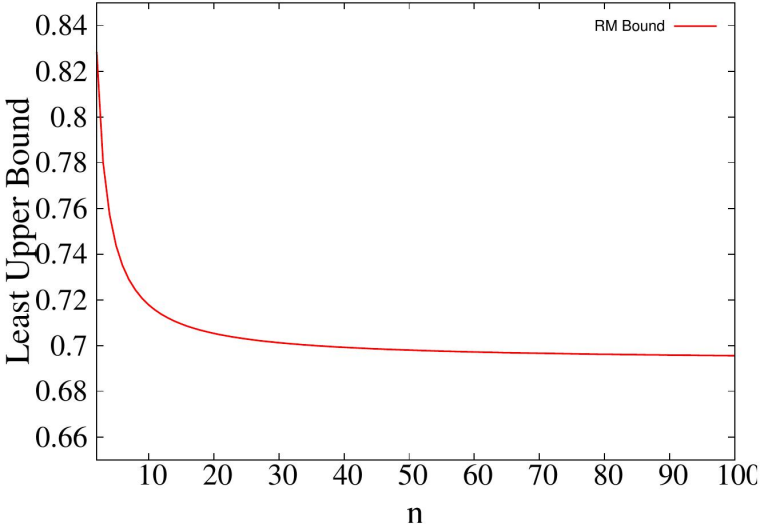
Theorem

[Liu and Layland] A set of n independent, preemptable periodic tasks with relative deadlines equal to their respective periods can be scheduled on a processor according to the RM algorithm if its total utilization U is at most $n(2^{\frac{1}{n}} - 1)$. In other words,

$$U_{lub}(RM, n) = n(2^{\frac{1}{n}} - 1) \geq 0.693.$$

n	$U_{lub}(RM, n)$	n	$U_{lub}(RM, n)$
2	0.828	3	0.779
4	0.756	5	0.743
6	0.734	7	0.728
8	0.724	9	0.720
10	0.717	∞	$0.693 = ln2$

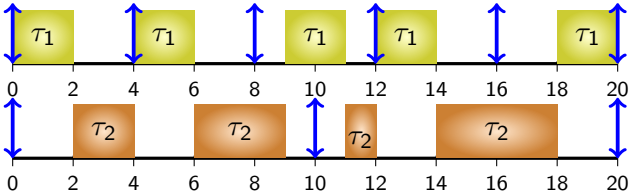
Least Upper Bound



Earliest-Deadline-First (EDF) Scheduling

At any moment, the system executes the job with the **earliest absolute deadline** among the jobs in the ready queue.

Example Schedule: $\tau_1 = (2, 4, 4)$, $\tau_2 = (5, 10, 10)$



Optimality and Schedulability of EDF

Theorem

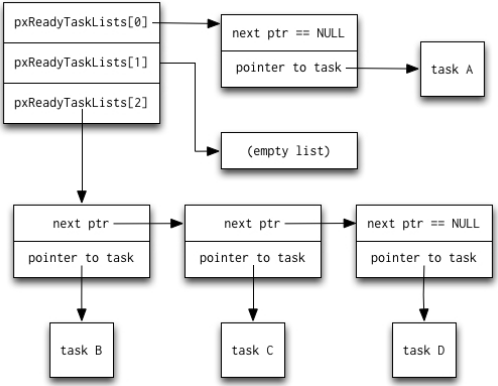
Liu and Layland: A task set $\mathbf{T}$ of independent, preemptable, periodic tasks with relative deadlines equal to their periods can be feasibly scheduled (under EDF) on one processor if and only if its total utilization U is at most one.

Note: EDF is optimal for timing satisfactions on uniprocessor systems.

