Département Informatique INSA de Lyon

Les exécutifs temps réel

Jean-Philippe Babau

Département Informatique, INSA Lyon

jean-philippe.babau@insa-lyon.fr

Département Informatique INSA de Lyon

jean-philippe.babau@insa-lyon.fr

Plan

- Introduction aux exécutif multitâches
- Rôle et familles d'exécutifs
- Les services
- Mise en œuvre
- Utilisations

jean-philippe.babau@insa-lyon.fr

Environnement de développement

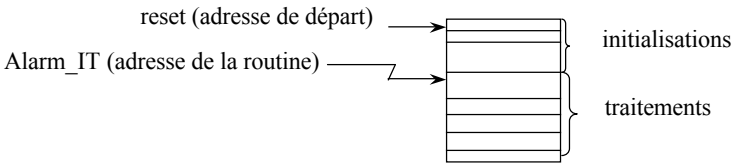
Machine hôte Machine cible

- Développement
  - programmation sur la machine hôte
  - simulation sur la machine hôte
  - téléchargement sur la machine cible
  -
- Exécution
  - activation
  - trace du programme à l'exécution
  - tests

jean-philippe.babau@insa-lyon.fr

Programmation

- langages de programmation au niveau système
  - assembleur, C
- programmation mono tâche (pas d'OS)
  - Largement répandu car simple de mise en œuvre
  - Gain de place
  - Une boucle = un cycle de régulation (lecture – traitements – écriture)
  - Programmation cyclique : une interruption périodique active le traitement

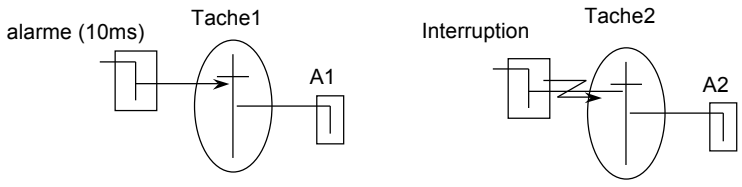


- programmation multitâches
  - plusieurs interruptions
  - temps réel (priorité)
- système mono ou multi-processus

jean-philippe.babau@insa-lyon.fr

Programmation multitâches

- une activité A1 périodique (10ms)
- une activité A2 sur interruption



- besoin de communication
- besoin de protection des données

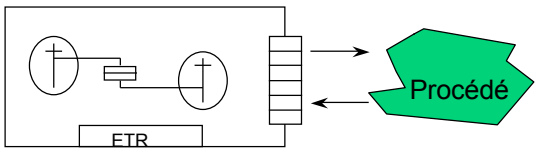
jean-philippe.babau@insa-lyon.fr

Plan

- Introduction aux exécutif multitâches
- Rôle et familles d'exécutifs
- Les services
- Mise en œuvre
- Utilisations

jean-philippe.babau@insa-lyon.fr

Rôle de l'exécutif temps réel



- Rôle de l'ETR à la conception
  - primitives de gestion de la concurrence (tâche, sémaphore, etc.)
- Rôle de l'ETR à l'exécution
  - gestion des interruptions, de l'état des objets temps réel
  - ordonnancement des tâches (priorité)
  - rôle classique d'un SE (BIOS, etc.)
- Fonctionnalités supplémentaires
  - librairies graphiques
  - communication réseau (TCP- IP, etc.)
  - Java, internet embarqué (html)
  - gestionnaire de fichiers
  - debbuger, trace temporelle

jean-philippe.babau@insa-lyon.fr

Systèmes d'exploitations

- Machine virtuelle (Java)
  - Extensions, limitations (PERC, JavaCard, EmbJava, RT-Java)
  - cohabitation de deux systèmes
  - gestion des IHM, de l'internet
- Système standard (Unix, NT)
  - Extensions temps réel (RTX, RT Linux)
  - Noyau temps réel (**Windows CE**)
- Système commercial
  - adaptable
  - standard
  - librairies
- Système propriétaire
  - coût
  - maintenance
  - taille (petit noyau : de 4 à 40 ko ; gros noyaux : 250 ko; pour info noyau Linux : 1Mo)
  - protection
- Pas d'exécutif

