Systèmes embarqués

M. Eleuldj, eleuldj@emi.ac.ma
Département génie informatique, EMI

Objectifs du cours

- Familiarisation avec:
  - Logiciels embarqués
  - Systèmes d'exploitation temps réels
  - Environnements de développement
  - Programmation de circuits FPGA
  - Conception ASIC
- Projets : Rapport (20 pages) + Exposé en binôme (15 mn) + Q/R
  - Système d'exploitation telps réel : μC OS
  - Plateforme J2ME
  - Consultation d'une base de données d'un téléphone mobile
  - Répondeur automatique d'un téléphone mobilele
  - Programmation d'un FPGA
  - CAO VLSI
- Référence
  - An Embedded Software Primer, D. E. Simon, Addison Wesley, 2003

Plan

I Introduction
II Interruptions
III Architectures logicielles
IV Systèmes d'exploitation temps réel
V Services supplémentaires des systèmes d'exploitation
VI Environnements de développement des applications embarquées
VII Architecture FPGA
VIII Introduction aux processeurs embarqués
IX Introduction aux mémoires embarquées
X Conception des circuits intégrés

Chapitre 1 : Introduction

- Evolution de la technologie
- Défis des systèmes embarqués
- Matériel typique

Evolution de la technologie

- Objectif de l'architecte des systèmes
  - augmentation du rapport (performance / coût)
- Progrès des circuits intégrés (VLSI)
  - plus denses (compact + rapide) et moins coûteux : 100 M tr/puce
  - plus d'implantations matérielles : ASIC, SoC, FPGA, …
- Codesign
  - conception d'une solution matérielle + logicielle (Hardware + Software)
- Exemples de systèmes embarqués
  - téléphones portables, imprimantes, récepteur satellitaires (set top box), répondeurs téléphoniques, téléviseurs numériques, montre numérique, ascenseur, air bag, jeux vidéo portables…
- Convergence des équipements
  - Télévision + Récepteur satellitaire + Internet + Enregistreur vidéo
  - Téléphone + Caméra + Télévision + Internet + Service à valeur ajoutée

Défis des systèmes embarqués (1/2)

- Débit
  - Traitement de beaucoup de données dans un temps court
- Temps de réponse
  - Réaction rapide aux événements (Algorithmes efficaces, structures de données, programmation,…)
- Testabilité
  - Sans écran, haut parleurs, voyant lumineux (non observabilité)
  - Erreur → arrêt du système
  - Difficulté de tester le système
- Mise au point (debugability)
  - Sans clavier, écran, … (non contrôlabilité + non observabilité)
  - Difficulté de la mise au point du système
- Fiabilité
  - Fonctionnement normal sans intervention humaine

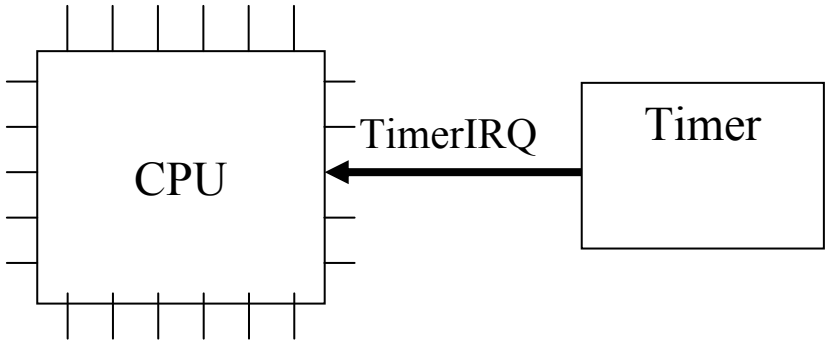
Défis des systèmes embarqués (2/2)

- Espace mémoire
  - Enregistrement du programme et données dans une mémoire limitée
- Programme d'installation
  - Nécessité d'environnement et/ou outils particuliers pour embarquer les logiciels et données
- Consommation d'énergie
  - Batterie (alimentation et poids) limitée pour les systèmes portables
- Charge du processeur
  - Traitement complexe → problème de temps de réponse
- Coût
  - Réduction du coût des systèmes embarqués (microprocesseurs, mémoires,…) → les logiciels s'exécutent sur du matériel à peine adéquat

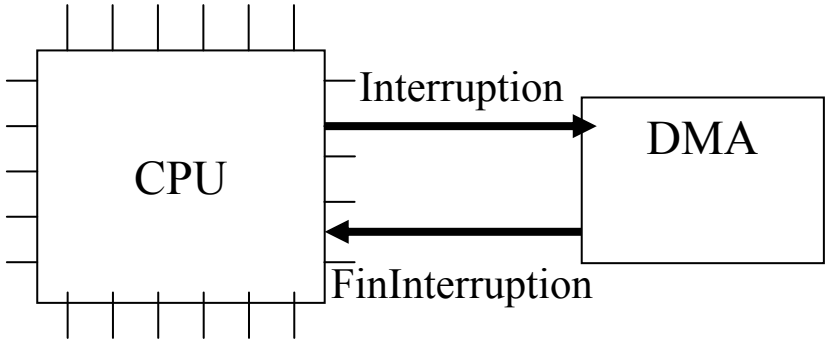
Matériel typique

Processeur	Largeur du bus (bits)	Mémoire externe	Périphériques Internes	Vitesse (MIPS)
Zilog Z8	8	64 KB	2 timers	1
Intel 8051	8	64 KB prog. + 64 KB données	3 timers + ports series	1
Zilog Z80	8	64 KB, 1 MB	Variables	2
Intel 80188	8	1 MB	3 timers + 2 DMA	2
Intel 80386	16	64 MB	3 timers +2 DMA + autres	5
Motorala 68000	32	4 GB	variables	10
Motorola PowerPC	32	64 MB	plusieurs	25

