Programmation Temps Réel

Interactions matériel/logiciel

Yann Thoma

Reconfigurable and Embedded Digital Systems Institute
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Septembre 2017

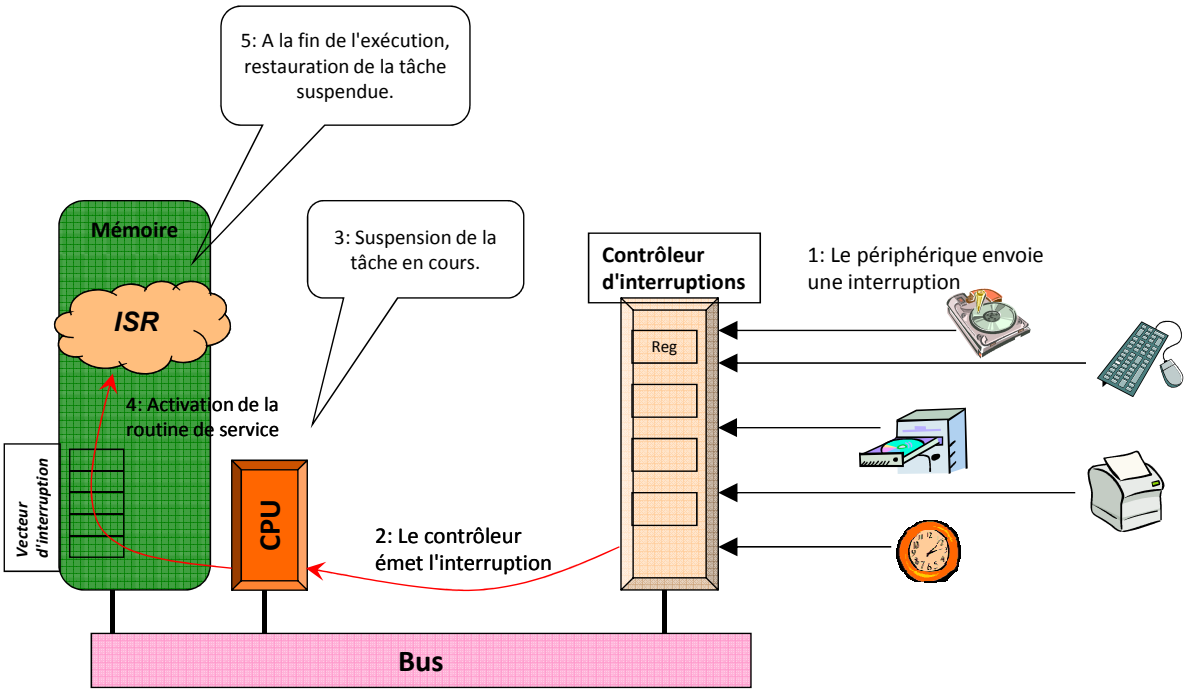
Plan

- Mécanismes d'interruptions
  - IRQ, ISR
- Temps de latence et gigue
- Priorités, masquage, tâches différées
- Gestion du temps
  - Service de timer, watchdog

Interruptions

- Les interruptions sont des mécanismes essentiels dans un système temps-réel.
  - Elles permettent de prendre en compte des événements externes au processeur de manière asynchrone.
  - Requête d'interruption
- A l'arrivée d'une interruption, une routine de traitement d'interruption (traitant ou handler) est exécutée.
  - **ISR** - *Interrupt Service Routine*
  - L'arrivée d'une interruption au processeur provoque l'arrêt de l'exécution en cours.
- Deux types d'interruptions
  - Interruption **matérielle**
  - Interruption **logicielle**