Outline

Task Priority and Ready List

- An array of task lists
  - static xList pxReadyTasksLists[configMAX_PRIORITIES];
    /* Prioritised ready tasks. */



[http://aosabook.org/en/freertos.html]

Lists in FreeRTOS (list.c)

- xListItem is a data structure for an element in double linked lists
- void vListInsertEnd(xList *pxList, xListItem *pxNewListItem)
  - insert an item pxNewListItem to the end of the list pxList
- void vListInsert(xList *pxList, xListItem *pxNewListItem)
  - insert an item pxNewListItem to the list pxList in a sorted (increasing) order
- void vListRemove(xListItem *pxItemToRemove)
  - remove the item pxItemToRemove from its associated list

Outline

xPortStartScheduler() and vPortStartFirstTask()

- The implementation of xPortStartScheduler() is hardware dependent

xPortStartScheduler() and vPortStartFirstTask()

- The implementation of xPortStartScheduler() is hardware dependent
  - Start the timer that generates the tick Interrupt Service Routine (ISR)

xPortStartScheduler() and vPortStartFirstTask()

- The implementation of xPortStartScheduler() is hardware dependent
  - Start the timer that generates the tick Interrupt Service Routine (ISR)
  - This function does not return until an executing task calls vTaskEndScheduler()

xPortStartScheduler() and vPortStartFirstTask()

- The implementation of xPortStartScheduler() is hardware dependent
  - Start the timer that generates the tick Interrupt Service Routine (ISR)
  - This function does not return until an executing task calls vTaskEndScheduler()
  - At least one task should be created by xTaskCreate() before calling vTaskStartScheduler()

xPortStartScheduler() and vPortStartFirstTask()

- The implementation of xPortStartScheduler() is hardware dependent
  - Start the timer that generates the tick Interrupt Service Routine (ISR)
  - This function does not return until an executing task calls vTaskEndScheduler()
  - At least one task should be created by xTaskCreate() before calling vTaskStartScheduler()
  - The idle task is created automatically

xPortStartScheduler() and vPortStartFirstTask()

- The implementation of xPortStartScheduler() is hardware dependent
  - Start the timer that generates the tick Interrupt Service Routine (ISR)
  - This function does not return until an executing task calls vTaskEndScheduler()
  - At least one task should be created by xTaskCreate() before calling vTaskStartScheduler()
  - The idle task is created automatically
  - Run the first task by running vPortStartFirstTask().

xPortStartScheduler() and vPortStartFirstTask()

- The implementation of xPortStartScheduler() is hardware dependent
  - Start the timer that generates the tick Interrupt Service Routine (ISR)
  - This function does not return until an executing task calls vTaskEndScheduler()
  - At least one task should be created by xTaskCreate() before calling vTaskStartScheduler()
  - The idle task is created automatically
  - Run the first task by running vPortStartFirstTask().

- The implementation of xPortStartFirstTask() is also hardware dependent

xPortStartScheduler() and vPortStartFirstTask()

- The implementation of xPortStartScheduler() is hardware dependent
  - Start the timer that generates the tick Interrupt Service Routine (ISR)
  - This function does not return until an executing task calls vTaskEndScheduler()
  - At least one task should be created by xTaskCreate() before calling vTaskStartScheduler()
  - The idle task is created automatically
  - Run the first task by running vPortStartFirstTask().

- The implementation of xPortStartFirstTask() is also hardware dependent
  - Locate the stack of the first task

xPortStartScheduler() and vPortStartFirstTask()

- The implementation of xPortStartScheduler() is hardware dependent
  - Start the timer that generates the tick Interrupt Service Routine (ISR)
  - This function does not return until an executing task calls vTaskEndScheduler()
  - At least one task should be created by xTaskCreate() before calling vTaskStartScheduler()
  - The idle task is created automatically
  - Run the first task by running vPortStartFirstTask().

- The implementation of xPortStartFirstTask() is also hardware dependent
  - Locate the stack of the first task
  - Enable the interrupts (as they were disabled)

xPortStartScheduler() and vPortStartFirstTask()

- The implementation of xPortStartScheduler() is hardware dependent
  - Start the timer that generates the tick Interrupt Service Routine (ISR)
  - This function does not return until an executing task calls vTaskEndScheduler()
  - At least one task should be created by xTaskCreate() before calling vTaskStartScheduler()
  - The idle task is created automatically
  - Run the first task by running vPortStartFirstTask().