Chapitre 2 : Interruptions

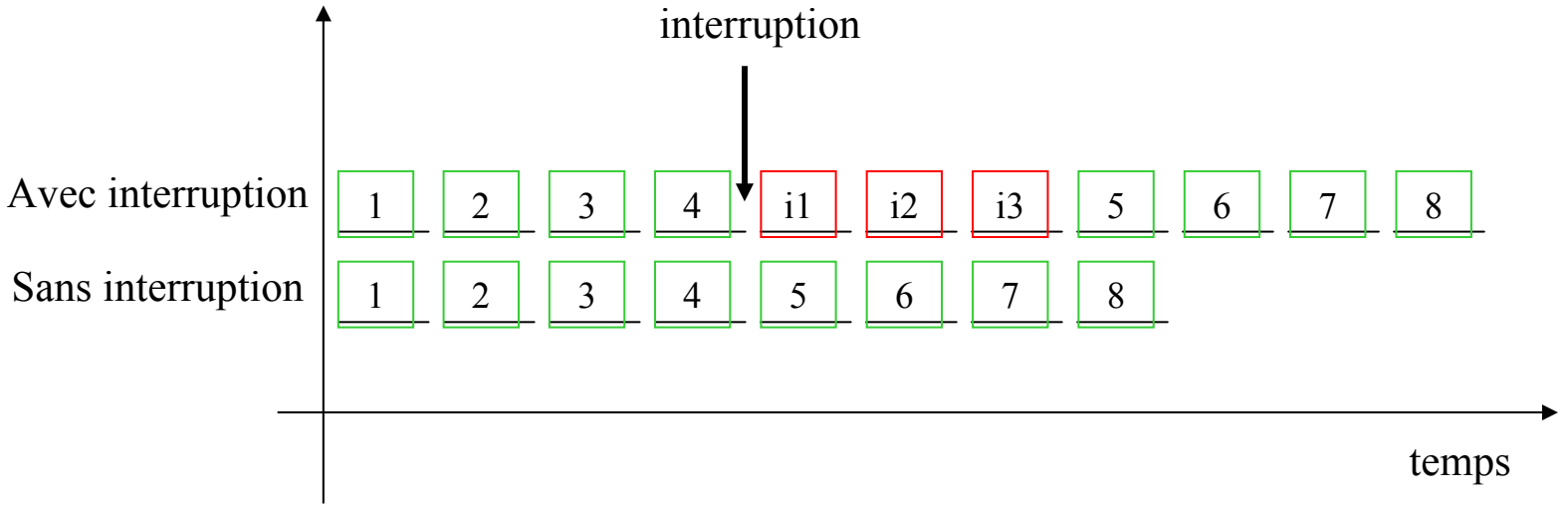


Monoprocesseur



parallélisme

Exemple d'interruption



Programme (tâche) est composé de 8 étapes : 1, 2,…, 8

Procédure d'interruption est composée de 3 étapes

e1 : sauvegarde du contexte

e2 : exécution de la procédure d'interruption

e3 : restauration du contexte

Interruption logicielle

- Interruption dans un programme
  - La majorité des compilateurs C des systèmes embarqués reservent le mot-clé interrupt pour la déclaration des procédures d'interruption
  - Le compilateur ajoute le code pour la sauvegarde et la restauration du contexte qui n'apparaît pas dans la déclaration
  - Le compilateur comprend une instruction pour inhiber (disable) et permettre (enable) les interruptions

- Exemple

```
void interrupt vHandleTimerIRQ (void)
{
        …
}
```

- Attention : Problèmes d'utilisation !!

Problème "classique“ de partage de données

```
static int iTemperatures[2];

void interrupt vReadTemperature (void)
{
    iTemperatures[0] = !!read in value from hardware
    iTemperatures[1] = !!read in value from hardware
}

void main (void)
{
    int iTemp0, iTemp1;
    while (TRUE)
    {
        iTemp0 = iTemperatures[0]                    /* 1 */
        iTemp1 = iTemperatures[1]                    /* 2 */
        if (iTemp0 != iTemp1)                        /* 3 */
            !! Set off howling alarme;
    }
}
```

Problème "difficile" de partage de données

```
static int iTemperatures[2];

void interrupt vReadTemperature (void)
{
    iTemperatures[0] = !!read in value from hardware
    iTemperatures[1] = !!read in value from hardware
}

void main (void)
{
    while (TRUE)
    {
        if (iTemperatures[0] != iTemperatures[1])
            !! Set off howling alarme;
    }
}
```

Atomicité d'une section

La condition (iTemperatures[0] != iTemperatures[1]) en langage C peut être traduite en langage d'assemblage par :

1 R1 ← iTemperatures[0]
2 R2 ← iTemperatures[1]
3 R3 ← R1 – R2
4 Retourner (R3 = 0)

où R1, R2 et R3 sont des registres

Question : Problème de partage de données est-il résolu ?

Définition : Une section (une ou plusieurs instructions) est atomique si elle ne peut pas être interrompue

Solution du problème de partage de données

```
static int iTemperatures[2];

void interrupt vReadTemperature (void)
{
    iTemperatures[0] = !!read in value from hardware
    iTemperatures[1] = !!read in value from hardware
}

void main (void)
{
    int iTemp0, iTemp1;
    while (TRUE)
    {
        disable ();   /* disable interrupts while we use the array */
        iTemp0 = iTemperatures[0]
        iTemp1 = iTemperatures[1]
        enable ();   /* enable interrupts */
        if (iTemp0 != iTemp1)
            !! Set off howling alarme;
    }
}
```

Exercice: Problème de partage de données?

```
static int iSeconds, iMinutes, iHours;

void interrupt vUpdateTime (void)
{   ++iSeconds;
    if (iSeconds >= 60)
    {       iSeconds = 0;
            ++iMinutes;
            if (iMinutes >= 60)
            {           iMinutes = 0;
                        ++iHours;
                        if (iHours >= 24)    iHours = 0;
            }
    }
    !! Do whatever needs to be done to the hardware
}

long lSecondsSinceMidnight (void)
{   return ((((iHours * 60) + iMinutes) *60) + iSeconds);
}
```

Latence d'interruption

- Interruption est déclenchée → procédure (ou fonction) d'interruption s'exécute
- Latence : temps nécessaire pour exécuter (répondre à) une interruption
- Réduction de la latence
  - Code de la procédure d'interruption doit être court (en langage d'assemblage)
  - S'arranger pour inhiber (Disable) les interruptions pour une courte période de temps

Chapitre 3: Architectures logicielles

- Systèmes embarqués
  - beaucoup d'événements
  - Traitements
  - Propriétés
  - Échéancier

- Architectures logicielles
  - A tour de rôle (Round Robin)
  - A tour de rôle avec interruptions
  - Queue des procédures d'interruptions
  - Système d'exploitation temps réel (RTOS)

A tour de rôle (Round-Robin Architecture)

```
void main (void)
{    while (TRUE)
     {       if (!! I/O Device A needs service)
             {           !! Take care of I/O Device A
                         !! Handle Data to or from I/O Device A
             }
             if (!! I/O Device B needs service)
             {           !! Take care of I/O Device B
                         !! Handle Data to or from I/O Device B
             }
             ...
             if (!! I/O Devise Z needs service)
             {           !! Take care of I/O Device Z
                         !! Handle Data to or from I/O Device Z
             }
     }
}
```

A tour de rôle (Caractéristiques)

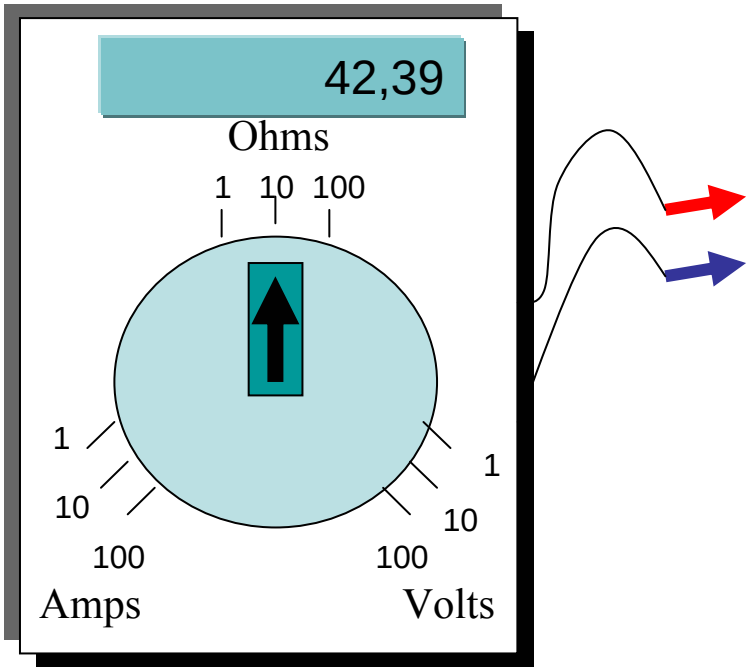
- Fonctionnement :
  - Vérifier les requêtes des organes (devices) d'E/S à tour de rôle (en boucle)
  - Répondre aux services

- Caractéristiques :
  - Architecture simple
  - Sans interruption
  - Sans latence

- Exemple : Système avec les organes A,B et C
  - temps d'exécution dans main pour A, B et C = 200 ms
  - Si C vient d'être exécuté alors C est moins prioritaire que A et B
  - Solution : modifier main en vérifiant les drapeaux dans l'ordre A, C, B, C, D, C.

