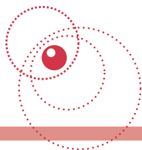


Systèmes d'exploitation embarqués

ELEC 344

Février 2010

Samuel TARDIEU <sam@rfc1149.net>

Licence de droits d'usage

TELECOM ParisTech

Plan

Introduction

Gestion de la mémoire

Gestion de la concurrence

Systèmes de fichiers

Quelques OS embarqués

FreeRTOS

Licence de droits d'usage

Samuel TARDIEU <sam@rfc1149.net>

TELECOM ParisTech

Système d’exploitation

Un système d’exploitation :

- fait le lien entre le logiciel (application) et le matériel ;
- abstrait certaines caractéristiques du matériel ;
- fournit des services communs (accès aux ressources, synchronisation, gestion de fichier) ;
- offre des possibilités de tests et de traces.

Licence de droits d’usage

Système embarqué

Généralement, un système embarqué :

- dispose de ressources limitées ;
- ne possède pas nécessairement de système de fichiers ;
- doit être le moins cher possible ;
- ne doit pas consommer d'énergie inutilement (batterie).

Licence de droits d'usage

Samuel Tardieu <sam@rfc1149.net>

Un OS est-il obligatoire ?

Absolument pas !

- Certains langages sont un OS à eux tout seul (Forth).
- Certains langages incluent des options de concurrence avancées (Ada).

Licence de droits d'usage

Samuel Tardieu <sam@rfc1149.net>

Un OS est-il utile ?

Absolument !

- La plupart des programmes embarqués ont des contraintes similaires (temps-réel, USB, TCP/IP, fichiers).
- Il est plus facile d'utiliser une API portable plutôt que de recoder des fonctionalités de base.
- Pourquoi réinventer la roue systématiquement ?

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

Plan

Introduction

Gestion de la mémoire

Gestion de la concurrence

Systèmes de fichiers

Quelques OS embarqués

FreeRTOS

Licence de droits d'usage

Samuel TARDIEU <sam@rfc1149.net>

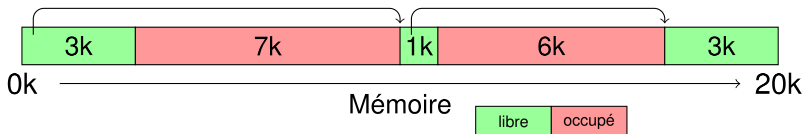
Gestion de la mémoire

- La mémoire est une ressource précieuse.
- La plupart des microcontrôleurs embarquent quelques k de mémoire :
  - 20k sur les STM32F103 ;
  - 1536 octets sur les PIC18F452.
- Ajouter de la RAM externe est coûteux :
  - utilisation d’entrées-sorties supplémentaires sur le processeur ;
  - complication du routage ;
  - intégrité du signal.

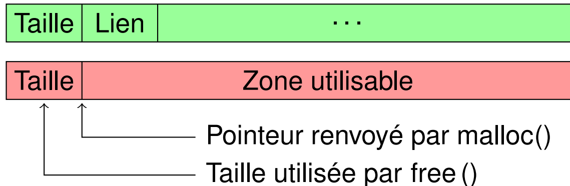
Licence de droits d’usage
Samuel Tardieu <sam@rfc1149.net>

Gestion dynamique de la mémoire

- Utilisation d'une liste chaînée des blocs libres (*free-list*)



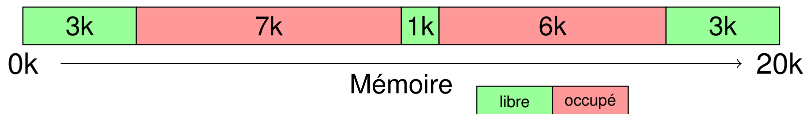
- Stockage de la taille réservée en mémoire (la taille libérée n'est pas passée à free ())



Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

Fragmentation

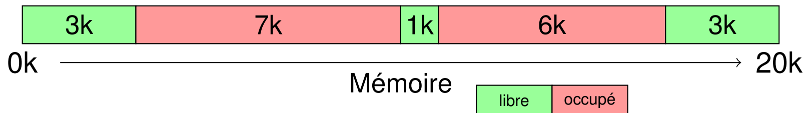
Au cours de son utilisation, la mémoire disponible peut devenir fragmentée.



Comment allouer 5k alors que seuls deux blocs non contigus de 3k et un de 1k sont disponibles ?

Politiques d’allocation

Dans la situation suivante, dans quel bloc allouer une zone de 600 octets demandée par le programme ?



Plusieurs stratégies possibles :

- *Best fit*
- *Worst fit*
- *First fit*
- *First fit* équivalent à une des deux premières solutions

Licence de droits d’usage
Samuel Tardieu <sam@rfc1149.net>

Gestion de la libération

Plusieurs stratégies possibles :

- Agrégation des blocs libres, peut nécessiter un tri de la liste ; peu déterministe.
- Libération sans agrégation des blocs libres, peut nécessiter un tri de la liste.
- Pas de libération.

