Multitasking Operating Systems

José Costa

Software for Embedded Systems

Departamento de Engenharia Informática (DEI)
Instituto Superior Técnico

2015-10-12

Outline

- Motivation for processes
- The process abstraction
- Context switching
- Multitasking

Reactive systems

- Respond to external events
  - Engine controller
  - Seat belt monitor
- Real-time response has implications
  - System architecture
  - Program implementation
- May require a chain reaction among multiple processors

Why Multiple Processes?

TÉCNICO LISBOA

Processes help us manage timing complexities

- Multiple rates
  - multimedia
  - automotive
- Asynchronous input
  - user interfaces
  - communication systems

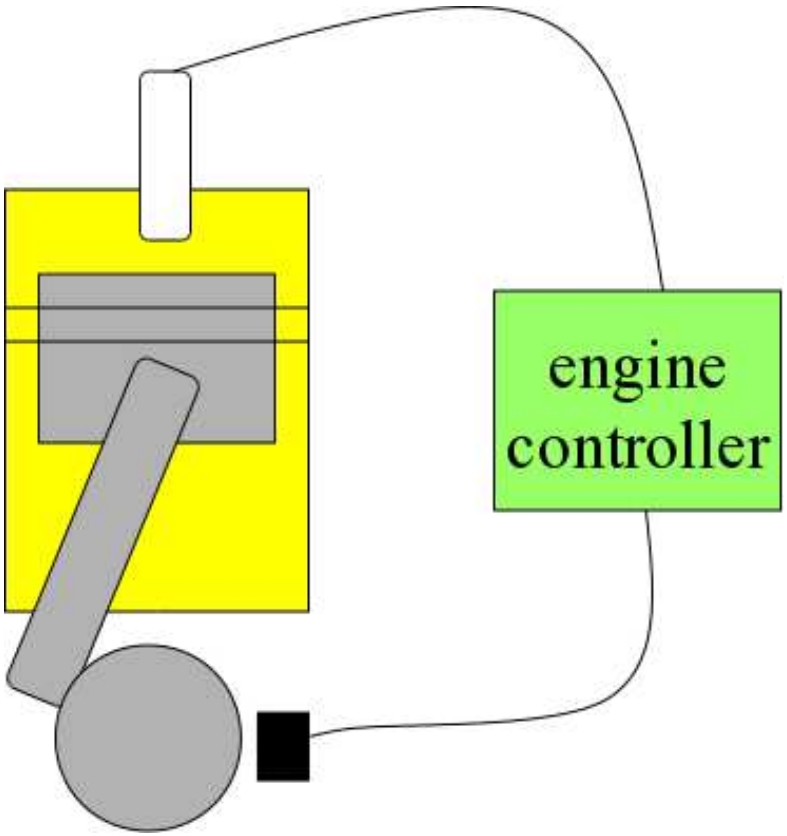
Multi-rate Systems

- Tasks may be synchronous or asynchronous
- Synchronous tasks may recur at different rates
- Processes run at different rates based on computational needs of the tasks

Period of a process

- Is the time between sucessive executions
- Is the inverse of the rate

Example: Engine Control



Tasks

- spark control
- crankshaft sensing
- fuel/air mixture
- oxygen sensor
- Kalman filter
- state machine

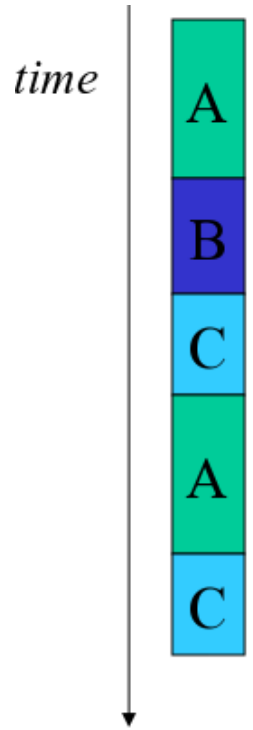
Typical Rates in Engine Control

TÉCNICO LISBOA

Variable	Full range time (ms)	Update period (ms)
Engine spark timing	300	2
Throttle	40	2
Air flow	30	4
Battery voltage	80	4
Fuel flow	250	10
Recycled exhaust gas	500	25
Status switches	100	20
Air temperature	Seconds	400
Barometric pressure	Seconds	1000
Spark (dwell)	10	1
Fuel adjustment	80	8
Carburetor	500	25
Mode actuators	100	100

