Noyaux temps réel

Prof. J.-D. Decotignie
CSEM Centre Suisse d'Electronique et de Microtechnique SA
Jaquet-Droz 1, 2007 Neuchâtel
jean-dominique.decotignie@csem.ch

Plan

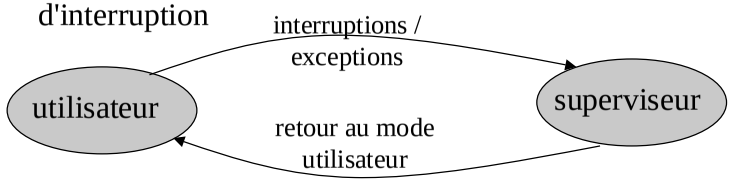
- introduction
- processus, tâches, threads
- noyaux et micro-noyaux
- principe de fonctionnement du noyau
- services offerts
- signalisation des événements
- communication
- exclusion
- conclusion

Protection par matériel

- deux modes d'opération
- protection de la mémoire
- protection des entrées et sorties
- protection du processeur

Deux modes d'opération

- mode utilisateur
  - pas accès à toutes les instructions
  - pas accès à tous les registres
- mode privilégié
  - donne accès à tout les processeur (instructions et registres)
  - on y passe automatiquement en cas d'exception ou d'interruption



Protection de la mémoire

- au moins pour les vecteurs d'exception et les routines de traitement des interruptions
- ceci peut être basé sur
  - des mécanismes matériels
    - on empêche l'accès à certaines adresses en mode utilisateur
  - un circuit de gestion de la mémoire
    - le programme ne peut accéder qu'à certaines adresses
    - la (les) gamme(s) est (sont) programmables
      - ces valeurs font partie du contexte du programme

csem

Protection des entrées et sorties

- les instructions d'entrées et sorties sont privilégiée
  - ne marche pas avec espace commun d'adressage
- on veut éviter qu'un programme en mode utilisateur
  - puisse prendre contrôle du processeur en passant en mode privilégié (en modifiant le vecteur d'exception par ex.)
  - puisse accéder directement aux entrées et sorties (souvent réalisé par protection mémoire)

csem

Protection du processeur

- on utilise un temporisateur qui interrompt ce qui est en cours après une période donnée
  - permet au système de reprendre le contrôle
  - le temporisateur est décrémenté à chaque coup d'horloge
  - quand il arrive à 0, une interruption est provoquée
- le temporisateur est souvent utilisé
  - pour le temps partagé "time sharing"
  - pour calculer l'instant courant
- il faut s'arranger pour que la modification du temporisateur soit une instruction privilégiée

csem

Le concept de tâche

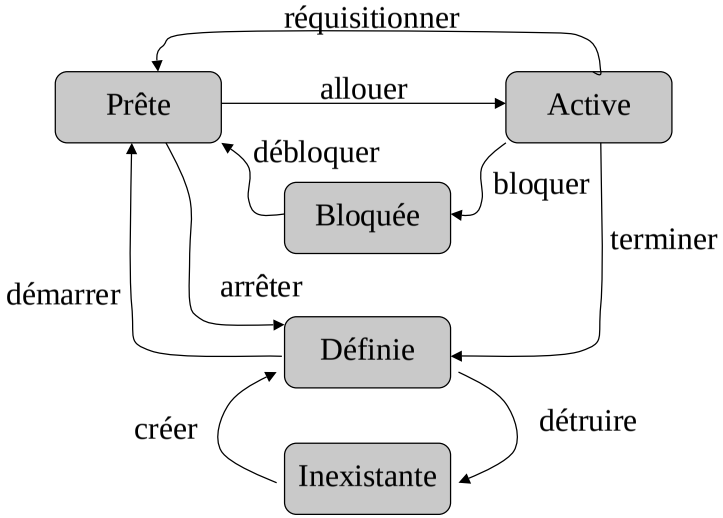
- un système d'exploitation exécute une variété de programmes
  - système par lot: jobs
  - systèmes multi-utilisateurs: programme utilisateur, process, threads
  - systèmes temps réel: tâches, threads
- une tâche inclut
  - un compteur ordinal
  - une pile
  - des données

csem

Le concept de tâche (2)

