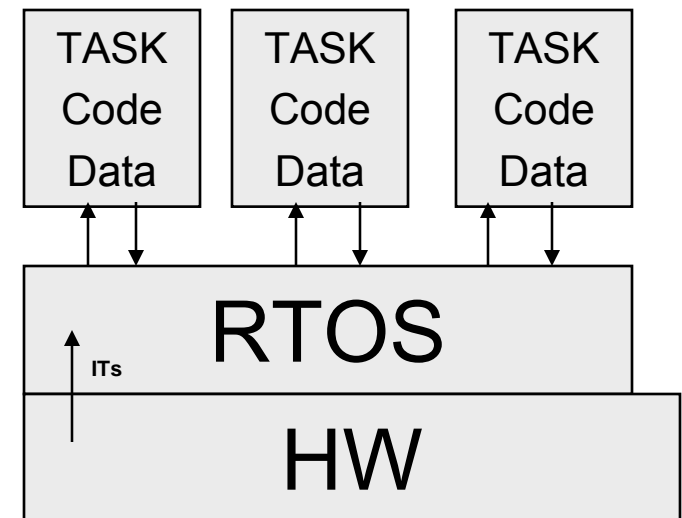
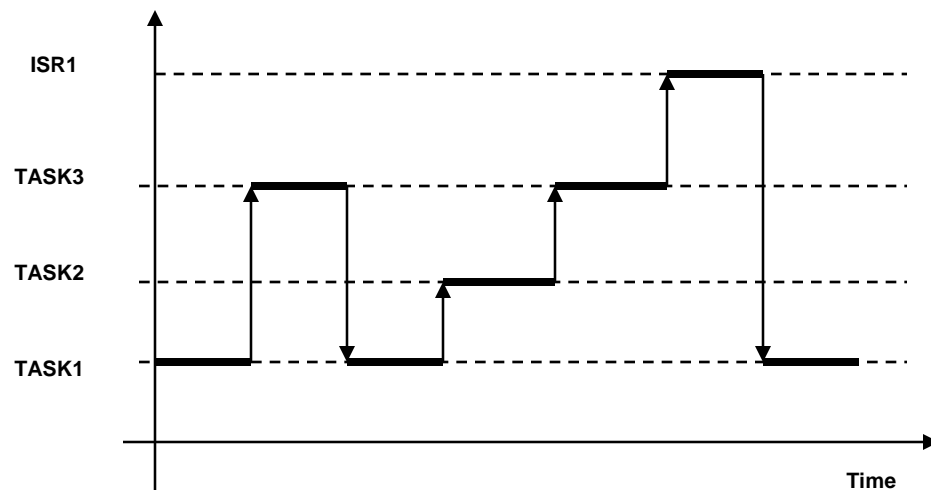

EECE 276

Embedded Systems

RTOS Basics
Process scheduling

RTOS Basics

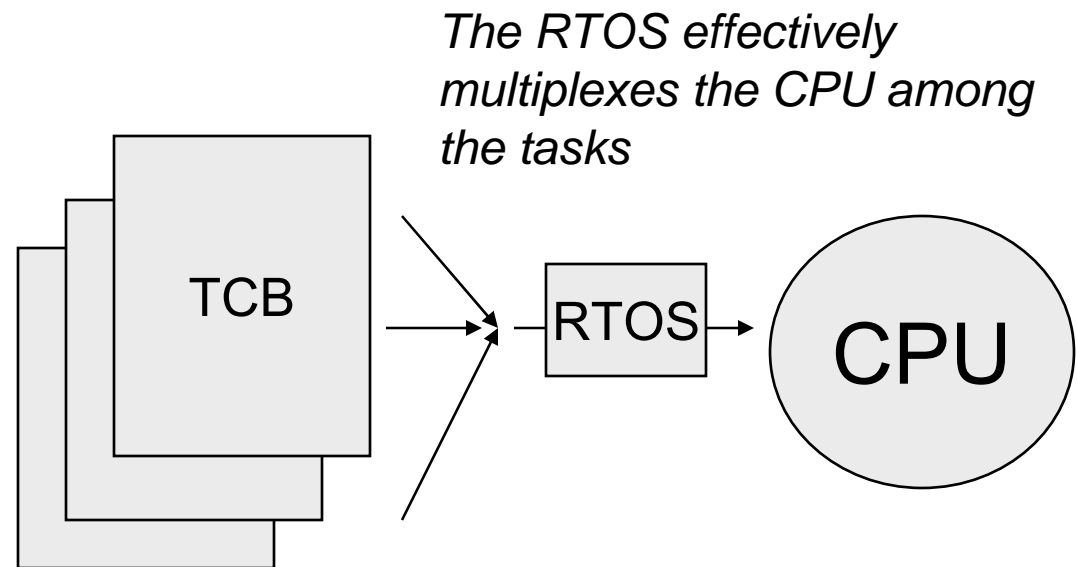
- Kernel: schedules tasks
- Tasks: concurrent activity with its own state (PC, registers, stack, etc.)



