Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Real-Time Architectures 2003/2004

Mapping on execution platform

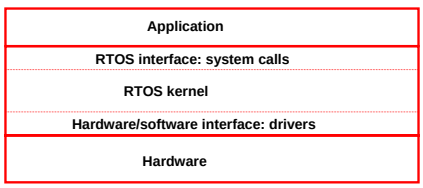
Johan Lukkien

17-05-2004

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Static organization

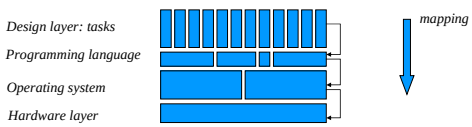
- Layering
- Architecture of Application: single executive or modular
- Architecture of RTOS: monolithic or micro kernel



Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Mapping between layers

- Usually there are different units of concurrency in each layer
- The mapping must be made explicit



Single executive ("cyclic executive")

- Application is written as a single repetition, fast enough
  - to execute time driven tasks
  - to execute event driven tasks
    - Interrupt service routines just record event occurrence (event flags)
- Advantages
  - needs little or no OS support
- Disadvantages:
  - lack of structure (not extensible)
    - spaghetti code, no clear task structure
  - repetition may grow to be inefficient
    - dependencies across several iterations
    - scalability: extension may destroy everything
  - cannot distribute

Single executive: example

```
while true do
    if ItIsTimeFor (Keyboard)        then /* check the keyboard */ fi;
    if ItIsTimeFor (Mouse)           then /* check the mouse    */ fi;
    .....
    if EventOccurred (TempSensor) then /* deal with event      */ fi;
    .....
    WaitFor (Minimum time to next check
             or Event interrupt);
od
```

- Notes:
  - in fact simulation of concurrent task execution
  - sometimes the only choice (dependent on what the RTOS supports)
    - no RTOS, just a simple library suffices
  - the idea may recur even in a multi-tasking context
    - e.g., this may be the way that a sensor and a handler communicate

Tasks

- Introduced in the analysis phase
- Associated with handling an event

- Reasons for additional decomposition: *design*
  - (partly) sharing between tasks
    - e.g. (actions on) a shared data structure
  - enable tweaking individual subtasks in a processing pipeline
    - e.g. scalable parts
  - assigning relative importance to subtasks
    - less important processing steps are easily left out in case of overload
  - deal with temporal input bursts
    - e.g. separate event registration and actual processing
  - frequency differences

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Task-related OS concepts

- Process (“program in execution”)
  - defines a data space
    - concurrency transparency on single processor
  - has at least one associated thread
  - unit of distribution
  - unit of fault containment
- Thread
  - unit of concurrency
  - unit of scheduling
    - though with one thread per process, a process is often said to be the unit of scheduling
  - several threads ‘live’ within shared address space
    - i.e., in a process

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Reasons to introduce processes

- Allow for distribution
- Introduce safety boundaries
- Follow solution structure
- (Increase concurrency level)
  - since more processes means more threads
  - hence, actually the purpose is to introduce more threads

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Reasons to introduce threads

- Increase concurrency level
  - performance
    - hide latency
    - exploit platform concurrency
  - discriminate importance levels in activities
    - e.g. interrupt routines
- Deal with the natural concurrency
  - natural organization, structure
    - e.g. thread per event
    - thread per resource
    - thread per (active) external interaction sequence

Mapping of tasks

- Each task mapped onto a process
  - and, by result, onto a thread
  - then tasks can share resources only through the OS
  - or they approach them using message passing
    - the resource is maintained by a separate process
- Each task mapped onto a thread
  - newly created or repeatedly restarted

Multiplexing

- Several threads per task
  - then the actions of a task are distributed across several threads
  - a logical consequence of having *active objects*
    - active objects: data + operations + independent, private behavior .... “object with private thread”
    - simplest form: message passing process
- Several tasks per thread
  - efficiency concern in the mapping, e.g. when the tasks exclude each other

Consequences for scheduling

- Assume FPS (e.g. RMS) at task level
  - direct mapping task on thread/process
    - OS must support same policy
  - one thread executes several tasks completely
    - OS must support dynamic priority assignment
    - priority setting derived from event
  - one task is executed by several threads
    - the threads act as resources to be acquired by the task
    - dynamic priority assignment requirement
      - the priority of a thread is determined by the task it is working for

Consequences for scheduling

- Assume DPS (e.g. EDF) at task level
  - direct mapping task on thread/process
    - OS must support same policy
  - one thread executes several tasks completely
    - OS must support same policy
      - the deadline will do the trick
  - one task is executed by several threads
    - need to determine deadlines for subtasks
    - no clear strategy

Validation

- Need to have values for task execution times
  - dependent on platform, OS, compiler
  - direct measurement
    - profiling
- Strict requirements on OS
  - discussed later
  - .... need timing information of relevant system calls and OS operations
- **Question**:
  - what would *portability* mean in this context?
  - is a platform-independent real-time design possible?
    - what would be needed?
    - what would be the parameterization?
    - what are typical problems?