Interruptions - Mécanisme



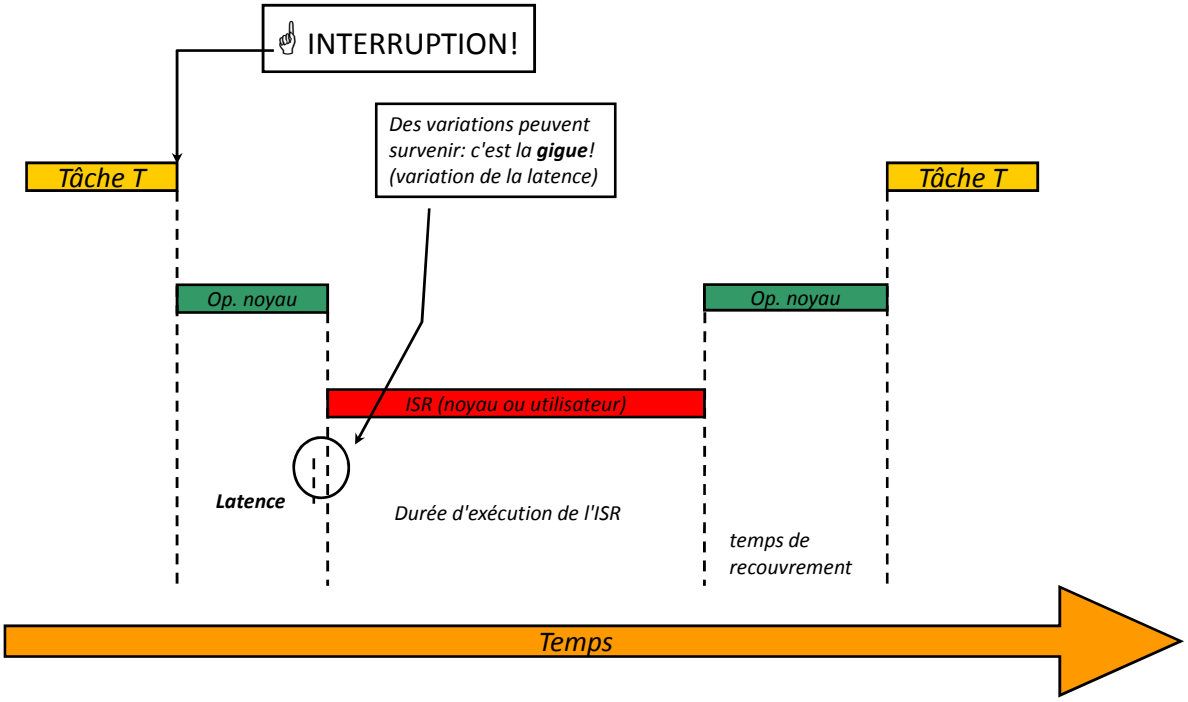
IRQs sur x86

INT (hex)	IRQ	Common Uses
00-01	Exception handlers	-
02	Non-maskable IRQ	Non-maskable IRQ (Parity Errors)
03-07	Exception handlers	-
08	Hardware IRQ0	System Timer
09	Hardware IRQ1	Keyboard
0A	Hardware IRQ2	Redirected
0B	Hardware IRQ3	Serial Comms. COM2/COM4
0C	Hardware IRQ4	Serial Comms. COM1/COM3
0D	Hardware IRQ5	Reserved/Sound card
0E	Hardware IRQ6	Floppy disk controller
0F	Hardware IRQ7	Parallel Comms.
10-6F	Softwre interrupts	-
70	Hardware IRQ8	Real time clock
71	Hardware IRQ9	Redirected IRQ2
72	Hardware IRQ10	Reserved
73	Hardware IRQ11	Reserved
74	Hardware IRQ12	PS/2 Mouse
75	Hardware IRQ13	Math's co-processor
76	Hardware IRQ14	Hard disk drive
77	Hardware IRQ15	Reserved
78-FF	Software interrupts	-

ISR - Particularités

- Une ISR n'a pas le statut de tâche
  - L'ISR s'exécute dans le contexte de l'interruption (contexte noyau); la pile utilisée est donc la pile du noyau.
- Restrictions sur les ISRs
  - Une ISR ne doit pas appeler de routines bloquantes
  - Une ISR doit absolument être courte.
    - Une ISR est activée après la préemption de la tâche en cours d'exécution.
    - Une ISR peut être préemptée par une autre interruption.
- Si un traitement de durée importante doit être effectué, on fait appel à une tâche spéciale (le traitement est alors différé).

