Systèmes embarqués

Introduction à la programmation micro-contrôleur

avec un OS temps réel

Julien DeAntoni

Merci à Jean-Philippe Babau pour m'avoir permi la réutilisation d'une partie de ses supports

Université Nice SOPHIA ANTIPOLIS
Polytech'Nice-Sophia
INRIA SOPHIA ANTIPOLIS
i3S

Contenu du cours

- Généralités
  - Les sytèmes considérés
  - Le développement de tels systèmes
- Programmation sans OS
- *Programmation avec un OS temps réel*

Contenu du cours

- Qu'est-ce qu'un OS temps réel (RTOS)
  - RTOS vs GPOS
  - Différents RTOS...
  - Les objets *communs* d'un RTOS
  - Implémentation d'un RTOS
- Programmation avec un OS temps réel
  - Comment utiliser les objets d'un RTOS ?
- Mise en Oeuvre
  - AvrX

Contenu du cours

- **Qu'est-ce qu'un OS temps réel (RTOS)**
  - **RTOS vs GPOS**
  - Différents RTOS...
  - Les objets *communs* d'un RTOS
  - Les objets *haut niveau* d'un RTOS
  - Implémentation d'un RTOS
- Programmation avec un OS temps réel
  - Comment utiliser les objets d'un RTOS ?
- Mise en Oeuvre

RTOS vs GPOS

- **L'essence de la discorde : Prédictabilité[1]**
  - RTOS
    - Multi-tâche / thread depuis toujours
    - Ses actions sont détermistes...
    - **...et interruptibles**
  - GPOS
    - Multi-tâche seulement récemment
    - Prends la main sur les tâches en cours...
    - Pour une durée indéterminée et non interruptible

1 : Ou Déterminisme...

Contenu du cours

- **Qu'est-ce qu'un OS temps réel (RTOS)**
  - RTOS vs GPOS
  - **Différents RTOS...**
  - Les objets *communs* d'un RTOS
  - Les objets *haut niveau* d'un RTOS
  - Implémentation d'un RTOS
- Programmation avec un OS temps réel
  - Comment utiliser les objets d'un RTOS ?
- Mise en Oeuvre

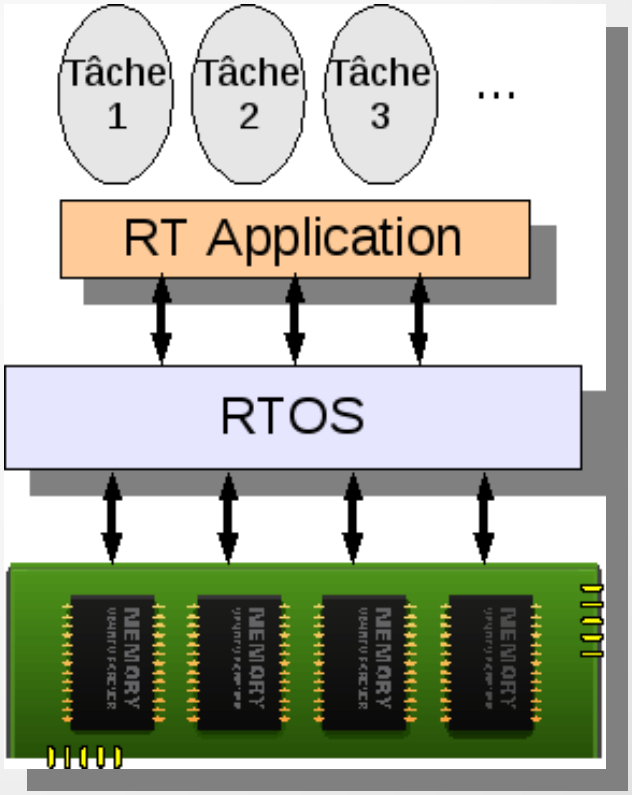
Différents RTOS

Le classique (1)

- Complétement développé au dessus du matériel, il fournit les objets standards d'un RTOS

- Utilisation de l'API du RTOS par l'application

- Tout est intercepté par le RTOS (même les interruptions)



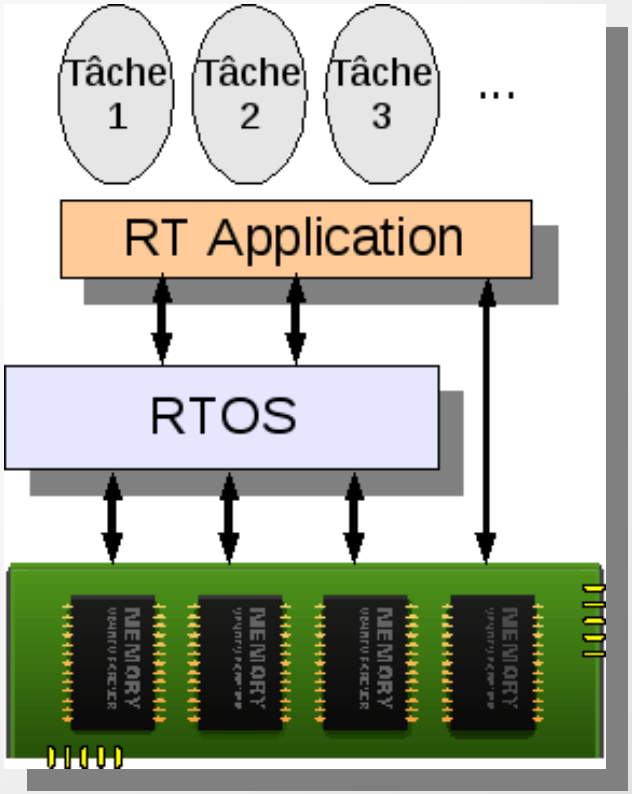
Différents RTOS

Le classique (2)

- Complétement développé au dessus du matériel, il fournit les objets standards d'un RTOS

- Utilisation de l'API du RTOS par l'application

- Laisse à l'application certains accès au matériel



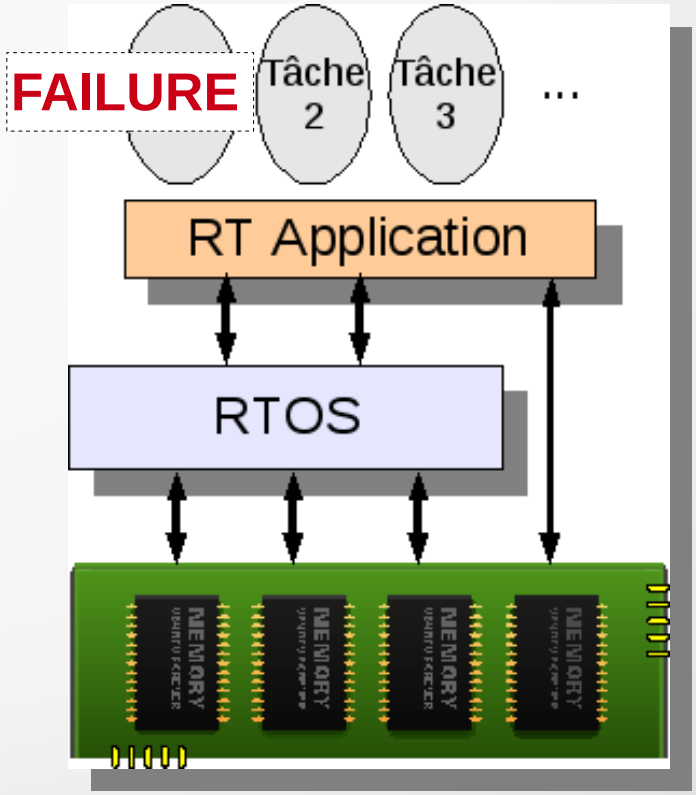
Différents RTOS

Le classique (2)

- Complétement développé au dessus du matériel, il fournit les objets standards d'un RTOS

- Utilisation de l'API du RTOS par l'application

- Laisse à l'application certains accès au matériel



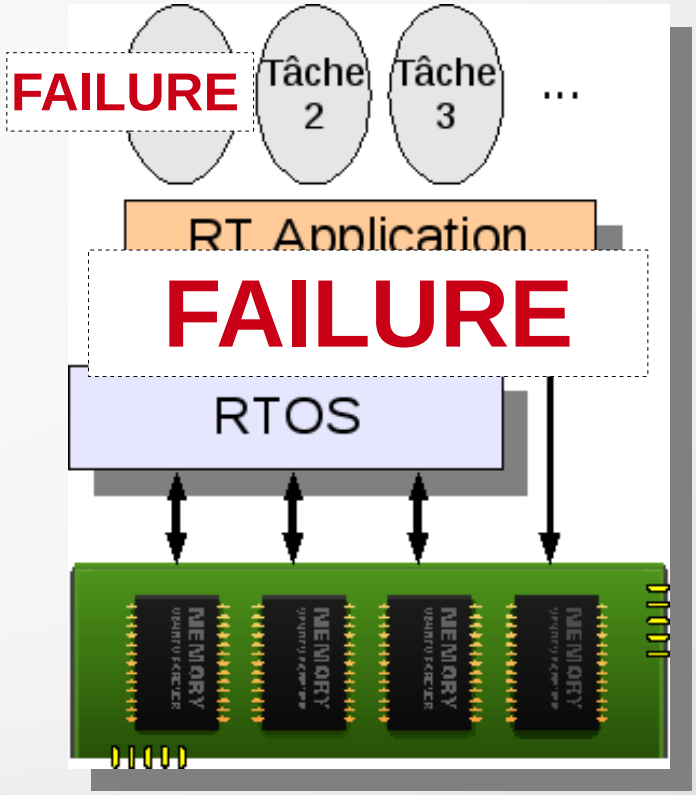
Différents RTOS

Le classique (2)

- Complétement développé au dessus du matériel, il fournit les objets standards d'un RTOS