Life Without Processes

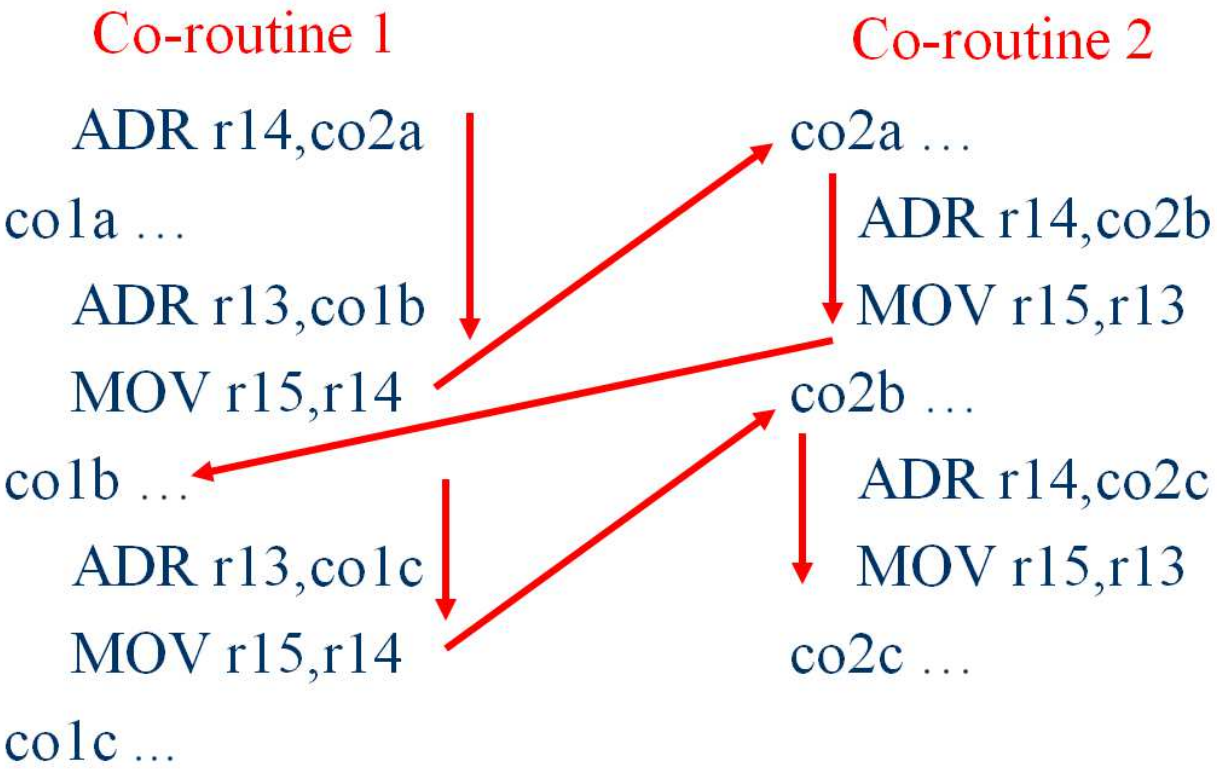
- Code turns into a mess
  - interruptions of one task for another
  - spaghetti code
- Although it can still be developed under a strong discipline (e. g. round robin architecture)




```
A_code();
...
B_code();
...
if (C) C_code();
...
A_code();
...
switch (x) {
   case C: C();
   case D: D();
   ...
```

Co-Routines

- Commonly used in the early days of embedded computing to handle multiple processes without processes
- Useful example of the complexities of not using processes for certain applications



Co-Routine Methodology

- Like subroutine, but caller determines the return address
- Co-routines voluntarily give up control to other co-routines
- Pattern of control transfers is embedded in the code

Co-Routine's Disadvantages

- Does not do nearly enough to help us construct complex programs with significant timing properties