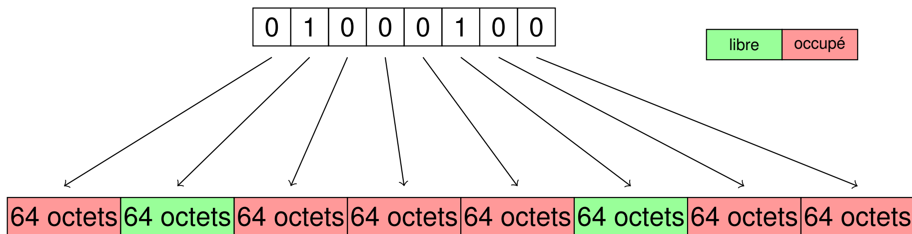
Toutes ces stratégies sont couramment utilisées. La dernière permet l'allocation dynamique en début de programme, qui ne commencera ses véritables fonctions qu'après que l'ensemble des allocations aient été effectuées.

Licence de droits d'usage
Samuel Tardieu <sam@rfc1149.net>

Gestion par *bitmaps*

La mémoire peut-être gérée avec des *bitmaps* :

- blocs de taille fixe et contigus ;
- un bit par bloc indique si le bloc est libre ou non ;
- possibilité d'utiliser plusieurs zones avec des blocs de taille différente.



Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

Recherche dans un *bitmap*

Comment, dans un mot machine, isoler un bit quelconque parmi les bits à 1 en O(1) ?

0	1	0	0	0	1	0	0

Licence de droits d'usage

Samuel TARDIEU <sam@rfc1149.net>

Recherche dans un *bitmap*

Comment, dans un mot machine, isoler un bit quelconque parmi les bits à 1 en O(1) ?

0	1	0	0	0	1	0	0

```
inline unsigned int lowest_bit(unsigned int word)
{
  return ((word ^ (word - 1)) >> 1) + 1;
}
```

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

Recherche dans un *bitmap*

Comment, dans un mot machine, isoler un bit quelconque parmi les bits à 1 en O(1) ?

0	1	0	0	0	1	0	0

```
inline unsigned int lowest_bit(unsigned int word)
{
  return ((word ^ (word - 1)) >> 1) + 1;
}
```

0	1	0	0	0	1	0	0	word
0	1	0	0	0	0	1	1	word – 1
0	0	0	0	0	1	1	1	^
0	0	0	0	0	0	1	1	>> 1
0	0	0	0	0	1	0	0	+ 1

Licence de droits d'usage

Samuel Tardieu <sam@rfc1149.net>

Allocation statique

L'absence d'allocation dynamique a des avantages :

- détermination de la position définitive de chaque bloc lors de l'édition de liens et temps d'accès réduit ;
- vérification de la disponibilité de la quantité nécessaire de mémoire lors de l'édition de liens ;
- aucune possibilité de fragmentation ou de manque de mémoire lors de l'exécution.

Cette solution doit être privilégiée lorsque c'est possible.

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

Utilisation de la MMU

L'utilisation d'une MMU (*Memory Management Unit*) permet :

- la protection des zones mémoire ;
- la réduction de la fragmentation par l'utilisation de pages et de la correspondance entre adresse logique et adresse physique ;
- la possibilité de disposer de zones *thread-local* sans indirection supplémentaire.

Licence de droits d'usage

Samuel TARDIEU <sam@rfc1149.net>

Plan

Introduction

Gestion de la mémoire

Gestion de la concurrence

Systèmes de fichiers

Quelques OS embarqués

FreeRTOS

Licence de droits d'usage

Samuel TARDIEU <sam@rfc1149.net>

Architectures et parallélisme

On trouve plusieurs types de systèmes embarqués :

- mono-processeur ;
- SMP (*symetric multiprocessing*) ;
- NUMA (*non-uniform memory architecture*).

Même dans les architectures mono-processeur, on souhaite souvent exécuter plusieurs activités de manière concurrente.

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

Boucle principale

Le schéma le plus simple est celui de la boucle principale :

```
void tache_1()
{
  if (capteur_1_actif())
    reagir_a_capteur_1();
}

...

int main()
{
  for (;;) {
    tache_1();
    ...
    tache_n();
  }
}
```

Inconvénients :

- gaspillage des ressources par l'utilisation systématique du *polling* ;
- pas de gestion de priorité ou de fréquence relative.

Licence de droits d'usage

Samuel TARDIEU <sam@rfc1149.net>

TELECOM ParisTech

Boucle événementielle

On peut également attendre un événement signalé par le matériel :

```
int main()
{
  for (;;) {
    switch (wait_for_event()) {
      case capteur_1:
        reagir_a_capteur_1();
        break;
      ...
      case capteur_n:
        reagir_a_capteur_n();
        break;
      }
    }
}
```

Inconvénients :

- pas de prioritisation des événements ;
- pas de possibilité de faire de longs calculs sans bloquer la gestion des autres événements.

Licence de droits d'usage

Samuel TARDIEU <sam@rfc1149.net>

Fonctionnement sur interruptions

On peut avoir :

- une tâche principale qui s'exécute en permanence ;
- des routines lancées lorsqu'une interruption survient.

C'est ainsi que fonctionnent des systèmes mono-tâche pour « émuler » le multi-tâches (MS-DOS par exemple, sur interruption d'horloge).

Inconvénients :

- une seule tâche principale.

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

Changement de contexte

On souhaiterait pouvoir :

- sauvegarder l’état d’un traitement donné pour en effectuer un autre et retrouver son état ;
- effectuer des traitements longs en donnant leur chance aux autres tâches à réaliser.