- The implementation of xPortStartFirstTask() is also hardware dependent
  - Locate the stack of the first task
  - Enable the interrupts (as they were disabled)
  - Run from the pointer of the stack

Ready Queue and Scheduling

- A ready queue maintains the TCB pointers of the tasks that are ready to be executed.

Ready Queue and Scheduling

- A ready queue maintains the TCB pointers of the tasks that are ready to be executed.
- The scheduler then selects the highest-priority job (task instance) in the ready queue for execution

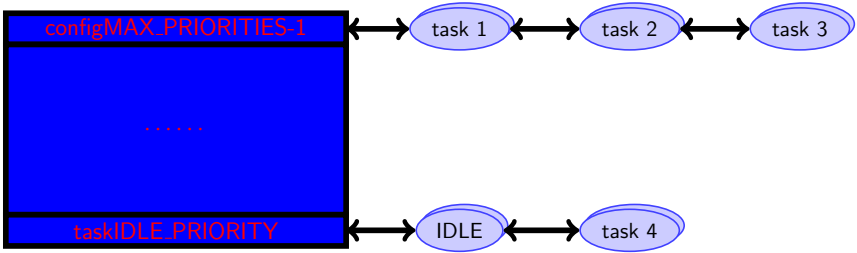
Ready Queue and Scheduling

- A ready queue maintains the TCB pointers of the tasks that are ready to be executed.
- The scheduler then selects the highest-priority job (task instance) in the ready queue for execution
- (Static-priority) fixed-priority scheduling
  - FreeRTOS uses fixed-priority (static-priority) scheduling
  - A task has a default priority value, defined when the task is created by using xTaskCreate()
  - All the instances of the task will then use the same priority for executing
  - If there are multiple task instances in the ready queue with the same priority, they share the processor and FreeRTOS uses a shared scheme to run these tasks

Ready Queue in FreeRTOS

- FreeRTOS uses multiple ready lists: one at each priority level.

```
1 #define prvAddTaskToReadyQueue( pxTCB )\
2 if( ( pxTCB )->uxPriority > uxTopReadyPriority )\
3 {\
4     uxTopReadyPriority = ( pxTCB )->uxPriority;\
5 }\
6 vListInsertEnd( ( xList * ) &( pxReadyTasksLists[ ( pxTCB )->uxPriority ] ),↩
      &( ( pxTCB )->xGenericListItem ) )
```



How does scheduler work - vTaskSwitchContext()

- System periodic tick interrupt calls vTaskSwitchContext()

```
...
/* Find the highest priority queue that contains ready tasks. */
while( listLIST_IS_EMPTY( &( pxReadyTasksLists[ uxTopReadyPriority ] ) ) )
{
  configASSERT( uxTopReadyPriority );
  --uxTopReadyPriority;
}
/* listGET_OWNER_OF_NEXT_ENTRY walks through the list, so the tasks of
the same priority get an equal share of the processor time. */
...

listGET_OWNER_OF_NEXT_ENTRY( pxCurrentTCB, &( pxReadyTasksLists
[ uxTopReadyPriority ] ) );
...
```

[http://aosabook.org/en/freertos.html]

How does scheduler work - vTaskSwitchContext()

- System periodic tick interrupt calls vTaskSwitchContext()
  - Selects the task with the highest priority in the ready list

```
...
/* Find the highest priority queue that contains ready tasks. */
while( listLIST_IS_EMPTY( &( pxReadyTasksLists[ uxTopReadyPriority ] ) ) )
{
  configASSERT( uxTopReadyPriority );
  --uxTopReadyPriority;
}
/* listGET_OWNER_OF_NEXT_ENTRY walks through the list, so the tasks of
the same priority get an equal share of the processor time. */
...

listGET_OWNER_OF_NEXT_ENTRY( pxCurrentTCB, &( pxReadyTasksLists
[ uxTopReadyPriority ] ) );
...
```

