Conceptes Avançats de Sistemes Operatius

Facultat d'Informàtica de Barcelona
Dept. d'Arquitectura de Computadors

Curs 2013/14 Q1

Suport per temps real

Índex

- Definicions i conceptes
- Polítiques de planificació
- Suport en Linux
- Sistemes per temps real
- Xenomai
- RT-Preempt
- Pthreads RT

Definició

- Real-time software
  - must deliver computation results
    - under application specific time constraints
  - fails when a result is made available too late
    - (or too early in some systems)
      - even if the result is otherwise correct

http://www.on-time.com/rtos-32-docs/rtkernel-32/programming-manual/tasking/real-time.htm

Temps real

- Execució dels processos
  - A temps, complint el deadline
  - Sense ser interromputs per altres processos (menys prioritaris)
- Hard
  - Perdre un deadline significa una fallada total del sistema
- Firm
  - Es poden tolerar algunes pèrdues de deadline
  - La utilitat dels resultats es nul·la si arriben després del deadline
- Soft
  - La utilitat dels resultats decau quan més tard arriben
  - Linux
- http://en.wikipedia.org/wiki/Real-time_computing
-

http://stackoverflow.com/questions/17308956/differences-between-hard-real-time-soft-real-time-and-firm-real-time

Exemples

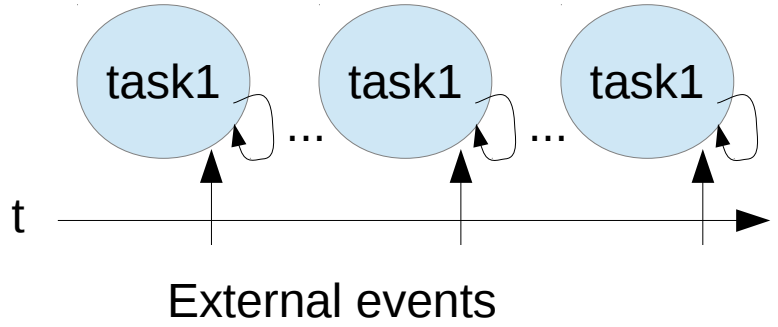
- Hard real-time
  - Control del motor d'un cotxe, frens anti-lliscament
  - Sistema de control d'un avió
  - Control d'un marcapassos
- Firm real-time?
  - *Video rendering* (també *soft*?)
- Soft real-time
  - Manteniment dels plans de vol de companyies aèries
    - Latència de segons admisible
  - Reproducció d'audio i vídeo (també *firm*?)
    - Degradació de la qualitat admisible

Conceptes

- Planificació per prioritats
  - Segons la importància de cada tasca
- Deadline
  - Assignat a una tasca, és el temps màxim en el qual la tasca s'ha d'haver executat, per tal que el sistema pugui continuar funcionant
- Deadline miss
  - Pèrdua del deadline en una tasca
  - Les conseqüències poden ser greus
    - hard, firm, soft...

Conceptes

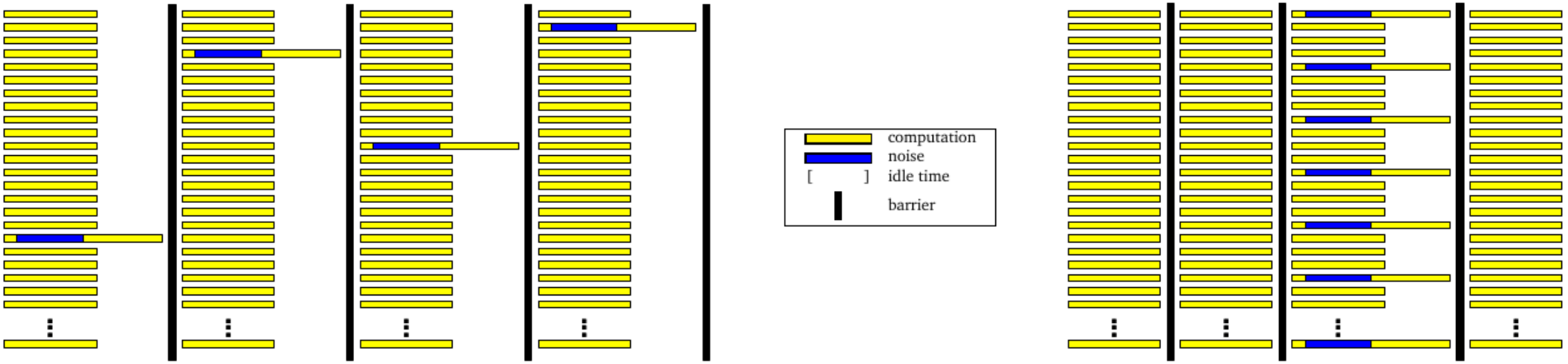
- Tasques periòdiques
  - Són aquelles que es repeteixen indefinidament
  - Seguint un període d'activació
  - Habitualment responen a un event extern



- Tasques aperiòdiques
  - Les tradicionals, que comencen i acaben, sense repetir-se necessàriament

Conceptes

- Jitter: la variació en el temps d'execució d'un procés, deguda a la seva interacció amb
  - Altres processos
  - Interrupcions



(a) Uncoordinated noise

(b) Coscheduled noise

F. Petrini, D.J. Kerbyson, S. Pakin, "The Case of the Missing Supercomputer Performance: Acheving Optimal Performance on the 8,192 Processors of ASCI Q", ACM Digital Lib, doi: http://dx.doi.org/10.1145/1048935.1050204

Comparativa OS - RTOS

- Multitasking
  - OS: per donar igual tractament a tots els usuaris/processos/fluxos (fairness, time sharing)
  - RTOS: usen les prioritats dels processos/fluxos de forma estricta
- Sobrecàrrega del sistema
  - OS: permesa
  - RTOS: no permesa
    - La sobrecàrrega anirà acompanyada de pèrdues de deadlines

http://www.on-time.com/rtos-32-docs/rtkernel-32/programming-manual/tasking/tasks.htm

RTOS Scheduler

- Té en compte la prioritat dels processos i el seu estat
  - Running en un processador
  - Ready no s'executen, però estan a punt
  - Suspended (task_suspend / task_resume)
  - Blocked esperant un event
  - Timed esperant un event, amb timeout
  - Delaying esperant que passi un temps (timer)
- Els canvis d'estat són produïts per events externs o per una altra task