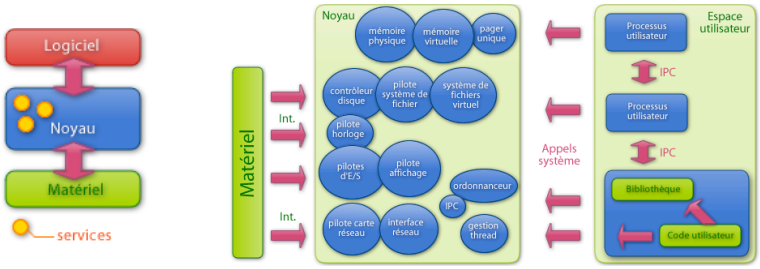
jean-philippe.babau@insa-lyon.fr

Systèmes d'exploitation temps réel

- Real-Time Operating System : 43 références dans Wikipedia !
- Exécutif temps réel du marché
  - VxWorks, QNX, LynxOS, Chorus de Chorus Systèmes, Nucleus RTOS, WindowsCE
- Exécutifs libres
  - MicroC/OS-II, FreeRTOS, RTEMS
  - Jean J. Labrosse " *MicroC/OS-II, The Real-Time Kernel* " Miller Freeman Inc. 1999
- Exécutifs à composants
  - Think (ObjectWeb), S.Ha.R.K
  - Think sur le lego® RCX2.0 pour une application pathfinder : 6 Ko
- Exécutifs dédiés
  - TinyOS : OS pour les réseaux de capteurs sans fil, architecture à composants
  - TIM micro-kernel du robot Khepera-II

jean-philippe.babau@insa-lyon.fr

Architectures des systèmes monolithiques



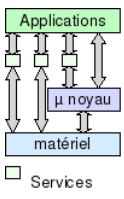
Schémas wikipédia

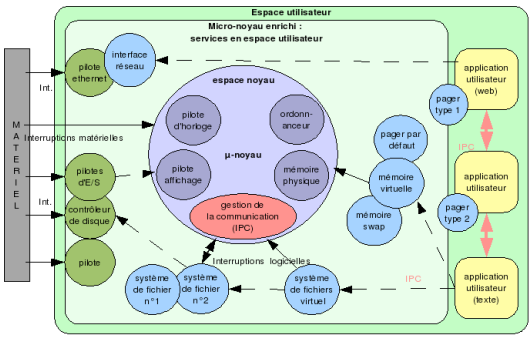
- Noyau non modulaire
  - Efficace
  - problème de maintenance, évolutions, adaptations
- Noyaux monolithiques modulaires : Linux

jean-philippe.babau@insa-lyon.fr

Architectures à micro-noyaux

- Noyau : limité au gestionnaire de mémoire, ordonnanceur, communication inter-processus
- Fonctions non critiques : espace utilisateur, mode protégé
- Micro-noyau enrichi



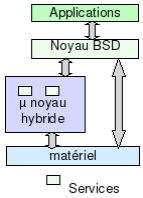


Schémas wikipédia

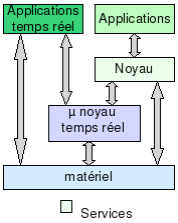
jean-philippe.babau@insa-lyon.fr

Architectures hybrides et temps réel

- Noyau hybride
  - noyau + système monolithique
  - Services « utiles » implémentés dans le noyau
- Noyaux temps réel
  - Noyau pour les exécutifs simples
  - Noyau enrichi pour offrir des services
  - Noyau hybride pour l'encapsulation des services
    - VxWorks, RT-Linux, RTAI



Schémas wikipédia



jean-philippe.babau@insa-lyon.fr

Département Informatique INSA de Lyon

Plan

- Introduction aux exécutif multitâches
- Rôle et familles d'exécutifs
- Les services
- Mise en œuvre
- Utilisations

jean-philippe.babau@insa-lyon.fr

Les objets du temps réel

- Le processus
  - regroupement d'objets temps réel
  - espace mémoire
- Les objets programmables
  - la tâche ou le thread ou processus léger
  - les routines d'interruption, l'alarme
- Les objets de communication
  - le sémaphore
    - synchronisation
    - exclusion mutuelle
  - l'événement
    - synchronisation
  - la boîte aux lettres
    - échange synchronisé d'informations
  - Le canal ou pipe
    - échange synchronisé d'informations entre processus

jean-philippe.babau@insa-lyon.fr

Manipulation des objets temps réel

- Une primitive de création
  - initialisation
- Une primitive de destruction
- Un jeton par objet
  - identifiant de création
  - handle sous NT
  - entier sous VxWorks

jean-philippe.babau@insa-lyon.fr

La tâche

- Fil d'exécution
  - identifiant
  - fonction à exécuter
  - priorité
  - état
    - prête, bloquée, en-cours
  - contexte
    - valeurs des registres, compteur de programme, pile
- Tâche au niveau programmation
  - procédure en général sans paramètre et sans retour
  - en général en boucle infinie
  - une attente dans la boucle

