Real-Time Kernels

Prof. J.-D. Decotignie
CSEM Centre Suisse d'Electronique et de Microtechnique SA
Jaquet-Droz 1, 2007 Neuchâtel
jean-dominique.decotignie@csem.ch

Outline

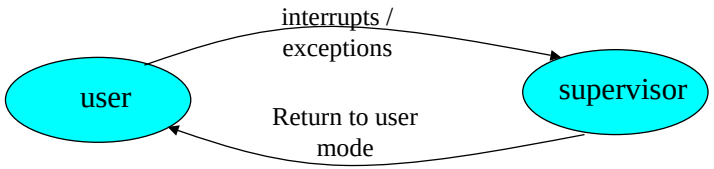
- introduction
- process, task, thread
- Kernel and micro-kernel
- Kernel operating principles
- Offered services
  - Event signalling
  - Communication
  - Exclusion
- conclusion

Hardware based protection

- Two operating modes
- Memory protection
- Input and output protection
- Processor protection

Two operating modes

- User mode
  - Not all instructions are available
  - Not all registers can be accessed
- Priviledged mode
  - All instructions and registers can be accessed
  - Automatic switch to this mode in case of interrupt or exception



Memory protection

- At least for the exception vectors and the interrupt service routines
- May be based upon
  - Hardware mechanisms
    - Some addresses are not accessible in user mode
  - A memory management unit
    - Only some address ranges can be accessed from the program
    - The range(s) is (are) programmable
      - They are part of the context of the program

csem

Input and output protection

- Input and output instructions are privileged
  - Impossible in case of uniform addressing space
- We want to avoid that a program in user mode
  - Could take control of the processor through the use of the priviledged mode (i.e. modifying the exception vectors)
  - Could access the inputs and outputs directly (often through memory protection)

csem

Processor protection

- Use of a timer that interrupts what is executing after a given delay
  - The system may then take control back
  - The time decrements regularly
  - When it reaches 0, this raises an interrupt
- A timer is often used
  - To share the processor "time sharing"
  - As a support for current time and date
- Modifying the time should be privileged

csem

The task concept

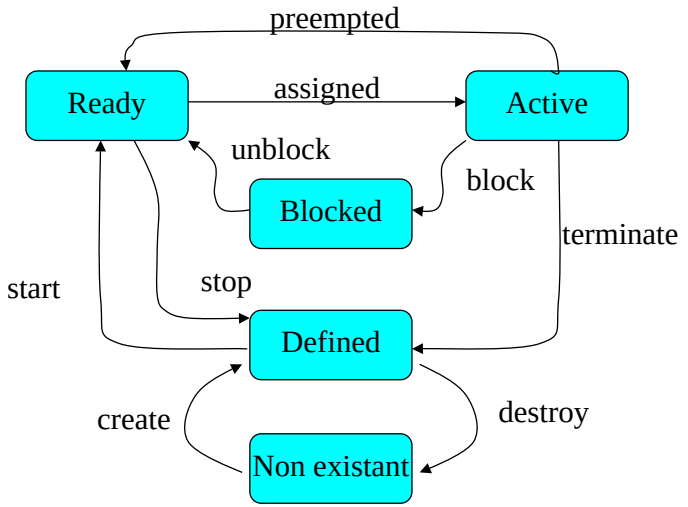
- An operating system executes a variety of programs
  - Batch systems: jobs
  - Multi-user systems: user programs, processes, threads
  - Real-time systems: tasks, threads
- A task includes
  - A program counter
  - A stack
  - Data

csem

The task concept (2)

- Along its execution, a task has a changing state:
  - It is created
  - It is given the processor
  - It is waiting for the processor
  - It is waiting for an event / some data
  - It has ended its execution
  - It is destroyed
- It can be only in a single of these states at any given time

Task states



The components of a task

- Memory zones
  - code
  - date
  - stack
- Processor registers
  - Program counter
  - General registers
  - Status register
  - co-processors registers (floating point, memory management, etc.)

Process control block (PCB)

- This is the information associated to a task
  - Its state
  - The program counter and the content of the registers
  - All the scheduling information
  - Memory management information
  - Statistics
  - Information on inputs and outputs
    - Open files, open connections, etc.