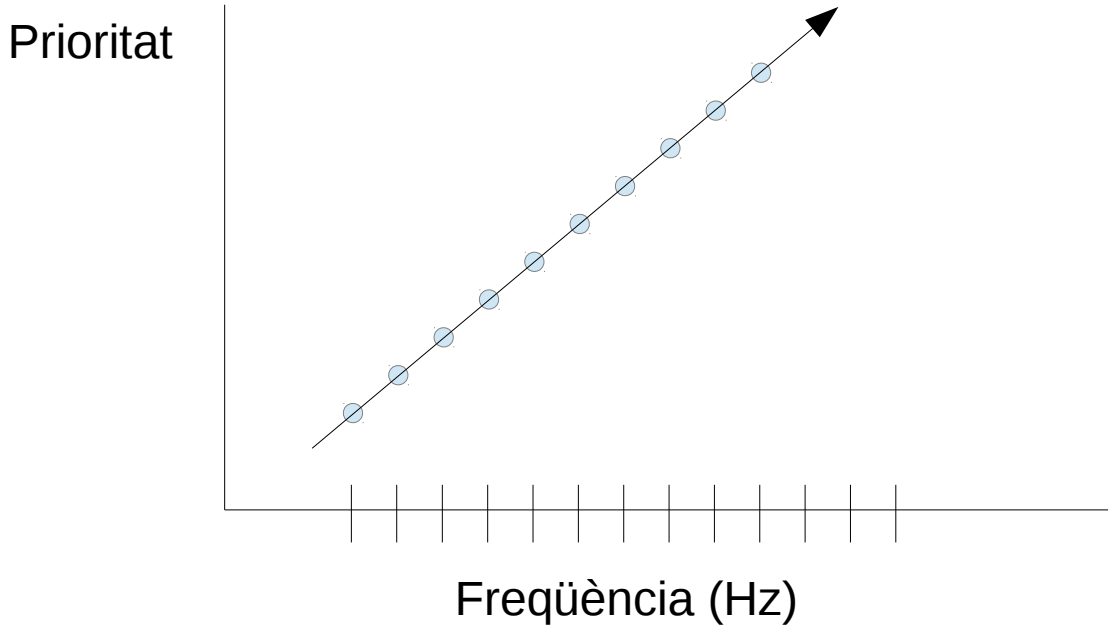
http://www.on-time.com/rtos-32-docs/rtkernel-32/programming-manual/tasking/scheduler.htm

RTOS Scheduler

- Regles que implementa
  - Els N processos (ready) amb més prioritat corren
    - tenint N processadors disponibles
  - Si s'ha de triar entre diversos processos que tenen la mateixa prioritat, s'escull aquell que fa més temps que no s'ha activat
  - Si diversos processos estan esperant (blocked) un event, s'activen quan passa l'event respectiu, en l'ordre fixat per la seva prioritat
  - En el moment en que la primera regla no és certa, es fa un canvi de context a un altre procés (més prioritari)

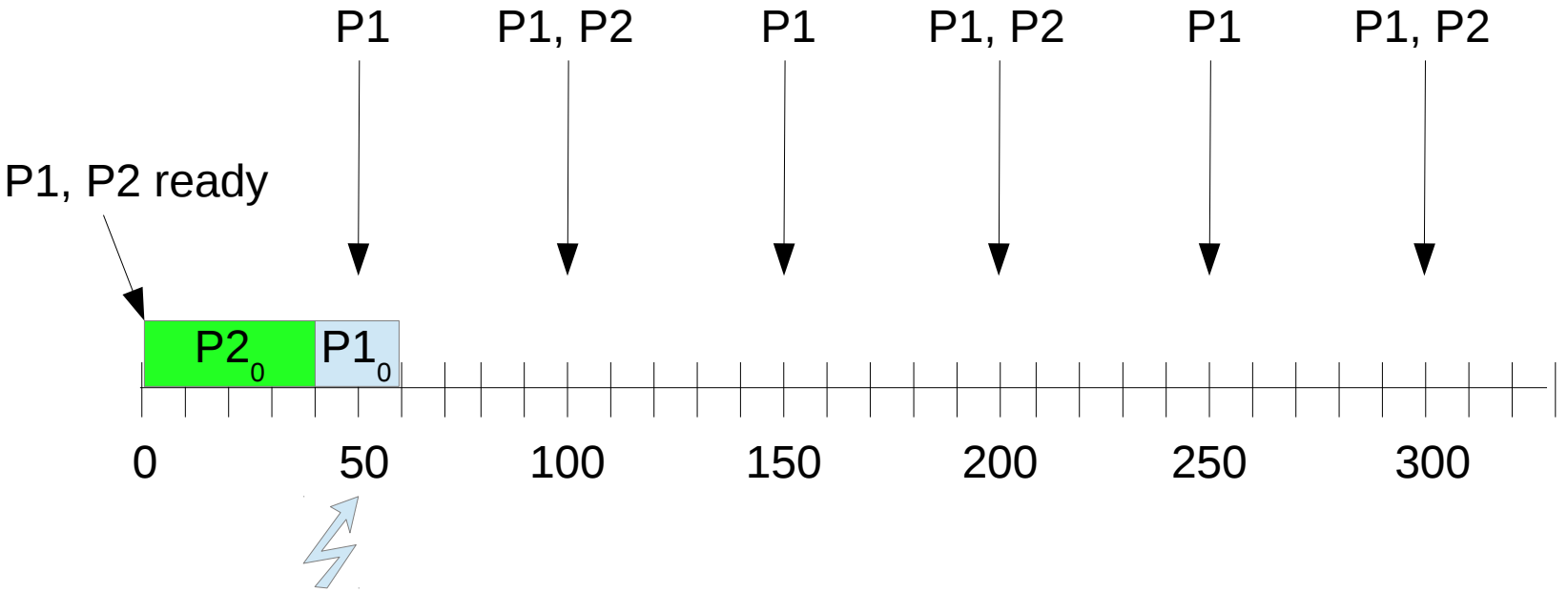
Polítiques de planificació

- Rate-monotonic scheduling (RMS)
  - Per tasques periòdiques
  - La prioritat és estàtica i és la freqüència de la tasca
  - Sempre s'executarà la tasca amb la freqüència més alta



Exemple

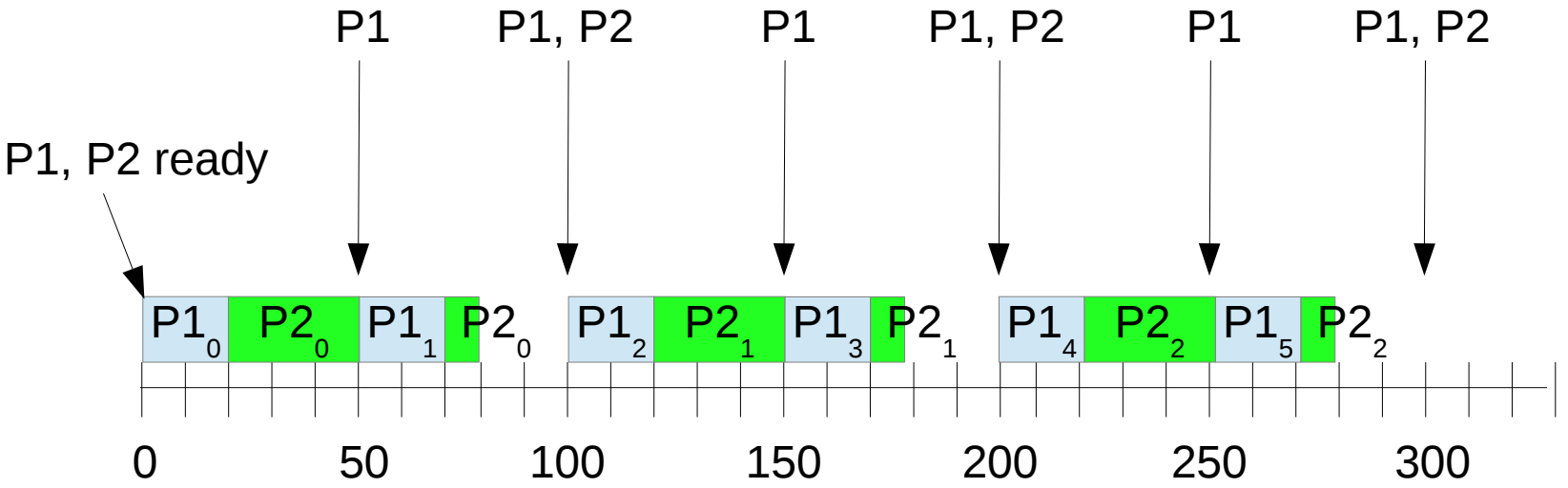
- P1: freqüència 2/100, durada: 20, prioritat "low"
- P2: freqüencia 1/100, durada: 40, prioritat "high"