jean-philippe.babau@insa-lyon.fr

Tâche : les primitives

- Appel d'une autre tâches
  - création (prête ou bloquée)
  - activation
  - réactivation
- Appel d'une autre tâche ou dans la tâche
  - destruction
  - suspension
  - demande du niveau de priorité
  - changement de priorité
  - jeton 0 pour l'appelant sous VxWorks
- Appel dans la tâche
  - mise en sommeil
  - appels de fonctions

jean-philippe.babau@insa-lyon.fr

Exemples

- VxWorks
  LOCAL int taskSpawn(char* "taskName",int priority,0,10000, functionName,0,0,0,0,0,0,0,0,0,0)
  LOCAL int taskInit(…)
  taskSafe ()
- iRMX
  static TOKEN rq_create_task(priority, (void far *) taskId, ...)
- Win32
  CreateProcess(NULL|appliName, "MonProcessFils", …)
  HANDLE CreateThread(lpThreadAttributes, dwStackSize, lpStartAddress, lpParameter, dwCreationFlags // CREATE_SUSPENDED, lpThreadId)
- RT-Linux
  int init_module(void)
  int pthread_create(pthread_t, pthread_attr_t , void *(**start_routine*)(void*), void **arg*)
  typedef struct { … int *attr_priority* … } pthread_attr_t
- TIM micro-kernel
  int install_task (char * taskName, int stackSize, void * functionAd);
- RTX_166 Tiny Real-Time
  os_create_task (int tasd_id);
  os_delete_task (int tasd_id);

jean-philippe.babau@insa-lyon.fr

Le sémaphore

- Rôle
  - synchronisation entre tâches
  - exclusion mutuelle
  -
- Principe
  - sémaphore binaire
  - sémaphore à compte
    - valeur initiale / valeur maximum
    - unité définie par l'utilisateur
      - une unité = 1 Ko
      - une unité = ressource
  - dépôt / retrait
  - liste d'attente
  -
- Paramètres du sémaphore
  - contenu maximum (peut bloquer le dépôt)
  - dépôt multiple dans un sémaphore à compte
  - file d'attente (FIFO ou PRIORITY)

jean-philippe.babau@insa-lyon.fr

Sémaphore : les primitives

- Création
  - Initialisation
- Destruction
  - impact sur l 'application
- Dépôt
  - nombre d'unité
- Retrait
  - nombre d'unité
  - temps d'attente (infini / fini)

jean-philippe.babau@insa-lyon.fr

Exemples

- VxWorks

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

- iRMX

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

- Win32

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

- TIM micro-kernel : pas de sémaphore

jean-philippe.babau@insa-lyon.fr

L'événement

- Rôle
  - synchronisation entre tâches
  - rendez-vous
- Principe
  - envoi
    - signalisation d 'un instant (information temporelle)
  - attente
    - atomicité du déclenchement
  - interrogation
  - événement composite : opérateurs
    - et, ou
- Paramètres de l 'événement
  - libération d 'une ou de toutes les tâches en attente
  - file d'attente (FIFO ou PRIORITY)
  - mémorisation de l 'événement
  - comptage des événements
  - consommation si plusieurs événements composites

jean-philippe.babau@insa-lyon.fr

Evénement : les primitives

- Création
  - initialisation
- Destruction
  - impact sur l 'application
- Envoi
- Attente
  - consommation de l 'événement
  - temps d'attente (infini / fini)

jean-philippe.babau@insa-lyon.fr

Exemples

- Win32

```
HANDLE CreateEvent ( NULL ,auto-reset, initial, "EventName" )
SetEvent ( h_MyEvent )
WaitForSingleObject ( sem , INFINITE | Time-out ) // timeout en ms
```

autoreset

TRUE : un événement consommé reste actif. Pour remettre l'évènement à l'état inactif, on doit appeler la fonction ResetEvent ( )