- Adaptée aux systèmes dans lesquels il y a :
  - peu de traitement (calcul)
  - délai dû au temps de réponse n'est pas primordial

Exemple : Multimètre numérique



Afficher la mesure des 3 organes (résistance, courant et tension) selon : Ohms_1, Ohms_10, hms_100, Amps_1, Amps_10, Amps_100, Volts_1, Volts_10 ou Volts_100

A tour de rôle avec interruptions (1/2)

```
BOOL fDeviceA = FALSE;
BOOL fDeviceB = FALSE;
…
BOOL fDeviceZ = FALSE;

void interrupt vHandleDeviceA (void)
{    !! Take care of I/O Device A
     fDeviceA = TRUE;
}

void interrupt vHandleDeviceB (void)
{    !! Take care of I/O Device B
     fDeviceB = TRUE;
}
…
void interrupt vHandleDeviceZ (void)
{    !! Take care of I/O Device Z
     fDeviceZ = TRUE;
}
```

A tour de rôle avec interruptions (2/2)

```
void main (void)
{    while (TRUE)
     {       if (fDeviceA)
             {            fDeviceA = FALSE;
                          !! Handle Data to or from I/O Device A
             }
             if (fDeviceB)
             {             fDeviceB = FALSE;
                          !! Handle Data to or from I/O Device B
             }
             …
             if (fDeviseZ)
             {             fDeviceZ = FALSE;
                          !! Handle Data to or from I/O Device Z
             }
     }
}
```

A tour de rôle avec interruptions (Caractéristiques)

- Exemple : systèmes avec les organes A, B et C
  - temps d'exécution dans main pour A, B et C = 200 ms
  - les 3 déclanchent une interruption → pire que à tour de rôle
  - Si C vient d'être exécuté alors C est moins prioritaire que A et B
  - Solutions :
    - inclure le code de l'organe C dans la procédure d'interruption
    - modifier main en vérifiant les drapeaux dans l'ordre A, C, B, C, D, C.

- Caractéristiques :
  - Architecture qui reste simple
  - Avec latence

- Adaptée pour des organes dans lesquels il y a peu d'interruptions simlutanées et peu de traitement (lecture de code à barre optique, …)

Queue des procédures d'interruption (Function Queue Scheduling Architecture)

```
!! Queue of function pointers
void interrupt vHandleDeviceA (void)
{     !! Take care of I/O Device A
      !! Put function_A on queue of function pointers
}
void interrupt vHandleDeviceB (void)
{     !! Take care of I/O Device B
      !! Put function_B on queue of function pointers
}
void main (void)
{     while (TRUE)
      {           while (!! Queue of function pointers is empty)
                  …
                  !! Call first function on queue
      }
}
void function_A (void)
{     !! Handle actions required by device A
}
void function_B (void)
{     !! Handle actions required by device B
}
```

Queue des procédures d'interruptions (Caractéristiques)

- Fonctionnement :
  - procédure d'interruption ajoute un pointeur de fonction à la queue
  - Le "main" lit la queue et appelle les fonctions

- Caractéristiques :
  - les priorités peuvent être implantée pendant l'insertion dans la queue
  - le pire cas d'attente de la procédure d'interruption la plus prioritaire est la longueur du plus code le plus long des fonctions
  - les fonctions de faible priorité peuvent ne jamais s'exécuter

Système d'exploitation temps réel (RTOS Architecture)

```
void interrupt vHandleDeviceA (void)
{    !! Take care of I/O Device A
     !! Set signal X
}
void interrupt vHandleDeviceB (void)
{    !! Take care of I/O Device B
     !! Set signal Y
}
void Task1 (void)
{    while (TRUE)
     {       !! Wait for signal X
             !! Handle Data to or from I/O Device A
     }
}
void Task2 (void)
{    while (TRUE)
     {       !! Wait for signal Y
             !! Handle Data to or from I/O Device B
     }
}
```

Système d'exploitation temps réel (RTOS)

- RTOS : Real Time Operating System

- Caractéristiques :
  - Prise en charge des interactions entre les procédures d'interruptions et le code des tâches (utilisation de signals et pas de variables partagées)
  - Choix par le RTOS de la tâche urgente à exécuter à tout moment (pas de "main")
  - Suspension par le RTOS d'une tâche au milieu de son traitement et l'exécution d'une autre

- Conséquences
  - Temps de réponse du système stable
  - RTOS Overhead : Prend une partie du temps global de traitement

- Disponibilité
  - Plusieurs versions sont disponibles sur le marché (+ outil de mise au point : debug)

Caractéristiques des architectures

Caractéristique \\ Architectures	**Priorité**	**Temps de réponse en pire cas**	**Stabilité de la réponse quand le code change**	**simplicité**
A tour de rôle	aucune	Accumulation du codes des tâches	Faible	Très simple
A tour de rôle avec interruptions	procédures dans l'ordre de priorité + tâches même priorité	Temps de toutes les tâches + temps des procédures d'interruption	Bonne pour les procédures Faible pour les tâches	Problème des données partagées
Queue des procédures d'interruptions	procédures dans l'ordre de priorité + tâches dans l'ordre de priorité	Temps de la plus longue tâche + temps des procédures	Relativement bonne	Problème des données partagées + Code de gestion des priorités
Système d'exploitation temps réel (RTOS)	procédures dans l'ordre de priorité + tâches dans l'ordre de priorité	Temps de la plus longue tâche + temps des procédures	Très bonne	Très complexe (à l'intérieur du RTOS)

Chapitre 4 :
Introduction aux systèmes temps réel (RTOS)

- Tâche et états d'une tâche
- Tâche et données
- Sémaphores

SE et RTOS

SE ≠ RTOS

- SE (Système d'exploitation ou système opératoire)
  - Au démarrage, le SE prend le contrôle
  - Compilation + Edition des liens + Exécution du programme
  - Multiprogrammation, multi-utilisateurs
- RTOS (Système d'exploitation temps réel)
  - Edition des liens de l'application et du RTOS
  - Au démarrage, l'application prend le contrôle et démarre le RTOS
  - RTOS et l'application sont fortement couplés
  - Aucune protection vis-à-vis de l'application → meilleure performance
  - Services limités aux Systèmes embarqués → réduction de taille mémoire
  - Configuration du RTOS : gestion des fichiers, pilotes des E/S, gestion de la mémoire, Outils,…

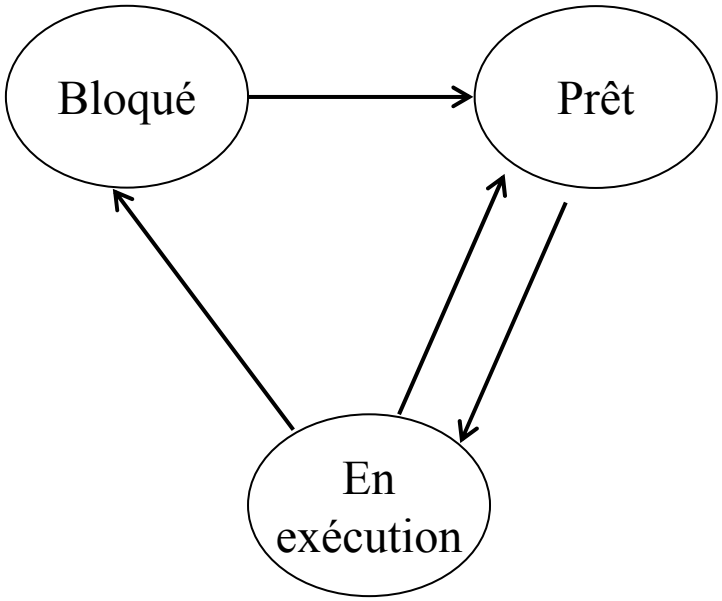
Exemples de RTOS

- Exemples
  - VxWorks, VTRX, pSOS, Nucleus, C Executive, Lynx OS, QNX, MultiTask!, AMX, μC/OS

- RTOS Spécifiques
  - Vitesse
  - Taille du code
  - Robustesse

- POSIX : Standard de IEEE pour une interface des systèmes d'exploitation temps réel