Utilisation de l'API du RTOS par l'application

Laisse à l'application certains accès au matériel



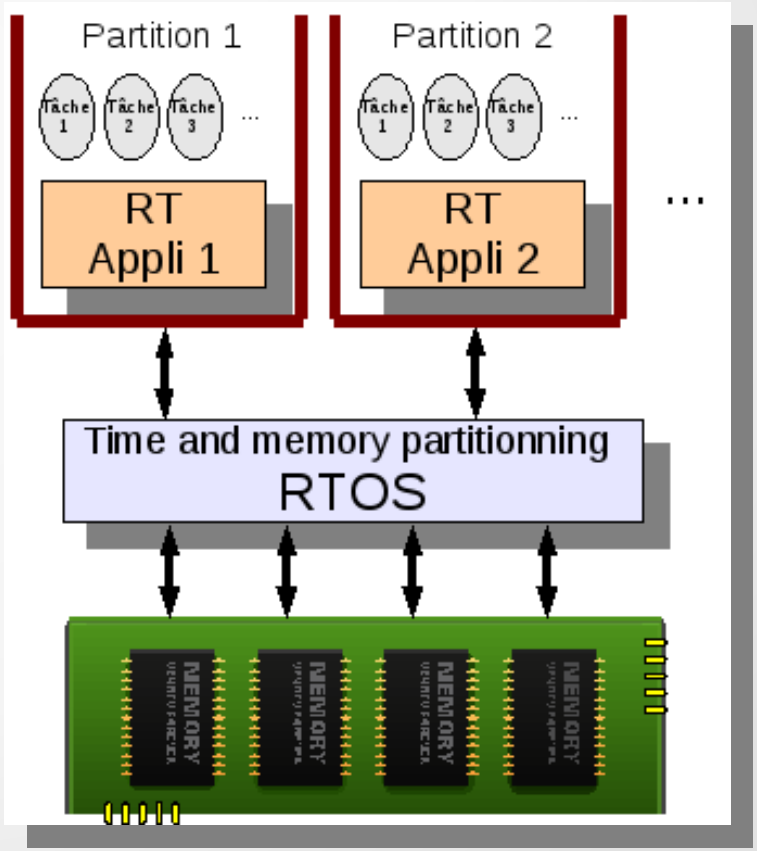
Différents RTOS

safety-critical

- Permet d'isoler temporellement et en mémoire des applications différentes

- Utilisation de l'API du RTOS par l'application

- S'assure que le partitionnement est respecté



Différents RTOS

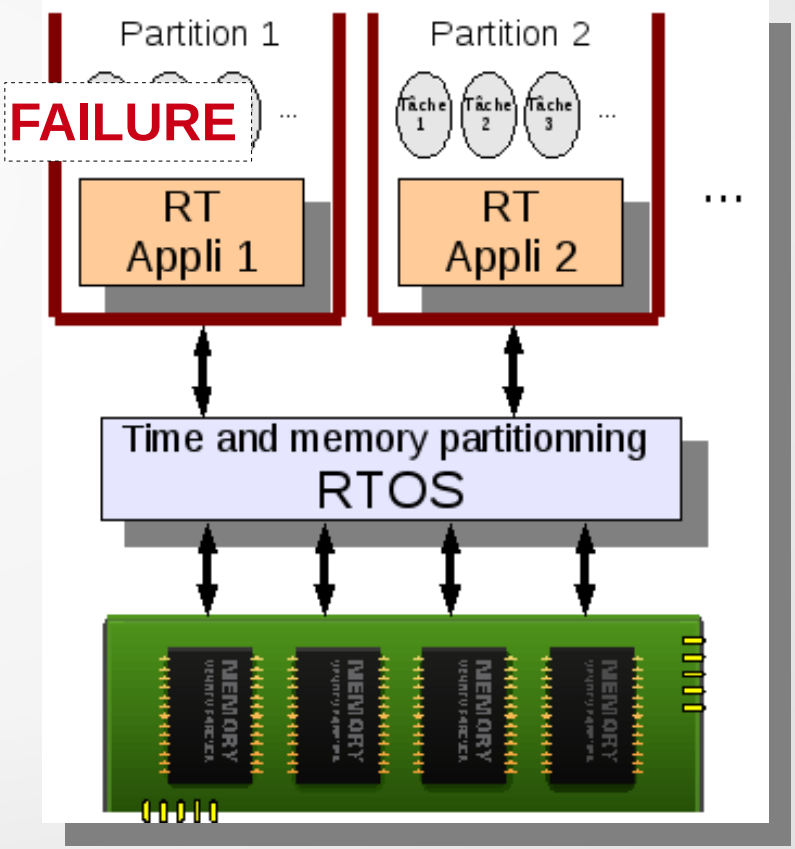
safety-critical

- Permet d'isoler temporellement et en mémoire des applications différentes

- Utilisation de l'API du RTOS par l'application

- S'assure que le partitionnement est respecté

➔ Éviter la propagation des pannes



Différents RTOS

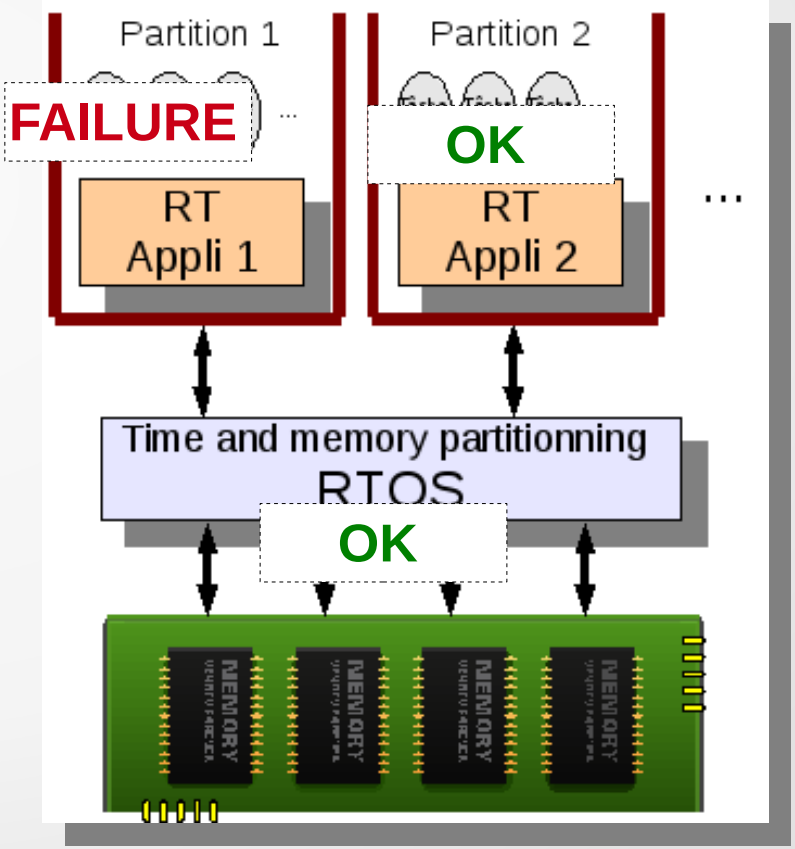
safety-critical

- Permet d'isoler temporellement et en mémoire des applications différentes

- Utilisation de l'API du RTOS par l'application

- S'assure que le partitionnement est respecté

➔ Éviter la propagation des pannes



Contenu du cours

- **Qu'est-ce qu'un OS temps réel (RTOS)**
  - RTOS vs GPOS
  - Différents RTOS...
  - **Les objets *communs* d'un RTOS**
    - Tâche
    - Sémaphore
    - Ordonnanceur
    - Les objets *haut niveau*
  - Implémentation d'un RTOS
- Programmation avec un OS temps réel
- Mise en Oeuvre

Les objets du temps réel
les basiques

- Les objets programmables
  - La tâche (ou le thread ou processus léger)
  - Les routines d’interruption, l’alarme
- Objet de communication
  - Le sémaphore
    - Synchronisation
    - Exclusion mutuelle

Les objets du temps réel
la tâche (1)

- Une tâche est un fil d'exécution
- Une tâche est représentée par une structure[1] contenant :
  - Un identifiant
  - Une référence vers son code
  - Une priorité
  - Un état :
    - Prête
    - Bloquée
    - En cours
    - ...
  - Un contexte
    - valeurs des registres
    - compteur de programme
    - pile

```
typedef struct
{
    void *r_stack;            // Start of stack (top address-1)
   void (*start) (void);      // Entry point of code
   pProcessID pid;            // Pointer to Process ID block
   unsigned char priority;    // Priority of task (0-255)
} FLASH const TaskControlBlock;


typedef struct ProcessID
{                                          //state is encoded in the scheduler
    struct ProcessID * next;
    unsigned char flags;
    unsigned char priority;
    void *ContextPointer;
#ifdef SINGLESTEPSUPPORT
    unsigned char * bp1;
    unsigned char * bp2;
#endif
} * pProcessId;
```

Exemple simplifié d'AvrX

1 : Souvent appelée TaskControlBlock

Les objets du temps réel
la tâche (2)

Primitives classiques :

- **Appel effectué depuis une autre tâche**
  - création (prête ou bloquée)
  - Lancement
  - Réveil
- **Appel effectué depuis une autre tâche ou dans la tâche elle-même**
  - Destruction
  - Suspension
  - Demande du niveau de priorité
  - Changement de priorité
- **Appel dans la tâche elle même**
  - mise en sommeil
  - appels de fonctions

```
INTERFACE void AvrXRunTask(TaskControlBlock *);
INTERFACE unsigned char AvrXInitTask(TaskControlBlock *);

INTERFACE void AvrXResume(pProcessID);
INTERFACE void AvrXSuspend(pProcessID);
INTERFACE void AvrXBreakPoint(pProcessID);
INTERFACE unsigned char AvrXSingleStep(pProcessID);
INTERFACE unsigned char AvrXSingleStepNext(pProcessID);

INTERFACE void AvrXTerminate(pProcessID);
INTERFACE void AvrXTaskExit(void);
INTERFACE void AvrXHalt(void);     // Halt Processor (error only)

INTERFACE void AvrXWaitTask(pProcessID);
INTERFACE Mutex AvrXTestPid(pProcessID);

INTERFACE unsigned char AvrXPriority(pProcessID);
INTERFACE unsigned char AvrXChangePriority(pProcessID, unsigned
char);
INTERFACE pProcessID AvrXSelf(void);
```

Les objets du temps réel
la tâche (3)