FALSE : un événement consommé devient inactif

initial

FALSE : non signalé

- RTX_166 Tiny Real-Time

```
os_send_signal  (int task_id) ;   /*Envoi d'un événement à une tâche*/
os_wait_signal  (void);           /*  Attente d'un événement */
os_clear_signal (int task_id) ;   /*Suppression d'un signal pour une tâche */
```

jean-philippe.babau@insa-lyon.fr

La boîte aux lettres

- Rôle
  - échange d'information synchronisées entre tâches
  - objet de communication entre lecteurs/écrivains
- Principe
  - pas de propriétaire
  - un ou plusieurs écrivains
  - un ou plusieurs lecteurs
  -
- Paramètres de la boîte aux lettres
  - contenu maximum (ignoré en pratique)
  - temps d'attente (infini / fini)
  - file d'attente des tâches (FIFO ou PRIORITY)
  - file des messages
  -
- Boite aux lettre de réponse
  - lors de l'envoi : définir une boite de réponse
  - lors de la réception : récupérer le lieu de la réponse

jean-philippe.babau@insa-lyon.fr

Boîte aux lettres : les primitives

- Création
  - initialisation
- Destruction
  - impact sur l 'application
- Dépôt
  - conditionnel (si plein)
- Retrait
  - conditionnel (si message présent)
  - temps d'attente (infini / fini)
- Création/destruction du message
  - allocation mémoire
  - libération mémoire

jean-philippe.babau@insa-lyon.fr

Exemples

- VxWorks

```
msgQId = msgQCreate(nbMaxMess, longueurMax, SEM_Q_FIFO | SEM_Q_PRIORITY)
status = msgQSend(msgQId,buffer,nOctets,temps | NO_WAIT| WAIT_FOREVER, MSG_PRI_NORMAL |
    MSG_PRI_URGENT )
val = msgQReceive(msgQId,buffer,nOctets,temps | WAIT_FOREVER | NO_WAIT )
```

- iRMX

```
balId_tk = rq_create_mailbox(flags,&status)
messId_tk = rq_create_segment(taille,&status)
rq_send_message(balId_tk,messId_tk,repId_tk,&status)
objet_tk = rq_receive_message(balId_tk,temps,repId_tk,&status)
rq_delete_segment(messId_tk,&status)
```

- Win32

```
Création implicite
BOOL PostThreadMessage(idThread, Msg, wParam, lParam)
BOOL GetMessage(lpMsg, hWnd, wMsgFilterMin, wMsgFilterMax)
```

jean-philippe.babau@insa-lyon.fr

La communication par pipe

- Rôle
  - échange d'information synchronisées entre tâches de processus distinct
  - objet de communication entre lecteurs/écrivains
- Principe
  - pas de propriétaire
  - un ou plusieurs écrivains
  - un ou plusieurs lecteurs
- Paramètres du pipe
  - accès en R / W / RW
  - nombre maximum de message
  - taille du message

jean-philippe.babau@insa-lyon.fr

Pipe : les primitives

- Création/Destruction
  - Initialisation
- Ouverture/Fermeture
  - R / W
- Ecriture
  - si plein
  - si message trop long
- Lecture
  - si vide

jean-philippe.babau@insa-lyon.fr

Les ressources

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

Protection des ressources

- sémaphore d'exclusion mutuelle
  - politique d'accès (cf. ordonnancement)
- passage en mode non préemptif ou super-priorité
  - gestion d 'une information temporelle
  - accès à un réseau
  - donnée à durée de vie/validité limitée
- opération atomique
- masquage/démasquage des interruptions
  - « pas longtemps »
- pas de protection
  - un seul utilisateur

La routine d'interruption

- interruption
  - matériel : niveau d 'interruption, it processeur
  - niveau logiciel : vecteur -> routine -> tâche d 'interruption
- gestion des interruptions
  - (0) arrivée d 'une interruption non masquée
  - (1) fin de l 'instruction de la tâche courante
  - (2) sauvegarde du contexte de la tâche
  - (3) acquittement de l 'interruption (contrôleur d'interruption)
  - (4) exécution de la routine d 'interruption associée
  - (5) ré-ordonnancement
    - retour à la tâche interrompue
    - activation nouvelle tâche
- principes
  - priorités(routines) > priorités(tâches)
  - possibilité de masquage/démasquage des its (perte d 'informations)
  - pas d'appel aux primitives de manipulation d'objets dans la routine d'interruption

jean-philippe.babau@insa-lyon.fr

Interruption : les primitives

- Création
  - initialisation
- Destruction
- Déclenchement
- Attente
- Masquage
  - niveau
- Démasquage
  - niveau