Pour cela, chaque tâche doit disposer de son contexte :

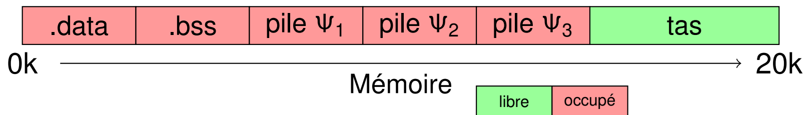
- l’état des registres du processeur dont le pointeur ordinal (*program counter*) et le pointeur de pile (*stack pointer*) ;
- la pile d’appel des sous-programmes.

Licence de droits d’usage
Samuel TARDIEU <sam@rfc1149.net>

Allocation de la pile

Dans une architecture multi-tâches, chaque tache Ψ_i nécessite une pile d'exécution. L'allocation de cette pile est critique :

- trop petite, elle causera une corruption mémoire ;
- trop grande, elle consommera trop de mémoire.



La taille à choisir dépend des profondeurs d'appel de chaque tâche et de la taille occupée par les variables locales.

Licence de droits d'usage

Samuel TARDIEU <sam@rfc1149.net>

Synchronisation

L'utilisation d'un système multi-tâches peut poser des problèmes de synchronisation :

```
void transfert(compte origine, compte destination, unsigned int montant)
{
  if (origine.solde >= montant) {
    /* (1) */
    origine.solde      -= montant;
    /* (2) */
    destination.solde += montant;
  }
}
```

Si cette routine est appelée deux fois avec les mêmes paramètres simultanément et que le changement de contexte se fait en (1), la vérification peut s'avérer incorrecte. Si une autre tâche effectue un total des soldes alors que le changement de contexte s'est fait en (2), le total sera inférieur à ce qu'il devrait être.

Licence de droits d'usage
Samuel Tardieu <sam@rfc1149.net>

Coroutines

Les coroutines sont un modèle de parallélisme coopératif :

- chaque tâche indique (avec `yield ()`) lorsqu'elle accepte de donner la main à une autre ;
- les problèmes de synchronisation n'existent pas, les points de synchronisation étant placés par l'utilisateur.

Certains modèles de coroutines sans pile existent :

- la consommation mémoire est réduite ;
- les variables locales sont interdites ;
- `yield ()` ne peut être appelé que depuis le sous-programme principal de la coroutine ;
- sur certaines architectures où la pile est située à un endroit fixe (PIC18F), le changement de contexte est bien plus rapide.

Licence de droits d'usage

Samuel Tardieu <sam@rfc1149.net>

Priorités

Les priorités permettent de choisir la prochaine tâche à exécuter :

- les priorités peuvent être statiques ou dynamiques ;
- le temps maximum entre deux changements de contexte indique le temps de réaction à une condition extérieure ;
- on peut créer une *idle task* de priorité minimale qui met le processeur en veille en attendant qu'un événement modifie le système ;
- cette même *idle task* peut également effectuer des opérations de nettoyage (agrégation des blocs mémoire libres) ;
- le passage régulier dans l'*idle task* peut indiquer une non-surcharge du système et servir à signaler un *watchdog*.

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

Systèmes préemptifs

Un système multi-tâches est dit « préemptif » lorsqu'une tâche peut être interrompue sans l'avoir elle-même demandé :

- réaction plus rapide à des événements asynchrones ;
- réveil d'une tâche plus prioritaire à l'expiration d'un délai fixé (alarme) ;
- possibilité ou non de *round-robin* entre des tâches de même priorité (quantum de temps) ;
- nécessité d'utiliser des outils de synchronisation pour utiliser des données communes ou communiquer entre les tâches.

Licence de droits d'usage

Samuel TARDIEU <sam@rfc1149.net>

Section critique

Une section critique représente un ou plusieurs chemins de code dans lesquels une seule tâche peut se trouver à la fois.

Implémentations possibles :

- inhibition temporaire des interruptions ;
- inhibition temporaire de l'ordonnanceur préemptif, si les routines d'interruption n'utilisent pas les données ;
- utilisation d'un sémaphore.

Dans toutes ces solutions, on risque de bloquer une tâche prioritaire.

Licence de droits d'usage
Samuel Tardieu <sam@rfc1149.net>

Sémaphore

Un sémaphore représente un ensemble de ressources et comprend :

- un compteur, qui indique le nombre de ressources disponibles ;
- une file d'attente, de type FIFO (*first in first out*) ou ordonnée par les priorités, représentant les tâches bloquées en attente d'une ressource.

Les opérations sur un sémaphore sont :

- P() ou take() : demande une ressource, et bloque l'appelant si aucune n'est disponible ; ne revient de l'appel que lorsque la ressource a été acquise ;
- V() ou release() : libère une ressource, jamais bloquant.

Un sémaphore avec une ressource unique est appelé « verrou » (*mutex*).

Licence de droits d'usage
Samuel Tardieu <sam@rfc1149.net>

Problème potentiel

Imaginons la situation suivante :

- un sémaphore `S` contenant initialement une ressource disponible ;
- une tâche Ψ_{basse} de basse priorité réclame le sémaphore à t_0 ;
- une tâche Ψ_{haute} de haute priorité démarre à t_1 et réclame le sémaphore à t_2 ;
- une tâche Ψ_{medium} démarre à t_3 (avant que la tâche Ψ_{basse} ait relâché le sémaphore) et dure très longtemps.

