Programming Real-Time Embedded systems : C/POSIX and RTEMS

1. Introduction
2. Operating systems for Real-Time applications
3. Market
4. POSIX 1003 Standard
5. RTEMS operating system
   (a) POSIX thread model of RTEMS and fixed priority scheduling
   (b) Synchronization tools
   (c) Clocks and timers management
6. Summary
7. References

Introduction

- **Properties/constraints of embedded real-time systems:**

  1. As any real-time systems: functions and timing behavior must be predictable.
  2. Extra requirements or constraints:
     - Limited resources: memory footprint, power, ...
     - Reduced accessibility for programmers.
     - High level of autonomy (predictability).
     - Interact with their environment, with sensors/actuators (predictability).

- **Various kinds of execution platforms.**

Summary

1. Introduction
2. Operating systems for Real-Time applications
3. Market
4. POSIX 1003 Standard
5. RTEMS operating system
   (a) POSIX thread model of RTEMS and fixed priority scheduling
   (b) Synchronization tools
   (c) Clocks and timers management
6. Summary
7. References

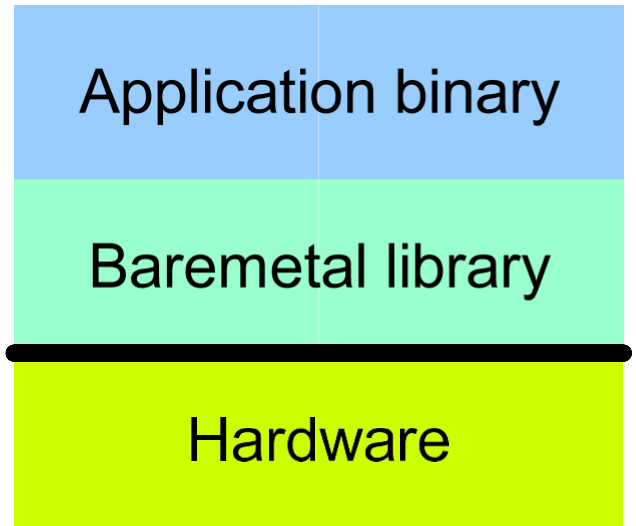
Execution platform (1)

- **Main features/criteria**
  - **Real-time abstractions:** tasks, scheduling, interrupts, synchronization and communication tools, ...
  - **Portability:** by the architecturen the standards (POSIX 1003, Ada 2005).
  - **Configurability:** mandatory versus optional parts. Adaptation to application requirements. Memory footprint.
  - **Isolation:** boundary of memory protection mechanisms.
  - Support of real-time languages: mainly C, C++ and Ada.
  - Ease access to hardware resources/devices.
  - Level of predictability.
  - Programming/compiling environment.

Execution platform (2)

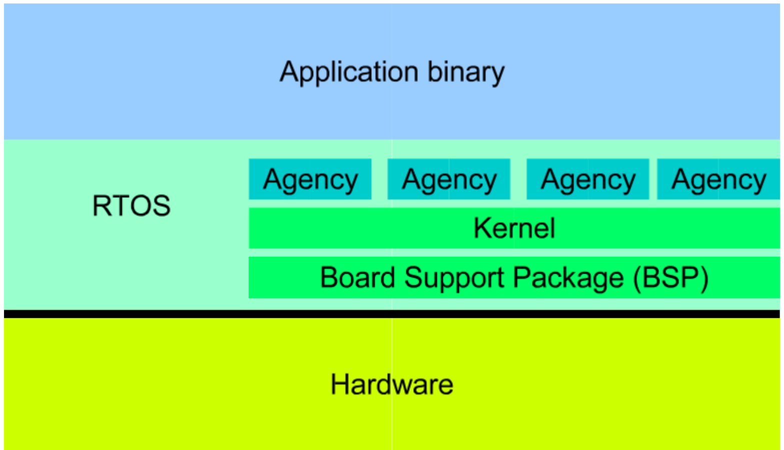
- **Types of execution platforms**
  - Bare-metal runtime: no operating system (OS).
  - Real-time operating systems (RTOS): OS but usually no system calls and memory protection.
  - Real-time unix: OS, system calls, memory protection, spacial isolation.
  - Time and space partitioned systems (TSP): both memory and time isolation (by scheduling).