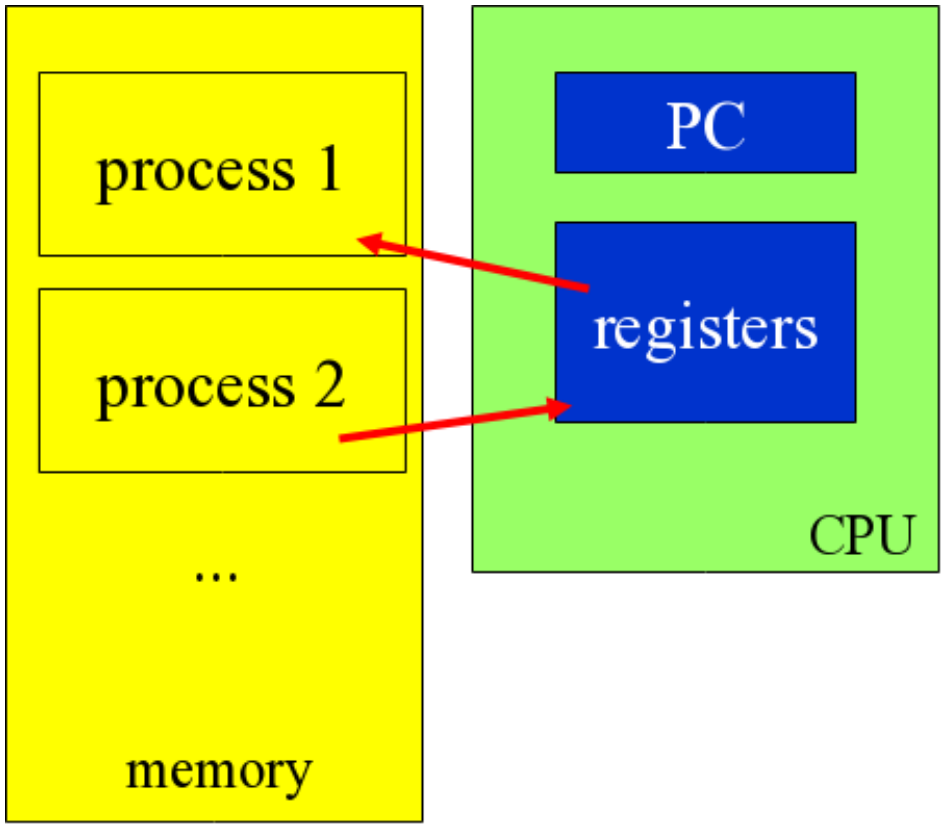
- Can be ugly to work with
  - it's hard to trace through the flow of control

Processes

- Processes allow us to handle complex systems
  - fundamental abstraction for dealing with multiple simultaneous operations
- A process is a unique execution of a program
  - defined by its data and code
  - several copies of a program may run simultaneously or at different times
- A process has its own state
  - registers
  - memory
- The operating system manages processes

Processes and CPUs

- Activation record: copy of process state
- Context switch
  - current CPU context goes out
  - new CPU context goes in



Terms

Thread (lightweight process)

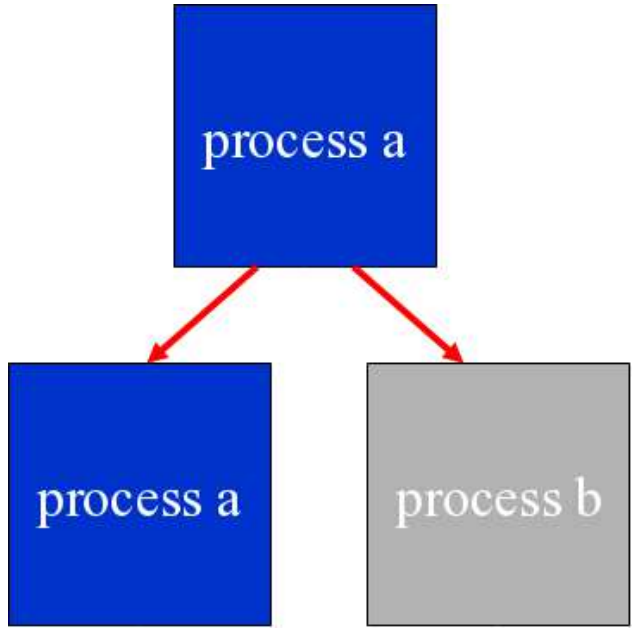
- a process that shares memory space with other processes

Reentrancy

- ability of a program to be executed several times with the same results
- programs are easier to debug if they are reentrant

Processes in POSIX

- Create a process with fork
  - parent process keeps executing old program
  - child process executes new program



fork()

- The fork process creates child:

```
childid = fork();
if (childid == 0) {
  /* child operations */
} else {
  /* parent operations */
}
```

execv()

- Overlays child code:

```
childid = fork();
if (childid == 0) {
  execv(“mychild”,childargs);
  perror(“execv”);
  exit(1);
}
```

Context Switching

- It's the mechanism for moving the CPU from one executing process to another

- A process should not be able to tell that it was stopped and then restarted

Questions

- Who controls when the context is switched?
- How is the context switched?