Tâche et états d'une tâche



• Tâche : procédure → processus léger (thread) dans un SE
• Etats : Bloqué, Prêt et En exécution
• Ordonnanceur : alloue le processeur en fonction de l'état, priorité et historique des tâches

Exemple : réservoirs d'une station d'essence

```
/* "Button Task" */
void  vButtonTask (void)  /* High priority */
{    while (TRUE)
     {        !! Block until user pushes a button
              !! Quick : respond to the user
     }
}

/* "Levels Task" */
void vLevelTask (void) /* Low priority */
{    while (TRUE)
     {        !! Read levels of floats in tank
              !! Calculate average float level
              !! Do some interminable calculation
              !! Do more interminable calculation
              !! Do yet more interminable calculation
              !! Figure out which tank to do next
     }
}
```

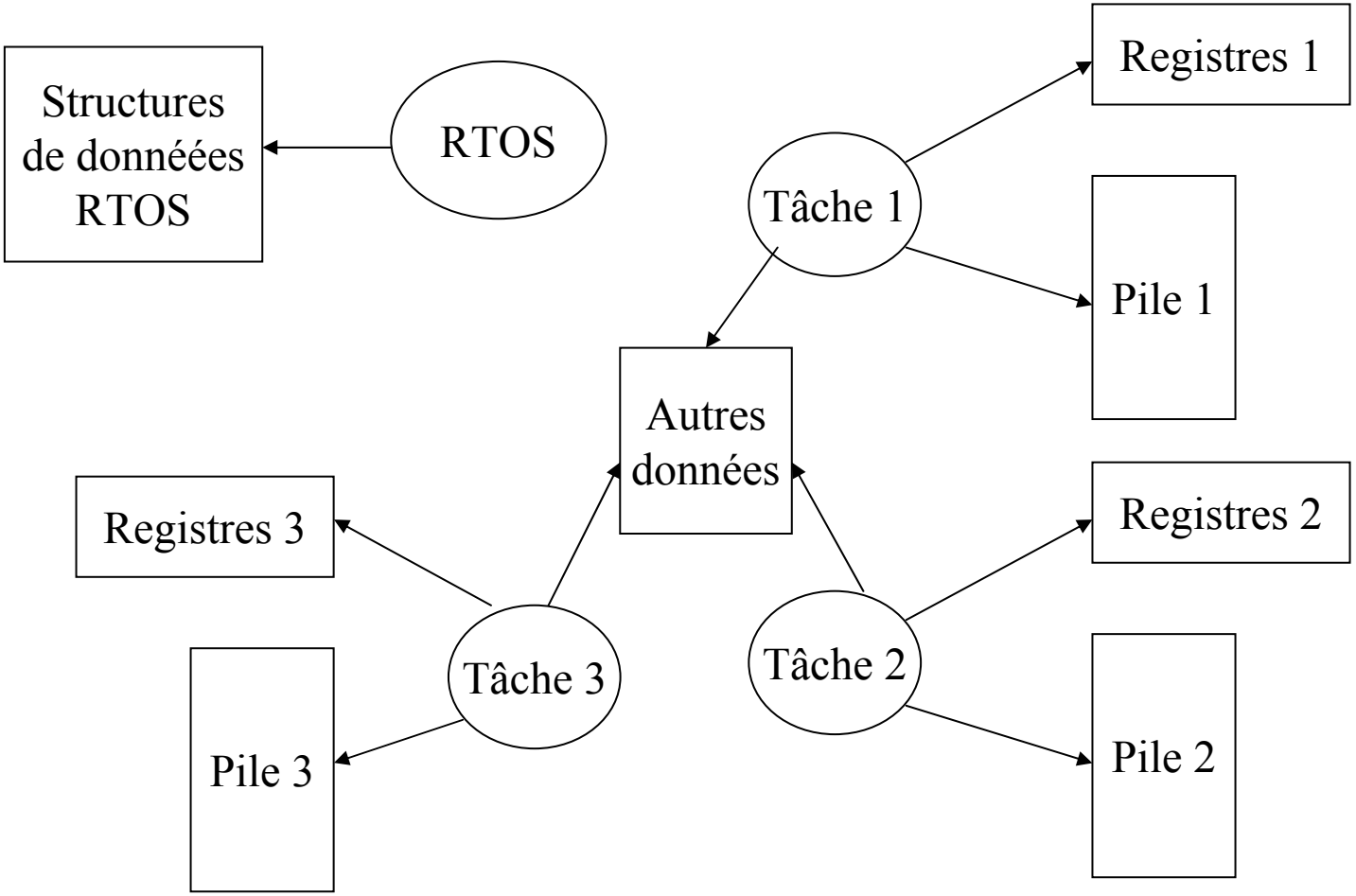
Code d'initialisation du RTOS

```
void  main (void)
{
    /* initilize (but do not start) the RTOS */
    initRTOS ();

    /* Tell the RTOS about our task */
    StarTask (vRespondToButton, HIGH_PRIORITY);
    StarTask (vCalculateTankLevels, LOW_PRIORITY);

    /* Start the RTOS (This function never return) */
    StartRTOS;
}
```

Les données dans un RTOS



Partage des données entre tâches

```
Struct
{     long     lTankLevel;
      long lTimeUpdated;
} tankdata[MAX_TANKS]
/* "Button Task “ */
void  vRespondToButton (void) /* High priority */
{     int i;
      while (TRUE)
      {          !! Block until user push a button
                 i = !! ID of button pressed
                 printf(“nTIME: %08ld   LEVEL: %08ld “,tankdata[i].lTimeUpdated, tankdata[i].lTankLevel);
      }
}
/* Levels Task */
void vCalculateTankLevel (void) /* Low priority */
{     i = 0;
      while (TRUE)
      {          !! Read levels of floats in tank
                 !! Do more interminable calculation
                 !! Do yet more interminable calculation
                 /* Store the result */
                 Tankdata[i].lTimeUpdated = !! Current time
                 Tankdata[i].lTankLevel = !! Result of calculation
                 !! Figure out which tank to do next
                 i = !! Something new
      }
}
```

Partage de code : fonction réentrante

```
void  Task1 (void)
{           .
            .
            vCountErrors (9);
            .
            .
}
void  Task2 (void)
{           .
            .
            vCountErrors (11);
            .
            .
}

Static int cErrors;
void  vCountErrors (int cNewErrors)
{           cErrors += cNewErrors;
}
```

Sémaphores

```
Struct
{     long     lTankLevel;
      long lTimeUpdated;
} tankdata[MAX_TANKS]
/* "Button Task “ */
void  vRespondToButton (void)  /*High priority */
{     int i;
      while (TRUE)
      {          !! Block until user push a button
                 i = !! ID of button pressed
                 TakeSemaphores ();
                 printf(“nTIME: %08ld   LEVEL: %08ld “,tankdata[i].lTimeUpdated,tankdata[i].lTankLevel);
                 ReleaseSemaphores ();
      }
}
/* Levels Task */
void vCalculateTankLevel (void) /* Low priority */
{     i = 0;
      while (TRUE)
      {          .
                 TakeSemaphores ();
                 !! Set tankDdata[i].lTimeUpdated
                 !! Set tankDdata[i].lTankLevel
                 ReleaseSemaphores ();
                 .
      }
}
```

Variantes des sémaphores

- Sémaphore de comptage (pris plusieurs fois)
- Sémaphores ressource (résolution du partage de données
- Sémaphore mutex (exclusion mutuelle: Mutexbegin et mutexend)

Chapitre 5
Services complémentaires des RTOS

- Communication inter tâches : queue, boite aux lettres (mailboxe) et tube (pipe)
- Temporisateur (Timer)
- Allocation mémoire
- Événements
- Interaction entre les procédures d'interruptions et le RTOS