  - Lliçó: segons com s'assignen les prioritats, no es compleixen els deadlines

Exemple Rate Monotonic

- P1: freqüència 2/100, durada: 20, prioritat 2
- P2: freqüencia 1/100, durada: 40, prioritat 1



  - Lliçó: executar primer la que pot tornar a demanar execució abans

Polítiques de planificació

- Earliest Deadline First
  - Per tasques periòdiques
  - La prioritat és dinàmica
    - Depèn de quan proper és el deadline de la tasca
      - Inversament proporcional al temps que falta pel deadline
    - Cada vegada que la tasca es desbloqueja ha d'indicar el seu deadline al sistema
- Exemple: P1, prioritat 1/50
  P2, prioritat 1/100
  1/50 > 1/100 → s'executa primer P1
  - Representació com en el dibuix anterior

Temps real

- Soft, linux amb suport per temps real
  - Arch Linux https://wiki.archlinux.org/index.php/Realtime_process_management
  - Debian http://www.pengutronix.com/software/linux-rt/debian_en.html
    http://www.ptxdist.org/software/ptxdist/index_en.html
  - Suse https://www.suse.com/products/realtime/
  - Gentoo http://www.gentoo.org/proj/en/desktop/sound/realtime.xml
  - RedHat http://www.redhat.com/products/mrg/realtime/
  - Ubuntu https://wiki.ubuntu.com/RealTime

PAM - pluggable authentication modules

- Permet configurar els limits dels processos
  - MEMLOCK, quantitat de memòria virtual que pot fixar-se a memòria física
  - NICE, màxim valor per a la prioritat d'usuari 0 .. 40
  - RTPRIO, màxim valor de prioritat de temps real que pot tenir el procés
  - Defineix classes de prioritats per a l'E/S
    - Realtime: IOPRIO_CLASS_RT, preferent per accedir al disc
    - Best effort: IOPRIO_CLASS_BE, classe per defecte
- Arch Linux, Gentoo

https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO
http://taste.tuxfamily.org/wiki/index.php?title=Build_PREEMPT-RT_kernel

RT-Preempt

- Proporciona característiques de hard real time a Linux
  - Permet que les regions crítiques puguin patir preempcions
    - Manté les que han de ser no-preemptibles
  - Implementa herència de prioritats pel kernel
    - spinlocks
    - semàfors
  - Converteix els gestors d'interrupcions en fluxos
    - Se'ls pot canviar la prioritat
  - Afegeix rellotges d'alta resolució
- Debian, Suse, Ubuntu

Real-time Ubuntu kernels

- -lowlatency
  - soft real-time
  - basat en el kernel per defecte, amb una configuració agressiva per reduir latències
- -realtime
  - hard real-time
  - based on PREEMPT-RT

Temps real

- Firm, hard, altres sistemes
  - Linuxworks LynxOS http://www.lynuxworks.com/
  - WxWorks (Wind River) http://www.windriver.com/products/vxworks/
  - QNX http://www.qnx.com/products/neutrino-rtos/index.html
  - HP-RT
  - Xenomai (Linux) http://www.xenomai.org/

Recomanacions

- WxWorks
  - Usar primitives de sincronització de baix nivell
    - operacions atòmiques (sync_test_and_set)
    - spin locks (per 10-20 línies de codi)
    - semàfors / mutex
  - Usar CPU affinity (sched_setaffinity)
  - Minimitzar desbalanceig en la feina dels fluxos
    - evitar que hi hagi recursos sobrecarregats

Best Practices: Adoption of Symmetric Multiprocessing Using VxWorks and Intel® Multicore Processors
http://www.windriver.com/products/platforms/vxworks_multicore/

Problemàtica

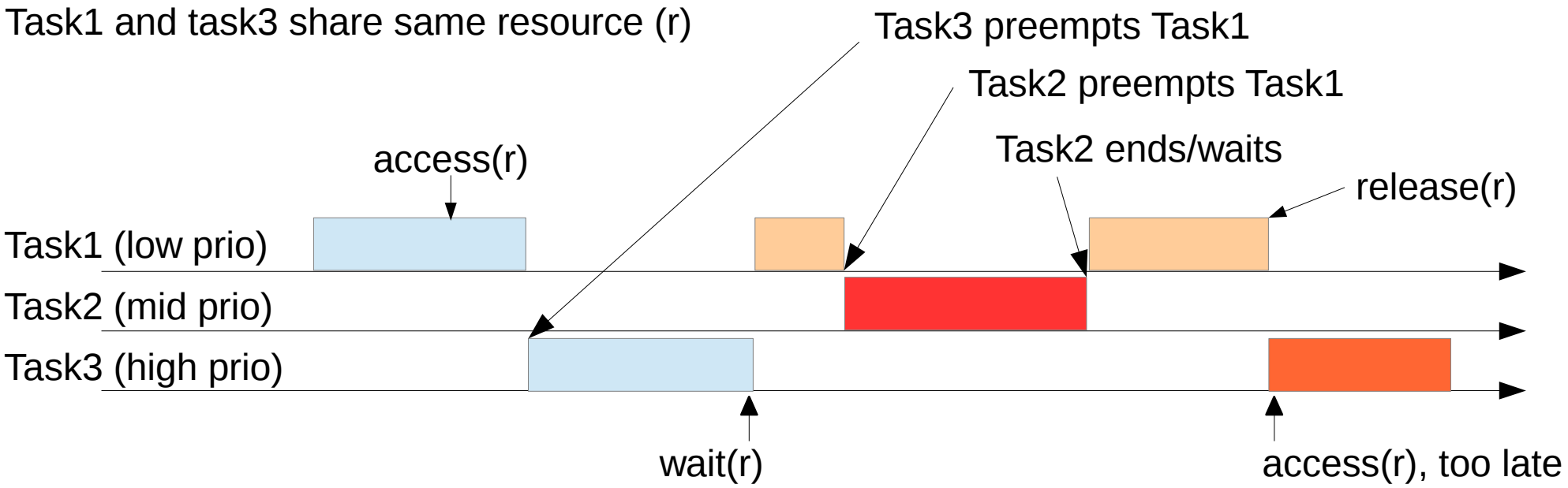
- Inversió de prioritats (priority inversion)
  - Es dóna quan un flux més prioritari espera per entrar en una regió crítica que té un flux menys prioritari
  - O bé espera per usar un recurs – o dispositiu – que està ocupat per un flux menys prioritari
- Solució: herència de prioritat
  - si es dóna el cas, transferir la prioritat del flux més prioritari, al flux menys prioritari per tal que surti de la regió tan aviat com pugui