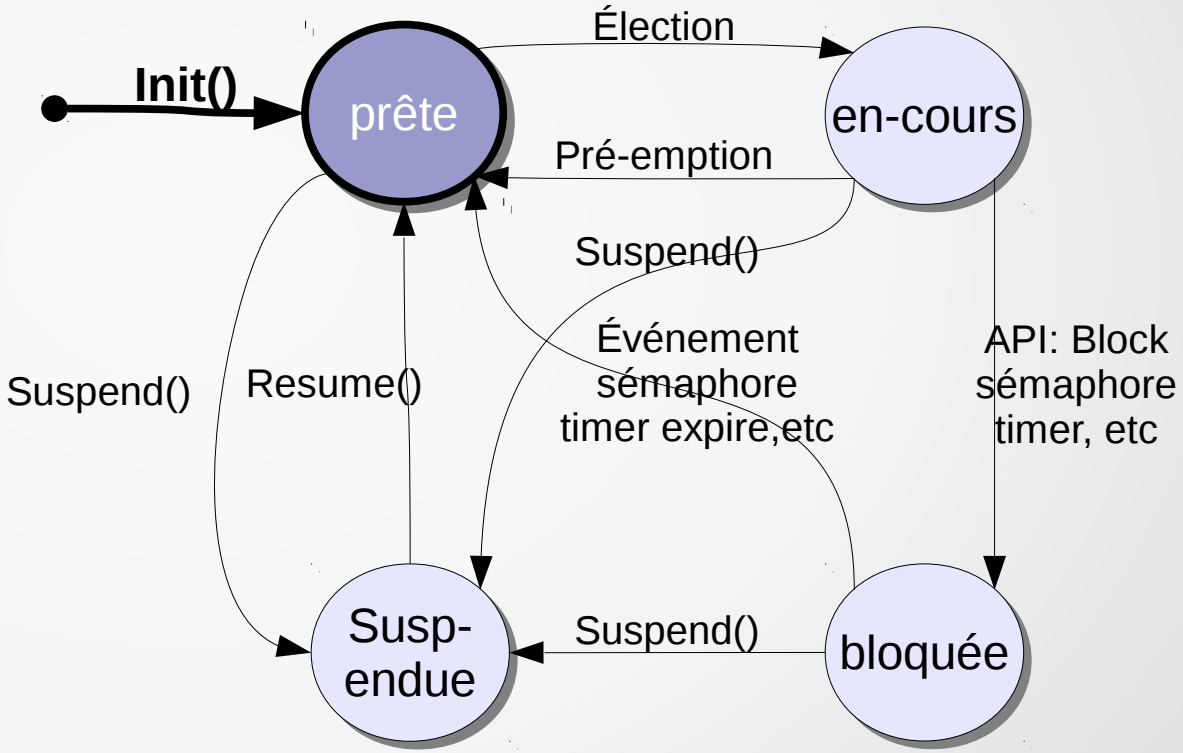
Exemple de primitives

- **VxWorks**
  - LOCAL int taskSpawn(char* "taskName",int priority,0,10000, functionName,0,0,0,0,0, \\ 0,0,0,0,0)
  - LOCAL int taskInit(…)
  - taskSafe ()
- **IRMX**
  - static TOKEN rq_create_task(priority, (void far *) taskId, ...)
- **Win CE**
  - CreateProcess(NULL|appliName, "MonProcessFils", …)
  - HANDLE CreateThread(lpThreadAttributes, dwStackSize, lpStartAddress, lpParameter, \\ dwCreationFlags, CREATE_SUSPENDED, lpThreadId)
- **RT-Linux**
  - int init_module(void)
  - int pthread_create(pthread_t, pthread_attr_t , void *(*start_routine)(void*), void *arg)
- **TIM micro-kernel**
  - int install_task (char * taskName, int stackSize, void * functionAd);
- **RTX_166 Tiny Real-Time**
  - os_create_task (int tasd_id);
  - os_delete_task (int tasd_id);

Les objets du temps réel
la tâche (4)

État d'une tâche :

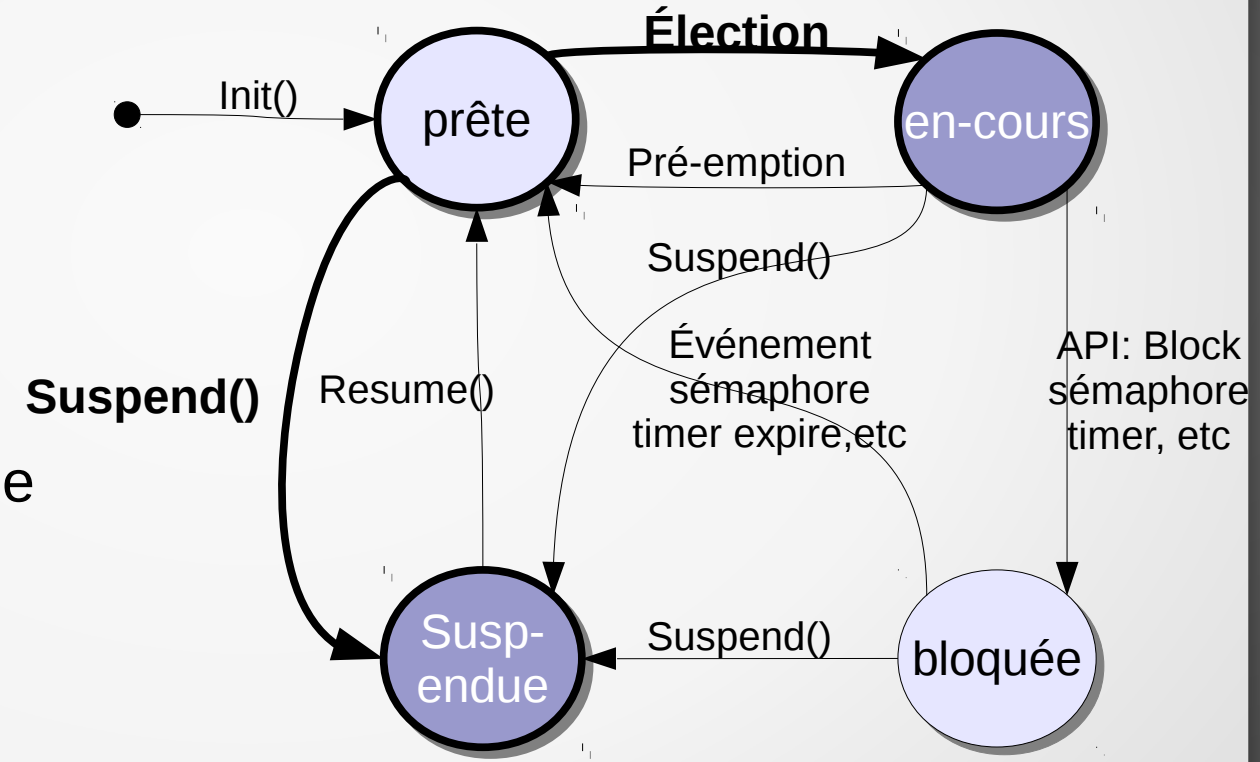
- prête
  - mémoire allouées
- bloquée
  - en attente d'une ressource ou d'un événement
- suspendue
  - en mémoire mais ne sera pas ordonnancée
- en-cours
  - Choisie par l'ordonnanceur pour s'exécuter
- *Morte*
  - Plus de mémoire allouée



Les objets du temps réel
la tâche (4)

État d'une tâche :

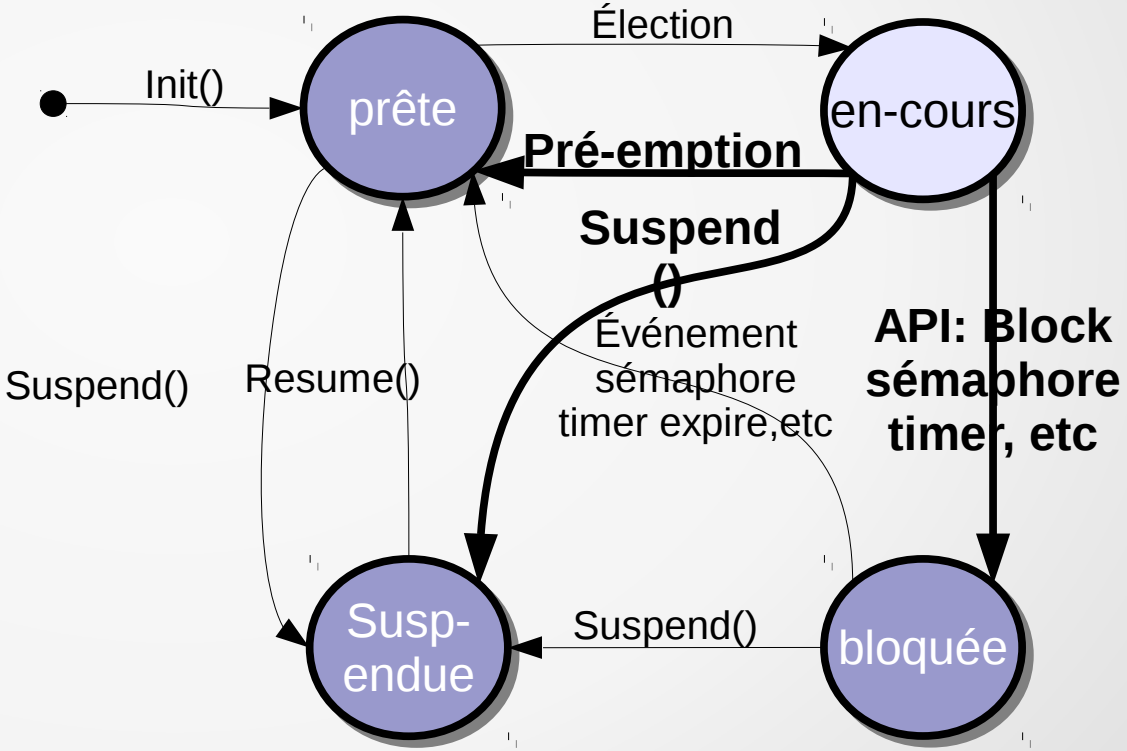
- prête
  - mémoire allouées
- bloquée
  - en attente d'une ressource ou d'un événement
- suspendue
  - en mémoire mais ne sera pas ordonnancée
- en-cours
  - Choisie par l'ordonnanceur pour s'exécuter
- *Morte*
  - Plus de mémoire allouée



Les objets du temps réel
la tâche (4)

État d'une tâche :

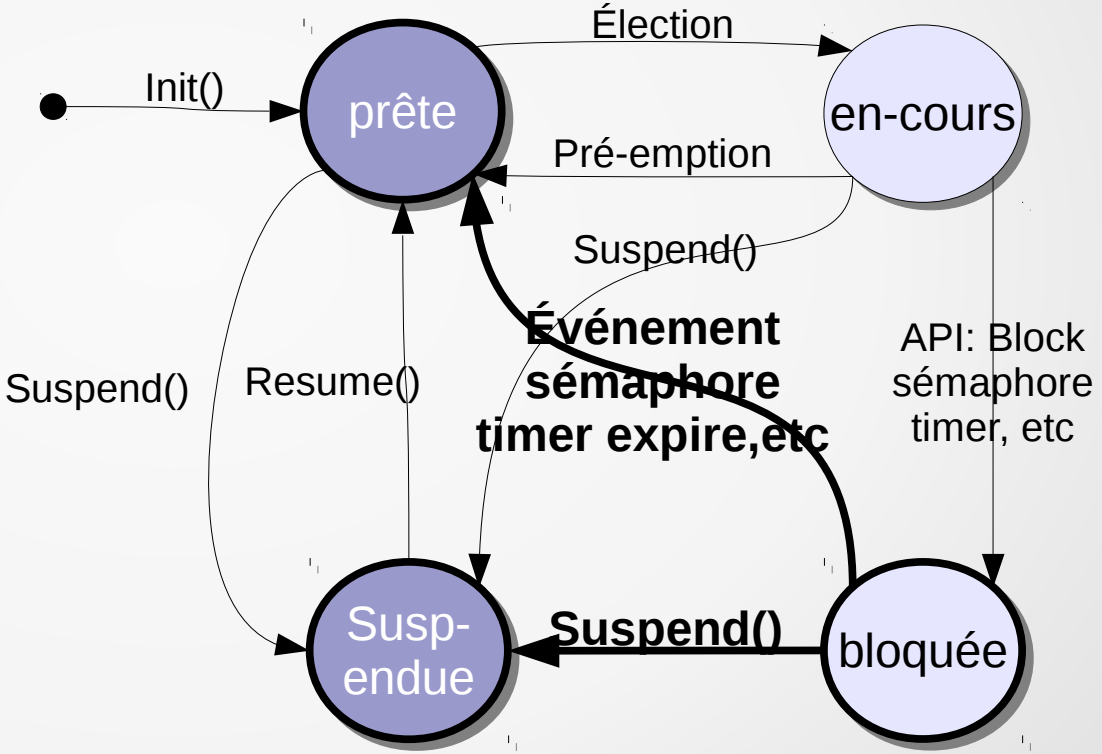
- prête
  - mémoire allouées
- bloquée
  - en attente d'une ressource ou d'un événement
- suspendue
  - en mémoire mais ne sera pas ordonnancée
- en-cours
  - Choisie par l'ordonnanceur pour s'exécuter
- *Morte*
  - Plus de mémoire allouée