Tasks

- Tasks = Code + Data + State (context)
- Task State is stored in a Task Control Block (TCB) when the task is not running on the processor
- Typical TCB:

ID
Priority
Status
Registers
Saved PC
Saved SP

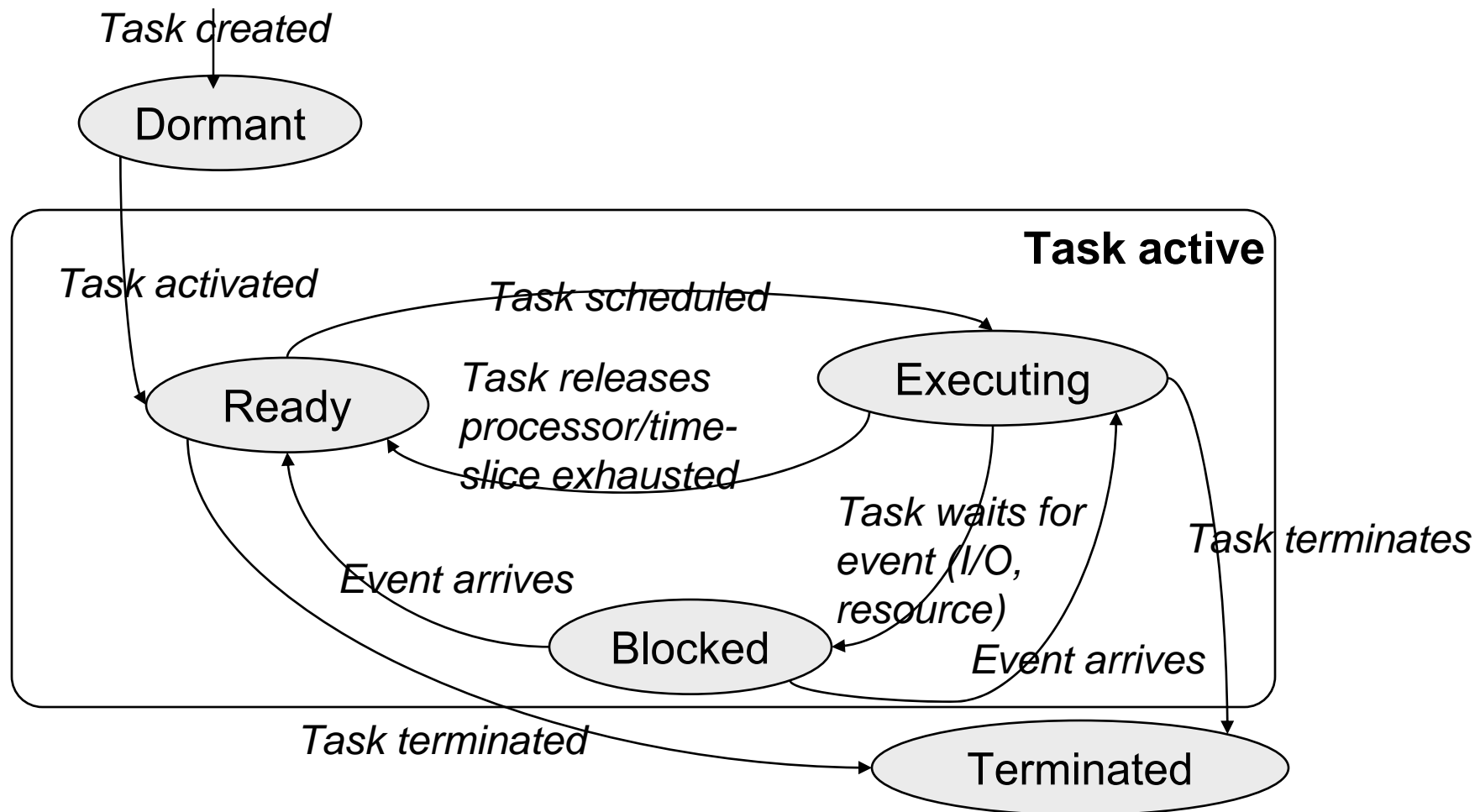


Task states

- **Executing:** running on the CPU
- **Ready:** could run but another one is using the CPU
- **Blocked:** waits for something (I/O, signal, resource, etc.)
- **Dormant:** created but not executing yet
- **Terminated:** no longer active

The RTOS implements a Finite State Machine for each task, and manages its transitions.

Task State Transitions



RTOS Scheduler

- Implements task state machine
- Switches between tasks
- Context switch algorithm:
 1. Save current context into current TCB
 2. Find new TCB
 3. Restore context from new TCB
 4. Continue
- Switch between EXECUTING -> READY:
 1. Task yields processor voluntarily: **NON-PREEMPTIVE**
 2. RTOS switches because of a higher-priority task/event: **PREEMPTIVE**

RTOS Tasks

- Run in the same memory space
- Can share data
 - » Data sharing problems as before!
 - » Cannot simply disable IT-s (this would stop the RTOS!)
- Can share code – Similar problems

```
void task1() { ... vCountErr(9); ... }  
void task2() { ... vCountErr(10); ...}  
static int cErrors;  
void vCountError(int addErr) {  
    cErrors += addErr;  
}
```

← ***Not an atomic operation***

RTOS Tasks: Code sharing