Problemàtica

- Inversió ilimitada de prioritats
  (unbounded priority inversion)
  - S'afegeix una tasca (o grup de tasques) intermitja que endarrereix encara més l'execució d'una tasca més prioritària
- Solucions:
  - herència de prioritat
  - sostre de prioritat (priority ceiling): cada recurs té una prioritat. Se li assigna un nivell superior (ceiling) al de la tasca més prioritària que l'usa. En usar-lo, totes les tasques s'executen en aquest nivell de prioritat

Exemple

- Del Mars Pathfinder, inversió ilimitada de prioritats



Inici de la missió: 4 de desembre de 1996
Arribada a Mart: 4 de juliol de 1997
Temps previst d'operations: entre 1 setmana i un mes
Temps final d'operacions: 3 mesos
Probable fallada: bateria

http://research.microsoft.com/en-us/um/people/mbj/Mars_Pathfinder/Mars_Pathfinder.html

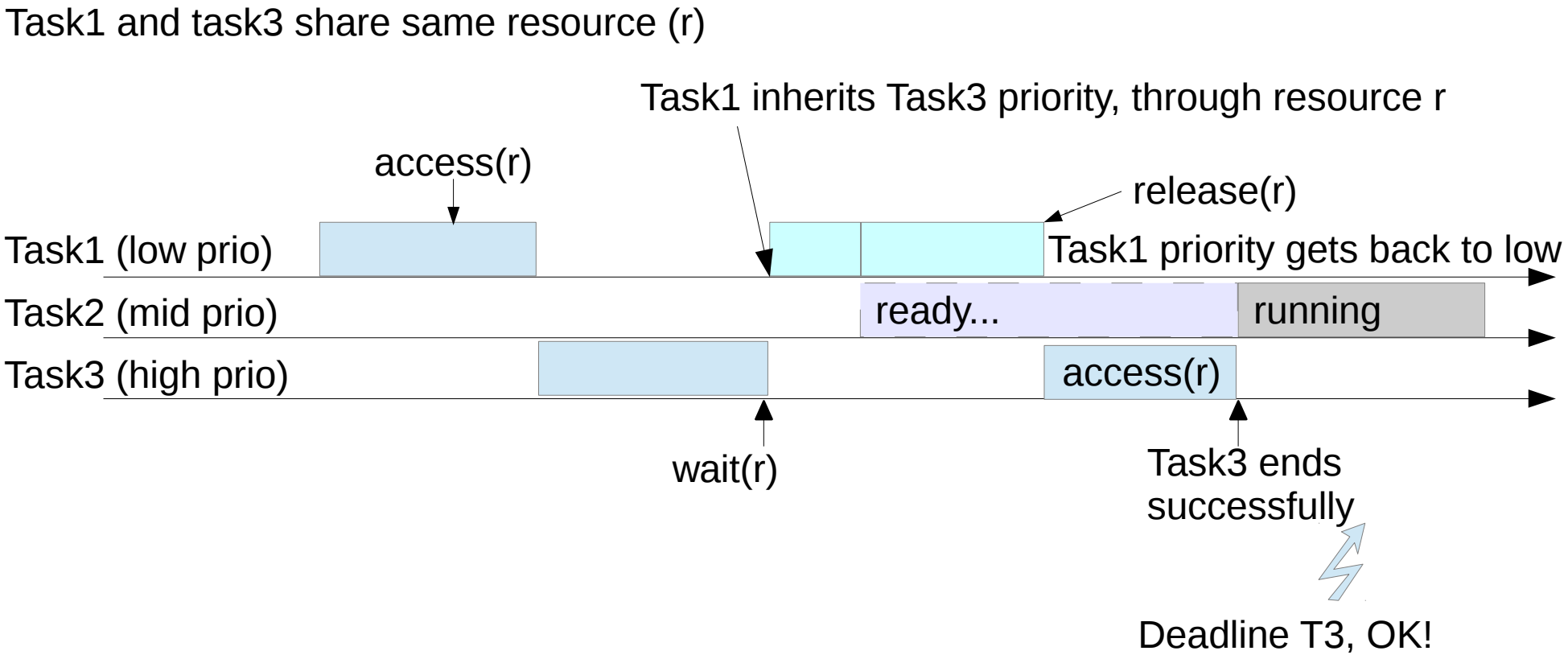
Exemple

- Tasques en el Mars Pathfinder
  - Information bus management task high priority
    - Bloqueja el bus i *posa/treu* dades
  - Communications task med priority
    - Pot trigar una estona...
    - No necessàriament usa el bus
  - Meteorological data collection task low priority
    - Bloqueja el bus i hi publica les dades recollides
- El mutex que protegeix el bus està inicialitzat sense herència de prioritats
- La solució va consistir en reinicialitzar el mutex amb herència de prioritats
  - VxWorks tenia l'opció de debug activada
  - Això va permetre canvia la inicialització del mutex

"The JPL engineers fortuitously decided to launch the spacecraft with this feature still enabled"

Exemple

- Del Mars Pathfinder, amb herència de prioritats



Exemple

- Del Mars Pathfinder, solució alternativa
  - Priority ceiling: assignar prioritats més altes també als recursos

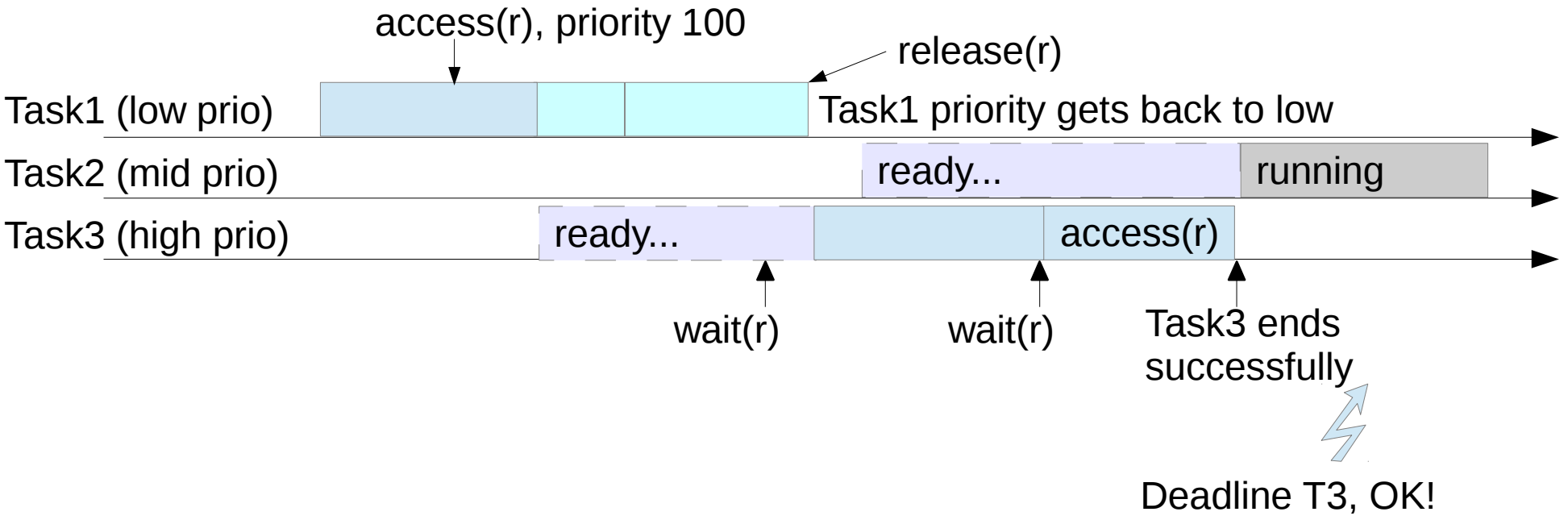
r1	200
r2	150
r3	125
r4	100
MAX_TASK_PRIORITY	99
HIGH_PRIORITY	75
MED_PRIORITY	50
LOW_PRIORITY	25
IDLE_PRIORITY	0