Execution platform (3)



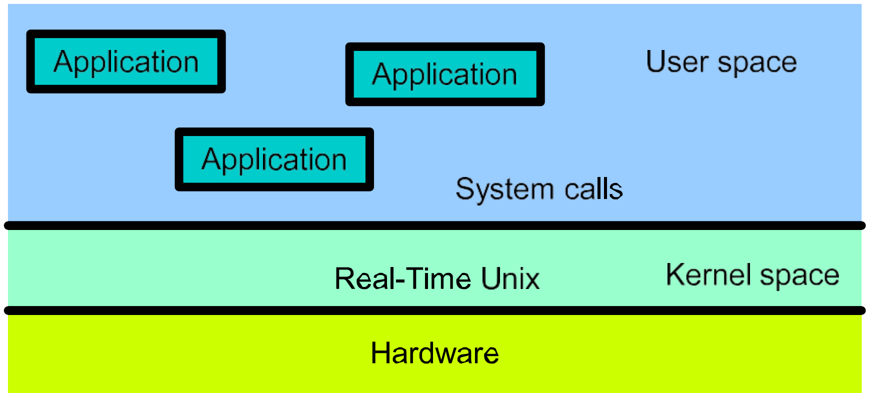
- **Bare-metal**
  - Highest level of predictability. Full access to hardware.
  - No memory protection : system and application are linked alltogether
  - High development cost if system re-design.
  - System services ; e.g. off-line scheduling, no concurrent tasks.
  - Think about a library linked with your application.

Execution platform (4)



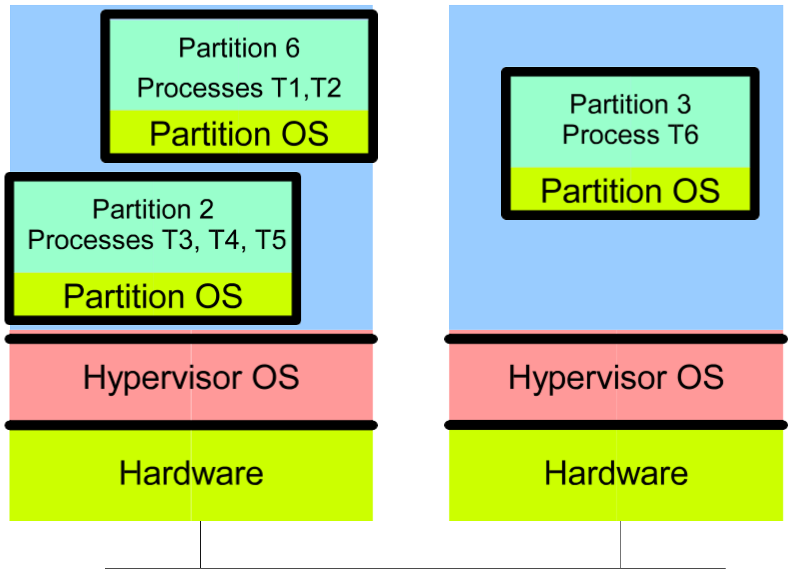
- **RTOS, Real-Time operating system**
  - High predictability. Full access to hardware.
  - Concurrent tasks and online ressource managment (scheduling).
  - System needs configuration (agencies). More flexible if system re-design. Portability brought by layers: kernel and BSP.
  - No memory protection. Both application tasks and kernel tasks share the same address space.

Execution platform (5)



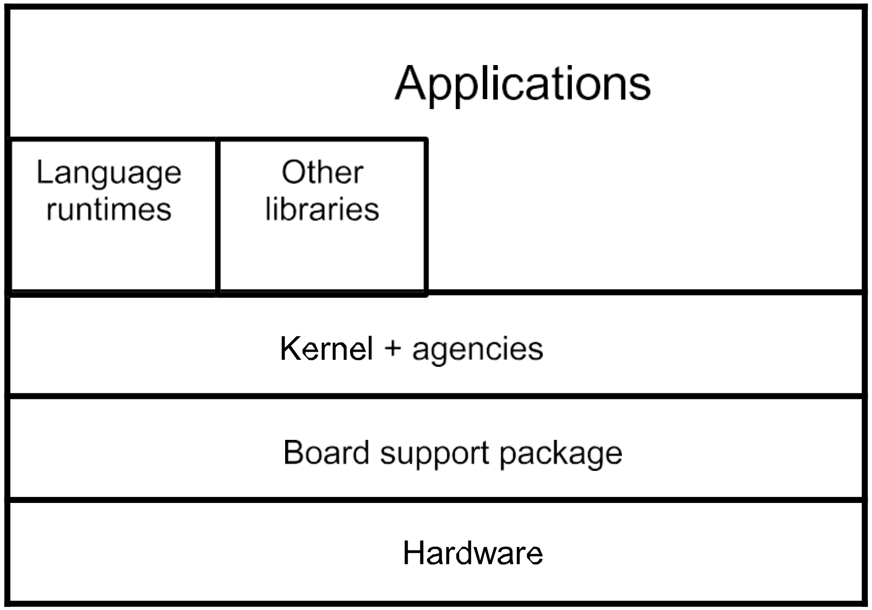
- **Real-time Unix**
  - A Unix, but with higher preemptivity and real-time scheduling features.
  - Lower level of predictibility.
  - Usual programming environnment (i.e. no need to cross-compiler).
  - Classical Unix Process/thread memory protection : kernel and user space + system calls.