Les objets du temps réel
la tâche (4)

État d'une tâche :

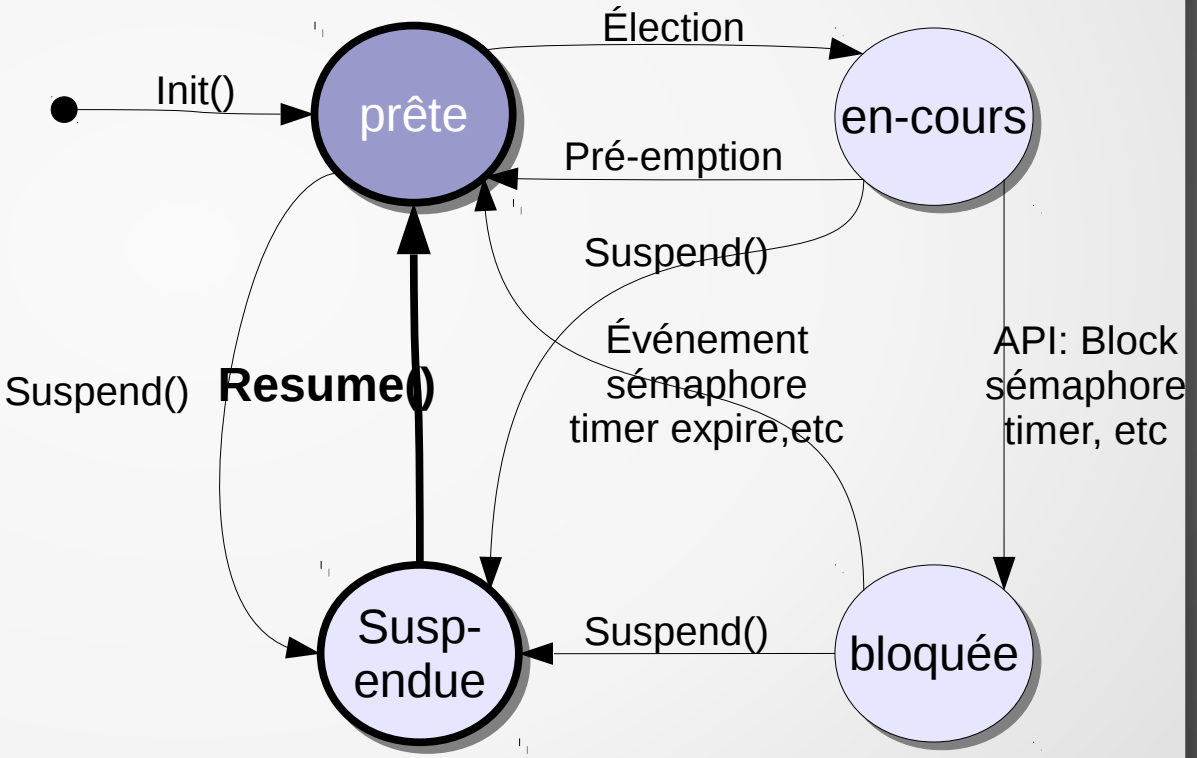
- prête
  - mémoire allouées
- bloquée
  - en attente d'une ressource ou d'un événement
- suspendue
  - en mémoire mais ne sera pas ordonnancée
- en-cours
  - Choisie par l'ordonnanceur pour s'exécuter
- *Morte*
  - Plus de mémoire allouée



Les objets du temps réel
la tâche (4)

État d'une tâche :

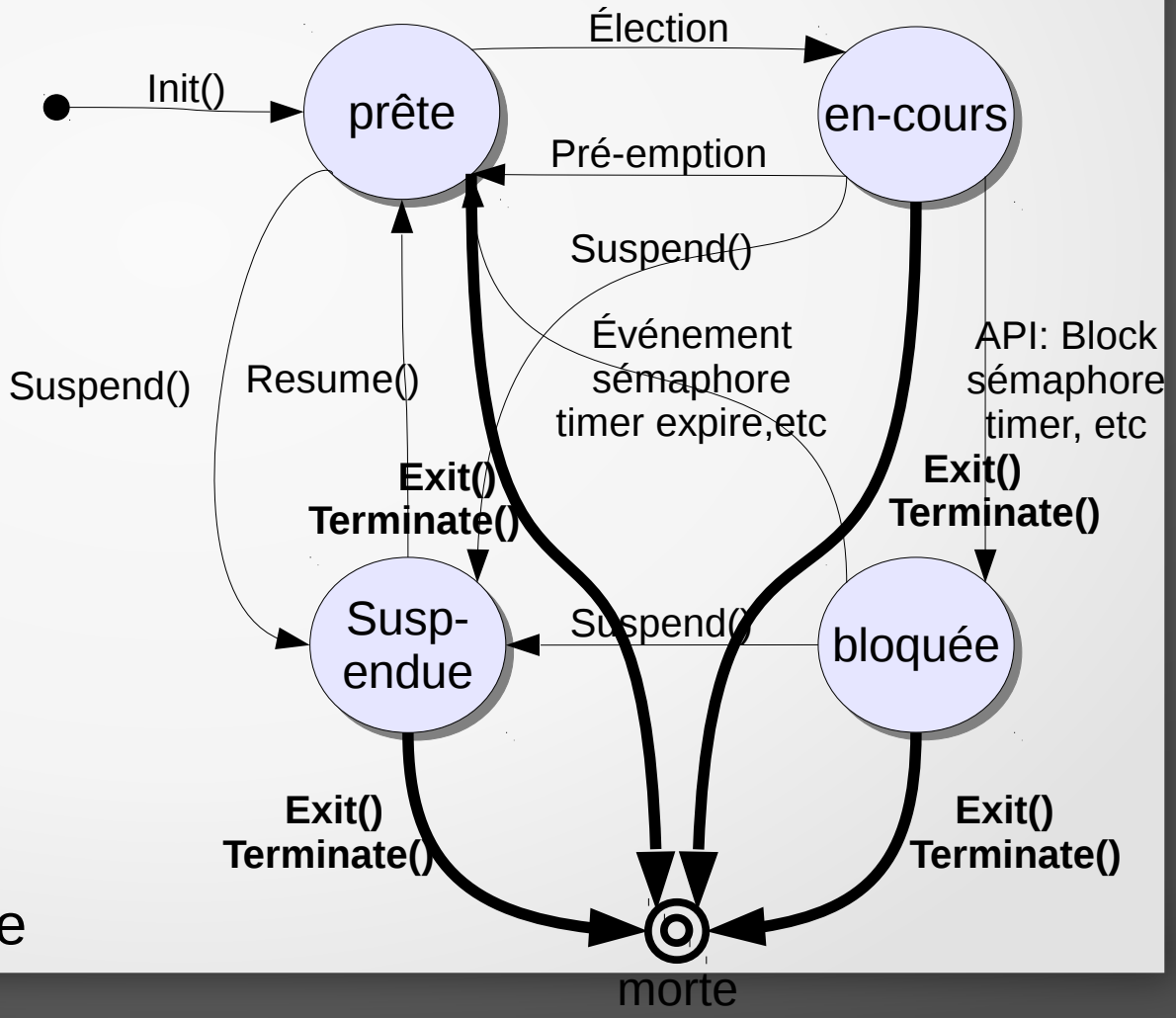
- prête
  - mémoire allouées
- bloquée
  - en attente d'une ressource ou d'un événement
- suspendue
  - en mémoire mais ne sera pas ordonnancée
- en-cours
  - Choisie par l'ordonnanceur pour s'exécuter
- *Morte*
  - Plus de mémoire allouée



Les objets du temps réel
la tâche (4)

État d'une tâche :

- prête
  - mémoire allouées
- bloquée
  - en attente d'une ressource ou d'un événement
- suspendue
  - en mémoire mais ne sera pas ordonnancée
- en-cours
  - Choisie par l'ordonnanceur pour s'exécuter
- *Morte*
  - Plus de mémoire allouée



Les objets du temps réel

la tâche (5)

Tâche classique :

- Initialisation (variables, etc)
- Boucle infinie
  - Faire quelques choses
  - Attendre
    - Ressource
    - Timer
    - Sémaphore

```
AVRX_GCC_TASKDEF(task1, 8, 3)
{
unsigned char valueD=0;
PORTD = valueD;
 while (1)
 {
  AvrXStartTimer(&timer1, 20); // 20 ms delay
  AvrXWaitTimer(&timer1);
  PORTD=++valueD;  // Modify PORTD
  }
}
```

Exemple AvrX

Les objets du temps réel

la tâche (5)

Tâche classique :

- Initialisation (variables, etc)
- Boucle infinie
  - Faire quelques choses
  - Attendre
    - Ressource
    - Timer
    - Sémaphore

```
AVRX_GCC_TASKDEF(task1, 8, 3)
{
unsigned char valueD=0;
PORTD = valueD;
 while (1)
 {
  AvrXStartTimer(&timer1, 20); // 20 ms delay
  PORTD=++valueD;  // Modify PORTD
  AvrXWaitTimer(&timer1);
  }
}
```

Exemple AvrX

Les objets du temps réel
routine d'interruption (5)

Interruption classique :

- Démasque des ITs (ou non)
- Faire le minimum de chose
  - MAJ d'une variable
  - Modification d'un sémaphore
- Fin

```
AVRX_SIGINT(TIMER0_OVF_vect)
{
    IntProlog();     // Save interrupted context, switch stacks
    TCNT0 = TCNT0_INIT;       // Reload the timer counter
    AvrXTimerHandler();         // Process Timer queue
    Epilog();       // Restore context of next running task
}
```