jean-philippe.babau@insa-lyon.fr

Exemple

- Windows CE

- une interruption IRQ1 est liée à une pin (front montant ou descendant, selon le matériel) et un niveau logique NL1
- l'arrivée d'une IRQ1 masque toutes les autres interruptions (pertes)
- dès que le niveau logique de l'IRQ1 est reconnu, la routine d'interruption correspondante est lancée. Durant son exécution, les autres interruptions sont mémorisées (selon la politique du matériel) sauf celle liées à NL1 (perte)
- la routine d'interruption active un (ou plusieurs) événement
- Si plusieurs routines sont à exécuter, elles le sont dans un ordre FIFO (politique a priori programmée)
- un thread de type IST (Interrupt Service Thread) se met en attente de l'événement
- ce thread a en charge de l'acquittement explicite de l'interruption (primitive InterruptDone(dwSysIntr))

jean-philippe.babau@insa-lyon.fr

L'alarme

- Rôle
  - activation à date fixe
  - attente
  - mise en place d'un time-out actif
- Principes
  - interruption programmée
  - logiciel ou matériel
  - interne ou externe
- Paramètres
  - temporisation
  - mono ou multi réveil
  - données applicatives

jean-philippe.babau@insa-lyon.fr

Alarme : les primitives

- Création
  - initialisation
- Destruction
- Activation
  - temps
  - nombre
- Arrêt

jean-philippe.babau@insa-lyon.fr

Exemples

- VxWorks

  wdId = wdCreate()

  status = wdStart(wdId,tempo,alRoutine,param)

- Win32

  wTimerID= timeSetEvent (delay, wTimerResolution // en millisecondes , callbackFunction, userData, TIME_ONESHOT | TIME_PERIODIC)

- RT-Linux

```
pthread_make_periodic_np (pthread_self(), date réveil, période);
```

- RTX_166 Tiny Real-Time

  os_wait (int typ, unsigned int timeout, int dummy);

  os_wait (K_TMO, 300, 0);
  /* tâche bloquée pour 300 ticks, un tick correspond à une période d'horloge du µc */
  os_wait (K_IVL, 1000, 0);
  /* tâche bloquée pour 1s */

jean-philippe.babau@insa-lyon.fr

Cadenceur

- Intervalle de temps
  - interruption périodique (top ou tick)
  - intervalle de 10 ou 200 ms
  - défini par défaut à 1/60 seconde en VxWoks
    - peut être modifié
- Base de temps unique pour le système
  - Attente
    - en ticks : taskDelay(3) => temps d'attente = [2, 3]
  - Alarme
    - Activation périodique
    - Contrôle d'échéance, de timeout
- Exemples
  - VxWorks
    int sysClkRateGet (void) /* renvoie le nombre de ticks par seconde */
    STATUS sysClkRateSet(int ticksPerSecond)
  - RT-Linux
    gethrtime() /* renvoie la précision en nanosecondes */
    clock_gethrtime()

jean-philippe.babau@insa-lyon.fr

Fonctions réentrantes (utilisables simultanément par plusieurs tâches)

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

OK

```
int coef  = 3 ;

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

OK

```
int coef  ;
int coefMini = 3;

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

pas OK

jean-philippe.babau@insa-lyon.fr

Gestion de la mémoire

- prédictibilité
  - taille, temps
  - allocation dynamique
- gestion par bloc
  - mémoire = ensemble de partitions
  - une partition = un ensemble de bloc de taille fixe
  - par exemple : 100 blocs de 32 octets
- gestion du bloc
  - allocation/désallocation par bloc
  - un bloc = un objet temps réel
  - primitives de création / destruction
  - primitives de demande/libération de bloc(s)

jean-philippe.babau@insa-lyon.fr

Plan

- Introduction aux exécutif multitâches
- Rôle et familles d'exécutifs
- Les services
- Mise en œuvre
- Utilisations