Execution platform (6)



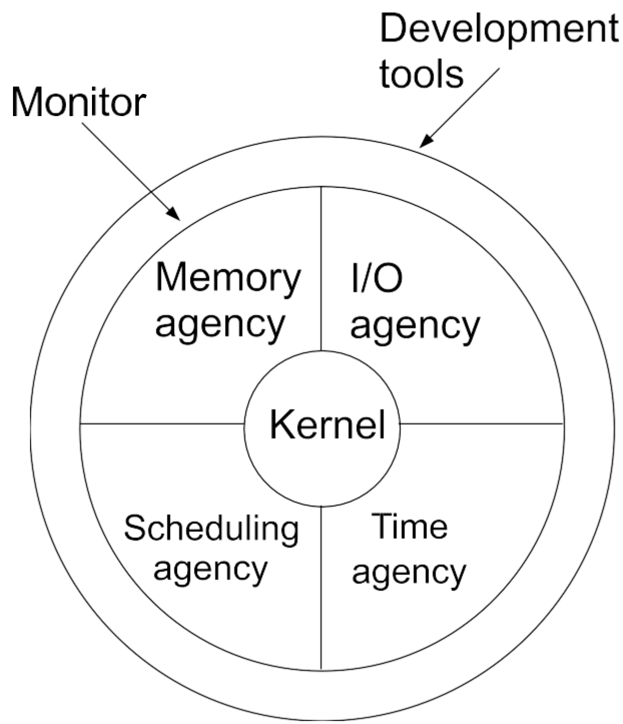
- **Time and space Paritionning execution platform**
  - Concepts of partitions and processes.
  - Enforce both temporal and space isolation.
  - 2 levels of OS, of scheduling (off-line partition scheduling + online process scheduling), of communication/synchronization (intra and inter partitions).

RTOS: Real-time operating system (1)



- **Portability of programs:** layered architecture to increase portability
  - Language runtimes: allow to run a program written with a given language (C or Ada).
  - BSP/Board support package: allows to port a system on different hardware devices/processors. Contains drivers.

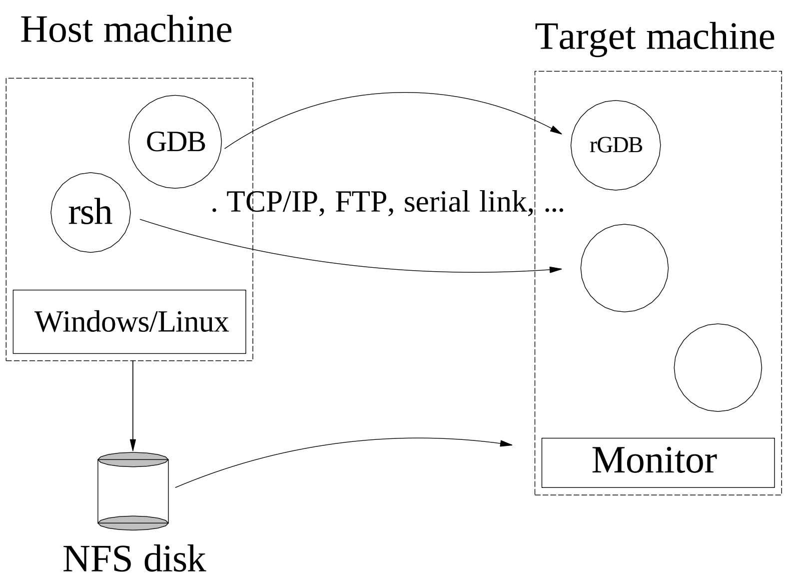
RTOS: Real-time operating system (2)



- **Configurability**: required because small amount of resources : we only put into the system the mandatory agencies.

  - **Kernel:** mandatory part of the monitor.
  - **Agencies:** optional parts, depending on the hardware, on the application/system requirements.

RTOS: Real-time operating system (3)



- **Cross-compiling**: because targets have a limited amount of resource (configurability) and are composed of specific hardware/software (timing behavior).
- **Host:** where we compile the program.
- **Target:** where we run the program.

RTOS: Real-time operating system (4)