Exemple AvrX

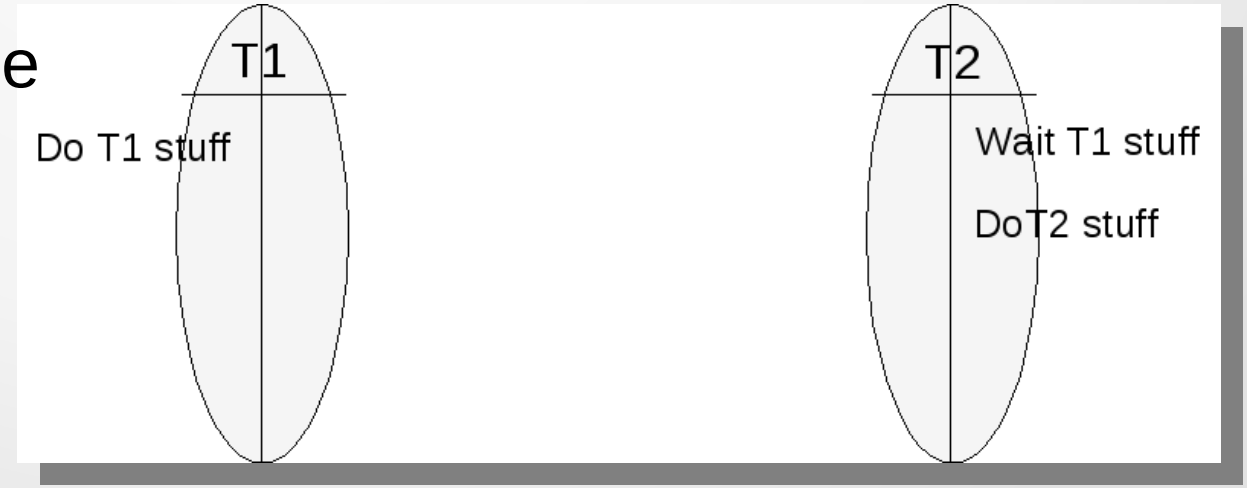
Les objets du temps réel
le sémaphore (1)

- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - synchronisation entre tâches
  - exclusion mutuelle
  - Interruption logicielle

Les objets du temps réel

le sémaphore (1)

- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - **synchronisation entre tâches**
  - exclusion mutuelle
  - Interruption logicielle

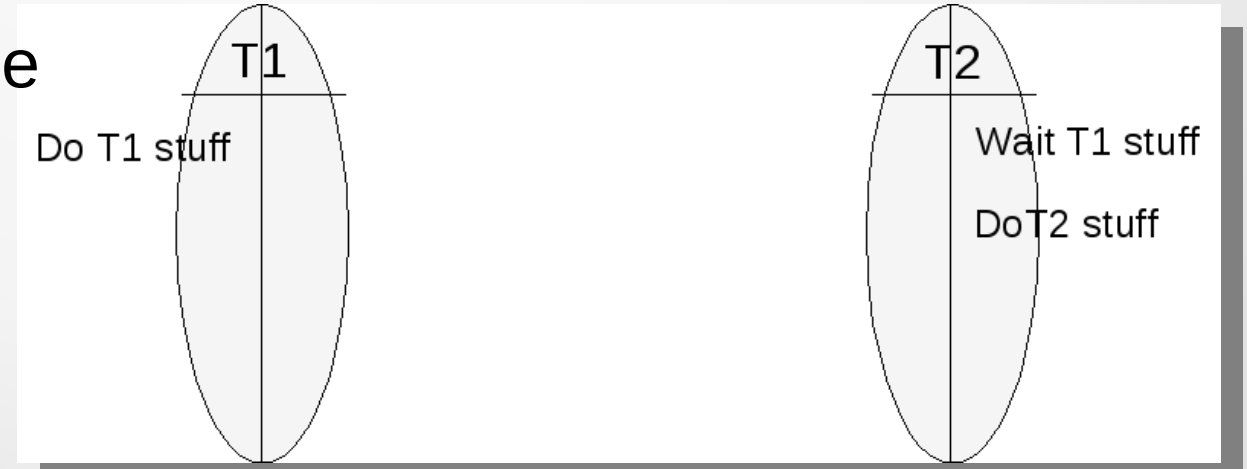


Les objets du temps réel

le sémaphore (1)

- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - **synchronisation entre tâches**
  - exclusion mutuelle
  - Interruption logicielle

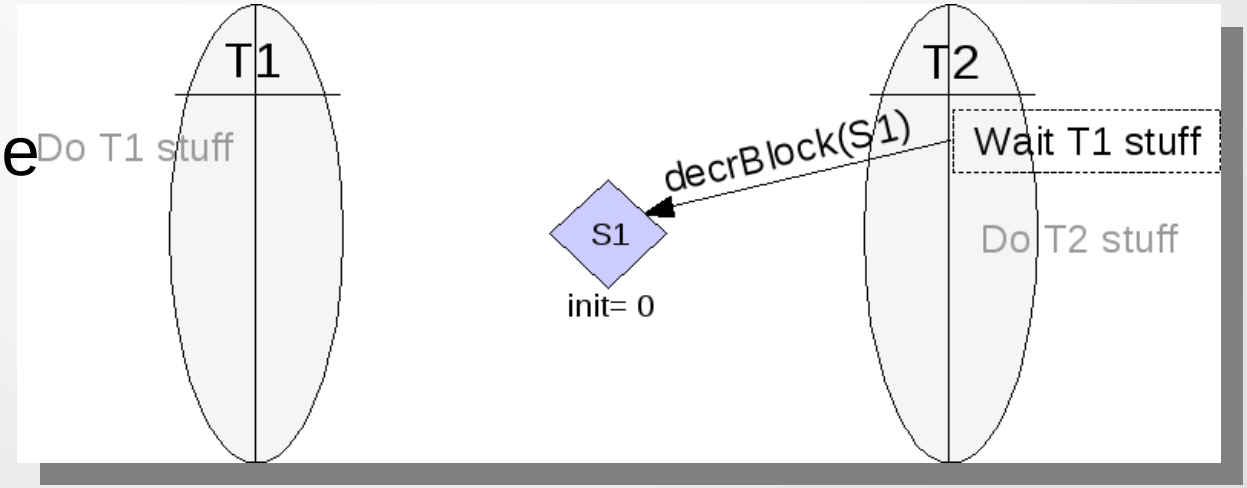
Lacatre: langage d'aide à la conception d'application temps réel
www.lisyc.univ-brest.fr/pages_perso/babau/cours/multitache.pdf



Les objets du temps réel

le sémaphore (1)

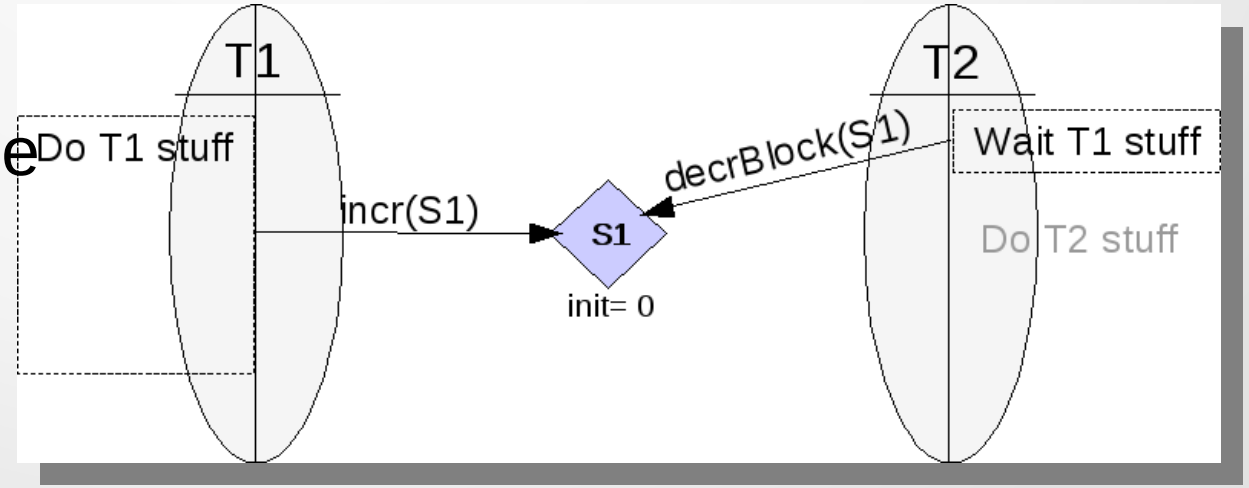
- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - **synchronisation entre tâches**
  - exclusion mutuelle
  - Interruption logicielle



Les objets du temps réel

le sémaphore (1)

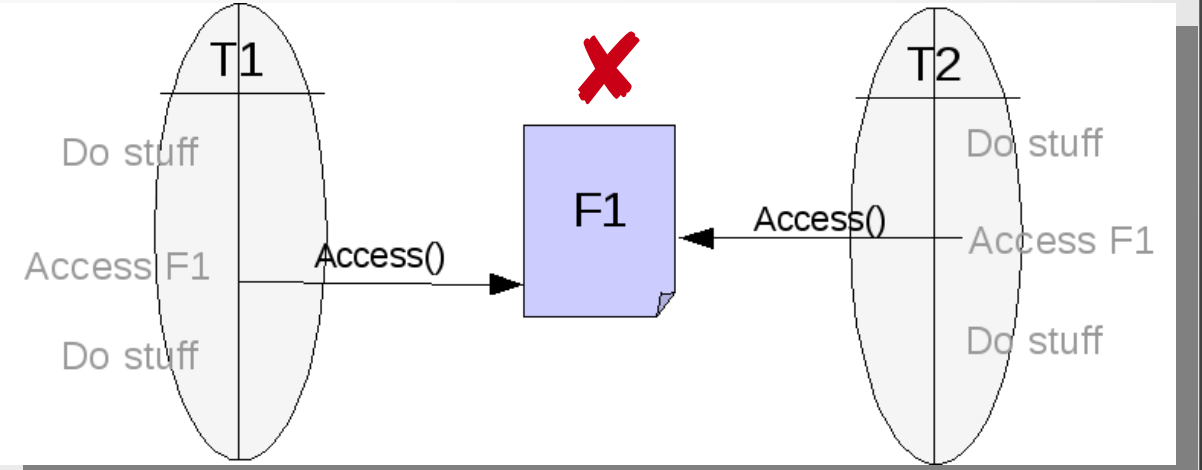
- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - **synchronisation entre tâches**
  - exclusion mutuelle
  - Interruption logicielle



Les objets du temps réel

le sémaphore (2)