Interruptions - Latence et gigue



Gestion des périphériques

- Il existe essentiellement deux modes de gestion des périphériques:
  - Avec interruption matérielle
    - Peu de latence dans le traitement des entrées-sorties
    - Peu de maîtrise des instants de demandes d'interruptions
    - Difficulté à intégrer dans les méthodes d'analyse d'ordonnançabilité
  - Scrutation (polling): examen périodique des registres d'état des périphériques
    - Davantage de latence dans le traitement des entrées-sorties
    - Plus facile à intégrer dans les méthodes d'analyse d'ordonnançabilité
    - Gaspillage du temps processeur.
    - Une tâche périodique peut effectuer les scrutations.

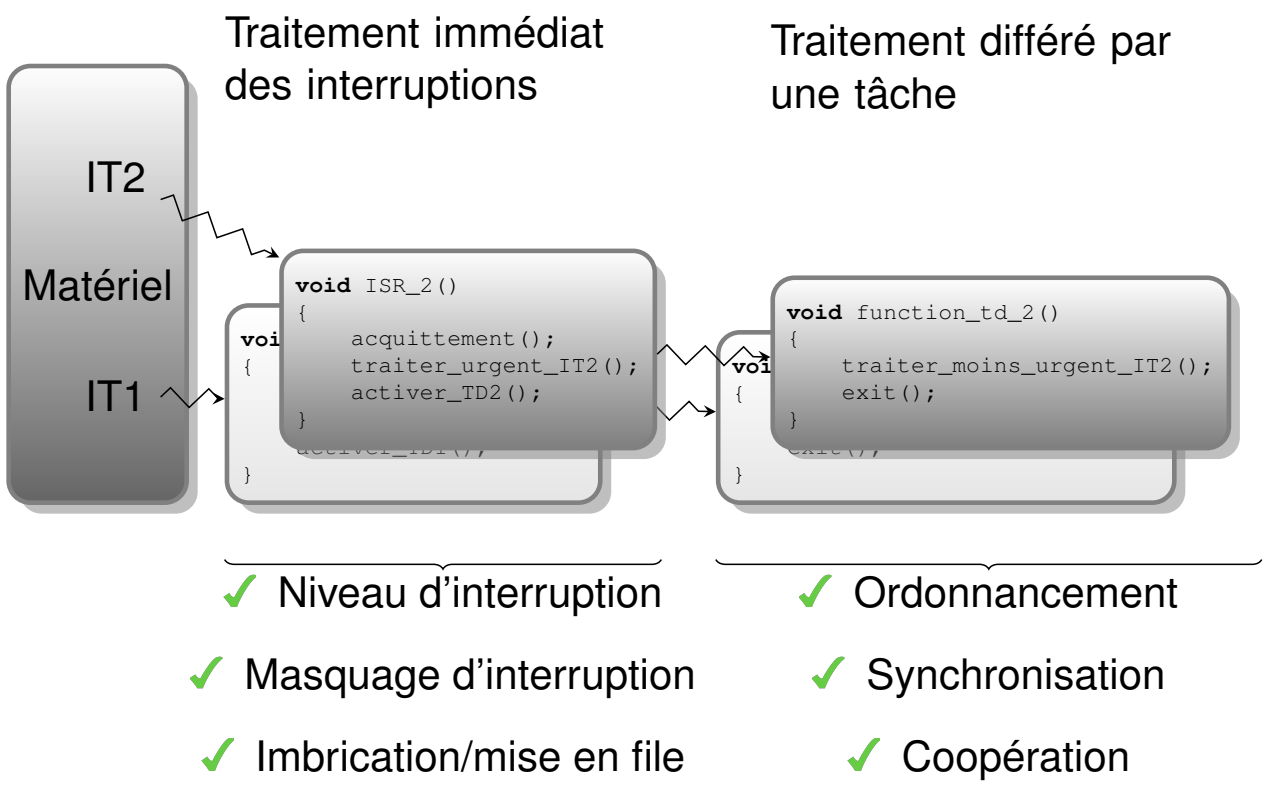
Masquage d'interruptions

- Connexion d'une ISR à un niveau d'interruption
  - Plusieurs ISRs peuvent être déclenchées à la suite (interruptions multi-niveau).
    - Une interruption survient durant le traitement d'une ISR, et cette interruption n'est pas masquée.
- Masquage / démasquage des interruptions
  - Utilisable pour mettre en oeuvre des sections critiques de code pour une courte durée

```
{
    traitement_standard1();
    Masquage();
    traitement_protege(); // Doit être court
    Demasquage();
    traitement_standard2();
}
```