jean-philippe.babau@insa-lyon.fr

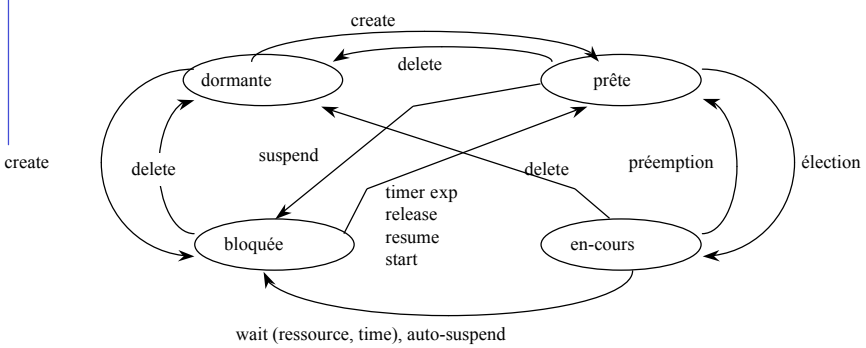
Implémentation d'une tâche

- Structure *Task Control Block* (*TCB*)
  - Eléments de la structure
    - identifiants (nom, entier ou jeton)
    - adresse de la fonction à exécuter
    - niveau de priorité
      - généralement de 0 à 255
      - plus petit niveau : plus forte priorité
    - état
    - compteur de programme, pointeur de pile, registres
    - données (paramètres de la fonction à exécuter, données spécifiques)
    - paramètres d'exécution (non préemptible, attente en tick, ressources utilisées, …)
  - Exemple VxWorks
    - Structure WIND_TCB
    - Primitives d'accès ou de consultation de la structure : taskShoW(), taskIsReady( )
- *TCB* initialisé à la création
- main
  - adresse de départ
  - tâche comme une autre avec un TCB par défaut
- Destruction
  - libération des ressources utilisées ?
    - sémaphores, mémoires

jean-philippe.babau@insa-lyon.fr

États d'une tâche

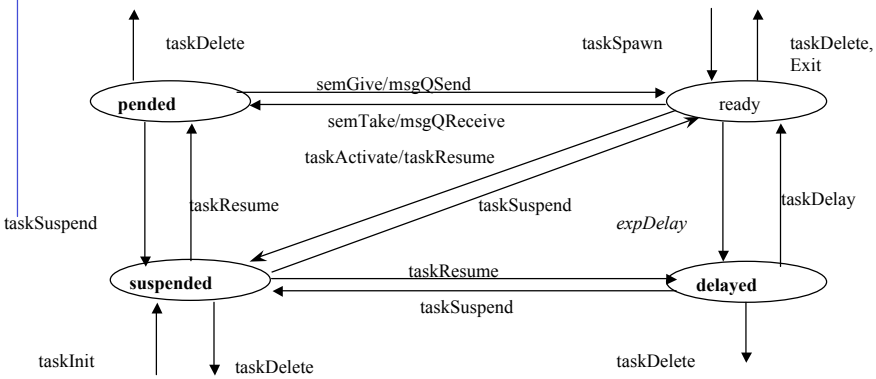
- état
  - dormante
    - en mémoire
  - en-cours
    - priorité maximum parmi les prêt
  - prête
    - zone allouées
  - bloquée
    - en attente, suspendue



jean-philippe.babau@insa-lyon.fr

Etat de la tâche sous VxWorks

- états
  - prêt / **bloqué**



jean-philippe.babau@insa-lyon.fr

Gestion des tâches

- Tableau ou liste de tâches (*TCB*)
  - allocation statique : nombre maximum de tâches (256 en général)
  - allocation dynamique (VxWorks) : limité par la mémoire

Num	Ident	Etat	Priorité
0	Acq	bloquée	10
1	Trait	prête	20
2	Cmde	prête	30

- Gestion de l'état
  - lors de chaque appel de primitive susceptible de changer l'état d'une tâche
  - lors d'opérations sur un objet de communication ou sur une tâche
- Ordonnanceur
  - parcours d'une liste
    - recherche, à partir de 0 (plus forte priorité), de la 1ère tâche dans l'état « prête »
  - priorité égales : partage de temps
  - modification de priorité : modification du tableau