Co-operative Multitasking

- Improvement on co-routines
  - hides context switching mechanism
  - still relies on processes to give up CPU
- Each process allows a context switch call
- Separate scheduler chooses which process runs next
- Implementable with a function-queue architecture

Problems with Co-operative Multitasking

Programming errors can keep other processes out

- Process never gives up CPU
- Process waits too long to switch, missing input

Context Switching

- Must copy all registers to activation record, keeping proper return value for PC

- Must copy new activation record into CPU state

Context Switching in ARM

- Save old process:

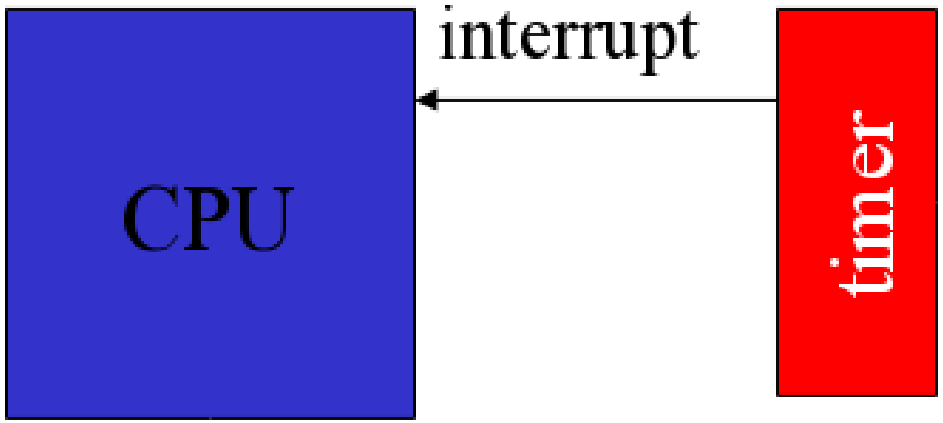
```
STMIA r13,{r0-r14}^ ; save registers
MRS r0,SPSR
STMDB r13,{r0,r15}  ; save status register and PC
```

- Start new process:

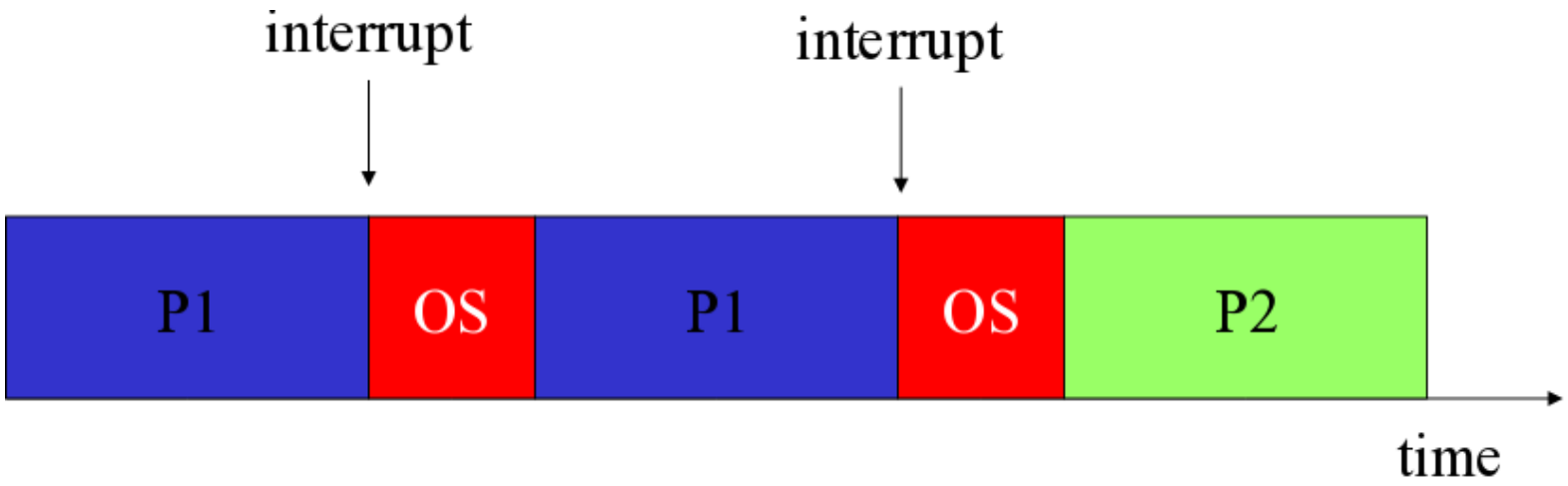
```
ADR r0,NEXTPROC     ;  get address of next process
LDR r13,[r0]
LDMDB r13,{r0,r14}
MSR SPSR,r0         ; set status register
LDMIA r13,{r0-r14}^ ; get registers
MOVS pc,r14         ; restore PC
```