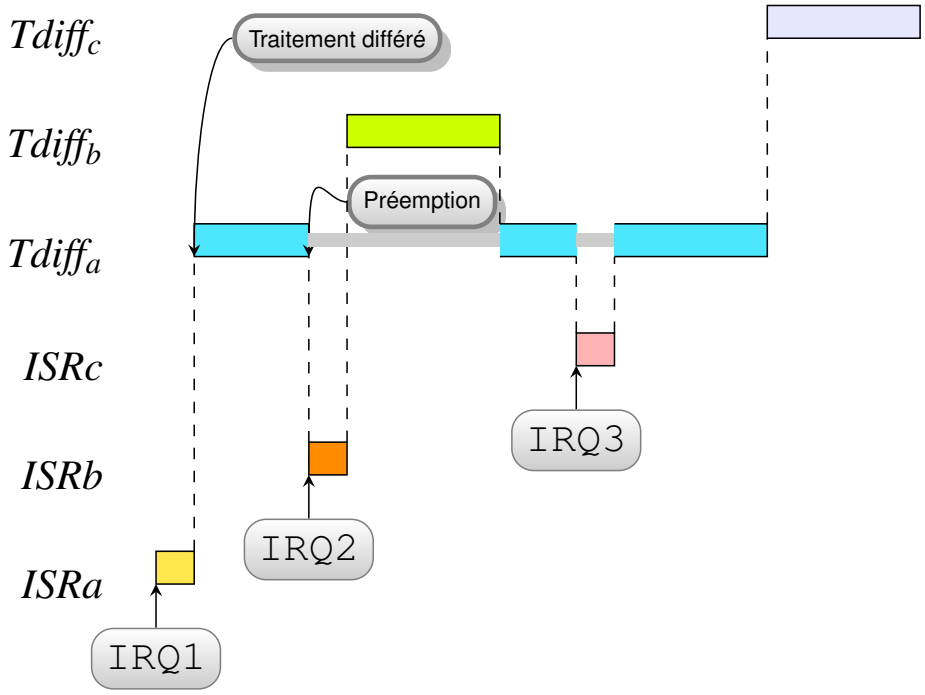
  - Mémorisation de l'état précédent dans le cas multi-niveau (gérée à l'aide de la pile système ou celle de l'application).

Tâches et interruptions (1/5) - Interruptions



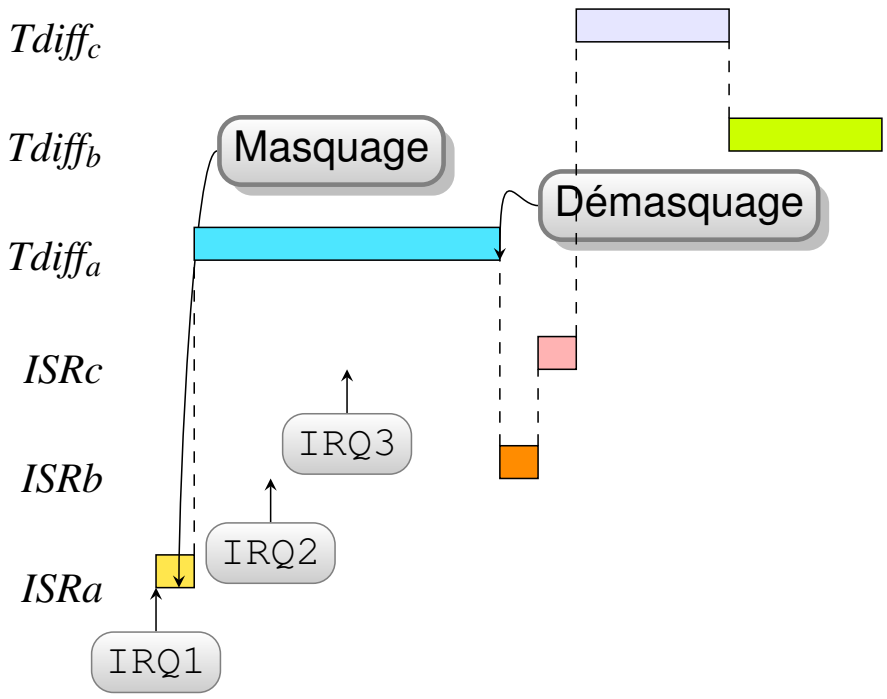
Tâches et interruptions (2/5) - Priorités