On peut arriver à une situation où la tâche Ψ_{medium} bloque de par sa seule existence la tâche plus prioritaire Ψ_{haute} sans pour autant posséder de ressource dont cette dernière a besoin pour progresser.

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

Inversion de priorité

L'inversion de priorité est une situation dans laquelle une tâche moins prioritaire bloque, indirectement, une tâche plus prioritaire, en empêchant la libération d'une ressource.

P(S)

Élu
Prêt
Bloqué

Ψ_{haute}

Ψ_{medium}

Ψ_{basse}

P(S)

t_0 t_1 t_2 t_3 t

Licence de droits d'usage

Samuel TARDIEU <sam@rfc1149.net>

Inversion de priorité

L'inversion de priorité est une situation dans laquelle une tâche moins prioritaire bloque, indirectement, une tâche plus prioritaire, en empêchant la libération d'une ressource.

P(S)

Ψ_{haute}

Ψ_{medium}

Ψ_{basse}

P(S)

Inversion de priorité

Élu
Prêt
Bloqué

t_0 t_1 t_2 t_3 t

Licence de droits d'usage

Samuel TARDIEU <sam@rfc1149.net>

Héritage de priorité

On utilise alors l'héritage de priorité : lorsqu'une tâche plus prioritaire attend une ressource possédée par une tâche moins prioritaire, cette dernière prend la priorité de la première.

Licence de droits d'usage

Samuel TARDIEU <sam@rfc1149.net>

Interblocage

Si plusieurs tâches utilisent plusieurs sémaphores et cherchent à réserver les ressources au même moment, on peut aboutir à des interblocages :

- un *deadlock* lorsqu'aucune des tâches ne peut progresser ;
- un *livelock* lorsque les deux tâches progressent mais passent leur temps uniquement à réserver les ressources.

Licence de droits d'usage
Samuel Tardieu <sam@rfc1149.net>

Synchronisation et interruptions

Les routines d'interruption, prioritaires, empêchent la progression normale du programme et limitent la gestion par priorités. Pour cela, on divise généralement le traitement en deux parties :

FLIH (*first-level interrupt handler*), consistant à débloquer une tâche qui effectuera le traitement complet de l'interruption et à enregistrer sa prise en compte au niveau matériel ;

SLIH (*second-level interrupt handler*), tâche ordinaire, disposant de sa priorité propre, qui effectue le traitement, possiblement long, de la condition signalée.

Un événement peu important sera acquitté rapidement au niveau du matériel mais sera possiblement traité beaucoup plus tard lorsqu'il ne restera rien de plus important à faire.

Licence de droits d'usage

Samuel Tardieu <sam@rfc1149.net>

Synchronisation et interruptions

La signalisation est faite à l'aide d'un sémaphore :

- le FLIH donne le sémaphore (ressources++) ;
- le SLIH consomme le sémaphore (ressources–).

Ici, ce n'est pas la même entité qui prend et redonne le sémaphore.

V(S)

Élu
Prêt
Bloqué

FLIH
Ψ_{haute}
Ψ_{SLIH}

P(S)
P(S)

t_0 t_1 t_2 t_3 t_4 t_5 t

Licence de droits d'usage
Samuel Tardieu <sam@rfc1149.net>

Différents types de sémaphore

Dans les systèmes embarqués, on trouve généralement différents types de sémaphores :

- sémaphores dont le nombre de ressources est plafonné, sans héritage de priorité, utilisé pour la synchronisation (FLIH/SLIH ou entre tâches) ;
- verrous avec héritage de priorité ;
- verrous multi-entrées avec héritage de priorité.

L'opération bloquante sur ces entités (P()) est généralement assortie d'un *timeout* :

- timeout à zéro : retour immédiat ;
- timeout non nul : temps d'attente limité ;
- timeout à une valeur spéciale : appel bloquant.

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

Files d’attente

Pour passer des informations de manière protégée, les files d’attente (ou *queue*) permettent de déposer et de récupérer de manière atomique des données ordonnées.

- Chaque file d’attente est créée avec une taille maximale et un type de données.
- L’écriture et la lecture sont bloquantes, suivant un modèle producteur/consommateur.
- Les opérations bloquantes sont assorties d’un *timeout*.

Licence de droits d’usage

Samuel TARDIEU <sam@rfc1149.net>

Interdépendances

Les files d'attente peuvent être implémentées à l'aide de sémaphores. Inversement, les sémaphores peuvent être implémentés à l'aide d'une file d'attente. Dans le modèle FLIH/SLIH :

- le FLIH met un élément dans la file d'attente à une place avec un retour immédiat s'il y a déjà un élément ;
- le SLIH consomme un élément dans la file d'attente de manière bloquante.

Ressources et priorités

Lorsqu'une tâche Ψ_1 libère une ressource sur laquelle une tâche Ψ_2 est en attente, Ψ_2 passe immédiatement dans l'état prêt. Si Ψ_2 est plus prioritaire que Ψ_1, cela induit un transfert de contrôle immédiat (changement de contexte) de Ψ_1 vers Ψ_2.