Exemple d'utilisation d'une queue (1/2)

```
/* RTOS queue function prototypes */
void AddToQueue (int iDATA);
void ReadFromQueue (int *p_iDATA);

void Task1 (void)
{               .
                .
                if (!!problem arises)
                                vLogError (ERROR_TYPE_X);
                !! Other things that need to be done soon
   .
   .
   }
void Task2 (void)
{               .
                .
                if (!!problem arises)
                                vLogError (ERROR_TYPE_Y);
                !! Other things that need to be done soon
   .
   .
   }
```

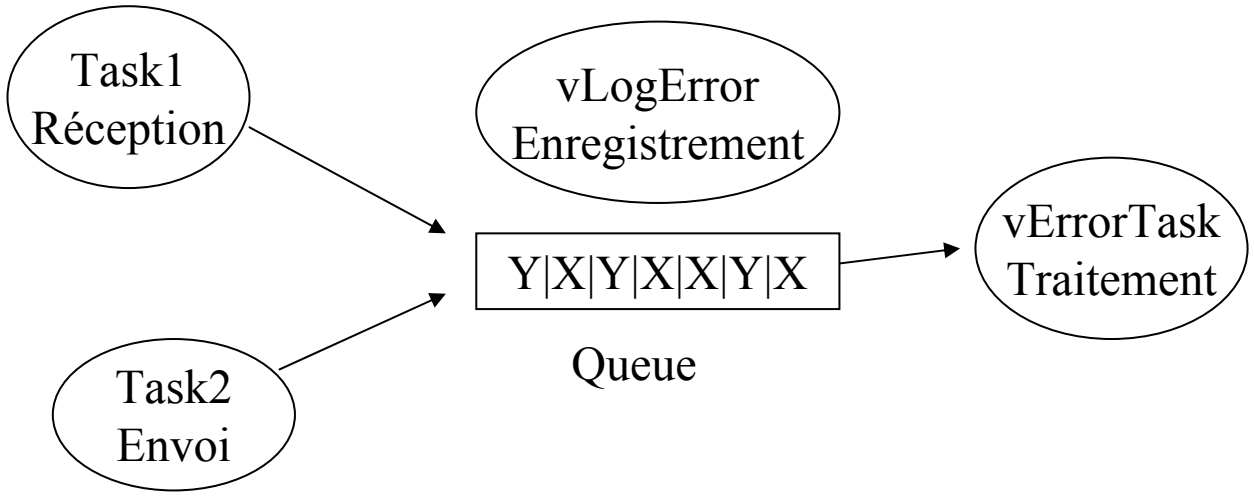
Exemple d'utilisation d'une queue (2/2)

```
void vLogError (int iErrorType)
{       AddToQueue (iErrorType);
}

Static int cErrors;

void ErrorsTask (void)
{       int iErrorType;
        while (FOREVER)
        {       ReadFromQueue (&iErrorType);
                ++cErrors;
                !! Send cErrors and iErrorType out on network
        }
}
```

Exemple : pont de communication (bridge)



Utilisation des queues

- Initialisation des queues avant le démarrage des tâches
- Allocation de la mémoire de la queue
- Spécification de l'identification de la queue dans le cas de plusieurs queues
- Si la queue est pleine, le RTOS peut retourner une erreur ou bloquer la tâche temporairement
- Si la queue est vide, la tâche de lecture retourne une erreur et bloque la tâche
- La queue doit contenir le nombre d'octets d'un pointeur de type void

Exemple complet d'une queue (1/3)

```
/* RTOS queue function prototypes */
OS_EVENT *OSQCreate (void **ppStart, BYTE bySize);
Unsigned char OSQPost(OS_EVENT,*pOse, WORD wTimeout, BYTE bysize);
void *OSQPend (OS_EVENT *pOse, WORD, WORD wTimeout, BYTE *pByErr);
#define WAIT_FOREVER 0

/* Our message queue */
Static OS_EVENT *pOseQueue;

/* The data space for our queue. The RTOS will manage this */
#define SIZEOF_QUEUE 25
void *apvQueue[SIZEOF_QUEUE];

void main (void)
{               .
                .
                /* the queue gets initialized before the tasks are started */
                pOsQueue = OSCreate (apvQueue, SIZEOF_QUEUE);
                .
                !! Start task1
                !! Start task2
                .                       .
                .
}
```

Exemple complet d'une queue (2/3)

```
void Task1 (void)
{           .
            .
            if (!!problem arises)
                        vLogError (ERROR_TYPE_X);
            !! Other things that need to be done soon
   .
   .
   }
void Task2 (void)
{           .
            .
            if (!!problem arises)
                        vLogError (ERROR_TYPE_Y);
            !! Other things that need to be done soon
   .
   .
   }
```

Exemple complet d'une queue (3/3)

```
void vLogError (int iErrorType)
{           BYTE byReturn;       /* Return code from writing to queue */
            /* write to the queue. Cast the error type as a void pointer to keepthe compiler happy */
            byReturn =  OSQPost (pOseQueue, (void *) iErrorType);
            if (byReturn != OS_NO_ERR)
                        !! Handle the situation that arises when the queue is full
}
Static int cErrors
void ErrorsTask (void)
{           int iErrorType;
            BYTE byErr;
            while (FOREVER)
            {           :*Cast the value received from the queue back to an int.
                        (Note that there is no possible error from this. So we ignore byErr) */
                        iErrorType) = (int) OSQPend(pOsQueue, WAIT_FOREVER, &byErr);
                        ++cErrors;
                        !! Send cErrors and iErrorType out on network
            }
}
```

Utilisation des boites aux lettres

- Plusieurs RTOS permettent un seul message à la fois
- Si c'est possible d'avoir plusieurs messages alors leur nombre est limité
- Certains RTOS permettent d'avoir plusieurs boites aux lettres
- Certains RTOS permettent de définir une priorité entre les boites aux lettres (→ entre les messages)

Utilisation des tubes

- RTOS peut créer des tubes (pipe), y écrire, les lire,…
- Longueur des messages fixe ou variable.
- Tubes sont "byte oriented“.
  - Si Tâche A écrit 11 octets et la Tâche B écrit 19 octets et si Tâche C lit 14 octets alors 11 seront de la Tâche A et 5 de la Tâche B.
- Utilisation des fonctions standard de C (fread et fwrite) pour lire et écrie dans les tubes.

Pièges

- Aucune restriction de lecture et écriture des queues, boite aux lettres et pipes → faire attention.
- Aucune garantie que les données écrites (queue, boite aux lettres ou tube) sont identifiées proprement au moment de la lecture.
- Manque d'espace (queue, boite aux lettres ou tube) est désastreux (→ réservation d'espace suffisant).
- Passage de pointeur d'une tâche à l'autre via une queue, boite aux lettres ou tube crée des données partagées.

Fonctions du temporisateur

- Nécessité d'un temporisateur (Timer) dans les systèmes embarqués (trace du temps écoulé).
- Initialisation du top d'horloge au moment de la configuration.
- Top d'horloge court → exactitude et interruption du processeur fréquemment.
- Top d'horloge lent → interruption peu fréquente du processeur

Fonction délai du RTOS: touches du tél.

```
/* Message queue for phone numberd to dial */
Extern MSG_Q_ID queuePhoneCall;
Void vMakePhoneCall(void)
{       #define MAX_PHONE_NUMBER 11
        Char a_chPhoneNumber[MAX_PHONE_NUMBER]; /*Buffer fo null-terminated ASCII number*/
        Char *p_chPhoneNumber;      /* Pointer into a-chPhoneNumber */
        .
        .
        While (TRUE)
        {       msgQreceive(queuePhoneCall, a_chPhoneNumber,
                                        MAX_PHONE_NUMBER, WAIT_FOREVER);
                /* dial each of th digits */
                p_chPhoneNumber = a_chPhoneNumber;
                while(*p_chPhoneNumber)
                {       taskDelay(100);             /* 1/10th of a second silence */
                        vDialingToneOn(*p_chPhoneNumber – '0');
                        taskDelay(100);             /* 1/10th of a second with tone */
                        vDialingToneOff(');
                        /* Go to the next digital in the phone number */
                        ++p_chPhoneNumber;
                }
        }
}
```

Evénements

- Gestion des événements est un service des RTOS
- Événement : drapeau booléen (flag) initialisé par une tâche et d'autres tâches peuvent être bloquées en l'attendant.
- Plus d'une tâche peut être bloquée en attendant un événement.
- Les RTOS permettent de former un groupe d'événements. Une tâche peut attendre un sous ensemble de ce groupe.
- Certains RTOS initialisent (reset) les événements automatiquement dans d'autres ce sont les tâches qui s'en chargent.