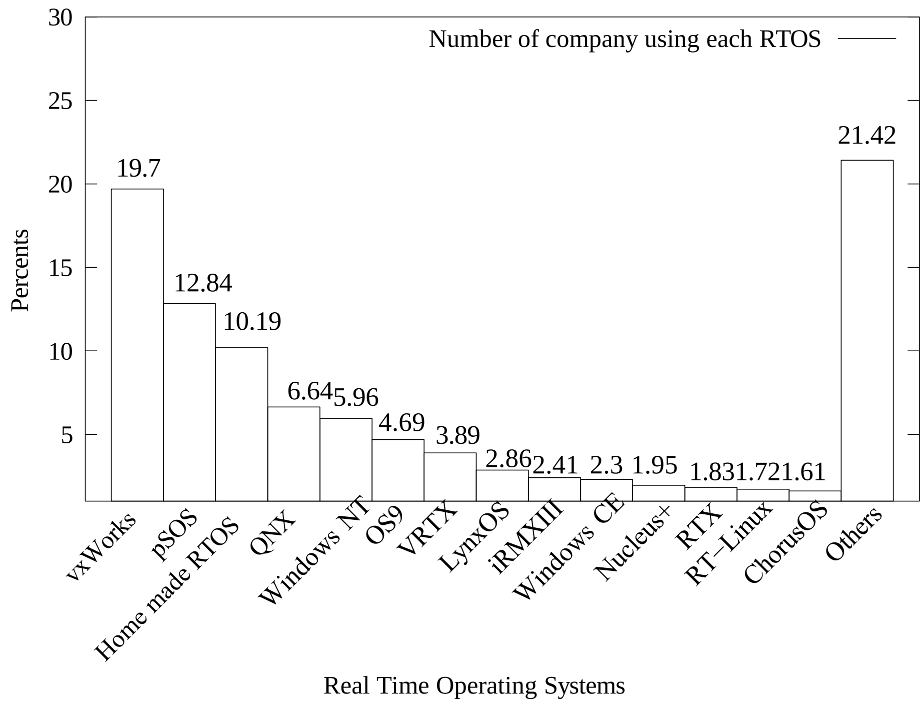
- **Performances are a priori known and deterministic**
  - Allow schedulability analysis (task capacities).
  - Use of benchmarks (e.g. *Rhealstone*, *Hartstone*, etc).

- **Main criteria**
  - Latency on interrupt.
  - Latency on context switches.
  - Latency on preemption.
  - Semaphore shuffle (latency between the release of a semaphore and its allocation by a waiting task).
  - Worst case response time of each system call, each subprogram of each library, ...
  - etc

Summary

1. Introduction
2. Operating systems for Real-Time applications
3. Market
4. POSIX 1003 Standard
5. RTEMS operating system
   (a) POSIX thread model of RTEMS and fixed priority scheduling
   (b) Synchronization tools
   (c) Clocks and timers management
6. Summary
7. References.

Market (1)



- **Specificities of this market [TIM 00]**
  - Large number of products: each product is devoted to a very few application types or domains.
  - Many "home made" products.

Market (2)

- **Commercial**
  - VxWorks (RTOS, large spectrum of use e.g. Pathfinder, french satellite).
  - pSOS (RTOS, mobile phone, military systems).
  - VRTX (RTOS, mobile phone, military systems).
  - LynxOs (real-time unix).
  - PikeOS (TSP).
- **Open-source**
  - OSEK-VDX (RTOS, automotive systems).
  - RTEMS (RTOS, space and military applications).
  - eCos (RTOS).
  - RT-Linux, RTAI, Xenomai (real-time unix).
  - POK (TSP).

Summary

1. Introduction
2. Operating systems for Real-Time applications
3. Market
4. POSIX 1003 Standard
5. RTEMS operating system
   (a) POSIX thread model of RTEMS and fixed priority scheduling
   (b) Synchronization tools
   (c) Clocks and timers management
6. Summary
7. References.

POSIX 1003 standard (1)

- Define a standardized interface of an operating system similar to UNIX [VAH 96].

- Published by ISO and IEEE. Organized in chapters:

Chapters	Meaning
POSIX 1003.1	System Application Program Interface (e.g. *fork*, *exec*)
POSIX 1003.2	Shell and utilities (e.g. *sh*)
POSIX 1003.1b [GAL 95]	Real-time extensions.
POSIX 1003.1c [GAL 95]	Threads
POSIX 1003.5	Ada POSIX binding
...	

- Each chapter provides a set of services. A service may be mandatory or optional.

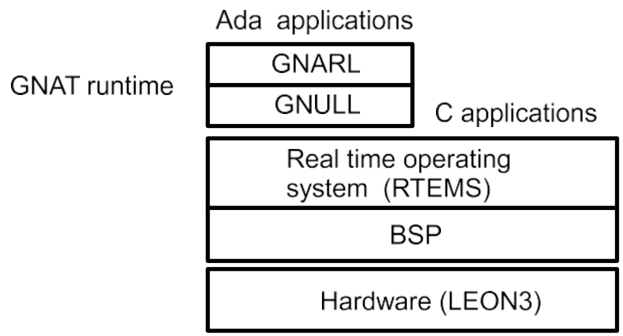
POSIX 1003 standard (2)

- Example of operating systems providing 1003.1b: Lynx/OS, VxWorks, Solaris, Linux, QNX, etc .. (actually, most of real-time operating systems).
- POSIX 1003.1b services :

Name	Meaning
_POSIX_PRIORITY_SCHEDULING	Fixed priority scheduling
_POSIX_REALTIME_SIGNALS	Real-time signals
_POSIX_ASYNCHRONOUS_IO	Asynchronous I/O
_POSIX_TIMERS	WatchDogs
_POSIX_SEMAPHORES	Synchronization tools
...	