Les raisons d'un changement de contexte sont donc :

- la disponibilité d'une ressource sur laquelle une autre tâche était en attente, depuis une autre tâche ou une routine d'interruption (FLIH) ;
- l'expiration d'un délai, qui consiste en fait à la libération d'une ressource déclenchée depuis une interruption liée à un *timer*, ce qui nous ramène dans le premier cas.

Une tâche en attente d'une ressource ne consommera pas inutilement de temps CPU.

Licence de droits d'usage

Samuel TARDIEU <sam@rfc1149.net>

Exemple : afficheur LCD

Un microcontrôleur pilote un afficheur LCD en lui envoyant des octets correspondant :

- à un caractère à afficher à la position courante du curseur ;
- à un ordre de déplacement spécifiant la ligne et la colonne.

On souhaite que plusieurs tâches puissent afficher à des endroits différents de l'écran.

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>
TELECOM ParisTech

Exemple : afficheur LCD

On utilise les entités suivantes :

- une file d'attente contenant les données à envoyer à l'afficheur LCD ;
- une tâche recevant successivement les octets de la file d'attente et les envoyant à l'afficheur ;
- un sémaphore permettant un accès exclusif à la file d'attente, pour que le remplissage se fasse de manière cohérente ;
- d'autres tâches prenant le sémaphore, mettant les données dans la file d'attente et relâchant ensuite le sémaphore.

Ainsi, les différentes chaînes de caractère à afficher ne peuvent pas se mélanger.

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

Synchronisation et priorités

Si la tâche qui envoie les données à l'afficheur est plus prioritaire que la tâche qui souhaite afficher quelque chose, l'affichage peut commencer dès l'entrée du premier octet dans la file d'attente et n'être limité que par les limitations de l'afficheur lui-même.

Si les priorités sont égales, il est conseillé de donner la main au consommateur, afin qu'il vide la file d'attente et limite les inversions de priorité par la suite.

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

Système temps-réel

Un système est dit temps-réel lorsque chaque événement est traité dans un délai maximum connu à l'avance :

- Un système réagissant la plupart du temps en 100ns mais dans 0,001% des cas en un temps non borné n'est pas temps-réel, bien qu'il soit rapide ;
- Un système réagissant systématiquement en moins de 10s à un événement est temps-réel, bien qu'extrêmement lent.

Licence de droits d'usage
Samuel Tardieu <sam@rfc1149.net>

Catégorisation des systèmes temps-réel

Il existe plusieurs types de systèmes temps-réel :

- temps-réel dur : un résultat arrivant après l'échéance est inutile (un *pacemaker* qui ne réagirait pas à temps) ;
- temps-réel mou : un résultat arrivant après l'échéance induit des performances dégradées (omissions d'images dans un décodeur vidéo).

La plupart des systèmes nécessitant du temps-réel comprennent un mélange de trois sous-composants :

- domaine temps-réel dur pour les opérations critiques ;
- domaine temps-réel mou ;
- domaine non-temps-réel, pour l'écriture des fichiers de traces par exemple.

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

Délais et tâches périodiques

Une tâche peut demander à attendre pendant un certain délai :

- relatif, c'est-à-dire un certain temps ;
- absolu, c'est-à-dire jusqu'à une date donnée.

Une tâche périodique utilisera l'un ou l'autre selon ses besoins :

- un délai absolu permet d'obtenir une exécution à un moment précis indépendamment des retards subis lors d'itération précédentes (gestion d'une horloge) ;
- un délai relatif permet d'espacer des événements d'un intervalle de temps donné (*keep-alive* sur un lien réseau).

Une tâche périodique est caractérisée par sa fréquence et son temps d'exécution d'une itération.

Licence de droits d'usage
Samuel Tardieu <sam@rfc1149.net>

Tâches périodiques et échéances

Étant donné un ensemble fini de tâches périodiques Ψ_i arrivant à intervalles T_i et nécessitant un temps d'exécution C_i, il est parfois possible de garantir qu'aucune tâche n'accumulera de retard en leur affectant des priorités statiques, à l'aide de l'algorithme RMS (*rate monotonic scheduling*).

On peut toujours trouver un tel jeu de priorités si

$$U = \sum_{i=1}^{n} \underbrace{\frac{C_i}{T_i}}_{u_i} \leq n\left(\sqrt[n]{2} - 1\right)$$

On remarquera qu'on a $\lim_{n->+\infty} n(\sqrt[n]{2} - 1) = \ln 2 \approx 69,31\%$. Cela signifie qu'avec un système avec une charge des tâches temps-réel U inférieure à 69,31% on trouvera toujours un moyen d'ordonnancer un tel jeu de tâches à base de priorités statiques.

Licence de droits d'usage

Samuel Tardieu <sam@rfc1149.net>

TELECOM ParisTech

Précautions à prendre

Lors de l'utilisation de l'algorithme RMS, deux précautions particulières doivent être prises :

- le temps nécessaire aux changements de contexte doit être ajouté au temps d'exécution des tâches ;
- si des moyens de synchronisation sont utilisés, ils doivent implémenter l'héritage de priorité pour éviter les inversions de priorité.

De plus, si toutes les tâches réussissent à remplir leur première échéance (c'est-à-dire à terminer leur premier cycle avant l'arrivée suivante), on peut prouver qu'elles y parviendront systématiquement.