Fonctions des événements dans AMX

- ajevcre (AMXID *p_amxidGroup, unsigned int uValueInit, char *p_chTag)
  Crée un groupe de 16 événements identifié par p_amxidGroup
  uValInit : contient les valeurs initiales des événements (set ou resset)
  p_chTag : nom de 4 octets utilisé lorsque p_amxidGroup n'est pas accessible

- ajevsig (AMXID *p_amxidGroup, unsigned int uMask, unsigned int uValueNew)
  initialise les événements (set ou resset) dans le groupe p_amxidGroup
  uMask : événements qui doivent être initialisés
  uValueNew : nouvelles valeurs des événements

- ajevwat (AMXID *p_amxidGroup, unsigned int uMask, unsigned int uMask, int uValue, long iTime out)
  permet à une tâche d'attendre 1 ou plusieurs événements dans le groupe
  uMask : événements à attendre
  uValue : valeurs des événements à attendre
  iMatch : tâches à débloquer quand tous les événements de uMask ont atteint uValue
  lTimeout : temps maximal d'attente

Utilisation des événements 1/2

```
/* Handle for the trigger group of eventd */
AMXID amxidTrigger;
/* Constants for use in the group */
#define TRIGGER_MASK    0x001;
#define TRIGGER_SET     0x001;
#define TRIGGER_RESET   0x000;
#define KEY_MASK        0x002;
#define KEY_SET         0x002;
#define KEY_RESET       0x000;
void main (void)
{       ...
        /* Create an event group with the trigger and keyboard events reset */
        ajevcre (&amxidTrigger, TRIGGER_MASK, TRIGGER_SET);
        …
}
void interrupt vTriggerISR (void)
{       /* The user pulled the trigger. Set the event */
        ajevsig (amxidTrigger, TRIGGER_MASK, TRIGGER_SET);
        }
void interrupt vKeyISR (void)
{       /* The user pressed a key. Set the event */
        ajevsig (amxidTrigger, KEY_MASK, KEY_SET);
        !! Figure out which key the user pressed and store that value
}
```

Utilisation des événements 2/2

```
void vScanTask (void)
{       …
        While (TRUE)
        {       /* wait for the user to pull the trigger */
                ajevwat (amxidTrigger, TRIGGER_MASK, TRIGGER_SET,
                                                WAIT_FOR_ANY, WAIT_FOREVER);
                /* Reset the trigger event */
                ajevsig (amxidTrigger, TRIGGER_MASK, TRIGGER_RESET);
                !!Turn on the scanner hardware and look for a scan
                …
                !! When the scan has been found, turn off the scanner
        }
}
 void vRadioTask (void)
{       …
        While (TRUE)
        {       /* wait for the user to pull the trigger or press a key*/
                ajevwat (amxidTrigger, TRIGGER_MASK | KEY_MASK, TRIGGER_SET | KEY_SET,
                                                WAIT_FOR_ANY, WAIT_FOREVER);
                /* Reset the key event (the trigger event will be reset by the ScanTask) */
                ajevsig (amxidTrigger, KEY_MASK, KEY_RESET);
                !!Turn on the radio
                …
                !! When the scan has been sent,, turn off the radior
        }
}
```

Allocation de la mémoire

- Fonctions C : malloc et free

Pool (réservoir)

Buffer1|Buffer2|……|Buffer n

Fonctions RTOS : Reqbuf, getbuf et Relbuf(release buffer)

Allocation de la mémoire (1/2)

```
#define LINE_POOL                   1;
#define MAX_LINE_LENGTH             40;
#define MAX_LINES                   80;
Static char a_lines[MAX_LINES][MAX_LINE_LENGTH];

void main (void)
{       .
        .
        init_mem_pool LINE_POOL, a_lines, MAX_LINES, MAX_LINE_LENGTH, TASK_POOL);
        .
        .
}
void vPrintFormat (void)
{       char *p_chLine;     /* Pointer to current line */
        …
        /* Format lines and send them to the vPrintOutputTask */
        p_chLine = getbuf(LINE_POOL, WAIT_FOREVER);
        sprintf(p_chLine, "INVENTORY REPORT ");
        sndmsg(PRINT_MBOX, pch_Line, PRIORITY_NORMAL);
        p_chLine = getbuf(LINE_POOL, WAIT_FOREVER);
        sprintf(p_chLine, "Date: %02/%02/%02, iMonth, iDay, iYear% 100);
        sndmsg(PRINT_MBOX, pch_Line, PRIORITY_NORMAL);
         p_chLine = getbuf(LINE_POOL, WAIT_FOREVER);
        sprintf(p_chLine, "Time: %02:%02, iHour, iMinute);
        sndmsg(PRINT_MBOX, pch_Line, PRIORITY_NORMAL);
        …
}
```

Allocation de la mémoire (2/2)

```
        sndmsg
        p_chLine =
        sprint
        sndmsg
        .
        .
}
void vPrintOutputTsk (void)
{       char *p_chLine;         /* Pointer to current line */
        While (TRUE)
        {       /* wait for a line too come in */
                p_chLine = rcvmsg (PRINT_MBOX, WAIT_FOREVER);

                !! Do what is needed to send the line to the printer

                /* Free the buffer back to the pool */
                relbuf (LINE_POOL, p_chLine);
        }
}
```

Procédures d'interruption dans les RTOS

```
/* Handle for the trigger group of eventd */
AMXID amxidTrigger;
Void vMakePhoneCall(void)
{       #define MAX_PHONE_NUMBER 11
        Char a_chPhoneNumber[MAX_PHONE_NUMBER]; /*Buffer fo null-terminated ASCII number*/
        Char *p_chPhoneNumber;      /* Pointer into a-chPhoneNumber */
        …
        While (TRUE)
        {       msgQreceive(queuePhoneCall, a_chPhoneNumber,
                                                MAX_PHONE_NUMBER, WAIT_FOREVER);
                /* dial each of th digits */
                p_chPhoneNumber = a_chPhoneNumber;
                while(*p_chPhoneNumber)
                {       taskDelay(100); /* 1/10th of a second silence */
                        vDialingToneOn(*p_chPhoneNumber – '0');
                        taskDelay(100); /* 1/10th of a second with tone */
                        vDialingToneOff(');
                        /* Go to the next digital in the phone number */
                        ++p_chPhoneNumber;
                }
        }
}
```

Chapitre 6
Outils de développement des systèmes embarqués

- Machine hôte et machine cible
- Editeur des liens/Localiseur
- Chargement d'une application embarquée