Summary

Introducing RTEMS (1)



- **RTEMS operating system**
  - RTEMS: GNU GPL real-time operating system for C and Ada small hard real-time systems.
  - Available for numerous BSP (included processor Leon : 32 bits, VHDL open-source, compliant with SPARC).
  - RTEMS has several API: native, Itron, POSIX and Ada (GNAT/Ada 2005 compiler from AdaCore).
  - Well adapted for space/aircraft applications.
  - Cross-compiling: compile on Linux, run on Leon.

Introducing RTEMS (2)

- **RTEMS model of concurrency**
  - Single process/address space and multiple threads
  - All flows of control (threads) share the same address space.
  - **Why one address space only**
    - Simple memory model implies more deterministic behavior.
    - Real-time system: only one application started when the system is switched on: no need to isolate several applications.
    - Ease flows of control communication and make them efficient.

Introducing RTEMS (3)

- **Process in Unix** = execution context + private address space (i.e. not shared with the other Unix processes).
- For safety, each process has its own address space.

```
#include <stdio.h>

int a=100;

int main(int argc, char* argv[]) {
        if(fork()==0)
                a+=100;
        else    a+=200;
        printf("a = %d\n",a);
}
```

- What is displayed with Unix?
- And with RTEMS?

Introducing RTEMS (4)

- **Simple RTEMS C program**

```
#define CONFIGURE_MAXIMUM_POSIX_THREADS 10
#define CONFIGURE_MAXIMUM_POSIX_MUTEXES 7
#define CONFIGURE_MAXIMUM_POSIX_TIMERS 16
#define CONFIGURE_MAXIMUM_POSIX_QUEUED_SIGNALS 40

#define CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER
#define CONFIGURE_APPLICATION_NEEDS_TIMER_DRIVER

#include <stdio.h>

void* POSIX_Init(void *argument) {
  printf("Hello world RTEMS\n");
  exit(0);
  return NULL;
}
```

Introducing RTEMS (5)

- $POSIX\_Init()$: main entry point. High priority level flow of control that initializes the application => the application starts at $POSIX\_Init()$ completion => critical instant (real-time scheduling theory).
- $exit()$: stops the application. We can switch off the board!
- C macros: to select embedded agencies and resource requirements (number of threads, number of semaphores) => constraints of embedded systems. Defined in $system.h$ in the sequel.

Introducing RTEMS (6)

- **Cross compiling**

1. Compile on Linux and generate a SPARC binary:

```
#make
sparc-rtems4.8-gcc --pipe -B/home/singhoff/ADA/rtems-4.8//sparc-rtem
-g -Wall  -O2 -g -g    -mcpu=cypress -msoft-float
-o o-optimize/hello.exe  o-optimize/init.o
sparc-rtems4.8-nm -g -n o-optimize/hello.exe > o-optimize/hello.num
sparc-rtems4.8-size o-optimize/hello.exe
   text    data     bss     dec     hex filename
 109840    3652    5360  118852   1d044 o-optimize/hello.exe

#file o-optimize/hello.exe
o-optimize/hello.exe: ELF 32-bit MSB executable, SPARC, version 1 (S

#file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.15, stripped
```

Introducing RTEMS (7)

- **Cross-compiling (cont)**

2. Send the binary to the Board/Leon processor (TCP/IP, serial link, ...).
3. Run the program on the board/Leon processor. Software emulator `tsim` (Leon 3 processor emulator).

```
#tsim o-optimize/hello.exe
TSIM/LEON3 SPARC simulator, version 2.0.15 (evaluation version)
allocated 4096 K RAM memory, in 1 bank(s)
allocated 32 M SDRAM memory, in 1 bank
allocated 2048 K ROM memory
read 2257 symbols
tsim> run
resuming at 0x40000000
** Init start **
** Init end **
Hello world RTEMS
Program exited normally.
tsim> quit
```

Summary

1. Introduction
2. Operating systems for Real-Time applications
3. Market
4. POSIX 1003 Standard
5. RTEMS operating system
   (a) POSIX thread model of RTEMS and fixed priority scheduling
   (b) Synchronization tools
   (c) Clocks and timers management
6. Summary
7. References.

POSIX threads with RTEMS (1)

- Compliant with chapter POSIX 1003.1c. Define both thread and synchronization tools.
- $POSIX\_Init()$: main thread of the application
- $exit()$: stops all threads. We can switch off the board!
- A thread inherit scheduling parameters from its creating thread.
- $system.h$: configure RTEMS kernel according to the number of threads (and semaphores too) => we cannot create threads as much as we want (deterministic system).

POSIX threads with RTEMS (2)

pthread_create	Spawn a thread. Parameters : code, attributes, arg.
pthread_exit	Terminate a thread. Parameters : return code.
pthread_self	Return thread id
pthread_cancel	Delete a thread. Parameters : thread id.
pthread_join	Wait for the completion of a son.
pthread_detach	Delete relationship between a son and its father.
pthread_kill	Send a signal to a thread.
pthread_sigmask	Change signal mask of a thread.