Licence de droits d'usage

Samuel TARDIEU <sam@rfc1149.net>

Tâches sporadiques

Une tâche sporadique représente l'exécution d'un code en réaction à un événement. Elle est caractérisée par son temps d'exécution et l'intervalle de temps minimal entre deux exécutions.
On peut faire rentrer les tâches sporadiques dans l'algorithme RMS en les transformant en tâches périodiques. Si nécessaire, il faut adopter une politique spécifique en cas de déclenchement trop fréquent :

- ignorer les déclenchements supplémentaires (en reportant éventuellement une erreur) ;
- sauver dans une file d'attente les déclenchements supplémentaires pour leur faire respecter l'intervalle minimal d'inter-arrivée.

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

Plan

Introduction

Gestion de la mémoire

Gestion de la concurrence

Systèmes de fichiers

Quelques OS embarqués

FreeRTOS

Licence de droits d'usage

Samuel TARDIEU <sam@rfc1149.net>

Pourquoi utiliser un système de fichiers ?

Un système de fichiers (FS) embarqué est utile :

- pour stocker des données de tailles connues ou inconnues de manière persistante ou temporaire ;
- pour stocker des informations de configuration ;
- pour stocker des programmes ;
- pour échanger des informations avec l'extérieur, sur un support interne ou un support amovible.

Licence de droits d'usage

Samuel Tardieu <sam@rfc1149.net>

Pourquoi ne pas utiliser un FS ?

On peut souhaiter ne pas utiliser de système de fichiers :

- quand les données sont peu nombreuses ;
- quand elles sont plus efficacement stockées dans des blocs de taille et d'adresse fixes ;
- quand la complexité d'un système de fichiers ne se justifie pas.

Licence de droits d'usage

Samuel TARDIEU <sam@rfc1149.net>

Les formats FAT

La famille de formats FAT (*file allocation table*) est la plus couramment utilisée pour les fichiers échangés avec le monde extérieur :

- simple à mettre en œuvre ;
- extrêmement répandue ;
- protégée par des brevets logiciels de Microsoft qui les exerce (à l'encontre de TomTom) depuis février 2009.

Plusieurs bibliothèques libres (par exemple FatFs) implémentent ces formats et peuvent être intégrées dans les applications si la présence de ces formats est souhaitée.

FAT sur mémoire flash

Les formats FAT sont peu adaptés à un stockage immédiat sur mémoire flash :

- absence de système de vérification d'intégrité ;
- absence de système de répartition d'écriture (*wear levelling*, temps d'effacement des blocs réécrits surtout sur flash NOR) ;
- favorise un système de stockage linéaire des blocs, où aller d'un bout à l'autre du disque prend du temps, ce qui n'est pas le cas avec la mémoire flash.

Licence de droits d'usage
Samuel Tardieu <sam@rfc1149.net>

FAT sur mémoire flash

On peut soit utiliser un format de stockage adapté, soit insérer un contrôleur (logiciel ou matériel) intermédiaire qui :

- efface les blocs inutilisés lorsqu'il n'a rien à faire ;
- stocke des informations de contrôle permettant de détecter des erreurs de lecture ou d'écriture (vérification *a posteriori*) ;
- répartit les écritures pour équilibrer le nombre d'écritures sur chaque bloc.

De tels contrôleurs sont inclus dans la plupart des clés USB et des disques flash.

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

YAFFS2

Des systèmes de fichiers adaptés aux mémoires flash sont disponibles. Par exemple, YAFFS2 :

- gère la flash en pages regroupées dans des blocs de 32 pages ;
- scanne les blocs lors du montage du système de fichiers pour reconstruire la structure en mémoire (peut prendre plusieurs secondes pour les gros volumes) ;
- utilise des identifiants d'objet et des numéros de séquence pour ne garder que la version la plus récente de chaque page ;
- peut regrouper des pages valides dans un nouveau bloc afin de libérer les anciens blocs si l'espace libre devient restreint.

La mise en œuvre d'un tel système de fichier est en général bien plus complexe que l'utilisation de FAT mais permet l'utilisation efficace des flashs (notamment des flashs NAND).

Mémoire flash et XIP

Pour exécuter du code stocké en mémoire flash, il faut qu'il soit directement accessible par le processeur. Pour cela, il faut l'une des conditions suivantes :

- le recopier en RAM après un décodage éventuel ;
- qu'il se trouve sous forme non compressée et non chiffrée dans une flash NOR (interne ou externe) mappée dans l'espace mémoire du processeur (XIP, *execute in place*).

Un corollaire est que la confidentialité (par chiffrement), la faible consommation d'espace logique (par compression) ou physique (par utilisation de flash NAND) augmentent nécessairement les besoins en mémoire vive.

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

SquashFS

SquashFS est un système de fichier pour Linux qui :

- permet uniquement la lecture ;
- compresse les fichiers, répertoires et i-nodes ;
- est combinable (avec UnionFS ou aufs) avec un système de fichier en mémoire, pour obtenir une version en lecture/écriture ;
- est intégré dans le noyau depuis la version 2.6.29 et donc peut être facilement utilisé comme système de fichiers initial ou unique.

Un nouveau système, AXFS, est en développement et permettra de stocker certaines pages non compressées pour permettre le XIP, et autorisera la répartition entre flash NOR (XIP) et flash NAND (le reste).