Machine hôte et cible

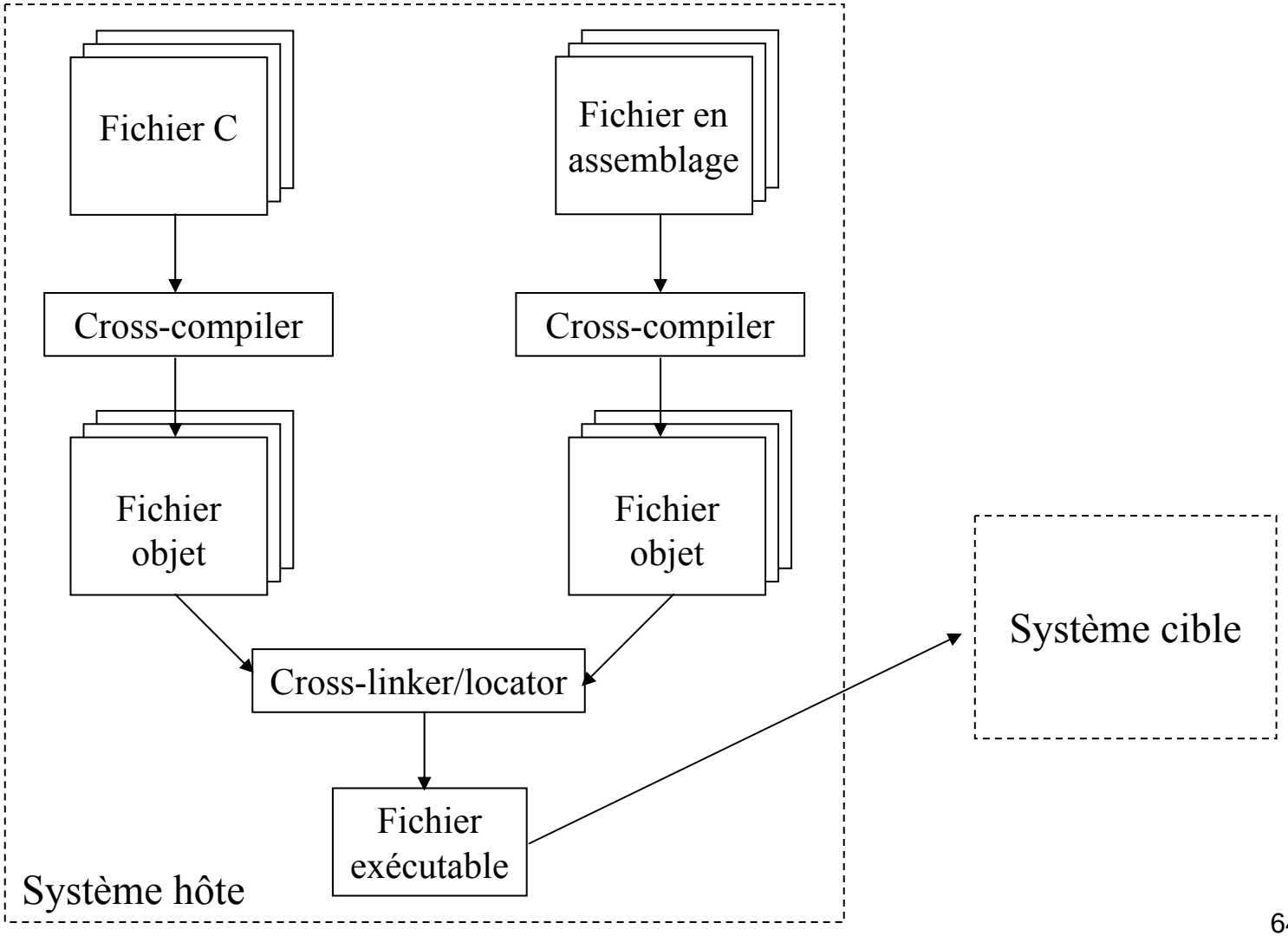
Chargement

Hôte (Intel Pentium + Outils)

Cible (Z80, M68000 ou MIPS)

- Outils natifs : compilateur + assembleur + éditeur de liens + chargeur
- Outils de croisement (Cross tools) : Cross-compiler + cross-assembler + Cross-linker/locator

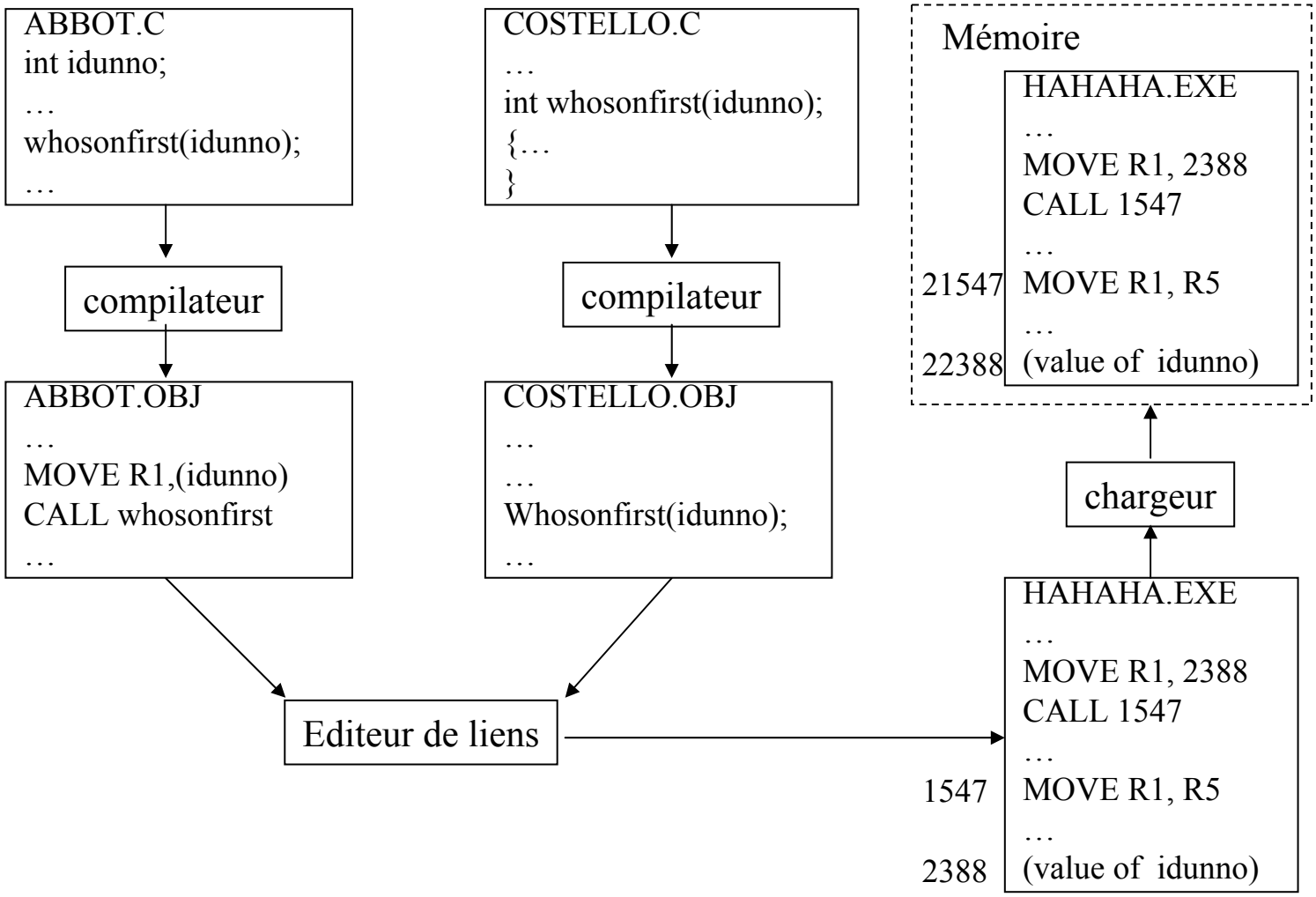
Chaîne d'outils de développement



Problèmes

- Portabilité de C : certaines fonctions ne sont pas reconnues par le système cible
- Exactitude du simulateur ou émulateur
- Différences entre un éditeur de liens dans un système classique (natif) et un éditeur de liens dans un système embarqué appelé Linker/locator :
  - Résolution des adresses
  - Localisation des composants du programme
  - Données initialisées et chaînes constantes
  - Tables d'allocation de la mémoire (Locator Maps)
  - Exécution dans la RAM