POSIX threads with RTEMS (3)

```
void* th(void* arg) {
        printf("Thread %d is running\n", pthread_self());
        pthread_exit(NULL);
}

void* POSIX_Init( void *argument) {
 pthread_t id1 ,id2;
 if(pthread_create(&id1,NULL,th,NULL)!=0)
  perror("pthread_create1");
 if(pthread_create(&id2,NULL,th,NULL)!=0)
  perror("pthread_create2");
 if(pthread_join(id1,NULL)!=0)
   perror("pthread_join 1");
 if(pthread_join(id2,NULL)!=0)
   perror("pthread_join 2");
 printf("End of the application\n");
 exit(0);
}
```

POSIX threads with RTEMS (4)

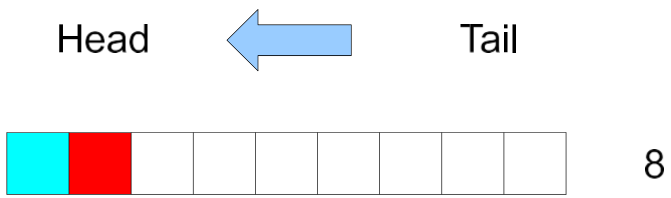
- Compile and run

```
#make
sparc-rtems4.8-gcc ...
#
#tsim o-optimize/join.exe
tsim> run
Thread 184614914 is running
Thread 184614915 is running
End of application
Program exited normally.
tsim> quit
```

POSIX 1003 scheduling (1)

- **POSIX real-time scheduling model**
  - Preemptive fixed priority scheduling. At least 32 priority levels.
  - Scheduling parameters are either inherited ($PTHREAD\_INHERIT\_SCHED$ attribute) of explicitly changed ($PTHREAD\_EXPLICIT\_SCHED$ attribute).
  - Two-levels scheduling:
    1. Choose the queue which has the highest priority level with at least one ready process/thread.
    2. Choose a process/thread from the queue selected in (1) according to a **policy**.

POSIX 1003 scheduling (2)



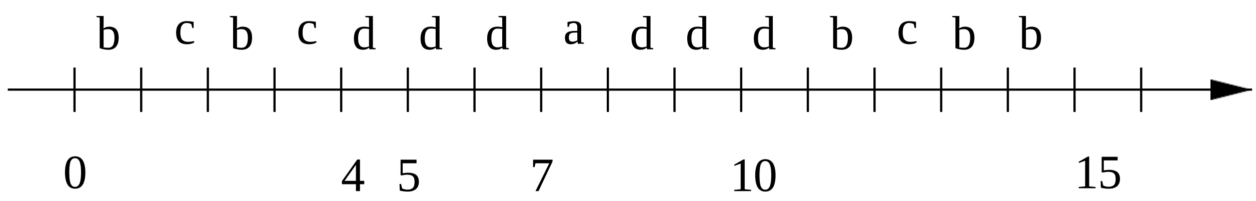
- **POSIX policies:**

  1. $SCHED\_FIFO$: when a thread becomes ready, it is inserted in the tail of its corresponding priority queue. Give the processor to the thread in the head of the queue. When blocked or terminated, a thread leaves the queue and the next process/thread in the queue gets the processor.
  2. $SCHED\_RR$: $SCHED\_FIFO$ with a time quantum. A time quantum is a maximum duration that a thread can run on the processor before preemption by an other thread of the same queue. When the quantum is exhausted, the preempted thread is moved to the tail of the queue.
  3. $SCHED\_OTHER$: implementation defined (may implement a time sharing scheduler).

POSIX 1003 scheduling (3)

- Example:

Task	C_i	S_i	Priority	Policy
a	1	7	1	FIFO
b	5	0	4	RR
c	3	0	4	RR
d	6	4	2	FIFO



- Quantum SCHED_RR = 1 unit of time.
- Highest priority level 1.

POSIX 1003 scheduling (4)

- **POSIX policy**

```
#define SCHED_OTHER      0
#define SCHED_FIFO       1
#define SCHED_RR         2
```

- **Scheduling parameters**

```
struct sched_param
  {
    int sched_priority;
    ...
  };
```

- **We can perform scheduling parameter updates**

1. When threads are created (with attribute or inheritance).
2. At any time (with specific POSIX functions).

POSIX 1003 scheduling (5)

sched_get_priority_max	Read maximum priority level
sched_get_priority_min	Read minimum priority level
sched_rr_get_interval	Read quantum
sched_yield	Release the processor
pthread_setschedparam	Assign priority/policy
pthread_getschedparam	Read priority/policy

Thread attributes (1)

- **Attributes:** properties of a thread that are set at thread creation.
  - Have a default value (e.g. stacksize).

Attribute name	Meaning
detachstate	*pthread_join* possible or not
schedpolicy	scheduling policy
schedparam	fixed priority (and other parameters)
inheritsched	inheriting scheduling parameters
stacksize	thread memory requirement
stackaddr	address of the thread stack

$\Longrightarrow$ Allow to customize threads for real-time systems

- Specification of resource requirements: memory/stack.
- Specification of scheduling parameters.