  - Les tasques hereden la prioritat del recurs al que accedeixen en exclusió mútua

Exemple

- Del Mars Pathfinder, amb priority ceiling

Task1 and task3 share same resource (r)



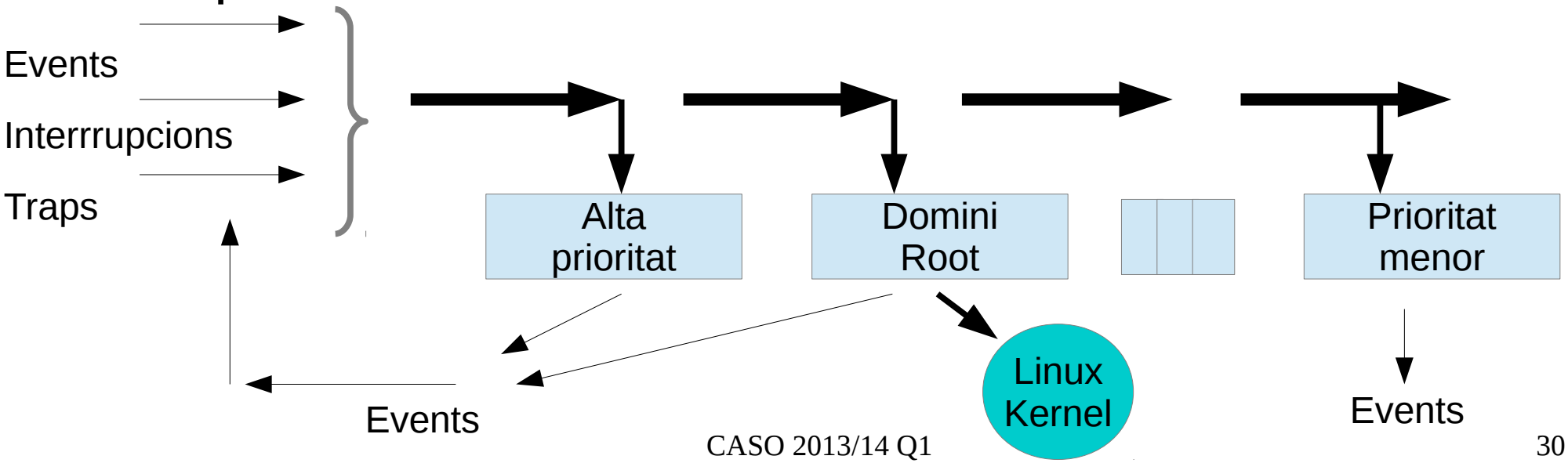
Xenomai

- Suport per temps real en Linux
  - Patch al kernel
  - Utilitats d'usuari
- Afegeix un conjunt de característiques de temps real a la configuració del kernel
- Permet executar serveis en temps real al costat d'aplicacions que no requereixen temps real

http://www.xenomai.org/index.php/Main_Page

Xenomai

- Implementa un ordre en la distribució d'events: Adeos
  - Basat en dominis de protecció
    - Cadascun amb una prioritat estàtica
  - Els events es distribueixen primer al domini més prioritari



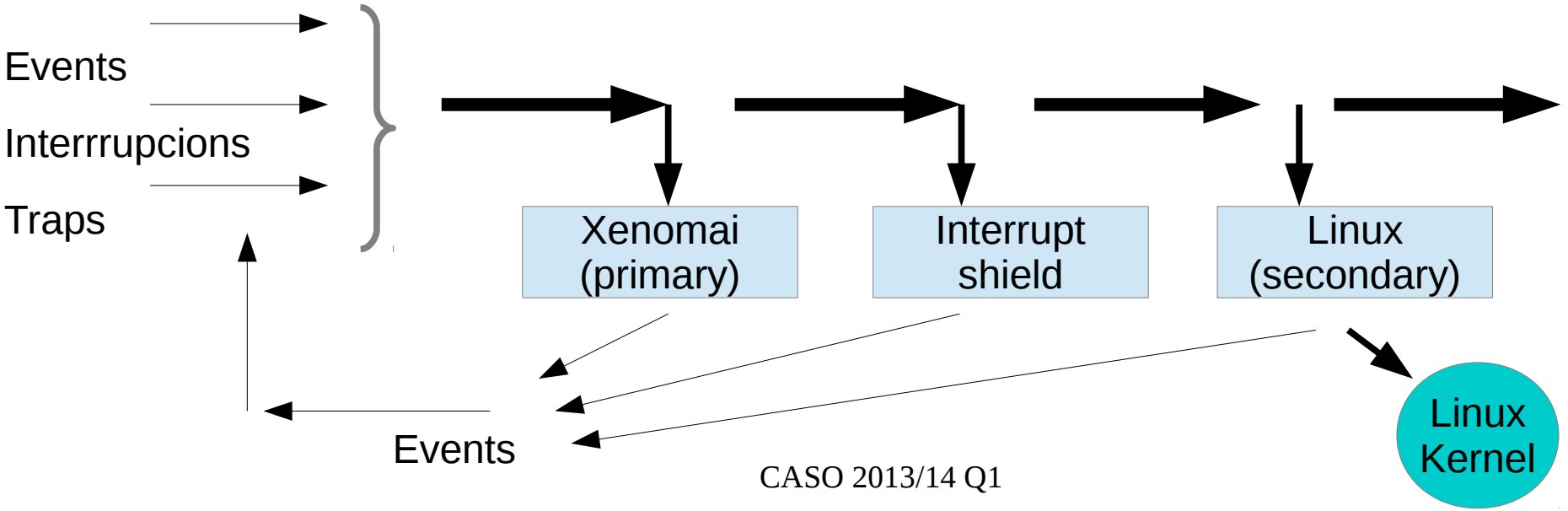
Xenomai

- Disseny dels dominis
  - eficients
  - amb ràfagues curtes d'execució
    - sobretot els més prioritaris
  - poden "inhibir" events i/o interrupcions
    - per exclusió mútua
    - llavors els events no arriben als dominis menys prioritaris fins que han estat processats

http://www.xenomai.org/documentation/branches/v2.3.x/pdf/Life-with-Adeos-rev-B.pdf