- Reentrancy: A function is called *reentrant* if it can be entered simultaneously, by multiple tasks.
- Rules: A reentrant function
 - » May not use variables in a non-atomic way (unless local variables or private variables of the calling task)
 - » May not call other, non-reentrant functions
 - » May not use the HW in a non-atomic way

Process scheduling

- Goal: to satisfy timing requirements
- Pre-run-time (*static*) scheduling: determine precise task schedules at design-time.
 - » Ex: TTA
- Run-time (*dynamic*) scheduling: scheduling is done dynamically, by the RTOS, based on priorities.

Task attributes (1)

A task model (periodic tasks):

- Precedence constraints: specify if any task(s) must precede other tasks.
 - Release (arrival) time - $r(i,j)$: The release time of the j -th instance of the i -th task
 - Phase: $\Phi(i)$: The release time of the first instance of the i -th task.
 - Response time: Time span between task activation and its completion
 - Absolute deadline - $d(i,j)$: The instant by which the j -th instance of the i -th task must complete
 - Relative deadline – $D(i)$: The maximum allowable response time for the task
-

Task attributes (2)

- Laxity type: Notion of urgency or leeway in a task's execution
- Period - $p(i)$: The minimum length of intervals between the release times of consecutive tasks.
- Execution time – $e(i)$: The (maximum) amount of time required to complete the execution of the i -th task when it executes alone and has all the resources it requires.