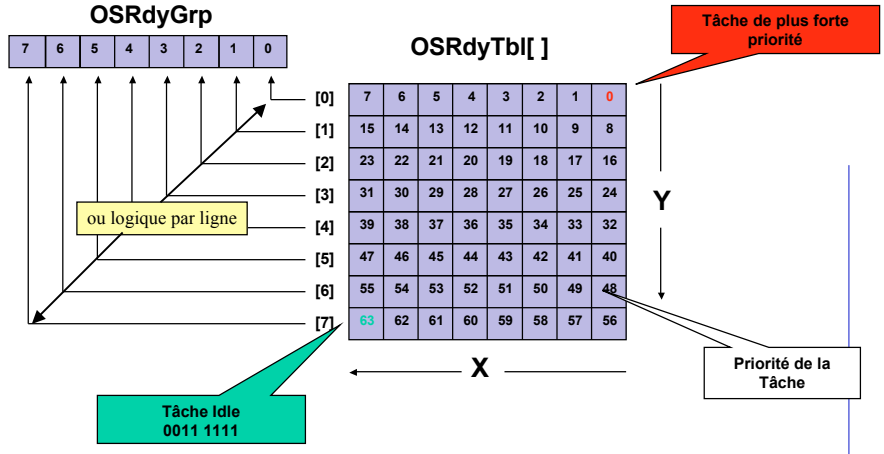
jean-philippe.babau@insa-lyon.fr

Ordonnanceur

- Mode préemptif
  - Appelé après chaque changement d'état
    - Dernier appel des primitives de communication
    - Dernier appel d'une routine (si déblocage possible d'une tâche)
- Mode non préemptif
  - Appelé à la fin de tâche
    - task_suspend(), task_exit()
- Blocage de l'ordonnanceur
  - Passage en mode non préemptif
  - Exécution de primitives *système*
  - Primitives spécifiques
    - VxWorks : taskLock(), taskUnlock()
    - TIM micro-kernel : tim_lock(), tim_unlock()

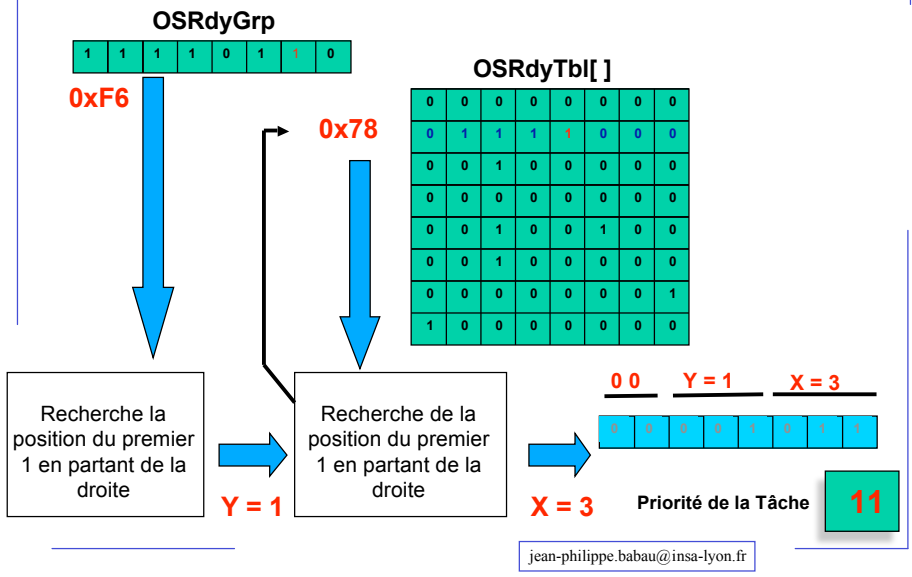
jean-philippe.babau@insa-lyon.fr

Ordonnanceur µC/OS II : table des tâches prêtes



jean-philippe.babau@insa-lyon.fr

Ordonnanceur µC/OS II : trouver la tâche à exécuter



jean-philippe.babau@insa-lyon.fr

Ordonnanceur µC/OS II : table pour la recherche

```
/*********************************************************
*            PRIORITY RESOLUTION TABLE
*********************************************************/
INT8U const OSUnMapTbl[] = {
  0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,  // 0x00-0x0F
  4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,  // 0x10-0x1F
  5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,  // 0x20-0x2F
  4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,  // 0x30-0x3F
  6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,  // 0x40-0x4F
  4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,  // 0x50-0x5F
  5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,  // 0x60-0x6F
  4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,  // 0x70-0x7F
  7, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,  // 0x80-0x8F
  4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,  // 0x90-0x9F
  5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,  // 0xA0-0xAF
  4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,  // 0xB0-0xBF
  6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,  // 0xC0-0xCF
  4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,  // 0xD0-0xDF
  5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,  // 0xE0-0xEF
  4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0   // 0xF0-0xFF
};

  INT8U  const  OSMapTbl [ ]  = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};
```

X = @ [0x78]
(i.e. 0x78 = OSRdyTbl[1])

Y = @ [0xF6]
(0xF6 =OSRdyGrp)

jean-philippe.babau@insa-lyon.fr

Ordonnanceur µC/OS II : les opérations

Trouver la tâche prête de plus haute priorité

```
Y = OSUnMapTbl[OSRdyGrp];

X = OSUnMapTbl[OSRdyTbl[X]];

prio = (Y << 3) + X;
```

Faire passer une tâche à l'état prêt

```
OSRdyGrp |= OSMapTbl[prio >> 3];                    /* prio >> 3 : Y */

OSRdyTbl[prio >> 3] |= OSMapTbl[prio & 0x07];       /* 0x07 = 0000 0111; prio & 0x07 : X*/
```