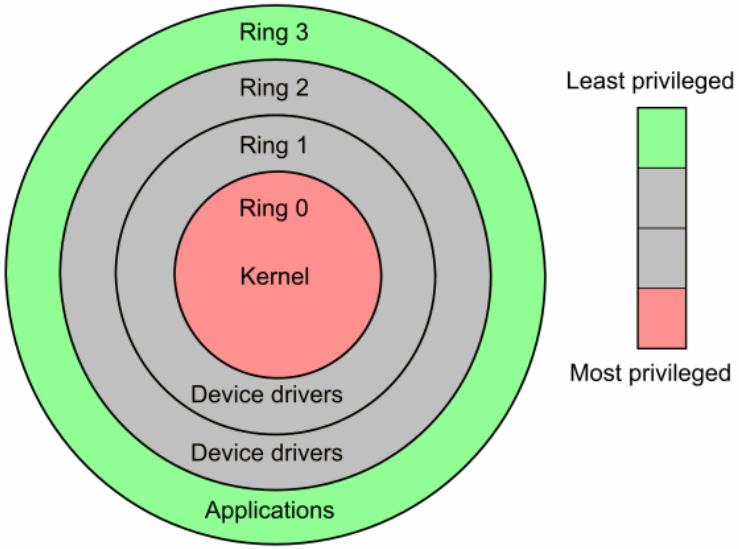
Xenomai

- Xenomai threads
  - Poden córrer en mode kernel o en usuari
    - També en el domini Linux
      - Amb latències més llargues
      - Però sense inversió de prioritats
- Entorn d'execució "standard" en Xenomai



Xenomai

- Implementació
  - aconseguir que Linux cedeixi el control a Adeos
  - per això es fa que el mòdul d'Adeos mogui Linux al *ring 1*
    - Les instruccions privilegiades causen una excepció



http://www.opersys.com/adeos/dox/adeos/node4.html

Xenomai

- Les instruccions són:

clts, clear task-switched flag
hlt, halt processor
cli, clear interrupt flag
sti, set interrupt flag
in, input from port
out, output to port
ins, input string from port
outs, output string from port
invd, invalidate cache
invlpg, invalidate
lgdt, load GDT register
lidt, load IDT register
lldt, load LDT register

lmsw, load machine status register
ltr, load task register
mov to/from CRn, move to control register n
mov to/from DRn, move to debug register n
wbinvd, writeback and invalidate cache
rdmsr, read model-specific registers
wrmsr, write model-specific registers
rdpmc, read performance-monitoring counter
rdtsc, read time-stamp counter

Com pot permetre Adeos que Linux les executi?

Xenomai

- Per cada instrucció, es pot executar de diverses maneres:
  - in, out, ins, outs: són molt usades pels drivers, cal executar-les eficientment
    - Modificar el descriptor de cada task (procés de Linux) per acomodar el vector de bits que permeti executar-les sobre els ports des del ring 1

Xenomai

- Per la resta d'instruccions
  - emulació
    - l'excepció ha de descodificar la instrucció i modificar l'estat del procés perquè sembli que l'ha executat
  - execució pas a pas
    - activa el bit de *single stepping* i deixa executar la instrucció
    - recupera el control després de la instrucció per desactivar el *single stepping*
  - no es fa un sti/cli sinó que Adeos s'apunta que s'ha fet

http://www.xenomai.org/index.php/Xenomai_quick_build_quide

Xenomai: exemple

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <sys/mman.h>
#include <native/task.h>
#include <native/timer.h>

int main(int argc, char* argv[])
{
        signal(SIGTERM, catch_signal);
        signal(SIGINT, catch_signal);
        /* Avoids memory swapping for this program */
        mlockall(MCL_CURRENT|MCL_FUTURE);
        /*
         * Arguments: &task,
         *            name,
         *            stack size (0=default),
         *            priority,
         *            mode (FPU, start suspended, ...)
         */
        rt_task_create(&demo_task, "trivial", 0, 99, 0);
        /*
         * Arguments: &task,
         *            task function,
         *            function argument
         */
        rt_task_start(&demo_task, &demo, NULL);
        pause();
        rt_task_delete(&demo_task);
        return 0;
}
```