Task model

Some basic identities:

- $\Phi(i) = r(i, 1)$ // First release time
- $r(i, k) = \Phi(i) + (k-1) * p(i)$ // Periodic tasks
- $d(i, j) = \Phi(i) + (j-1) * p(i) + D(i)$ // Abs. Deadline
- If $D(i) == p(i)$ then
 $d(i, k) = r(i, k) + p(i) = \Phi(i) + k * p(i)$

Task model

Simple task model:

- All tasks are strictly periodic.
- The relative deadline of a task is equal to its period.
- All tasks are independent – no precedence constraints
- No tasks have non-preemptible sections – cost of preemption is negligible
- Only processing requirements count – memory and I/O requirements are negligible.

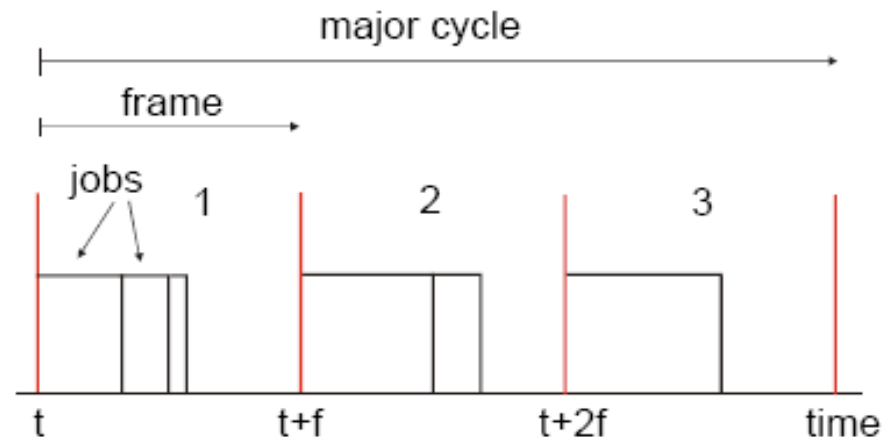
Scheduling techniques

- Round-robin scheduling:
 - » Each task is assigned a fixed time quantum (slice).
 - » Fixed-rate timer generates a periodic IT. Rate is equal to the slice
 - » Task runs until completion or slice expiration
 - » If task does not complete, it is placed at the end of a queue of executable (“ready”) tasks.
- Fair scheduling
- All tasks have the same priority

Scheduling techniques

- Cyclic executive:

- » Schedule consists of (pre-run-time scheduled) *frames*. A sequence of frames in which all tasks execute, all deadlines and periods are satisfied is called a *hyperperiod* ('*major cycle*'). It's length is $lcm(p(i))$ over all i-s.



Scheduling techniques

- Cyclic executive:

- » A frame (f) is long enough s.t. every task can start and complete within the frame – No preemption within the frame.
- » Scheduler makes scheduling decisions only at the beginning of each frame.
 - Cond #1: $f \geq \max(e(i))$: “Frame is long enough”.
 - Cond #2: $\text{floor}(p(i)/f) - p(i)/f = 0$: “Hyperperiod has integer number of frames”
 - Cond #3: $2*f - \text{gcd}(p(i),f) < D(i)$: “There is at least one frame between the release time and deadline of each task.”

Scheduling techniques

- Fixed priority – Rate Monotonic Scheduling

Given a set of periodic tasks and a preemptive priority scheduling, then assigning priorities s.t. tasks with the shorter periods have higher priorities (rate-monotonic), yields a feasible scheduling algorithm. (RM Rule).

Utilization: $u(i) = e(i)/p(i)$

A set of N periodic, independent tasks is RM-schedulable [Liu73] if

$$U = \sum_{i=1}^N u(i) \leq N * (2^{1/N} - 1)$$

Note: If $N \rightarrow \text{infinity}$ $\lim () = \ln 2 \sim 0.69$.

Scheduling techniques

- Dynamic priority – Earliest Deadline First Scheduling: *The ready task with the earliest deadline has the highest priority.*
- EDF Bound theorem:
A set of N tasks, each of whose relative deadline equals to its period, can be feasibly scheduled by EDF if and only if

$$\sum_{i=1}^N (e(i) / p(i)) \leq 1$$

EDF is optimal for a single processor (with preemption allowed)