Priorité de $Tdiff_b$ > Priorité de $Tdiff_a$ > Priorité de $Tdiff_c$



Tâches et interruptions (3/5) - Masquage

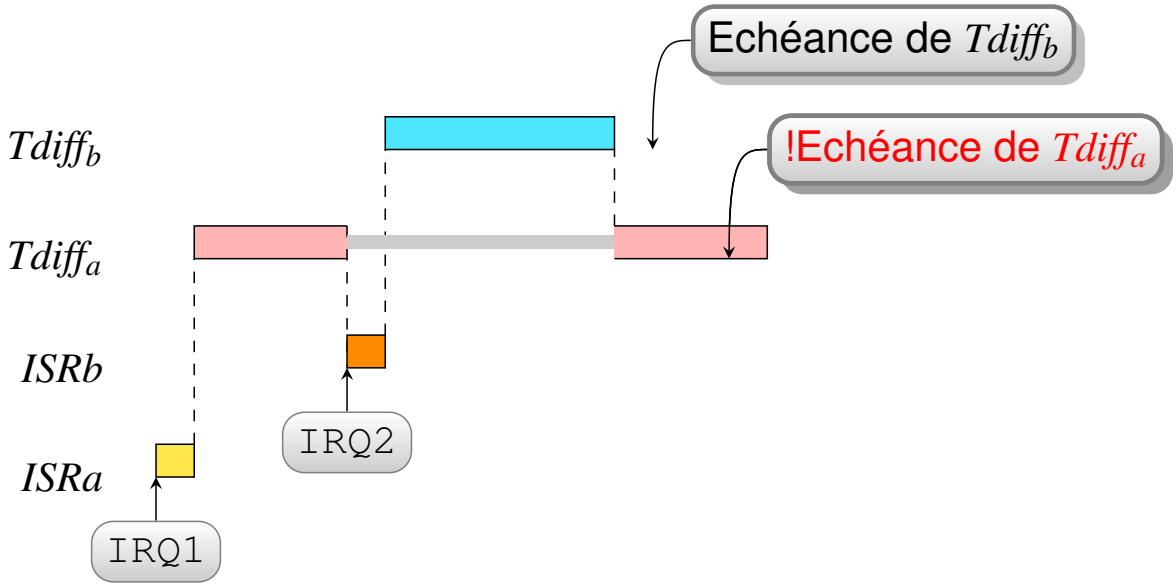
Priorité de $Tdiff_a$ > Priorité de $Tdiff_c$ > Priorité de $Tdiff_b$
Priorité de $IRQ1$ > Priorité de $IRQ2$ > Priorité de $IRQ3$