Process, task, thread

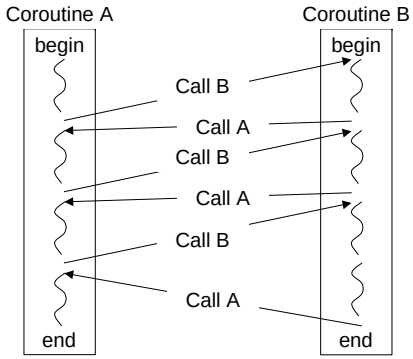
- UNIX process
  - Different spaces, memory protection
- light weight processes or threads (POSIX, Solaris, Java)
  - Shared space
- Tasks in real-time systems
  - Shared space, protection through language
- coroutines
  - Shared space
  - Not managed by the system (cooperative)

csem

Coroutines

- Invented by Conway

"an autonomous program that communicates with adjacent modules as if they were input and output procedures. Coroutines are thus procedures that are at the same hierarchical level, each one acting as it was the main program when, in fact there is no main program"



csem

Coroutine features

- local data values persist between the instant at which a coroutine looses control of processor and the next instant at which it gains control
- execution of a coroutine is suspended when it looses control of the processor and will not resume before it will be given the control back at a later time
- control is explicitly transferred from one coroutine to another one. This results in the currently executing coroutine to be suspended and to resume the execution of the target coroutine. This is what is often called quasi-parallelism.

csem

Coroutines in Modula-2

- Creation

```
NEWPROCESS (   Procedure_Name : PROC;
               Work_Space : ADDRESS;
               Work_Space_Size: CARDINAL;
               VAR       Coroutine_Handle : ADDRESS);
```

- Transfer of control

```
TRANSFER (VAR Handle_Coroutine1,
              Handle_Coroutines 2: ADDRESS);
```

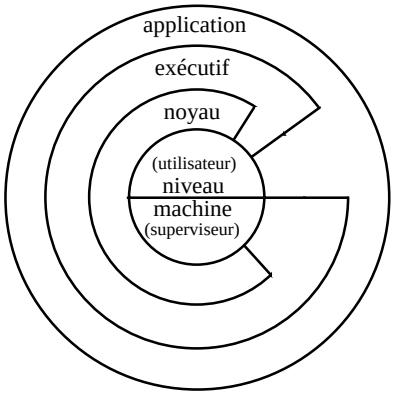
csem

Example of coroutines

```
MODULE Coroutine1;

FROM SYSTEM IMPORT ADDRESS,
    NEWPROCESS, TRANSFER;

VAR PP, C1, C2, C3 : ADDRESS;

PROCEDURE P1;
BEGIN
  LOOP
    ...
    TRANSFER (C1, C2);
    ...
  END;
END P1;

PROCEDURE P2;
BEGIN
  LOOP
    ...
    TRANSFER (C2, C3);
    ...
  END;
END P2;

PROCEDURE P3;
BEGIN
  LOOP
    ...
    TRANSFER (C3, C1);
    ...
  END;
END P3;

BEGIN (* main program *)
  ...
  NEWPROCESS(P1,...,...,C1);
  NEWPROCESS(P2,...,...,C2);
  NEWPROCESS(P3,...,...,C3);
  ...
  TRANSFER (PP, C1);
END Coroutine1.
```


System software structuring

- Onion model
  - processor
  - kernel
  - executive
- Groups the functions in families
- Each family corresponds to an abstraction level
- Visibility rules



Kernel

- Gives access to a virtual machine (processor)
- Stays in central memory
- handles
  - Task management
  - Optimal processor allocation
  - Communication between tasks
  - Task synchronization and mutual exclusion
  - Time management
  - Memory management
  - Communication with the external world

Kernels, micro-kernels

- UNIX like kernels
  - heavy
  - Numerous functions
- Lean kernels or micro kernels
  - Only the necessary minimum
  - Reduced but sufficient functionality

Operating principle

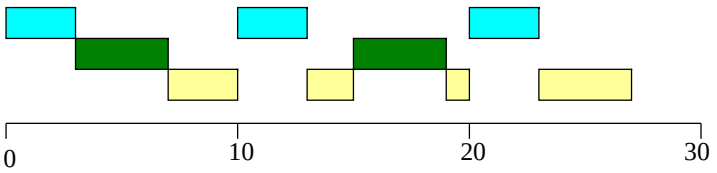
- pseudo parallelism
- All events should go through the kernel
  - interrupts
  - Mutual exclusion requests
  - synchronizations
  - communications
  - timers
-