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

Plan

Introduction

Gestion de la mémoire

Gestion de la concurrence

Systèmes de fichiers

Quelques OS embarqués

FreeRTOS

Licence de droits d'usage

Samuel Tardieu <sam@rfc1149.net>

PalmOS

PalmOS (de PalmSource) est axé sur les assistants personnels (PDA) :

- logiciel propriétaire ;
- basé sur un noyau AMX 68000, noyau multi-tâches ;
- n'a pas le droit d'exposer l'API multi-tâches, en raison de son contrat de license, mono-tâche de fait pour les développeurs ;
- avant la version 5, tourne sur m68k ;
- depuis la version 5, émule le code m68k sur ARM ;
- intègre la reconnaissance d'écriture ;
- offres des services de communication et de synchronisation.

Licence de droits d'usage

Symbian OS

Symbian OS gère principalement des téléphones mobiles et des « tablettes Internet » :

- logiciel propriétaire, devant être libéré en 2010 ;
- multi-tâches préemptif ;
- protection de la mémoire ;
- centré sur la protection des données et la conservation de l'énergie ;
- basé sur des callbacks et des événements ;
- offre une API éloignée de celle de POSIX, bien qu'une surcouche adaptative soit disponible.

Licence de droits d'usage

Samuel TARDIEU <sam@rfc1149.net>

Linux

Le noyau Linux, utilisé sur les ordinateurs de bureau et les serveurs, est largement configurable, permettant de l'utiliser dans une version légère dans des systèmes embarqués :

- logiciel libre ;
- supporte un grand nombre d'architectures de processeurs ;
- disponibilité d'un grand nombre de gestionnaires de périphériques (*device drivers*), et excellente documentation disponible pour en écrire de nouveaux ;
- système multi-tâches préemptif et multi-utilisateur (utile pour la protection du système) ;
- n'offre pas de garantie de temps-réel dur ;
- plus lourd que les systèmes dédiés à l'embarqué, à l'exception peut-être de Windows CE.

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

RTEMS

RTEMS (de OAR) est un système d'exploitation supportant les applications temps-réel :

- logiciel libre, disponible pour de nombreuses architectures ;
- exécutif léger, se combinant lors de la compilation et de l'édition de liens avec l'application de l'utilisateur ;
- système multi-tâche préemptif ;
- support de la programmation concurrente en Ada ;
- implémente tous les services POSIX d'un système mono-processus ;
- très utilisé dans le milieu des expérimentations physiques, notamment en milieu spatial ;
- utilisé dans le projet *Mars Reconnaissance Orbiter*.

Licence de droits d'usage

FreeRTOS

FreeRTOS est un système d'exploitation temps-réel ciblant tout spécialement les micro-contrôleurs :

- logiciel libre ;
- supporte un grand nombre de processeurs ;
- multi-tâches préemptif, coroutines sans pile ou les deux à la fois ;
- se combine avec l'application finale lors de la compilation et de l'édition de liens ;
- très petit, très rapide et très bien documenté ;
- possède deux versions supplémentaires :

  OpenRTOS : licence ne nécessitant pas d'indiquer qu'on utilise FreeRTOS ni de distribuer les sources de FreeRTOS ;

  SafeRTOS : version certifiée selon plusieurs normes.

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

Etc., etc., etc.

Il existe beaucoup d'autres systèmes d'exploitation pour systèmes embarqués non décrits ici, entre autres :

- Windows CE, Windows mobile, Windows 7 mobile, iPhone OS ;
- eCos ;
- VxWorks.

Nous allons plus particulièrement nous intéresser à FreeRTOS.

Licence de droits d'usage

Plan

Introduction

Gestion de la mémoire

Gestion de la concurrence

Systèmes de fichiers

Quelques OS embarqués

FreeRTOS

FreeRTOS : nommage

Pour réduire les risques liés à l'utilisation du C, FreeRTOS a utilise une stratégie de nommage :

- les préfixes `v`, `c`, `s`, `l`, `d`, `f`, `e` et `x` désignent des entités de type respectif `void`, `char`, `short`, `long`, `double`, `float`, type énuméré et autre, ou des fonctions retournant ces types ;
- des préfixes additionnels `p` et `u` représentent respectivement les pointeurs et les versions non signées des types précédents ;
- les fonctions privées à un fichier sont préfixées par `prv` ;
- le nom des fonctions, après leur préfixe, contient le nom du fichier d'où elles sont tirées.

De plus, le type `portBASE_TYPE` représente le type naturel (le plus efficace) de l'architecture sous-jacente.

Licence de droits d'usage
Samuel Tardieu <sam@rfc1149.net>

FreeRTOS : configuration

La configuration de FreeRTOS se fait dans un fichier `FreeRTOSConfig.h`, et définit notamment :

- les fonctionalités utilisées (préemption, sémaphores, files d'attente, priorités dynamiques ou non, délais relatifs, délais absolus, etc.) ;
- le nombre de niveau de priorités utilisées, et la plage réservée à l'exécutif de FreeRTOS et aux interruptions ;
- la taille de pile minimale pour chaque tâche, la taille de la mémoire disponible pour le tas ;
- la fréquence du *tick* système, qui représente la granularité des délais offerts au programmeur et la fréquence du changement de tâche active entre tâches de même priorité.