Tâches et interruptions (4/5) - Délais

- Les durées de traitement impactent sur les échéances des tâches

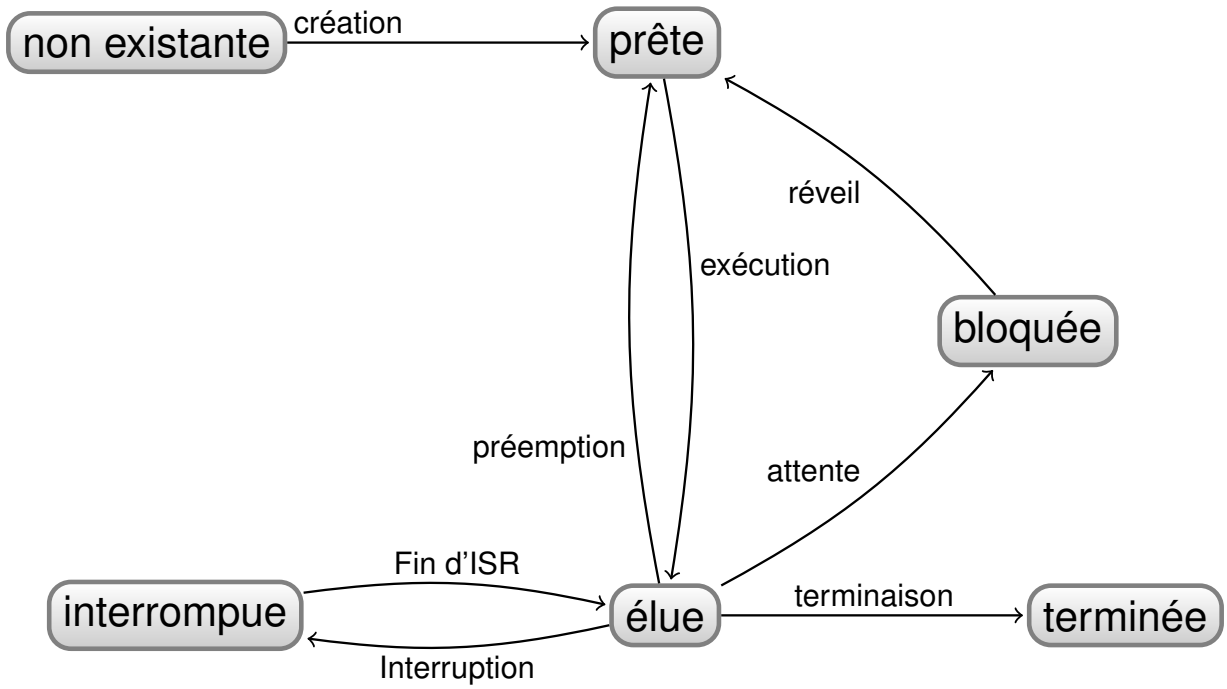
Priorité de $Tdiff_b$ > Priorité de $Tdiff_a$



Tâches et interruptions (5/5) - Remarques

- La programmation à l'aide d'interruptions asynchrones (matérielles) rend plus aisé la programmation de tâche complexes. Cependant la prédictibilité d'un système temps-réel devient plus difficile à garantir.
  - Cohabitation d'activités périodiques / apériodiques / sporadiques
- La gestion des tâches différées et la prise en compte des interruptions nécessite un noyau multitâche temps-réel.
  - Prise en compte de l'asynchronisme
  - La programmation de tâche temps-réel va dépendre de l'interface (API) avec le noyau temps-réel.
  - Un noyau temps-réel offre des objets prédéfinis tels que: tâches, objets de communication inter-tâche, objets de synchronisation inter-tâche, gestionnaire de temps, etc.
- La programmation de tâches temps-réels nécessite d'établir un schéma de priorité des tâches.
  - Notion d'ordonnançabilité d'un système multitâche.

Tâches et interruptions: diagramme de transitions



Interruptions - API Xenomai

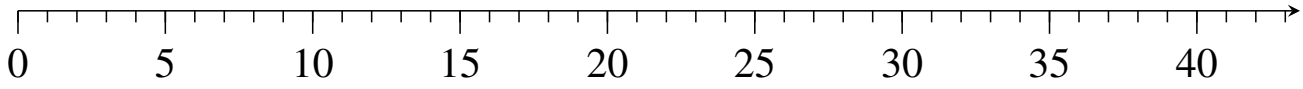
Fonction	Description	Kern.	User
`int rt_intr_create(RT_INTR *intr,const char *name, unsigned irq,rt_isr_t isr, rt_iack_t iack,int mode)`	Création d'un descripteur d'interruption	✓	
`int rt_intr_create(RT_INTR *intr,const char *name, unsigned irq,int mode)`	Création d'un descripteur d'interruption		✓
`int rt_intr_wait(RT_INTR *intr,RTIME timeout)`	Suspend la tâche en cours d'exécution jusqu'à la prochaine interruption		✓
`int rt_intr_enable(RT_INTR *intr)`	Démasquage de l'interruption	✓	✓
`int rt_intr_disable(RT_INTR *intr)`	Masquage de l'interruption	✓	✓
`int rt_intr_delete(RT_INTR *intr)`	Destruction d'un descripteur d'interruption	✓	✓

```
typedef int (*rt_isr_t)(struct xnintr *intr);
```

Tâches et interruptions - Exercice

Tâche	Arrivée	Priorité	Durée (ms)
T_a	2	1	22
T_b	12	3	5
T_c	(différée)	2	4
$IRQ_1 \rightarrow ISR\ I$	10	2	5
$IRQ_2 \rightarrow ISR\ J$	13	1	4

- La tâche T_c est le traitement différé de l'ISR I.
- Les interruptions sont démasquées durant l'exécution des ISRs.
- Etablir le chronogramme des événements de ce système temps-réel.