```
RT_TASK demo_task;
/* NOTE: error handling omitted. */
void demo(void *arg)
{
        RTIME now, previous;
        /*
         * Arguments: &task (NULL=self),
         *            start time,
         *            period (here: 1 s)
         */
        rt_task_set_periodic(NULL, TM_NOW, 1000000000);
        previous = rt_timer_read();
        while (1) {
                rt_task_wait_period(NULL);
                now = rt_timer_read();
                /*
                 * NOTE: printf may have unexpected impact on the timing of
                 *       your program. It is used here in the critical loop
                 *       only for demonstration purposes.
                 */
                printf("Time since last turn: %ld.%06ld ms\n",
                       (long)(now - previous) / 1000000,
                       (long)(now - previous) % 1000000);
                       previous = now;
        }
}
```

https://www.rtai.org/ – RealTime Application Interface - 4.0 (2013)

Interfície de Xenomai

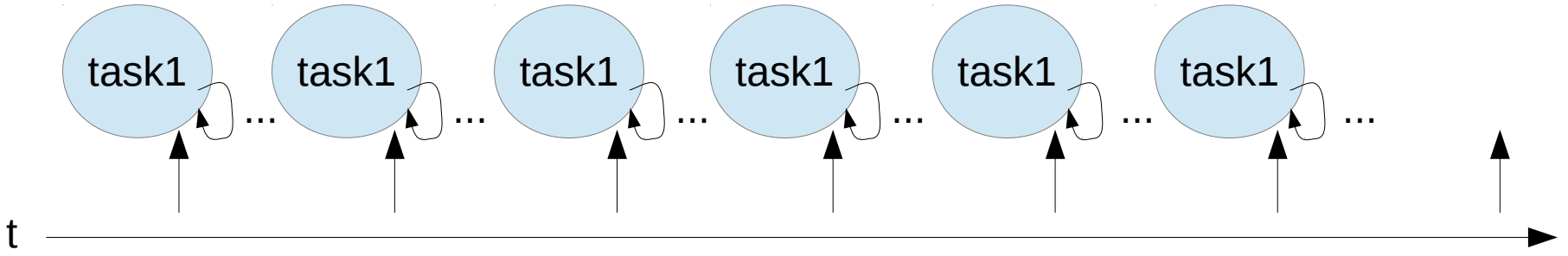
int rt_task_create (RT_TASK *task, const char *name, int stksize, int prio, int mode)
 Create a new real-time task.
int rt_task_start (RT_TASK *task, void(*entry)(void *cookie), void *cookie)
 Start a real-time task.
int rt_task_suspend (RT_TASK *task)
 Suspend a real-time task.
int rt_task_resume (RT_TASK *task)
 Resume a real-time task.
int rt_task_delete (RT_TASK *task)
 Delete a real-time task.
int rt_task_yield (void)
 Manual round-robin.
int rt_task_set_periodic (RT_TASK *task, RTIME idate, RTIME period)
 Make a real-time task periodic.
int rt_task_wait_period (unsigned long *overruns_r)
    Wait for the next periodic release point.
...

Interfície de Xenomai

- rt_task_create()
  - Crea i deixa la tasca en suspens
  - Retorna un handle (RT_TASK)
- rt_task_start()
  - Engega la tasca
- rt_task_spawn()
  - Combina creació i arrancada

Interfície de Xenomai

- rt_task_set_periodic()
  - programa el primer dispar de la tasca i el seu període
  - Un cop s'ha executat, la tasca s'aturarà en un *rt_task_wait_period()*



Interfície de Xenomai: resum

- Gestió de tasques
- Gestió del temps
- Sincronització
- Memòria compartida
- Pas de missatges
- Notificacions asíncrones
  - alarmes i interrupcions
- Gestors de dispositius
- Suport a depuració

RT-Preempt

- Incorpora idees de temps real a Linux
  - Patch al kernel, bastant actualitzat (3.10)
  - Suporta herència de prioritats
    - Soluciona el problema d'inversió de prioritats
- Xenomai pot basar-se en RT-Preempt
  - i proporcionar interfícies diferents
    - VxWorks
    - QNX
    - ...

https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO

https://rt.wiki.kernel.org/index.php/CONFIG_PREEMPT_RT_Patch

RT-Preempt

- Herència de prioritats en el pthread_mutex
  - Basada en un nou atribut
    - PTHREAD_PRIO
      - NONE … mutex normal
      - INHERIT … el flux executant la regió crítica ha de tenir la més alta entre les prioritats (seva, dels altres fluxos esperant a qualsevol dels mutex amb PTHREAD_PRIO_INHERIT que tingui el flux)
      - PROTECT …
  - I la implementació ha d'assegurar que la prioritat es transmet recursivament si el flux que ha heredat la prioritat es bloqueja en un altre mutex

RT-Preempt

- Herència de prioritats
  - PTHREAD_PRIO_PROTECT
    - usa la prioritat indicada a l'atribut *ceiling* del mutex
    - per evitar inversió de prioritats, inicialitzar el ceiling com a més gran o igual a les prioritats dels fluxos que poden agafar aquest mutex
    - i si em bloquejo en un mutex amb *ceiling* inferior, mantinc la prioritat superior!!!

RT-Preempt

- cycletest
  - Linux 3.10.17-rt12-smp PREEMPT RT – Atom N455

$ cyclictest -a -t -n -p 99

policy: fifo: loadavg: 1.98 1.60 0.99 2/329 1299

T: 0 (1144) P:99 I:1000 C: 503964 Min: 4 Act: 26 Avg: 33 **Max: 3354**
T: 1 (1145) P:99 I:1500 C: 335976 Min: 11 Act: 64 Avg: 40 **Max: 2535**

policy: fifo: loadavg: 0.65 0.78 0.83 2/355 1339

T: 0 (1301) P:99 I:1000 C: 816333 Min: 10 Act: 22 Avg: 31 **Max: 3803**
T: 1 (1302) P:99 I:1500 C: 544222 Min: 7 Act: 24 Avg: 33 **Max: 2787**

policy: fifo: loadavg: 0.58 0.91 1.16 1/387 3784

T: 0 (1919) P:99 I:1000 C:2363722 Min: 8 Act: 33 Avg: 30 **Max: 4550**
T: 1 (1920) P:99 I:1500 C:1575814 Min: 7 Act: 35 Avg: 29 **Max: 3860**

RT-Preempt

- cycletest
  - Linux 3.9.10-smp SMP – Atom N455

$ cyclictest -a -t -n -p 99

policy: fifo: loadavg: 0.77 0.58 0.42 2/432 2826

T: 0 (2742) P:99 I:1000 C: 232419 Min: 11 Act: 28 Avg: 32 Max: 10589
T: 1 (2743) P:99 I:1500 C: 154946 Min: 11 Act: 31 Avg: 37 Max: 19370

policy: fifo: loadavg: 1.68 0.94 0.57 1/377 2871

T: 0 (2828) P:99 I:1000 C: 119921 Min: 10 Act: 22 Avg: 40 Max: 15604
T: 1 (2829) P:99 I:1500 C: 79947 Min: 11 Act: 28 Avg: 32 Max: 11860

policy: fifo: loadavg: 1.57 1.24 0.77 2/370 2899

T: 0 (2873) P:99 I:1000 C: 234043 Min: 10 Act: 28 Avg: 36 Max: 16849
T: 1 (2874) P:99 I:1500 C: 156029 Min: 10 Act: 21 Avg: 36 Max: 15801

RT-Preempt

- cycletest
  - Linux 3.10.17-smp SMP – Atom N455

```
$ cyclictest -a -t -n -p 99

policy: fifo: loadavg: 0.08 0.17 0.27 2/308 1235

T: 0 ( 1234) P:99 I:1000 C:  18505 Min:     12 Act:   22 Avg:   49 Max:   16688
T: 1 ( 1235) P:99 I:1500 C:  12335 Min:     12 Act:   40 Avg:   34 Max:    6511

policy: fifo: loadavg: 0.47 0.29 0.30 2/309 1239

T: 0 ( 1238) P:99 I:1000 C: 134952 Min:     11 Act:   26 Avg:   35 Max:   15992
T: 1 ( 1239) P:99 I:1500 C:  89968 Min:     11 Act:   31 Avg:   30 Max:    4339

policy: fifo: loadavg: 0.47 0.38 0.33 4/308 1242

T: 0 ( 1241) P:99 I:1000 C: 258467 Min:     11 Act:   33 Avg:   32 Max:   16334
T: 1 ( 1242) P:99 I:1500 C: 172311 Min:     10 Act:   67 Avg:   38 Max:   16283
```

RT-Preempt

- cycletest
  - Linux 3.2.23 SMP – Core2 Duo P9400

```
$ cyclictest -a -t -n -p 99

T: 0 (30397) P:99 I:1000 C: 145316 Min:     4 Act:   18 Avg:   18 Max:   15646
T: 1 (30398) P:99 I:1500 C:  96877 Min:     4 Act:   13 Avg:   10 Max:    2670
```

RT-Preempt

- signaltest
  - Linux 3.10.17-rt12-smp SMP PREEMPT RT – Atom

$ signaltest

T: 0 (2579) P: 0 C: 851904 Min: 25 Act: 44 Avg: 69 **Max: 3537**

  - Linux 3.10.17-smp SMP – Atom

$ signaltest

T: 0 (1257) P: 0 C: 744656 Min: 19 Act: 56 Avg: 63 Max: 5329

Pthreads RT

- Barriers
  - pthread_barrier_init (&barrier, attr, expected_count)
  - pthread_barrier_wait (&barrier)

```
top - 12:36:23 up  3:50,  9 users,  load average: 0.55, 0.41, 0.42
Tasks: 328 total,   3 running, 325 sleeping,   0 stopped,   0 zombie
Cpu0  :  2.2%us, 15.7%sy,  0.0%ni, 82.1%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  :  1.8%us, 15.2%sy,  0.0%ni, 82.5%id,  0.0%wa,  0.4%hi,  0.0%si,  0.0%st
Mem:   1935004k total,  1769548k used,   165456k free,   209924k buffers
Swap:  5119996k total,     4696k used,  5115300k free,   824528k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  WCHAN     COMMAND
 3792 xavim     20   0 22832  380  304 R   42  0.0   0:07.64 -         barrier
 3793 xavim     20   0 22832  380  304 R   41  0.0   0:07.62 futex_wai barrier
 2112 root      20   0  160m  39m  15m S    1  2.1   2:52.81 poll_sche X
 3777 xavim     20   0 19764 1592 1060 R    1  0.1   0:00.79 -         top
    3 root      20   0     0    0    0 S    0  0.0   0:00.26 run_ksoft ksoftirqd
    9 root      20   0     0    0    0 S    0  0.0   0:00.26 run_ksoft ksoftirqd
 2440 xavim     20   0  351m  21m  16m S    0  1.1   0:00.29 poll_sche korgac
```

Pthreads RT

- Barriers
  - Atributs
    - PTHREAD_PROCESS_PRIVATE
    - PTHREAD_PROCESS_SHARED

Pthreads RT

- Estructura del barrier del barrier (I)