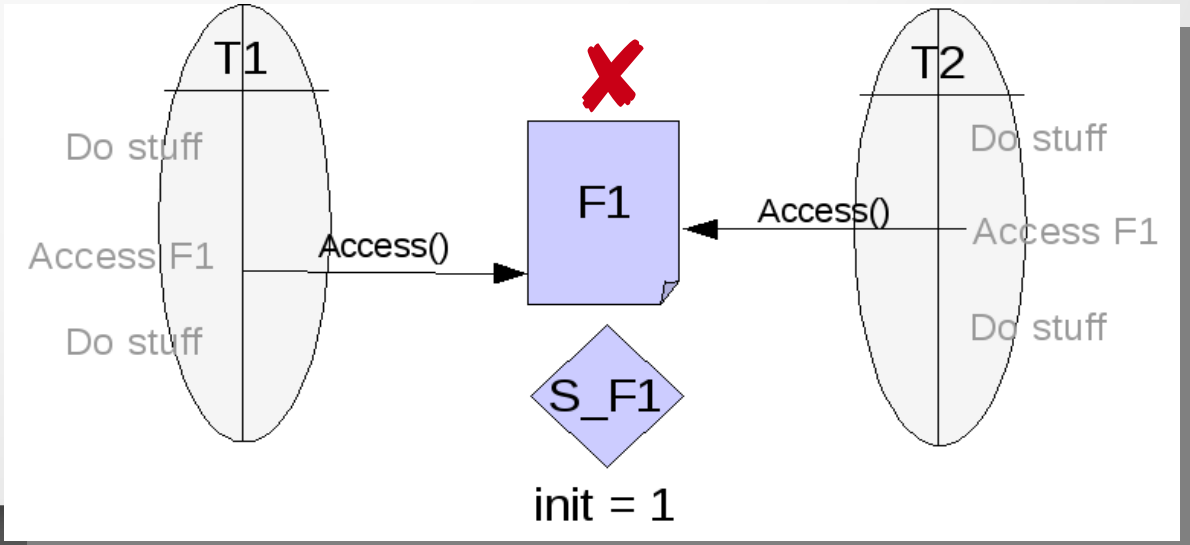
- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - synchronisation entre tâches
  - **exclusion mutuelle**
  - Interruption logicielle



Les objets du temps réel

le sémaphore (2)

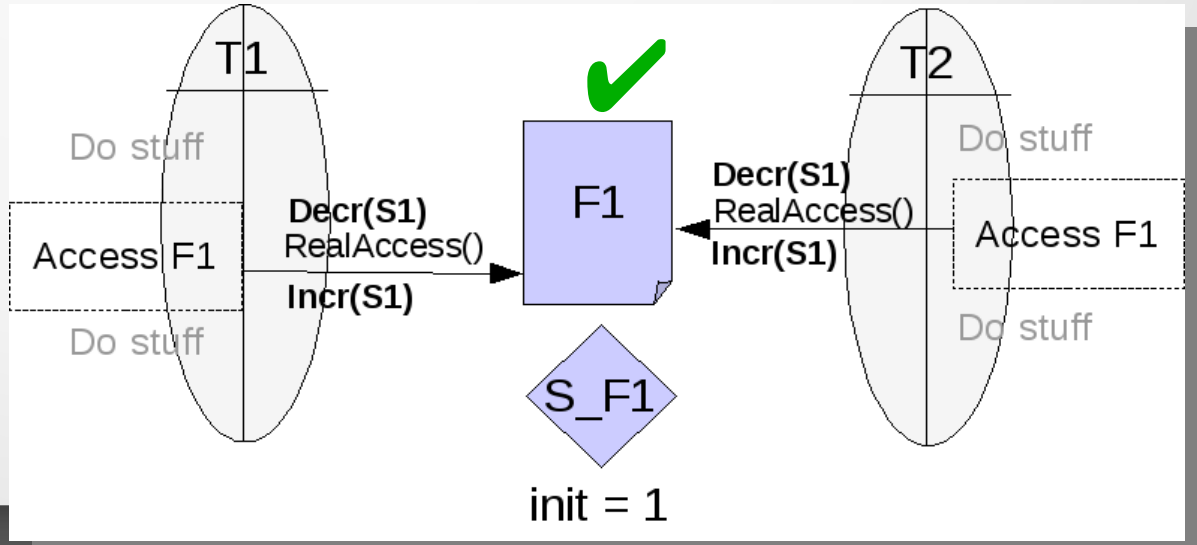
- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - synchronisation entre tâches
  - **exclusion mutuelle**
  - Interruption logicielle



Les objets du temps réel

le sémaphore (2)

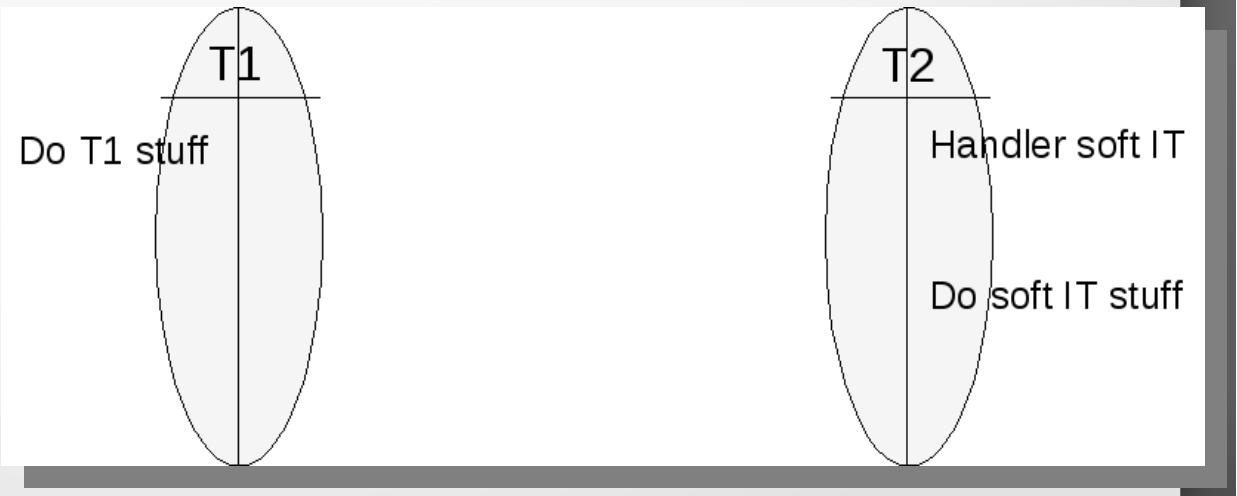
- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - synchronisation entre tâches
  - **exclusion mutuelle**
  - Interruption logicielle



Les objets du temps réel

le sémaphore (3 ?)

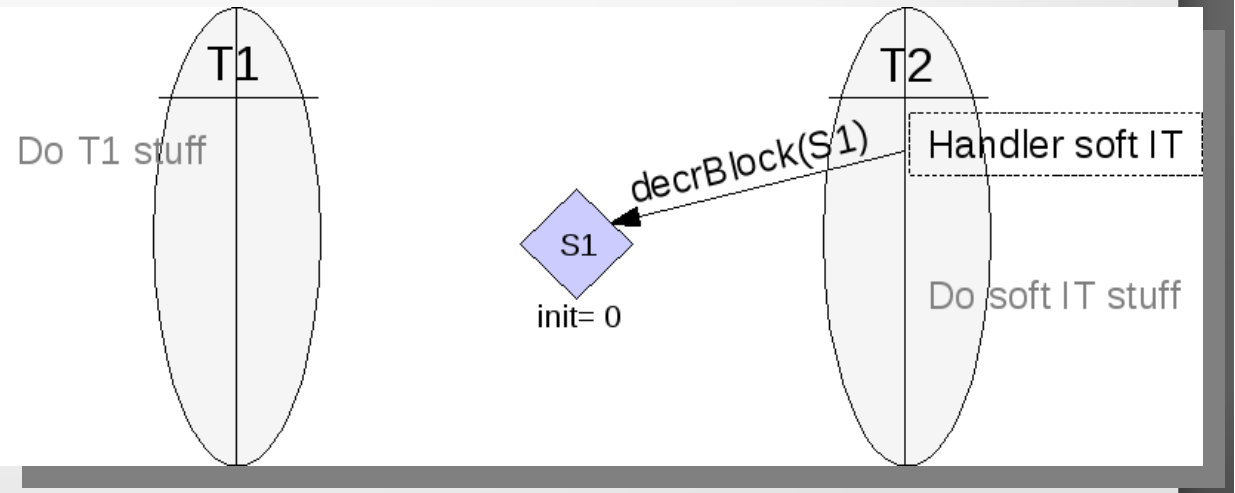
- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - synchronisation entre tâches
  - exclusion mutuelle
  - **Interruption logicielle**



Les objets du temps réel

le sémaphore (3 ?)

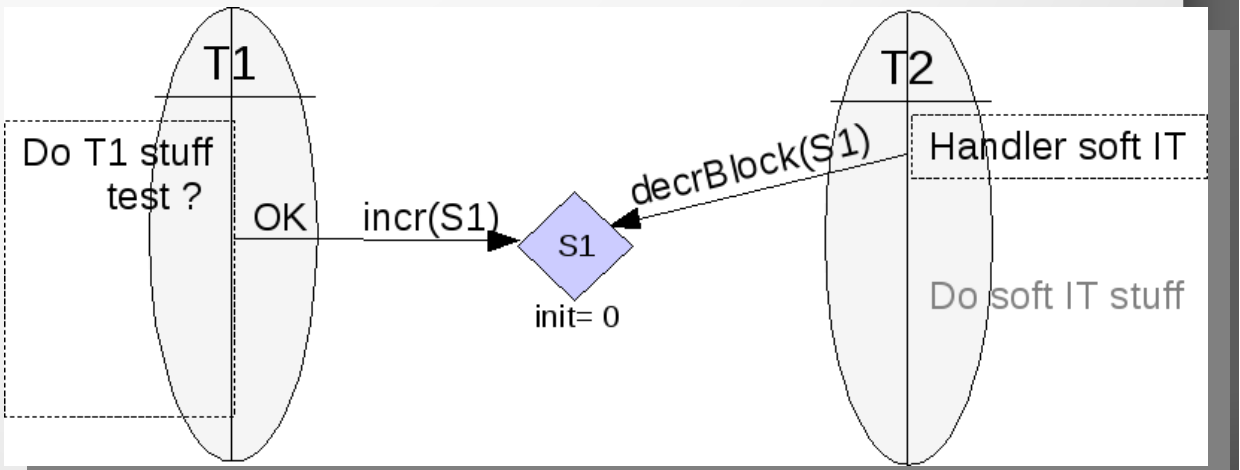
- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - synchronisation entre tâches
  - exclusion mutuelle
  - **Interruption logicielle**



Les objets du temps réel

le sémaphore (3 ?)

- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - synchronisation entre tâches
  - exclusion mutuelle
  - **Interruption logicielle**



Les objets du temps réel

le sémaphore (3)

- Type : booléen ou à compte
- Possède des primitives particulières :
  - Ces primitives sont atomic !!
    - Décrémenter() *//ne peut pas être < 0*
    - Incrémenter()
- Rôle
  - synchronisation entre tâches
  - exclusion mutuelle
  - Interruption logicielle
- **Accès par file d'attente**
  - **FIFO**
  - **PRIORITY**

Les objets du temps réel
le sémaphore (4)

- Primitives classiques
  - Création
    - Initialisation
  - Destruction
    - impact sur l ’application
  - Dépôt (incrémentation)
    - nombre d’unité
  - Retrait (décrémentation)
    - nombre d’unité
    - temps d’attente
      - Infini
      - fini

```
void AvrXSetSemaphore(pMutex);        //blocking
void AvrXIntSetSemaphore(pMutex);     //non blocking
void AvrXWaitSemaphore(pMutex);
Mutex AvrXTestSemaphore(pMutex);      //blocking
Mutex AvrXIntTestSemaphore(pMutex); //non blocking
void AvrXResetSemaphore(pMutex);
void AvrXResetObjectSemaphore(pMutex);
```

Exemple d'AvrX

Les objets du temps réel
le sémaphore (5)

Exemples de primitives

- **VxWorks**

```
semId = semBCreate(SEM_Q_FIFO | SEM_Q_PRIORITY, SEM_FULL | SEM_EMPTY) ;
semId = semCCreate(SEM_Q_FIFO | SEM_Q_PRIORITY, initCount) ;
semId = semMCreate(SEM_Q_FIFO | SEM_Q_PRIORITY |SEM_DELETE_SAFE|
    SEM_INVERSION_SAFE)
status = semGive(semId)
status = semFlush(semId) ; /* déblocage de toutes les tâches en attente
status = semTake(semId, temps | WAIT_FOREVER | NO_WAIT) ;
```

- **iRMX**

```
semId_tk = rq_create_semaphore(valInit,valMax,flags,&status) ;
rq_send_units(semId,nbUnite,&status)
reste = rq_receive_units(semId_tk,nbUnite,temps,&status)
```

- **Win32**

```
HANDLE CreateSemaphore( LPSECURITY_ATTRIBUTES, InitialCount, MaximumCount,
    lpName);
ReleaseSemaphore ( semhandle , unitNumber , 0 )
WaitForSingleObject ( sem , INFINITE | Time-out ) // timeout en ms
```

- **TIM micro-kernel** : pas de « sémaphore »

Les objets du temps réel
l'ordonnanceur (1)

- Mode préemptif
  - Ordonnanceur appelé après chaque changement d'état
    - Appel des primitives de communication
    - Après une routine (si déblocage possible d'une tâche)
- Mode non préemptif
  - Ordonnanceur appelé à la fin de tâche / boucle
    - task_suspend(), task_exit()
- Ordonnancement classique
  - Round Robin
  - Round Robin par priorités
  - EDF (earliest deadline first) / RMA (rate monotonic analysis)

Les objets du temps réel
l'ordonnanceur (2)

- Mode préemptif
  - Ordonnanceur appelé après chaque changement d'état
    - Appel des primitives de communication
    - Après une routine (si déblocage possible d'une tâche)
- Mode non préemptif
  - Ordonnanceur appelé à la fin de tâche / boucle
    - task_suspend(), task_exit()
- Blocage de l'ordonnanceur
  - Passage en mode non préemptif
  - Exécution de primitives *système*
  - Primitives spécifiques

Les objets du temps réel
timer logiciel (1)

- Pas de nombre limité (théoriquement)
- attente non active (pour le micro-contrôleur)
- Différentes stratégies :
  - 1 tâche spécifique qui modifie plusieurs sémaphores lors de l'expiration d'un timer (pas forcément les même à chaque fois) (période ≅ timer)
  - Appel d'un timer dans la tâche concernée (période dépendante du temps d'exécution des calculs)

Les objets du temps réel

objets *haut-niveau*

- **Événement**
  - synchronisation entre tâches
  - rendez-vous
- **Boîte aux lettres (FIFO)**
  - échange d'information synchronisées entre tâches
  - objet de communication entre lecteurs/écrivains
- **Pipe**
  - échange d'information synchronisées entre tâches de processus distinct
  - objet de communication entre lecteurs/écrivains

Contenu du cours

- **Qu'est-ce qu'un OS temps réel (RTOS)**
  - RTOS vs GPOS
  - Différents RTOS...
  - Les objets *communs* d'un RTOS
  - **Implémentation d'un RTOS**
- Programmation avec un OS temps réel
- Mise en Oeuvre

Implémentation d'un RTOS
gestion du temps (1)

- Base de temps unique pour le système
  - ➔ Mise en place d'une interruption périodique : compteur de ticks
    - Activation d'un timer
    - intervalle de X ms (10, 200, ...)
    - peut souvent être modifiée
- Tous les ticks *//si pré-emptif* :
  - L'OS prend la main
  - L'OS incrémente le compteur de tick
  - Vérifie si des timers logiciels ont expiré
    - Modifie des sémaphores ou l'état des tâches selon oui ou non
  - Ré-ordonnance si des changement ont eu lieu
  - L'OS rend la main

Implémentation d'un RTOS

L'ordonnanceur (1)

- Sans priorité, non pré-emptif
  - Tient à jour une liste (un tableau) des tâches dans l'état ”prête”
  - Lance l'exécution de la prochaine tâche dans la liste dès que la tâche en cours rend la main
- Sans priorité, pré-emptif
  - Tient à jour une liste (un tableau) des tâches dans l'état ”prête”
  - Lance l'exécution de la prochaine tâche dans la liste dès que la tâche en cours passe dans l'état 'bloquée', 'suspendue' ou 'morte'

Implémentation d'un RTOS

L'ordonnanceur (2)

- Avec priorité, pré-emptif
  - Tient à jour plusieurs listes des tâches dans l'état ”prête” : une par priorité
  - Lance l'exécution de la prochaine tâche dans la liste de plus haute priorité non vide (dès qu'une tâche de plus haute priorité est ”prête”) et la tâche en cours passe dans l'état 'bloquée', 'suspendue' ou 'morte')

Implémentation d'un RTOS

L'ordonnanceur (2)

- Avec priorité, pré-emptif
  - Tient à jour plusieurs listes des tâches dans l'état ”prête” : une par priorité
  - Lance l'exécution de la prochaine tâche dans la liste de plus haute priorité non vide (dès qu'une tâche de plus haute priorité est ”prête”) et la tâche en cours passe dans l'état 'bloquée', 'suspendue' ou 'morte')
- Inversion de priorité ?
  - Si une tâche de haute priorité est en attente d'une ressource bloquée par une basse priorité, la basse priorité devient prioritaire pour que la haute priorité puisse s'exécuter

Implémentation d'un RTOS
gestion d'interruption (1)

(1) Arrivée d'une interruption non masquée

(2) Fin de l’instruction de la tâche courante

(3) Sauvegarde du contexte de la tâche

(4) Acquittement de l ’interruption (contrôleur d’interruption)

(5) Exécution de la routine d ’interruption associée

(6) Ré-ordonnancement

- retour à la tâche interrompue
- activation nouvelle tâche

Contenu du cours

- **Qu'est-ce qu'un OS temps réel (RTOS)**
  - RTOS vs GPOS
  - Différents RTOS...
  - Les objets *communs* d'un RTOS
  - Implémentation d'un RTOS
- **Programmation avec un OS temps réel**
- Mise en Oeuvre

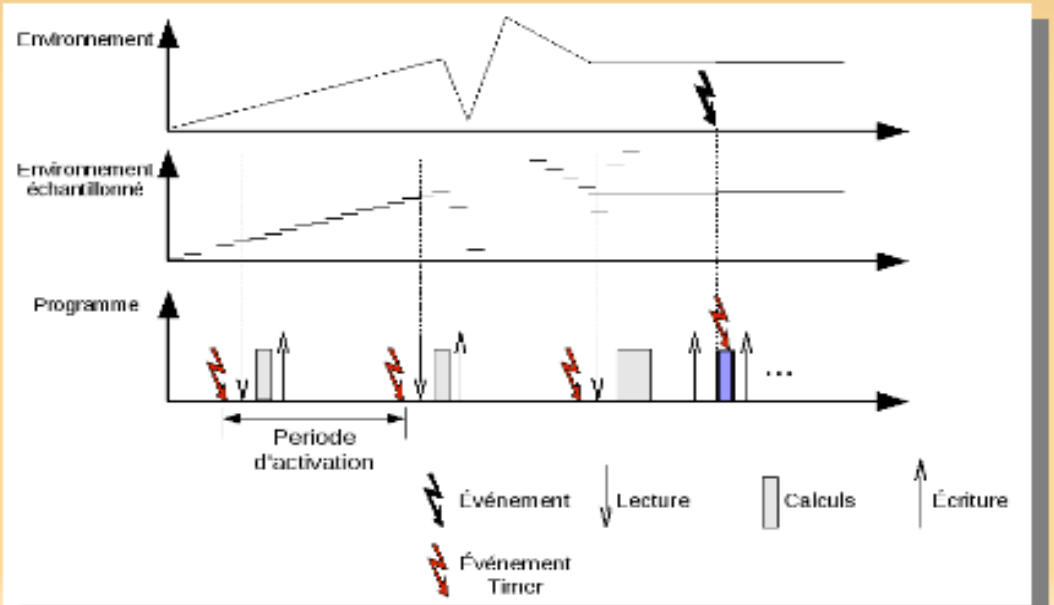
Programmation avec un RTOS
les tâches (1)

Rappel :

Micro-contrôleur sans OS

comment ?

2) Avec interruption
   - Inconvénient :
     - Demande un peu de technique



Programmation avec un RTOS
les tâches (1)

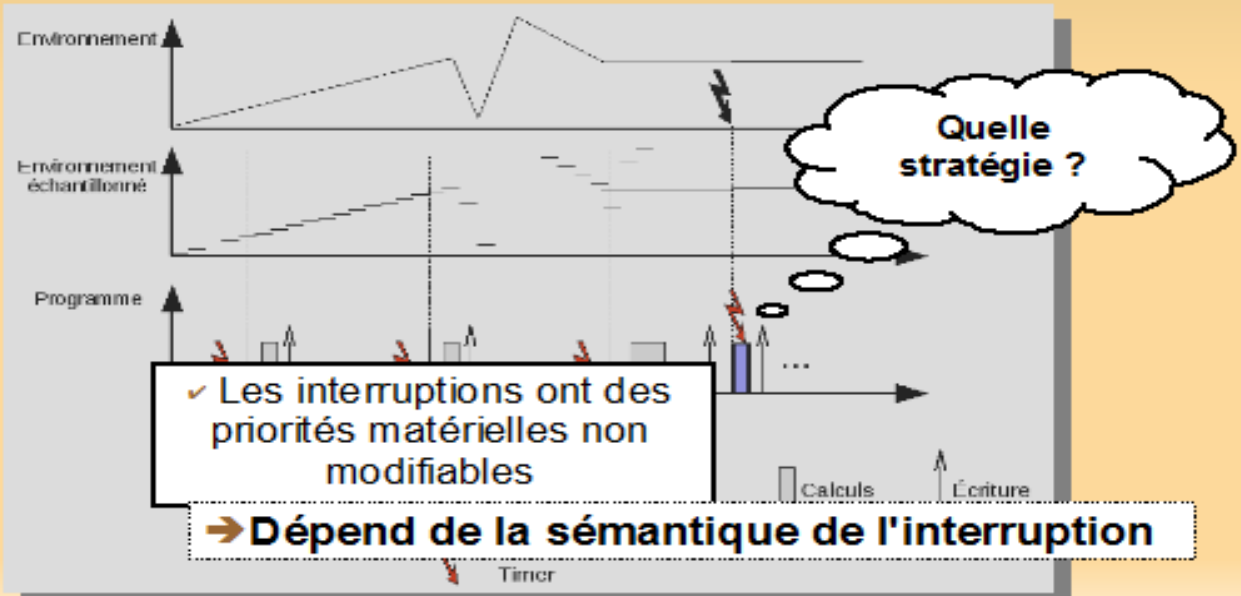
Rappel :

Micro-contrôleur sans OS

comment ?