ISR - Appels système et Exceptions

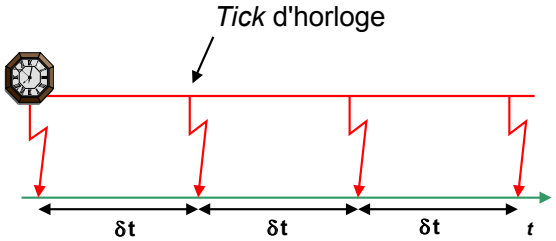
- Tous les appels système sont réalisés via une interruption logicielle
- Mais pas tous les appels à la bibliothèque standard C sont des appels système
- Les exceptions sont très similaires:
  - Utilisées pour signaler un problème
  - Exemples: Page fault, Floating Point Exceptions, etc...

Gestion du temps

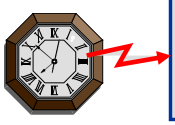
- Un exécutif temps-réel doit pouvoir offrir des services afin de manipuler le temps.
  - Programmation de timers
  - Granularités différentes
    - s, ms, μs, ns
- En programmation temps-réel, il est nécessaire de considérer les délais potentiels dus aux retards et aux gigues.
  - Mesures empiriques
  - Spécification du matériel
  - Propriétés physiques

Gestion du temps - Mécanismes

- Mécanisme de gestion du temps
  - Un dispositif (timer) signale l'écoulement d'un intervalle de temps.
  - Interruption associée
  - IRQ 0: Horloge système
  - IRQ 8: Horloge temps-réel
  - Résolution d'horloge : δt



- A chaque tick, une interruption est générée, donc une ISR spécifique est activée.



ISR de gestion du timer

```
void IRQ0_timer ( )
{ incrémenter_heure_courante ( ) ;
   . . .              /* divers */ ;
}
```

Gestion du temps - Service de timer

- Bien entendu, le noyau temps-réel utilise le timer pour se réguler. Il reçoit un tick d'horloge à une certaine fréquence. Dans le cadre de la régulation du noyau temps-réel, l'intervalle entre deux ticks s'appelle un jiffy.
- Le service de gestion de timer permettant de créer et de gérer des événements basés sur la notion de temps.
- Plusieurs réalisations existent:
  - Le service de timer se base sur le timer système: la gestion des timers dépend de la fréquence à laquelle le noyau est régulée. Cette considération est très importante, car elle déterminera la précision que l'on pourra attendre des timers.
  - Une autre possibilité est d'utiliser une horloge temps-réel (RTC). Dans ce cas, il s'agit d'une autre source d'interruption, et le timer peut être indépendant.
  - Finalement, on peut avoir recours à d'autres timers matériels embarqués dans le processeur/microcontrôleur.

One-Shot ou périodique?

- Timer One-Shot
  - Avantages
    - Meilleure précision (résolution pouvant aller jusqu'à la ns)
  - Désavantages
    - Besoin de "réarmer" le timer à chaque itération.
    - Nécessite quelques instructions supplémentaires.
    - Introduit par conséquence des délais au niveau du noyau.
- Timer périodique
  - Avantages
    - Pas besoin de réarmer le timer.
  - Désavantages
    - La résolution du timer est fixe et peut engendrer des problèmes de précision lorsque des délais sont nécessaires.