Preemptive Multitasking

- Most powerful form of multitasking
- OS controls when contexts switches
- OS determines what process runs next
- Uses timer to call OS and to switch contexts



Flow of Control with Preemption



Preemptive Context Switching

- Timer interrupt gives control to OS, which saves interrupted process's state in an activation record
- OS chooses next process to run
- OS installs desired activation record as current CPU state

Why Not Use Interrupts?

How

- We could change the interrupt vector at every period
- When there was an interrupt the corresponding process would run

But

- We would need management code anyway
- We would have to know the next period's process at the start of the current process

Evaluating Performance

- May want to test
  - context switch time assumptions
  - scheduling policy

- Can use OS simulator
  - to exercise process set
  - trace system behavior

Processes and Caches

- Processes can cause additional caching problems
  - even if individual processes are well-behaved, processes may interfere with each other

- Worst-case execution time with bad cache behavior is usually much worse than execution time with good cache behavior

Examples of Operating Systems

- There are lots of operating systems targeted at embedded systems
- Used in a myriad of applications
- With more or less features
- Ported to great number of microprocessors

TinyOS

- Started as a collaboration between the University of California, Berkeley in co-operation with Intel Research and Crossbow Technology
- Targeted at Wireless Sensor Networks
- Programmed in nesC
- Interfaces and components for common abstractions
  - packet communication, routing, sensing, actuation and storage

DSPnano

- embedded real-time operating system
- 100% compatible with POSIX
  - capabilities for threads, communication, synchronization and I/O
- Programmed in C
- Targeted at high performance real time digital signal processing

ThreadX

- Real-time operating system
- Multitasking kernel with preemptive scheduling, fast interrupt response, memory management, interthread communication, mutual exclusion, event notification, and thread synchronization features
- Programmed in C
- Used in printer products of HP
  - targeted also at consumer electronics, medical devices, data networking applications, and SoC development

NetBSD

- Open-source Unix-like operating system
- Derived from the BSD
- Ported to a large number architectures
  - "Of course it runs NetBSD"
- Used in routers, switchers and “a toaster”

FreeRTOS

- Real-time operating system
- Designed to be small and simple
  - only three C source files
- Provides methods for multiple threads or tasks, mutexes, semaphores and software timers
- Good portability

uClinux

- Pronounced "you-see-Linux"
- Fork of the Linux kernel for microcontrollers
  - without a memory management unit
- Used in network routers, security cameras, DVD/MP3 players, VoIP phone or Gateways, scanners and card readers.

Contiki

- Provides multitasking and built-in TCP/IP stack
- Run on devices that are severely constrained in terms of memory, power, processing power, and communication bandwidth
  - focus on low-power wireless internet of things devices
- Used in street lighting systems, sound monitoring for smart cities, radiation monitoring systems, and alarm systems

Others

- Smartphones
  - Android, iOS, ...
- Cisco IOS
- brickOS, leJOS
- many, many more

Outline

- Motivation for processes
- The process abstraction
- Context switching
- Multitasking

References

Computers as Components: Principles of Embedded Computing System Design , Marilyn Wolf. Morgan Kaufman. Ch. 6.1-6.4

Next Class

- Scheduling policies