- au fur et à mesure de son exécution, une tâche voit son état changer:
  - on la crée
  - elle se voit octroyer le processeur
  - elle est en attente du processeur
  - elle est en attente d'un événement / d'une données
  - elle a terminé son exécution
  - elle est détruite
- elle ne peut être que dans un état à la fois

Etats d'une tâche



Les composants d'une tâche

- des zones de mémoire
  - code
  - données
  - pile
- les registres du processeur
  - le compteur ordinal
  - les registres généraux
  - le mode d'état
  - les registres des co-processeurs (virgule flottante, gestion de mémoire, etc.)

Process control block (PCB)

- c'est l'information associée à chaque tâche
  - l'état de la tâche
  - le compteur ordinal et les registres du processeur
  - l'information d'ordonnancement
  - l'information sur la gestion de mémoire
  - l'information statistique sur l'utilisation
  - l'information d'état des entrées et sorties
    - fichiers ouverts, connexions ouvertes, etc.

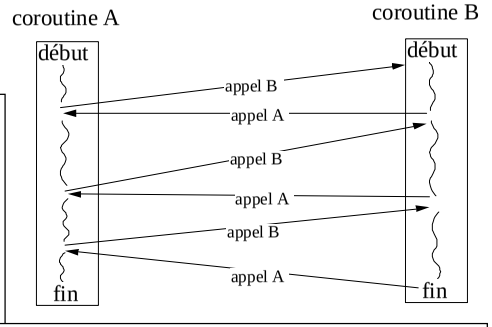
Processus, tâche, thread

- processus UNIX
  - espaces différents, protection mémoire
- light weight processes ou threads (POSIX, Solaris, Java)
  - espace commun
- tâches dans un système temps-réel
  - espace commun, protection par le langage
- coroutines
  - espace common
  - pas gérées par le système (coopératif)

Coroutines

- inventées par Conway

"un programme autonome qui communique avec des modules adjacents comme s'ils étaient des procédu-res d'entrée et de sortie.



Les coroutines sont donc des procédures toutes au même niveau hiérarchique, chacune agissant comme si elle était le programme principal alors qu'en fait il n'y a pas de programme principal"

Caractéristiques de coroutines

- les valeurs des données locales à une coroutine persistent entre l'instant où elle perd le contrôle et l'instant suivant où elle le reçoit de nouveau
- l'exécution d'une coroutine est suspendue lorsqu'elle perd le contrôle du processeur et ne sera reprise que lorsqu'on lui redonnera ce contrôle à un instant ultérieur
- le contrôle est transféré explicitement d'une coroutine à une autre. Ceci a pour effet de suspendre l'exécution de la coroutine en cours d'exécution et de reprendre l'exécution de la coroutine cible . On parlera alors de quasi-parallélisme.

Coroutines en modula-2

- création

```
NEWPROCESS ( Nom_de_la_procédure : PROC;
             Espace_de_travail : ADDRESS;
             Taille_de_l_espace : CARDINAL;
          VAR Coroutine_Handle : ADDRESS);
```

- passage de contrôle

```
TRANSFER (VAR Handle_Coroutine1,
              Handle_Coroutines 2: ADDRESS);
```

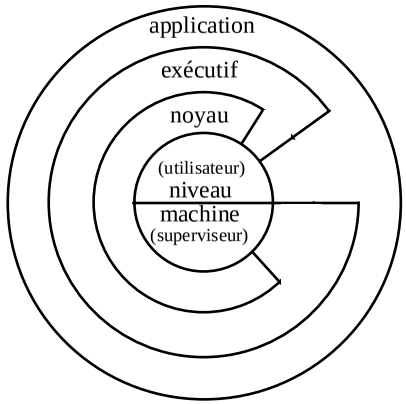

Exemple de coroutines

```
MODULE Coroutine1;

FROM SYSTEM IMPORT
 ADDRESS, NEWPROCESS, TRANSFER;
VAR PP, C1, C2, C3 : ADDRESS;

PROCEDURE P1;
BEGIN
   LOOP
        ...
        TRANSFER (C1, C2);
        ...
   END;
END P1;

PROCEDURE P2;
BEGIN
   LOOP
        ...
        TRANSFER (C2, C3);
        ...
   END;
END P2;
```

```
PROCEDURE P3;
BEGIN
   LOOP
        ...
        TRANSFER (C3, C1);
        ...
   END;
END P3;

BEGIN (* programme principal *)
   ...
   NEWPROCESS(P1,...,...,C1);
   NEWPROCESS(P2,...,...,C2);
   NEWPROCESS(P3,...,...,C3);
   ...
   TRANSFER (PP, C1);
END Coroutine1.
```