Time - API Linux

```
#include <time.h>

time_t time(time_t *t); // returns  the time since the Epoch

#include <sys/time.h>
struct timeval {
       time_t      tv_sec;     /* seconds */
       suseconds_t tv_usec;    /* microseconds */
   };

int gettimeofday(struct timeval *tv, struct timezone *tz);
```


Time - Posix

```
int clock_gettime (clockid_t which_clock,
                           struct timespec *tp);
int clock_getres (clockid_t which_clock,
                           struct timespec *tp);
CLOCK_REALTIME
   Systemwide realtime clock.

CLOCK_MONOTONIC
   Represents monotonic time. Cannot be set.

CLOCK_PROCESS_CPUTIME_ID
   High resolution per-process timer.

CLOCK_THREAD_CPUTIME_ID
  Thread-specific timer.

CLOCK_REALTIME_HR
  High resolution version of CLOCK_REALTIME.

CLOCK_MONOTONIC_HR
  High resolution version of CLOCK_MONOTONIC.
```

Timer - API Linux

```
#include <sys/time.h>

int getitimer(int which, struct itimerval *value);
int setitimer(int which, const struct itimerval *value,
              struct itimerval *ovalue);

ITIMER_REAL    decrements in real time, and delivers SIGALRM
               upon expiration.

ITIMER_VIRTUAL decrements only when the process is executing,
               and delivers SIGVTALRM upon expiration.

ITIMER_PROF    decrements both when the process executes and
               when the system is executing on behalf of the
               process. Coupled with ITIMER_VIRTUAL, this timer
               is usually used to profile the time spent by the
               application in user and kernel space. SIGPROF is
               delivered upon expiration.
```


Timer - API Posix

```
int timer_create (clockid_t which_clock,
                  struct sigevent *timer_event_spec,
                  timer_t *created_timer_id);
int timer_settime (timer_t timer_id, int flags,
                   const struct itimerspec *new_setting,
                   struct itimerspec *old_setting);
```

Timer - API Xenomai

Fonction	Description	Kern.	User
`int rt_timer_set_mode(RTIME nstick)`	Démarre le timer système. L'argument correspond à la période en ns pour un timer périodique, soit l'argument vaut TM_ONE_SHOT pour un timer apériodique.	✓	✓
`void rt_timer_spin(RTIME ns)`	Effectue de l'attente active durant un certain nombre de ns.	✓	✓
`SRTIME rt_timer_ns2ticks(SRTIME ns)`	Convertit une durée exprimée en ns en nombre de ticks système. Il y a d'autres fonctions de conversion.	✓	✓
`RTIME rt_timer_read(void)`	Retourne la valeur courante du timer temps-réel, en clock ticks (ou ns).	✓	✓

- **`RTIME`** est de type **`unsigned long long`**
- **`SRTIME`** est de type **`long long`**

Alarmes

- Une alarme est une association entre
  - Un compteur
  - Une action (handler) déclenchée à la fin du compteur.
    - Le fonctionnement est similaire à une ISR; c'est le noyau qui invoque le traitant à la réception de l'interruption du timer (IRQ 0).
- Type de déclenchement
  - Cyclique (récurrent) ou unique
  - Valeur relative ou absolue du compteur (pour le premier déclenchement seulement dans le cas d'alarmes cycliques)

Alarmes - API Linux

```
#include <unistd.h>

unsigned alarm(unsigned seconds);

If there is a previous alarm() request with time remaining, alarm()
shall return a non-zero value that is the number of seconds until
the previous request would have generated a SIGALRM signal.
Otherwise, alarm() shall return 0.
```


Alarmes - API Xenomai

Fonction	Description	Kern.	User
`int rt_alarm_create(RT_ALARM *alarm,const char *name, rt_alarm_t handler,void *cookie)`	Création d'un descripteur d'alarme avec association au *handler*	✓	
`int rt_alarm_create(RT_ALARM *alarm,const char *name)`	Création d'un descripteur d'alarme dans le user space		✓
`int rt_alarm_start(RT_ALARM *alarm,RTIME value, RTIME interval)`	Démarre le compte à rebours pour une certaine alarme	✓	✓
`int rt_alarm_wait(RT_ALARM *alarm)`	Attend la prochaine occurrence d'une certaine alarme		✓
`int rt_alarm_stop(RT_ALARM *alarm)`	Arrête le compte à rebours d'une certaine alarme	✓	✓
`int rt_alarm_delete(RT_ALARM *alarm)`	Destruction d'un descripteur d'alarme	✓	✓