Thread attributes (2)

- $pthread\_attr\_t$ type: store attribute data. Must be initialized before thread creation.

$pthread\_attr\_init$	Allocate an attribute
$pthread\_attr\_delete$	Remove an attribute
$pthread\_attr\_setATT$	Set a value to an attribute
$pthread\_attr\_getATT$	Read the value of an attribute

with ATT, the name of the attribute.

Thread attributes (3)

```
void* th(void* arg) ...

void* POSIX_Init(void *argument) {
 pthread_attr_t attr;
 pthread_t id;
 struct sched_param param;

 pthread_attr_init(&attr);
 if (pthread_attr_setinheritsched(&attr,PTHREAD_EXPLICIT_SCHED)!=0)
  perror("pthread_attr_setinheritsched");
 if (pthread_attr_setschedpolicy(&attr,SCHED_RR)!=0)
  perror("pthread_attr_setschedpolicy");
 param.sched_priority=130;
 if (pthread_attr_setschedparam(&attr,&param)!=0)
  perror("pthread_attr_setschedparam");

 if (pthread_create(&id,&attr,th,NULL)!=0)
  perror("pthread_create");
```

Summary

1. Introduction
2. Operating systems for Real-Time applications
3. Market
4. POSIX 1003 Standard
5. RTEMS operating system
   (a) POSIX thread model of RTEMS and fixed priority scheduling
   (b) Synchronization tools
   (c) Clocks and timers management
6. Summary
7. References

Synchronization tools (1)

- **Different types**

1. Mutexes
2. Counting semaphores
3. Conditional variables

Mutexes versus semaphores (1)

- **Mutexes**
  - Optimized for critical section only:
    - P() and V() of the same mutex in this order only.
    - The task who does the last P() of a mutex must be the one who will do the next V() on that mutex.
    - Inner P() of nested P() of a mutex, called by the same task, may be not blocking.
    - Cannot be used otherwise.
  - Most of the time based on efficient hardware mechanisms, i.e. spinlock, test-and-set.

Mutexes versus semaphores (2)

- **Counting semaphores**
  - Less efficient than mutexes.
  - No priority inheritance (no PCP like protocol).
  - Can be used to build any synchronization
    - P() and V() can be called in any order.
    - P() and V() of the same semaphore can be called by different tasks.

Mutex (1)

- Semaphores that are optimized for critical section.
- Composed of a queue and a boolean.
- Semaphore queue : threads are sorted according to their priority if $SCHED\_FIFO$ or $SCHED\_RR$.
- Behavior can be tailored with attributes:

Attribute name	Meaning
protocol	Inheritance protocol
pshared	not used with RTEMS
prioceiling	PCP/PIP priority ceiling

- *protocol* can have the following values:
  - $PTHREAD\_PRIO\_NONE$: blocking order is FIFO.
  - $PTHREAD\_PRIO\_INHERIT$: blocking order is priority with PIP.
  - $PTHREAD\_PRIO\_PROTECT$: blocking order is priority with PCP.

Mutex (2)

pthread_mutex_init	Initialize a mutex
pthread_mutex_lock	Lock ; may be blocking
pthread_mutex_trylock	Try to lock ; unblocking primitive
pthread_mutex_unlock	Unlock
pthread_mutex_destroy	Delete a mutex
pthread_mutexattr_init	Initialize an attribute
pthread_mutexattr_setATT	Set an attribute
pthread_mutexattr_getATT	Read an attribute

with ATT, the name of the attribute.

Counting semaphore (1)

- Can be used for any synchronization, and not only critical section.
- Semaphore composed of a queue and an integer.
- No attribute.
- Semaphore queue: threads are sorted according to their priority if $SCHED\_FIFO$ or $SCHED\_RR$.

Counting semaphore (2)

sem_init	Initialize a semaphore
sem_destroy	Delete a semaphore
sem_post	Unlock semaphore.
sem_wait	Lock a semaphore ; may be blocking
sem_trywait	Unblocking locking semaphore

Counting semaphore (3)

- **Example:**

```
sem_t sem;

void* POSIX_Init( void *argument) {
  pthread_t id;   struct timespec delay;

  if(sem_init(&sem,0,0)!=0)
   perror("sem_init");

  if(pthread_create(&id,NULL,th,NULL)!=0)
    perror("pthread_create");

  delay.tv_sec=4;   delay.tv_nsec=0;
  nanosleep(&delay,NULL);

  printf("Main thread %d : unlock thread %d\n",pthread_self(),id);
  if(sem_post(&sem)!=0)
  ...
```

Counting semaphore (4)

- **Example (cont):**

```
void* th(void* arg) {
   printf("thread %d is blocked\n",pthread_self());
   if(sem_wait(&sem)!=0)
    perror("sem_wait");
   printf("thread %d is released\n",pthread_self());
}
```

- **Compile and run:**

```
$make
sparc-rtems4.8-gcc ...
$
$tsim o-optimize/sem.exe
tsim>run
thread 184614914 is blocked
Main thread 184614913 : unlock the thread 184614914
thread 184614914 is released
```

Summary

1. Introduction
2. Operating systems for Real-Time applications
3. Market
4. POSIX 1003 Standard
5. RTEMS operating system
   (a) POSIX thread model of RTEMS and fixed priority scheduling
   (b) Synchronization tools
   (c) Clocks and timers management
6. Summary
7. References

Clocks and Timers (1)

- **We look for means to**
  - Set and read clocks, sometimes with different levels of precision/accuracy.
  - Suspend the execution (sleep) of a task.
  - Implement periodic releases of periodic tasks.