Structuration du logiciel système

- structure en pelure d'oignon
  - processeur
  - noyau
  - exécutif
- regroupe les fonctions en familles
- chaque famille correspond à un niveau d'abstraction
- règles de visibilité



Le noyau

- donne à l'exécutif accès à une machine virtuelle
- réside en mémoire centrale
- assure
  - la gestion des tâches
  - l'allocation optimale du processeur
  - la communication entre tâches
  - la synchronisation des tâches et exclusion mutuelle
  - la gestion du temps
  - la gestion de la mémoire
  - la communication avec l'extérieur

Noyaux, micro-noyaux

- noyaux à la UNIX
  - lourds
  - fonctionnalités nombreuses
- versions allégées ou micro noyaux
  - juste le minimum nécessaire
  - fonctionnalités réduites mais suffisantes

Principe de fonctionnement

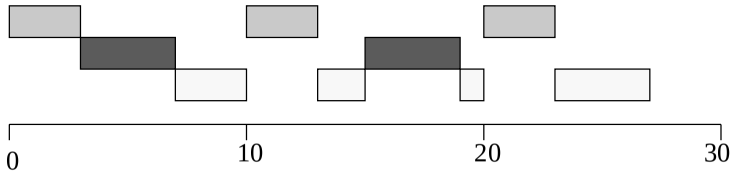
- pseudo parallélisme
- tout événement doit passer par le noyau
  - interruptions
  - demandes d'exclusion mutuelle
  - synchronisations
  - communications
  - temporisations
-

  => support d'un temporisateur HW

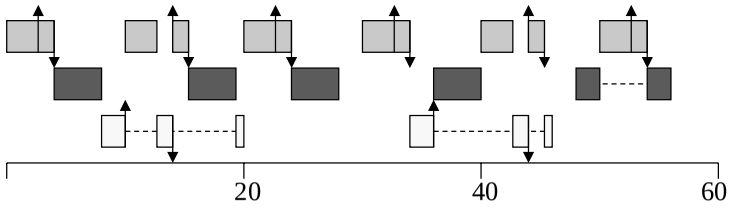
Exemple

- à chaque commutation correspond un événement

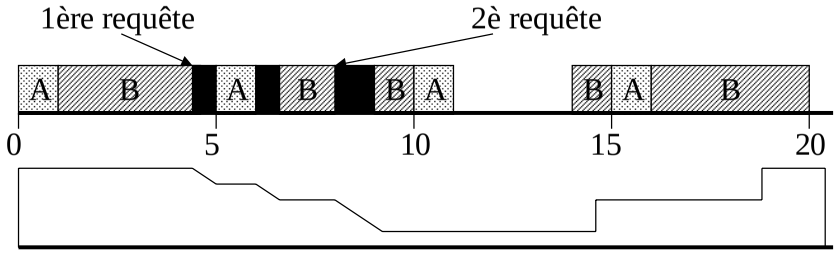
Task	T	D	C
A	10	10	3
B	15	15	4
C	30	30	10