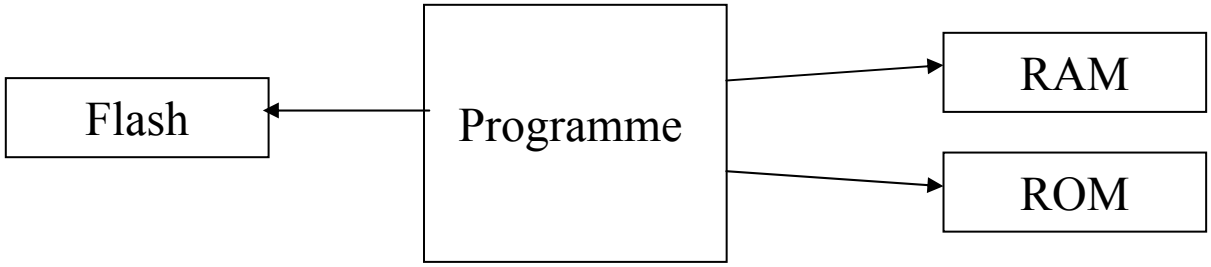
Chaîne d'outils natifs de développement



Résolution d'adresses

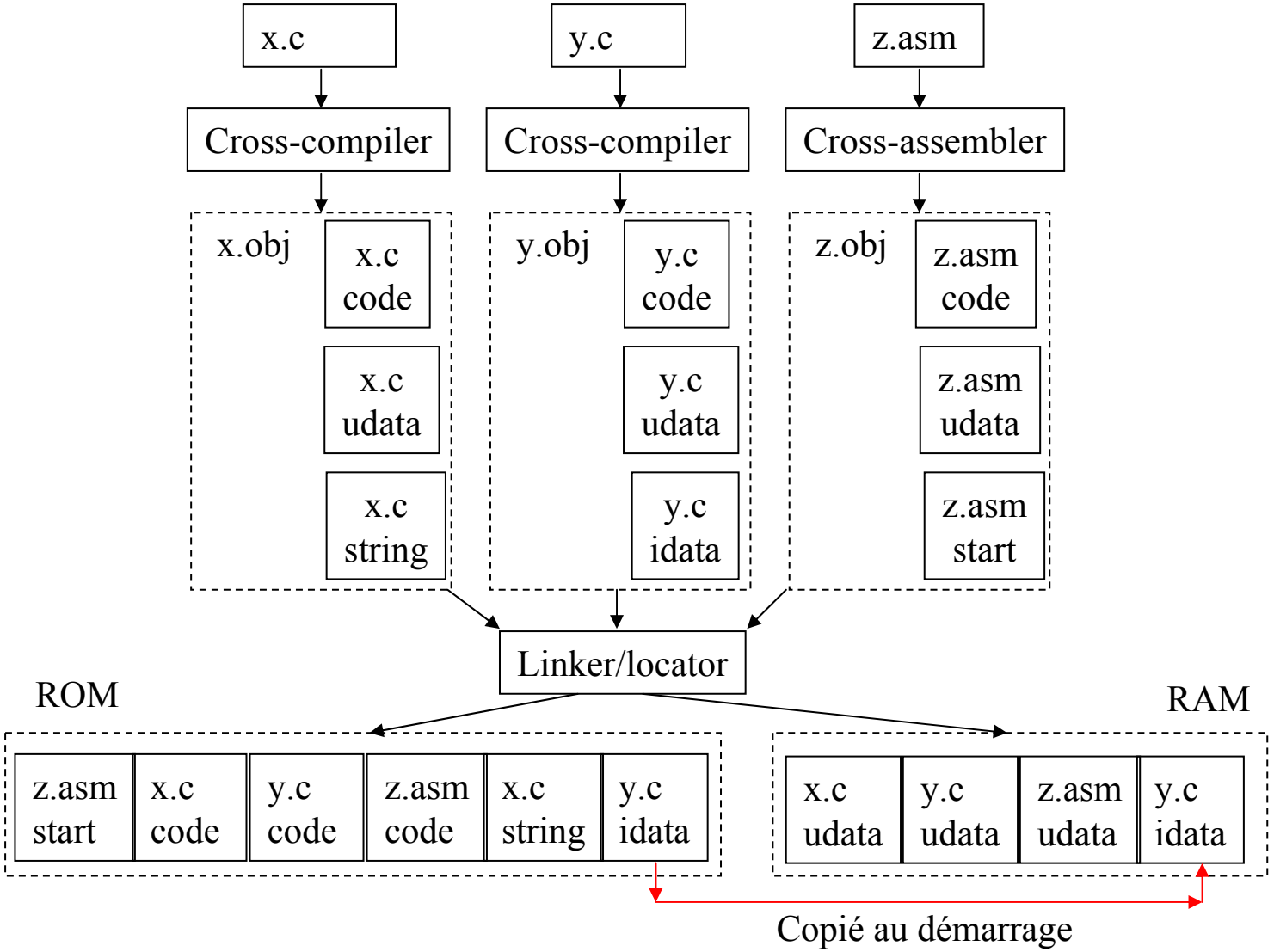
- Allocation de la mémoire (adresse) aux identificateurs de variables, procédures ou fonctions, constantes,…
- Plusieurs types de mémoires : RAM, ROM,…
- Il n'existe pas de chargeur (loader) dans les systèmes embarqués

Allocation de la mémoire au programme



- Décomposition du programme en segments (composants) :
  - Instructions (code)
  - Données initialisées (idata)
  - Données non initialisées (udata)
  - Chaînes constantes (string)
- Spécification de l'adresse du démarrage du programme (start)
- Spécification des zones de la RAM ou ROM utilisées
- Adresse de fin d'un segment (une pile par exemple)
- Créer un groupe de segments et affecter des zones par groupe

Exemple d'allocation de la mémoire



Données initialisées et chaîne constante

```
//Donnée initialisée
#define FREQ_DEFAULT 2410
…
static int iFreq = FREQ_DEFAULT;
…
Void sSetFreq(int iFresNew)
{
     iFreq = iFreqNew);
}

//Chaîne constante
char *sMsg = "Reactor is melting!";
printf(" Problem : %s ", sMsg);
strcpy(&sMsg[11]], "OK");
```

Tables du localiseur (Locator Maps)

Fichier de sortie du linker/locator contenant :

- Adresses des segments dans les différentes mémoires (RAM, ROM,…)
- Adresses des fonctions prédéfinies
- Adresses des données

→ vérification de l'emplacement (RAM ou ROM) au moment de la mise au point (debugging)

Réduction du temps d'exécution

- La RAM est plus rapide que la ROM et la mémoire Flash.
- Si le microprocesseur est rapide (exemple RISC), le programme est d'abord chargé dans la ROM et au moment du démarrage, il est copié dans la RAM et exécuté.

Chargement dans le système cible

- PROM : Utilisation dans le cas :
  - Stade de développement
  - Volume est peu élevé
  - Mises à jour sont planifiées
- Emulateur ROM
  - Organe (device) matériel qui remplace la mémoire ROM
- Mémoire Flash
  - Mise à jour des programmes (versions) lorsque le système est chez lz client
  - Plus lente que la RAM
  - Nombre d'accès est limité à environ 10000
  - Unité de transfert est 1K octets
- Moniteur (Monitor)
  - Programme résident dans la ROM cible permettant de charger de nouveau programmes à travers un port de communication