[http://aosabook.org/en/freertos.html]

How does scheduler work - vTaskSwitchContext()

- System periodic tick interrupt calls vTaskSwitchContext()
  - Selects the task with the highest priority in the ready list
  - pxCurrentTCB holds a pointer to the TCB of the selected task

```
...
/* Find the highest priority queue that contains ready tasks. */
while( listLIST_IS_EMPTY( &( pxReadyTasksLists[ uxTopReadyPriority ] ) ) )
{
  configASSERT( uxTopReadyPriority );
  --uxTopReadyPriority;
}
/* listGET_OWNER_OF_NEXT_ENTRY walks through the list, so the tasks of
the same priority get an equal share of the processor time. */
...

listGET_OWNER_OF_NEXT_ENTRY( pxCurrentTCB, &( pxReadyTasksLists
[ uxTopReadyPriority ] ) );
...
```

[http://aosabook.org/en/freertos.html]

How does scheduler work - vTaskSwitchContext()

- System periodic tick interrupt calls vTaskSwitchContext()
  - Selects the task with the highest priority in the ready list
  - pxCurrentTCB holds a pointer to the TCB of the selected task
  - Returns to the hardware-dependent code that starts running that task

```
...
/* Find the highest priority queue that contains ready tasks. */
while( listLIST_IS_EMPTY( &( pxReadyTasksLists[ uxTopReadyPriority ] ) ) )
{
  configASSERT( uxTopReadyPriority );
  --uxTopReadyPriority;
}
/* listGET_OWNER_OF_NEXT_ENTRY walks through the list, so the tasks of
the same priority get an equal share of the processor time. */
...

listGET_OWNER_OF_NEXT_ENTRY( pxCurrentTCB, &( pxReadyTasksLists
[ uxTopReadyPriority ] ) );
...
```

[http://aosabook.org/en/freertos.html]

Other Misc

- vTaskPrioritySet(pxTask, uxNewPriority): set the priority of a task
  - It may cause a context switch

Other Misc

- vTaskPrioritySet(pxTask, uxNewPriority): set the priority of a task
  - It may cause a context switch
- uxTaskPriorityGet(pxTask) : get the priority of a task

Other Misc

- vTaskPrioritySet(pxTask, uxNewPriority): set the priority of a task
  - It may cause a context switch
- uxTaskPriorityGet(pxTask) : get the priority of a task
- vDeleteTask(pxTaskToDelete): delete a task
  - The task is just removed from the ready queue, blocked queue after the call. The memory recycling is done by the idle task.

Other Misc

- vTaskPrioritySet(pxTask, uxNewPriority): set the priority of a task
  - It may cause a context switch
- uxTaskPriorityGet(pxTask) : get the priority of a task
- vDeleteTask(pxTaskToDelete): delete a task
  - The task is just removed from the ready queue, blocked queue after the call. The memory recycling is done by the idle task.
- vTaskIncrementTick(void)
  - Called by the portable layer each time a tick interrupt occurs
  - Increments the tick
  - Checks to see if the new tick value will cause any tasks to be unblocked

Summary: Task Management Functions

Creation

- xTaskCreate
- vTaskDelete

Control

- vTaskDelay
- vTaskDelayUntil
- uxTaskPriorityGet
- vTaskPrioritySet
- vTaskSuspend
- vTaskResume
- xTaskResumeFromISR

Utilities

- xTaskGetTickCount
- uxTaskGetNumberOfTasks
- vTaskList
- vTaskGetRunTimeStats
- vTaskStartTrace
- ulTaskEndTrace
- uxTaskGetStackHighWaterMark
- vTaskSetApplicationTaskTag
- xTaskGetApplicationTaskTag
- xTaskCallApplicationTaskHook