2) Avec interruption
   - Inconvénient :
     - Demande un peu de technique

Limite du sans OS ?



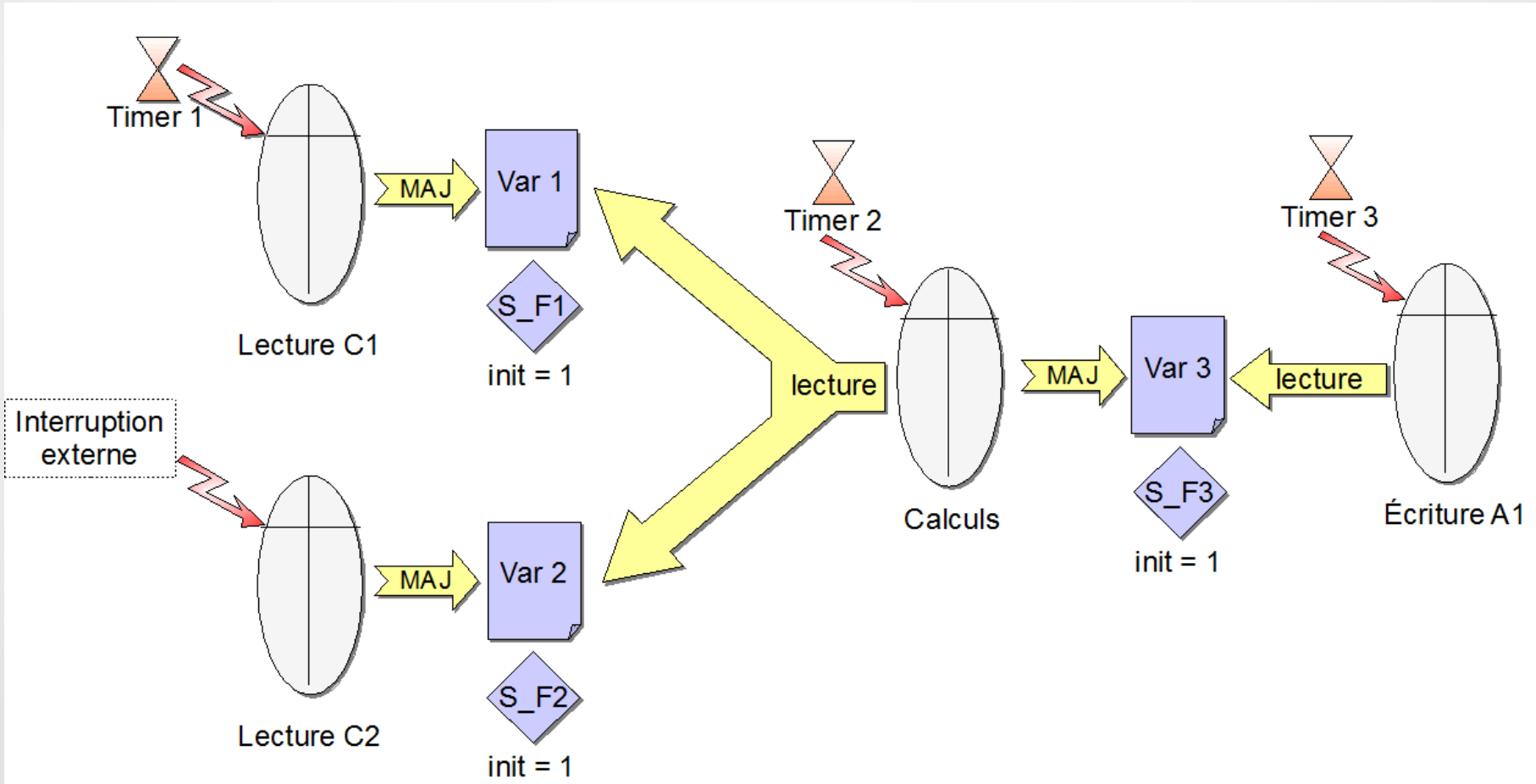
Programmation avec un RTOS
les tâches (2)

Principe :

- Une tâche par préoccupation, exemple :
  - Lecture des capteurs : 1 (ou plusieurs) tâche
  - Calcul : 1 (ou plusieurs) tâches
  - Écriture sur actionneurs : 1 (ou plusieurs) tâche
- Chaque tâche a son propre rythme
  - Périodique (déclenchée par timer)
  - Déclenchée par des handlers d'interruptions externes
- Éviter la création de tâche dynamique

Programmation avec un RTOS
les tâches (2)

Principe :



Programmation avec un RTOS
les ressources (1)

- Rôle
  - entité logique ou physique
  - peut être partagée par plusieurs tâches
- Objet non présent en tant que tel dans un OS
- Type de ressource
  - structure de données / zone mémoire
  - fichier
  - dispositif physique
  - réseau

Programmation avec un RTOS
les ressources (2)

- Sémaphore d'exclusion mutuelle
- Passage en mode non préemptif ou super-priorité
- Masquage / démasquage des interruptions
  - Pas trop longtemps !
- Opération atomique
  - Difficile à assurer si on réutilise un OS existant

Programmation avec un RTOS

fonctions réentrantes (1)

Utilisables simultanément par plusieurs tâches

➔ **1 contexte d'appel par tâche (pile)**

```
int somme (int a , int b)
{
    return (a+b) ;
}
```

```
int coef  = 3 ;

int correction (int a)
{
     return ( a * coef ) ;
}
```

OK

```
int coef  ;
int coefMini = 3;

int correction (int a, int c )
{
     if (c > coef) {coef = c ;}
     else { coef = coefMini ; }
     return ( a * coef ) ;
}
```

KO

OK si protection par mutex de *coef*

OK si protection par mutex de l'appel de la fonction

Programmation avec un RTOS

fonctions réentrantes (1)

Utilisables simultanément par plusieurs tâches

➔ **1 contexte d'appel par tâche (pile)**

```
int somme (int a , int b)
{
    return (a+b) ;
}
```

```
int coef  = 3 ;

int correction (int a)
{
     return ( a * coef ) ;
}
```

Attention il faut souvent estimer la taille de la pile !

```
int coef  ;
int coefMini = 3;

int correction (int a, int c )
{
     if (c > coef) {coef = c ;}
     else { coef = coefMini ; }
     return ( a * coef ) ;
}
```

OK si protection par mutex de *coef*

OK si protection par mutex de l'appel de la fonction

Programmation avec un RTOS

fonctions non réentrantes (2)

non utilisables simultanément par plusieurs tâches

➔ **Pas de contexte d'appel par tâche (pas de pile)**

```
int somme (int a , int b)
{
      return (a+b) ;
}
```

```
int coef  = 3 ;

int correction (int a)
{
       return ( a * coef ) ;
}
```

OK si macro
(recopie du code dans chaque tâche)

```
int coef  ;
int coefMini = 3;

int correction (int a, int c )
{
      if (c > coef) {coef = c ;}
      else { coef = coefMini ; }
      return ( a * coef ) ;
}
```

pas OK

OK si protection par mutex de l'appel de la fonction

Choix d'un RTOS
critères (1)

- Normes
  - SCEPTRE (1982)
  - POSIX (Unix)
  - OSEK/VDX (Automobile)
  - Profil MARTE de l'OMG ?
  - Autosar ?
- Domaine
  - Ferroviaire : l'OS préconisé est QNX
- Supports
  - Matériel : processeurs supportés, cartes, mémoire (4 Go sous CE)
  - Drivers (série, LCD, CAN), piles de protocole
  - Fournisseurs et suivi des versions

- Coûts
  - Environnement de développement
    - (Tornado 15 000 € sans fonctionnalités)
  - Royalties (royalty free)
  - Développement supplémentaires
    - Drivers, protocole

Choix d'un RTOS
critères (2)

- Optimisation des ressources
  - Critères de taille et/ou de performance
  - Le comportement de l'applicatif est connu / estimé
  - Le comportement du support est configurable par l'application
- Réutilisation dans plusieurs contextes applicatifs
  - Services variés, services de haut niveau
  - Utilisation de librairies
- Adaptabilité pour les systèmes ouverts
  - Gestion dynamique de composants ou de services
    - Installation, ajout/retrait, …

- Maintenabilité
  - Accès au code
  - traçabilité des appels
- Portabilité
  - Respect de normes

Contenu du cours

- **Qu'est-ce qu'un OS temps réel (RTOS)**
  - RTOS vs GPOS
  - Différents RTOS...
  - Les objets *communs* d'un RTOS
  - Implémentation d'un RTOS
- Programmation avec un OS temps réel
- **Mise en Oeuvre**
  - **AvrX**

Mise en oeuvre

AvrX (1)

- Petit RTOS dédié :
  - Développé par une personne (Larry Barello)
  - Destiné à la famille AT90
  - Compilation avr-gcc et icc-avr
  - Principalement écrit en assembleur
  - Taille : environ 500 à 1000 octets selon les objets utilisés
- Mais déjà un vrai RTOS :
  - RTOS Pré-emptif, 255 niveaux de priorités
  - Nombre de tâche limité uniquement par la SRAM disponible
  - RTOS Fournissant tous les objets *communs* d'un RTOS + l'objet FIFO
  - Performance : pour le suivi d'un timer et un tick basé sur 1/1000 de l'oscillateur externe : AvrX utilise 20% du temps processeur

Conclusion

micro-contrôleur avec un RTOS

- Des concepts génériques...
  - Tâches, sémaphores, etc
- … mais mises en œuvre spécifiques

➔ **Bien lire les spécifications**

- Implémentation
  - Liée aux spécificités du matériel (mémoire, IT)
    - ➔ Une couche de portage spécifique par cible
  - Simple et prédictible en temps
    - ➔ Allocations statiques

➔ **Maitrise du HW, prédiction a priori**

- Choix selon l'utilisation
  - Critères : coût (environnement et royalties, formation des équipes de développement), accès au code, développements spécifiques, taille, prédictibilité, durée de vie du produit, qualité des fournisseurs

➔ **Phase à ne pas négliger**