  => HW timer support

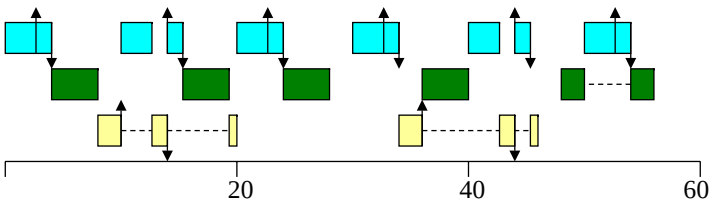
Example

- Each task switch corresponds to an event

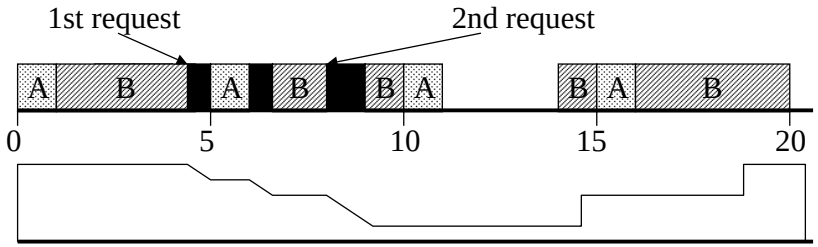
Task	T	D	C
A	10	10	3
B	15	15	4
C	30	30	10



Example with semaphores



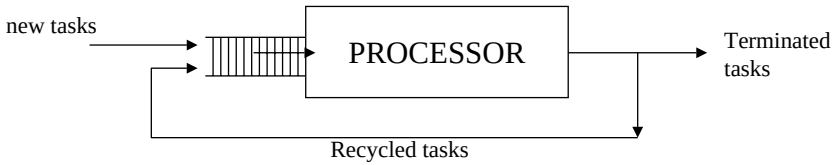
Example – Sporadic server



Task	T	C	use
A	5	1	20%
SpS	10	2.5	25%
B	14	6	42.9%

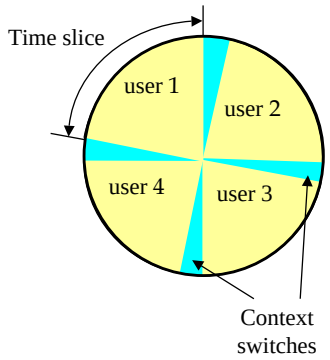
Processor allocation

- Round robin
  - One queue for waiting tasks
  - The processor is allocated to each task
    - For a fixed maximum duration
    - Or until it blocks
  - Is then put at the end of the waiting queue



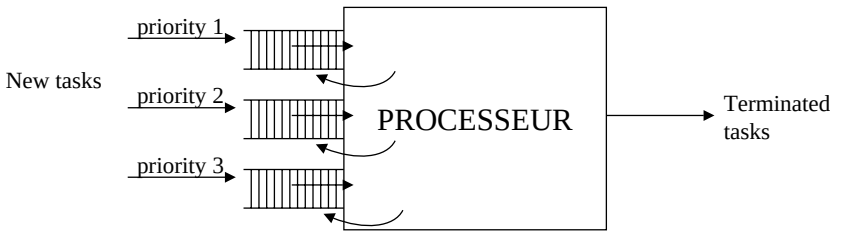
Time slices

- First linked to "time sharing"
- Usefull when there is no preemption
- Normally useless in a real-time kernel, but !



Processor allocation (2)

- According to priorities
  - One queue per priority level
  - Round robin within a priority level



Offered services

- Creation and destruction of tasks
- Synchronization services
- Mutual exclusion services
- Communication services
- Time management
- Operation on scheduling
- Memory management

- Others (network, files, ...)

Offered services (2)

- There are standards
  - POSIX
  - SCEPTRE
- Services selection
  - Ease of design and realization
  - Expression power
  - Taste
  - Sides aspects should not be forgotten (development tools, debugging tools)

Examples of offered services

Task Management	StartTask	Task	Start the task execution
	StopTask	Task	Stop the execution of the task
	ResumeTask	Task	Resume the execution of the task
	TerminateTask		End the execution of the task
	StateOf	Task	Give the current task state
	PriorityOf	Task	Give the current task priority
	Current		Returns the identity of the current task
	SetPriority	Task, Priority	Alter the priority of a task
Signalisation	SignalEvent	Event, task	Indicates an event to a given task
	WaitEvent	Event list	Wait ≥1 event occurence
	TOWaitEvent	Same +duration	Same but stops after a maximum waiting time
Synchronisation	IsEvent	Event list	True if all event have occurred
	ClearEvent	Event	Reset the state of an event

Examples of offered services (2)