Exemple avec sémaphores



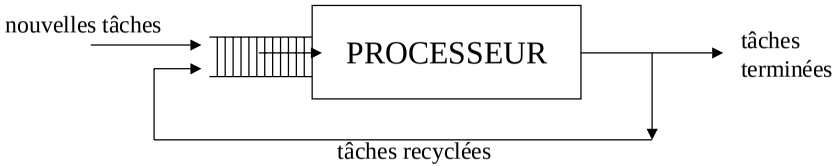
Exemple - Serveur sporadique



Task	T	C	use
A	5	1	20%
SpS	10	2.5	25%
B	14	6	42.9%

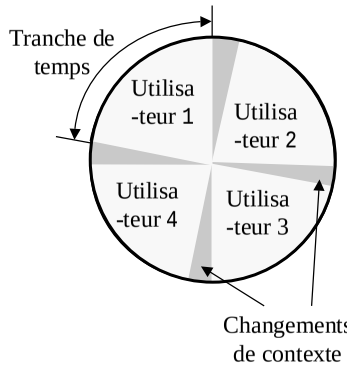
Allocation du processeur

- algorithme du tourniquet
  - une file pour les tâches prêtes
  - chaque tâche reçoit le processeur
    - pour une durée fixe
    - jusqu'à ce qu'elle se bloque
  - retourne ensuite en fin de file



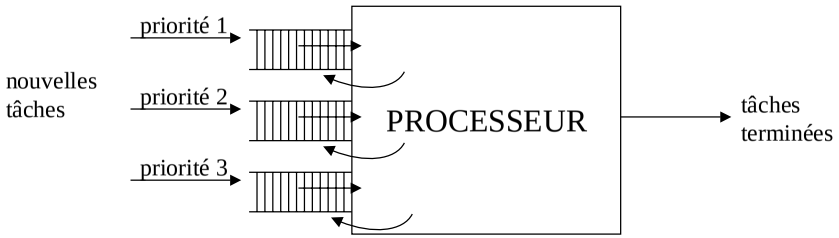
Time slices

- d'abord lié au "time sharing"
- concept utile dans le cas où il n'y a pas de réquisition
- en principe inutile dans un noyau temps réel, mais !



Allocation du processeur (2)

- allocation par priorité
  - une file par priorité
  - règle du tourniquet à l'intérieur de la même file



Services offerts

- création et destruction des tâches
- services de synchronisation
- services d'exclusion mutuelle
- services de communication
- gestion du temps
- intervention sur l'ordonnancement
- gestion de mémoire

- autres (réseau, fichiers, ...)

Services offerts (2)

- il existe des normes
  - POSIX
  - SCEPTRE
- choix des services
  - facilité de réalisation
  - pouvoir d'expression
  - affaire de goût
  - ne pas oublier les aspects annexes (outils de développement et de déverminage)

Exemples de services offerts

Gestion des tâches	StartTask	Tâche	lance l'exécution de la tâche
	StopTask	Tâche	arrêt de la tâche
	ResumeTask	Tâche	reprise de l'exécution de la tâche
	TerminateTask		termine l'exécution de la tâche
	StateOf	Tâche	donne l'état courant de la tâche
	PriorityOf	Tâche	donne la priorité d'une tâche
	Current		identifie la tâche courante
	SetPriority	Tâche, Priorité	modifie la priorité de la tâche
Signalisation Synchronisation	SignalEvent	Evén, tâche	Signale à la tâche "événement arrivé"
	WaitEvent	Liste d'évén.	Attend ≥1 événement soit arrivé
	TOWaitEvent	idem + durée	idem mais cesse d'attendre après le délai
	IsEvent	Liste d'événements	vrai si tous les événements de la liste sont arrivés
	ClearEvent	Evénement	Met un événement dans l'état non arrivé

Exemples de services offerts (2)

Communication	SendElement	Elément, file	Mettre l'élément au bout de la file
	GetElement	Elément, file	Prendre l'élément en tête de file
	TOGetElement	idem + durée	idem mais cesse d'attendre après le délai
	IsEmpty	File	Vrai si la file est vide
	IsFull	File	Vrai si la file est pleine
Exclusion	Lock	Verrou	Demande la prise d'un verrou
	TOLock	Verrou, délai	idem avec abandon au bout d'un délai
	Unlock	Verrou	Libère un verrou pris auparavant