Licence de droits d'usage
Samuel Tardieu <sam@rfc1149.net>

FreeRTOS : synchronisation

FreeRTOS offre les outils suivants pour la synchronisation entre les tâches et avec les routines d'interruptions :

- files d'attente ;
- sémaphores binaires ou à valeur maximale fixée, sans héritage de priorité ;
- verrous avec héritage de priorité.

Tous ces outils peuvent être utilisés depuis des tâches avec des *timeouts*, ou depuis des interruptions avec des primitives dédiées non bloquantes.

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

FreeRTOS : concurrence

Pour la gestion de programmes concurrents, FreeRTOS propose :

- un système de tâches préemptif avec gestion de la priorité ;
- un système de coroutines avec gestion de la priorité ;
- un système de *hooks* permettant de spécifier du code exécuté à chaque *tick* du noyau ou lorsque le système exécute l'*idle-task*.

Ces systèmes peuvent être utilisés seuls ou combinés. Il est par exemple possible d'utiliser des coroutines lors de l'exécution de l'*idle-task* et des tâches en mode préemptif sinon.

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

FreeRTOS : gestion du temps

Le système de *ticks* de FreeRTOS permet :

- à une tâche d'être suspendue pendant un nombre entier de ticks ;
- à une tâche d'être suspendue jusqu'à une date donnée (en nombre de *ticks* depuis le démarrage) ;
- à l'ordonnanceur de changer de tâche active parmi les tâches de plus haute priorité lors du déclenchement du *tick*.

Plus la fréquence du *tick* est grande, plus la gestion du temps est précise, mais plus le surcoût lié à cette gestion est important.

Pour des événements plus précis, il est possible d'utiliser des *timers* externes à FreeRTOS qui débloqueront des tâches à l'aide d'un sémaphore binaire.

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

FreeRTOS : gestion de la mémoire

FreeRTOS a besoin de routines malloc() et free () pour allouer la mémoire des différentes entités. En addition, trois implémentations sont livrées avec FreeRTOS :

- `heap_1.c` : allocation possible, désallocation ignorée ;
- `heap_2.c` : allocation possible, désallocation possible sans agrégation des zones libres contiguës ;
- `heap_3.c` : utilise les fonctions malloc() et free () fournies, mais les rend *thread-safe*.

Si les tâches et les objets (sémaphores, files d'attentes, etc.) sont créés au début du programme, toutes les allocations disponibles seront faites avant de démarrer l'ordonnanceur. Il sera par la suite impossible de manquer de mémoire.

Licence de droits d'usage

Samuel TARDIEU <sam@rfc1149.net>

FreeRTOS : outils de trace

FreeRTOS est doté d'outils de trace et de *debug* permettant de suivre le fonctionnement du système et de s'assurer de son bon déroulement :

- les tâches peuvent être nommées et la liste des tâches consultées ;
- des données supplémentaires peuvent être attachées aux tâches, et consultées lors du changement de contexte (par exemple en changeant une sortie analogique pour représenter une tension différente pour chaque tâche) ;
- un certain nombre d'attributs peuvent être consultés ;
- des fonctionalités de vérification de non-débordement de la pile peuvent être employés (vérification lors du changement de contexte et canari).

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

FreeRTOS : anatomie d'une application

Une application FreeRTOS aura généralement la forme suivante :

1. Initialisation sommaire du matériel (assez pour les deux étapes suivantes)
2. Copie de la section `.data` et initialisation de la section `.bss`
3. Exécution du programme principal (à partir de l'étape suivante)
4. Fin de l'initialisation du matériel (horloges supplémentaires, *wait states*, etc.)
5. Initialisation des périphériques
6. Création des structures FreeRTOS (tâches et structures de contrôle)
7. Démarrage de l'ordonnanceur

Licence de droits d'usage
Samuel TARDIEU <sam@rfc1149.net>

Et maintenant ?

Il ne reste plus qu’à mettre tout cela en œuvre !

Licence de droits d’usage

Samuel TARDIEU <sam@rfc1149.net>

Licence de droits d'usage

Contexte académique } avec modifications

Par le téléchargement ou la consultation de ce document, l'utilisateur accepte la licence d'utilisation qui y est attachée, telle que détaillée dans les dispositions suivantes, et s'engage à la respecter intégralement.

La licence confère à l'utilisateur un droit d'usage sur le document consulté ou téléchargé, totalement ou en partie, dans les conditions définies ci-après, et à l'exclusion de toute utilisation commerciale.

Le droit d'usage défini par la licence autorise un usage dans un cadre académique, par un utilisateur donnant des cours dans un établissement d'enseignement secondaire ou supérieur et à l'exclusion expresse des formations commerciales et notamment de formation continue. Ce droit comprend :
– le droit de reproduire tout ou partie du document sur support informatique ou papier,
– le droit de diffuser tout ou partie du document à destination des élèves ou étudiants,
– le droit de modifier la forme ou la présentation du document,
– le droit d'intégrer tout ou partie du document dans un document composite et de le diffuser dans ce nouveau document, à condition que :
  – L'auteur soit informé,
  – Le document dérivé soit diffusé dans un cadre académique.

Les mentions relatives à la source du document et/ou à son auteur doivent être conservées dans leur intégralité.

Le droit d'usage défini par la licence est personnel et non exclusif. Tout autre usage que ceux prévus par la licence est soumis à autorisation préalable et expresse de l'auteur : `sitepedago@telecom-paristech.fr`