```
union sparc_pthread_barrier
{
  struct pthread_barrier b;
  struct sparc_pthread_barrier_s
    {
      unsigned int curr_event;      // número de barriers
                                    // que hem passat
      int lock;                     // protegeix "left"
      unsigned int left;            // quants en falten per arribar
      unsigned int init_count;      // quants n'han d'arribar
      unsigned char pshared;        // compartit entre processos
                                    // o privat
    } s;
};
```

Inicialització: init_count = nthreads
left = nthreads

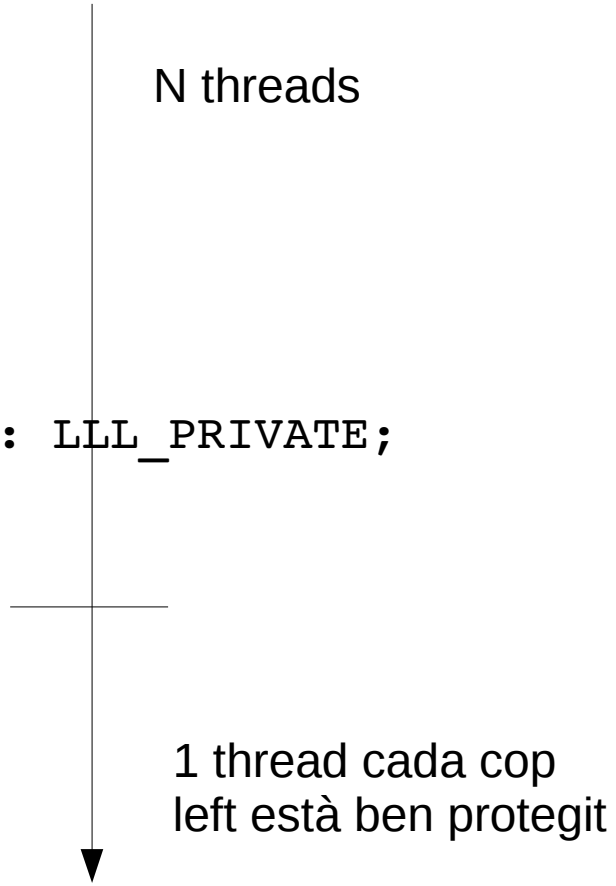
Pthreads RT

- Implementació del barrier (I)

```
/* Wait on barrier.  */
int
pthread_barrier_wait (barrier)
     pthread_barrier_t *barrier;
{
  union sparc_pthread_barrier *ibarrier
    = (union sparc_pthread_barrier *) barrier;
  int result = 0;
  int private = ibarrier->s.pshared ? LLL_SHARED : LLL_PRIVATE;

  /* Make sure we are alone.  */
  lll_lock (ibarrier->b.lock, private);

  /* One more arrival.  */
  --ibarrier->b.left;
```



Pthreads RT

- Implementació del barrier (II)

```
/* Are these all?  */
if (ibarrier->b.left == 0)
  {
    /* Yes. Increment the event counter to avoid invalid wake-ups and
       tell the current waiters that it is their turn.  */
    ++ibarrier->b.curr_event;

    /* Wake up everybody.  */
    lll_futex_wake (&ibarrier->b.curr_event, INT_MAX, private);

    /* This is the thread which finished the serialization.  */
    result = PTHREAD_BARRIER_SERIAL_THREAD;
  }
```

L'últim thread en arribar entra aquí...

... i desperta la resta

... aquest tornarà `PTHREAD_BARRIER_SERIAL_THREAD`

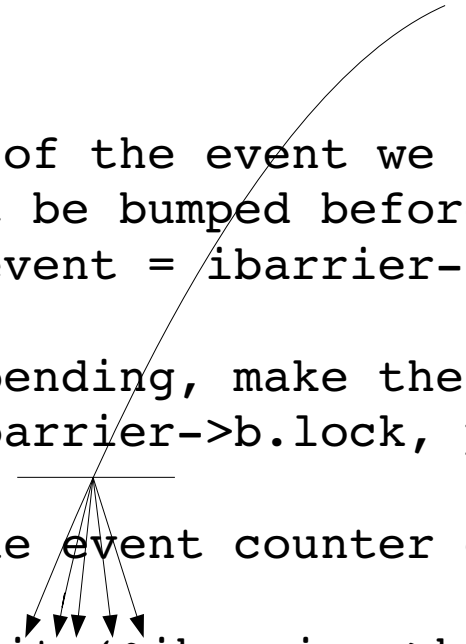
Pthreads RT

- Implementació del barrier (III)

```
else
  {
    /* The number of the event we are waiting for.  The barrier's event
       number must be bumped before we continue.  */
    unsigned int event = ibarrier->b.curr_event;

    /* Before suspending, make the barrier available to others.  */
    lll_unlock (ibarrier->b.lock, private);

    /* Wait for the event counter of the barrier to change.  */
    do
      lll_futex_wait (&ibarrier->b.curr_event, event, private);
    while (event == ibarrier->b.curr_event);
  }
```



Pthreads RT

- Implementació del barrier (IV)

```
  /* Make sure the init_count is stored locally or in a register.  */
  unsigned int init_count = ibarrier->b.init_count;

  /* If this was the last woken thread, unlock.  */
  if (atomic_increment_val (&ibarrier->b.left) == init_count)
    /* We are done.  */
    lll_unlock (ibarrier->b.lock, private);

  return result;
}
```

L'últim obre el lock per poder tornar-hi

Left està protegit contra decrements pel lock i contra increments per l'operació atòmica

Pthreads RT

- Low level lock
  - Implementat amb *lock cmpxchg*
  - Pot comprovar si no hi ha més threads en el procés
    - i llavors estalvar-se el *lock*

```
#define lll_trylock(futex) \
  ({ int ret;                                                          \
     __asm __volatile (cmpl $0, __libc_multiple_threads(%%rip)\n\t"    \
                       "je 0f\n\t"                                     \
                       "lock; cmpxchgl %2, %1\n\t"                     \
                       "jmp 1f\n\t"                                    \
                       "0:\tcmpxchgl %2, %1\n\t"                       \
                       "1:"                                            \
                     : "=a" (ret), "=m" (futex)                        \
                     : "r" (LLL_LOCK_INITIALIZER_LOCKED), "m" (futex), \
                       "0" (LLL_LOCK_INITIALIZER)                      \
                     : "memory");                                      \
     ret; })
```

%1

%2

Pthreads RT

- pthread_spin
  - init
  - lock
  - trylock
  - unlock
- pthread_getcpuclockid (pthread_t, *clockid)

Altres productes

- Microsoft Windows Embedded Compact 2013
  - x86 i ARM
- IntervalZero RTOS platform
  - Windows, www.intervalzero.com
- OnTime RTOS-32 platform
  - Win32 apps
- RTXC-Quadros
  - ARM, PPC..., www.quadros.com
- QNX
  - Intel, ARM... www.qnx.com