Sychronisation par événements

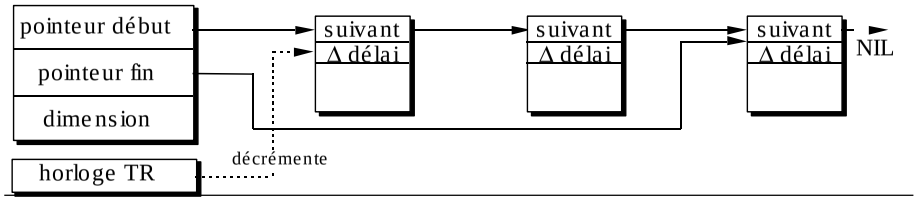
- SignalEvent
  - 1.met à jour fanion associé à l'événement
  - 2.réévalue condition d'attente
  - 3.si bon, passe la tâche dans les "prêtes"
  - 4.commute sur la tâche la plus prioritaire
- WaitEvent (peut porter sur plusieurs événements)
  - 1.si événement présent, la tâche continue
  - 2.si non, elle est mise dans les "bloquées"
  - 3.commute sur la tâche la plus prioritaire
- ClearEvent

Sychronisation par événements (2)

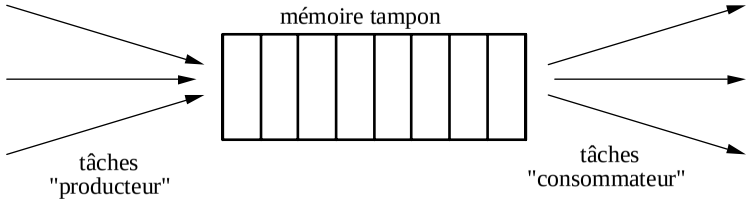
- TOWaitEvent (peut porter sur plusieurs événements)
  - 1. si événement présent, la tâche continue
  - 2. si non, elle est mise dans les "bloquées"puis on arme un temporisateur
  - 3. commute sur la tâche la plus prioritaire

Attente avec temps de garde

- exemple : TOWaitEvent
- fin de temporisation peut être signalée par un événement
- à chaque "tick" d'horloge (interruption)
  - 1. décrémente premier temporisateur
  - 2. si à zéro, retire le bloc de la liste
  - 3. met la tâche bloquée dans la liste des tâches prêtes



Communication



- SendElement
  - 1. met à jour la file d'attente
  - 2. si tâches en attente, passe la plus prioritaire dans les "prêtes"
  - 3. extrait le premier élément
  - 4. commute sur la tâche la plus prioritaire

Communication (2)

- GetElement
  - 1. si élément présent, on extrait le 1er élément et la tâche continue
  - 2. si non, elle est mise dans les "bloquées"
  - 3. elle est ajoutée dans la liste associée à la file
  - 4. commute sur la tâche la plus prioritaire
- IsEmpty, IsFull
- TOGetElement

Verrou

- Lock
  - 1. si verrou libre, il est marqué comme pris, une référence à la tâche est associée au verrou et la tâche continue
  - 2. si non, une référence à la tâche est mise dans la liste d'attente du verrou
  - 3. la tâche est mise dans les "bloquées"
  - 4. la raison du blocage est indiquée dans son descripteur
  - 5. commute sur la tâche la plus prioritaire + traitement de l'héritage de priorité

Verrou (2)

- UnLock
  - 1. rétablit la priorité de la tâche qui libère
  - 2. on enlève la référence à cette tâche qui est mise dans la liste des "prêtes"
  - 3. la 1ère des tâche en attente mise dans "prêtes"
  - 4. le verrou est marqué comme pris, une référence à cette tâche est associée au verrou
  - 5. commute sur la tâche la plus prioritaire + traitement de l'héritage de priorité
- TOLock

Conclusion

- principe de fonctionnement relativement simple mais
  - le choix des services doit être mûrement pesé
- implantation assez complexe (détails, exceptions, ...)
- il est souvent préférable de faire appel à des produits éprouvés
  - souvent inutilement complexes
  - peu prévisibles
  - attention aux outils
  - disposer des sources

Références

- H. Nussbaumer, "Informatique industrielle II", PPUR, Lausanne, 1987.
- B. Gallmeister, "POSIX.4: Programming for the Real World", O'Reilly & Associates, Bonn, 1995.
- A. Tanenbaum, "Modern Operating Systems", Prentice Hall, Upper Saddle River, 1992.
- J. Labrosse, "µC/OS the Real-Time Kernel", R&D publications, Lawrence, Kansas, 1992.