Communication	SendElement	Element, queue	add the element at the end of the queue
	GetElement	Element, queue	retrieve the element which is at the head of the queue
	TOGetElement	idem + duration	same but abort after duration
	IsEmpty	File	true if queue is empty
	IsFull	File	true if queue is full
Exclusion	Lock	Lock	request a lock
	TOLock	Lock, duration	same but abort after duration
	Unlock	Lock	free a (previously taken) lock

Event based sychronization

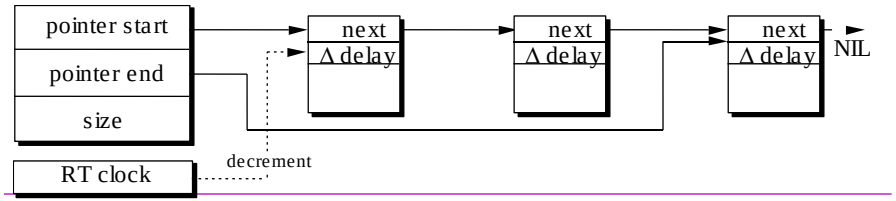
- SignalEvent
  - 1.updates the flag associated to the event
  - 2.recalculate the waiting condition
  - 3.if OK, put the waiting task in ready queue
  - 4.switch to the highest « priority » task
- WaitEvent (may be on more than one event)
  - 1.if event is signaled, task continues
  - 2.if not, it is put in the blocked queue
  - 3. switch to the highest « priority » task
- ClearEvent

Event based sychronization (2)

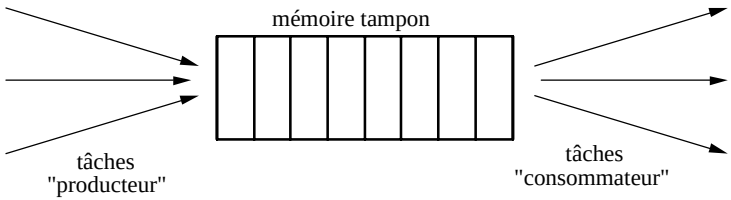
- TOWaitEvent (may be on more than one event)
  - 1. if event is signaled, task continues
  - 2. if not, it is put in the blocked queue and a timer is started
  - 3. switch to the highest « priority » task

Waiting with timeout

- example : TOWaitEvent
- End of timer may be signaled by an event
- At each clock tick (interrupt)
  - 1. Decrement first timer in the chained list
  - 2. If zero, removes the timer block from the list
  - 3. Insert the blocked task into the ready list



Communication



- SendElement
  - 1. Update the queue
  - 2. If waiting tasks, put the highest priority one in ready queue
  - 3. Remove first element in the queue
  - 4. switch to the highest « priority » task

Communication (2)

- GetElement
  - 1. If there is an element, it is removed and the task continues
  - 2. if not, it is put in the blocked queue
  - 3. It is added to the list associated to the queue
  - 4. switch to the highest « priority » task
- IsEmpty, IsFull
- TOGetElement

Locks

- Lock
  - 1. If lock is free, it is marked as taken and a reference to the task is made, the task continues
  - 2. If not, a reference to the task is added to the list associated to the lock
  - 3. The task is inserted in the blocked queue
  - 4. The blocking reason is indicated in the TCB
  - 5. switch to the highest « priority » task + priority inheritance

csem

Verrou (2)

- UnLock
  - 1. Set the task priority to its original value
  - 2. Reference to the task is removed and the task is inserted in the ready queue
  - 3. The « first » waiting task (if any) is inserted in the ready queue
  - 4. The lock is marked as taken and a reference to that task is made
  - 5. switch to the highest « priority » task + priority inheritance
- TOLock

csem

Conclusion

- Operating principles are rather simple, but …
  - The selection of services must be carefully weighted
- Implementation rather complicated (details, exceptions, ...)
- It is often better to use off the shelf proven kernels
  - However often too complex
  - Lack of predictability
  - Check the tools
  - Better to have source code at hand

csem

Références

- H. Nussbaumer, "Informatique industrielle II", PPUR, Lausanne, 1987.
- B. Gallmeister, "POSIX.4: Programming for the Real World", O'Reilly & Associates, Bonn, 1995.
- A. Tanenbaum, "Modern Operating Systems", Prentice Hall, Upper Saddle River, 1992.
- J. Labrosse, "µC/OS the Real-Time Kernel", R&D publications, Lawrence, Kansas, 1992.

csem