Clocks and Timers (2)

- Real-time system may have specific clock hardware. POSIX 1003.1b provides a generic interface, for any hardware/operating system.
- **Real-time extensions of clock service from POSIX 1003.1b**
  - A system may have several "real-time" clocks ($CLOCK\_REALTIME$ identifier).
  - Any POSIX 1003.1b must have at least one "real-time" clock.
  - Constraints on accuracy/precision: at least 20 ms. But actual precision depends on hardware and operating system.
  - Clocks can be used to create timers.

Clocks and Timers (3)

- **What is a timer**
  - A timer is an entity that is counting down events.
  - A timer as an initial value. When it reaches zero, it usually triggers the execution of a suprogram: RTEMS/POSIX triggers a signal in this case.

- **What is a signal**
  - Signal: event/message asynchronously sent to a process or a thread. Each signal has a known number (e.g. signal.h).
  - Signals can be ignored/masked, pended or delivered. Behavior can be specified by the programmer (signal table).

Clocks and Timers (4)

clock_gettime	Return current time
clock_settime	Give a value to a clock
clock_getres	Read precision of a clock
timer_create	Create a timer
timer_delete	Delete a timer
timer_getoverrrun	Return the number of pending signal for a timer
timer_settime	Start the timer
timer_gettime	Read remaining time before a timer has exhausted
nanosleep	Block a thread for an amount of time

Clocks and Timers (5)

- **Example of a timer with SIGALRM signal**

```
void *POSIX_Init( void *argument) {

  timer_t myTimer;
  struct timespec waittime;
  struct sigaction sig;
  struct itimerspec ti;
  struct sigevent event;
  sigset_t mask;

  sig.sa_flags=0;
  sig.sa_handler=handler;
  sigemptyset(&sig.sa_mask);
  sigaction(SIGALRM,&sig,NULL);

  sigemptyset(&mask);
  sigaddset(&mask,SIGALRM);
  sigprocmask(SIG_UNBLOCK,&mask,NULL);
```

Clocks and Timers (6)

- **Example of a timer with SIGALRM signal (cont)**

```
  event.sigev_notify=SIGEV_SIGNAL;
  event.sigev_value.sival_int=0;
  event.sigev_signo=SIGALRM;
  timer_create(CLOCK_REALTIME,&event,&myTimer);

  ti.it_value.tv_sec=1;
  ti.it_value.tv_nsec=0;
  ti.it_interval.tv_sec=0;
  ti.it_interval.tv_nsec=0;
  timer_settime(myTimer,0,&ti,NULL);

  printf("Wait for timer ...\n");
  waittime.tv_sec=10;
  waittime.tv_nsec=0;
  nanosleep(&waittime, NULL);

  exit(0);
  return NULL;
}
```

Clocks and Timers (7)

- **Example of a timer with SIGALRM signal (cont)**

```
void handler(int sig)
{
        printf("Signal %d received : timer exhausted\n",sig);
}
```

- **Compile and run:**

```
$make
sparc-rtems4.8-gcc ...
$tsim o-optimize/alarm.exe
tsim> run
resuming at 0x40000000
Wait for timer ...
Signal 14 received : timer exhausted

Program exited normally.
tsim> q
```

Summary

1. Introduction
2. Operating systems for Real-Time applications
3. Market
4. POSIX 1003 Standard
5. RTEMS operating system
   (a) POSIX thread model of RTEMS and fixed priority scheduling
   (b) Synchronization tools
   (c) Clocks and timers management
6. Summary
7. References

Summary

- **RTOS:** portability (architecture), configurability (resource available),cross-compiling, RTOS adapted to each domain/application.
- **RTEMS:** one process/several threads, several API including POSIX.
- **POSIX API for real-time systems :** thread and fixed priority scheduling, semaphore/mutex and inheritance protocols, timer/clock and periodic thread releases. $\Longrightarrow$ **may lead to the development of real-time applications that can be compliant with real-time scheduling theory.**

Summary

References

[GAL 95] B. O. Gallmeister. *POSIX 4 : Programming for the Real World* . O'Reilly and Associates, January 1995.

[TIM 00] M. Timmerman. « RTOS Market survey : preliminary result ». *Dedicated System Magazine*, (1):6–8, January 2000.

[VAH 96] U. Vahalia. *UNIX Internals : the new frontiers*. Prentice Hall, 1996.