Retirer une tâche de l'état prêt

```
if ( (OSRdyTbl[prio >> 3] &= ~OSMapTbl[prio & 0x07])  = = 0 )

OSRdyGrp &= ~OSMapTbl[prio >> 3];
```

jean-philippe.babau@insa-lyon.fr

Objets de communication : le sémaphore

- Structure *Semaphore Control Block (SCB)*
  - Structure
    - état
    - type
      - booléen / à comptes
      - FIFO / à priorité
    - tâche courante
    - liste d'attente
  - Exemple VxWorks
    - semShow()
- *SCB* initialisé à la création
- Attente avec timeout
  - activation du timer
  - lors du réveil
    - libération de la tâche
- Destruction
  - libération des tâches en attente
  - attente sur liste vide

jean-philippe.babau@insa-lyon.fr

Un sémaphore proposé par TIM micro-kernel …

```
/** Semaphore initialization **/
void sem_init(unsigned sem_id)
{
      if( sem_id > 31)                    32 maximum
            return BAD_ID;                pas de contrôle <0

      tim_lock();
            semaphoreList&= (!(0x00000001 << sem_id));
      tim_unlock();

}
```

```
/** Semaphore Release **/
void sem_v(unsigned sem_id)
{
      tim_lock();
            semaphoreList&= (0x00000001 << sem_id);
      tim_unlock();
}
```

Pas de déclenchement des tâches en attente

```
/** Semaphore Request **/
int sem_p(unsigned sem_id)
{
      tim_lock();

      if( (0x00000001 << sem_id) & semaphoreList)
      {
      /** Token available - reserve token **/
            semaphoreList&= (!(0x00000001 << sem_id));

      tim_unlock();
      return 1;
      } else
      {
      /** Token reserved - do nothing **/

      tim_unlock();          Pas d'attente
      return 0;
      }
}
```

jean-philippe.babau@insa-lyon.fr

Département Informatique INSA de Lyon

Un sémaphore proposé par TIM micro-kernel

- Codes des tâches appelantes

```
/** Wait until resource 0 is available **/
while( !sem_p(0) );                    ! Boucle sans fin !
      use_resource();
sem_v(0);
```

```
/** Try to access resource 0 and continue **/
if( sem_p(0) )
{
use_resource();
sem_v(0);
}
```

jean-philippe.babau@insa-lyon.fr

Plan

- Introduction aux exécutif multitâches
- Rôle et familles d'exécutifs
- Les services
- Mise en œuvre
- Utilisations

jean-philippe.babau@insa-lyon.fr

Choix d'un système d'exploitation temps réel

- Normes
  - SCEPTRE (1982)
  - POSIX (Unix)
  - OSEK/VDX (Automobile)
  - Profil MARTE de l'OMG
- Domaine
  - Ferroviaire : l'OS préconisé est QNX
- Supports
  - Matériel : processeurs supportés, cartes, mémoire (4 Go sous CE)
  - Drivers (série, LCD, CAN), piles de protocole
  - Fournisseurs et suivi des versions
- Coûts
  - Environnement de développement (Tornado 15 000 € sans fonctionnalités)
  - Royalties (royalty free)
  - Développement supplémentaires
    - Drivers, protocoles

jean-philippe.babau@insa-lyon.fr

Choix vis-à-vis d'un objectif

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

jean-philippe.babau@insa-lyon.fr

Développements

- Portage sur une cible spécifique
  - Gestion de la mémoire
  - Gestion des interruptions
  - Gestion des entrées / sorties
  - Séquence de boot
- Drivers de haut niveau
  - cf. cours sur les pilotes
- Programmation de services supplémentaires
  - Services manquants pour les micro-noyaux dédiés
    - Sémaphores, boîte aux lettres, …
  - Services de haut niveau pour les exécutifs
    - Activation périodique
    - Suivi d'échéances
    - cf. cours multitâches
  - Mise en place d'intergiciels pour supporter des paradigmes de haut niveau
    - Objet actifs UML, composants
  - Attention à bien assurer l'atomicité des services

jean-philippe.babau@insa-lyon.fr

Conclusion

- RTOS : concepts génériques
  - Tâches, sémaphores, …
- … mais mises en œuvre spécifiques

➔ Bien lire les spécifications

- Implémentation
  - Programmation au niveau noyau
    - Espace non protégé
  - Liée aux spécificités du matériel (mémoire, IT)
    - Une couche spécifique par cible
  - Simple et prédictible en temps
    - Allocation statique

jean-philippe.babau@insa-lyon.fr