UBO

lab-sticc.univ-brest.fr/~babau/

Les exécutifs temps réel

Jean-Philippe Babau

Département Informatique, UFR Sciences, UBO
Laboratoire Lab-STICC

jean-philippe.babau@univ-brest.fr

UBO

jean-philippe.babau@univ-brest.fr

UBO

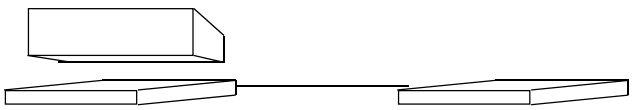
Plan

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

jean-philippe.babau@univ-brest.fr

UBO

Environnement de développement



Machine hôte Machine cible

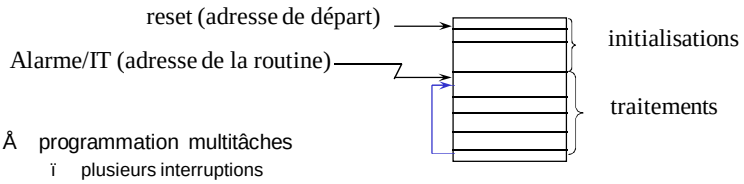
- Développement
  - programmation sur la machine hôte
  - simulation sur la machine hôte
  - téléchargement sur la machine cible
  -
- Exécution
  - activation
  - trace du programme à l'exécution
  - tests

jean-philippe.babau@univ-brest.fr

UBO

Programmation

- langages de programmation au niveau système
  - assembleur, C
- programmation mono tâche (pas de 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
    - boucle infinie



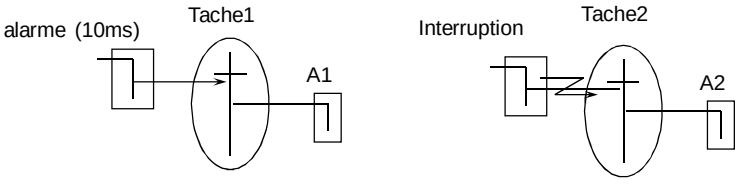
- programmation multitâches
  - plusieurs interruptions
  - temps réel (priorité)
- système mono ou multi-processus

jean-philippe.babau@univ-brest.fr

UBO

Programmation multitâches

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



- besoin de communication
- besoin de protection des données

jean-philippe.babau@univ-brest.fr

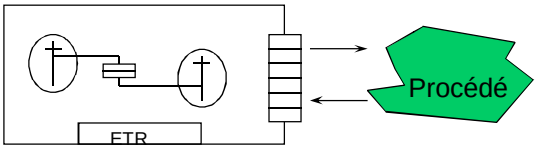
UBO

Plan

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

UBO

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

UBO

Systèmes dôexploitations

- Machine virtuelle (Java)
  - Extensions, limitations (PERC, JavaCard, EmbJava, RT-Java)
  - cohabitation de deux systèmes
  - gestion des IHM, de lôinternet
- Système standard (Linux, Windows)
  - Extensions temps réel (RT Linux, RTX)
  - 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 dôexécutif

jean-philippe.babau@univ-brest.fr

UBO

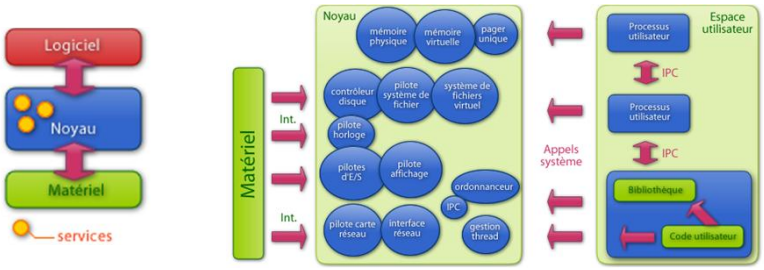
Systèmes dôexploitation temps réel

- 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
  - NXC pour le robot lego® NXT é

jean-philippe.babau@univ-brest.fr

UBO

Architectures des systèmes monolithiques



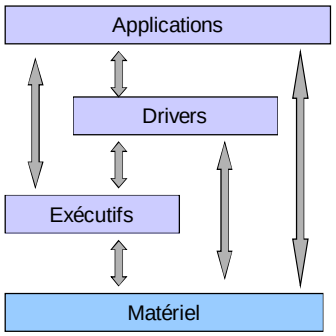
Schémas wikipédia

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

UBO

Architectures temps réel

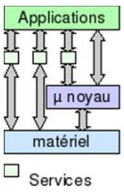
- Exécutifs temps réel
  - Librairie pour l'exécution concurrent
    - utilitaire
  - Exécutif, drivers et applications au même niveau
  - Pas de protection mémoire
    - Un seul espace d'adressage
  - Accès direct au matériel

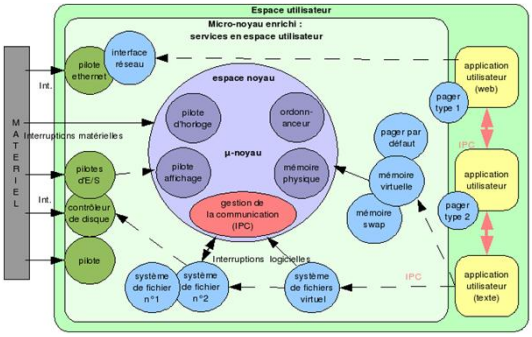


UBO

Architectures à micro-noyaux

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



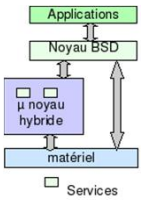


Schémas wikipédia

UBO

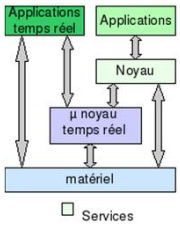
Architectures hybrides et temps réel

- 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 lêncapsulation des services
    - VxWorks, RT-Linux, RTAI

Schémas wikipédia



UBO

Plan

- Introduction aux exécutif multitâches
- Rôle et familles d'exécutifs
- Les services de l'exécutif multitâches
- Mise en î uvre
- Utilisations

UBO

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

UBO

Manipulation des objets temps réel

- Un objet
  - Une structure de donnée (*typedef struct*)
- Une liste dôobjets
  - Tableau fixe ou liste chainée
  - Prédictibilité : nombre dœléments bornés
- Une primitive de création
  - Initialisation des données avec vérification
    - Taille de pile positive, é
  - Vérification et mise à jour de la liste
    - Pas deux noms égaux, pas de priorités égales
    - Pas de dépassement de capacité, indice non déjà attribué, é
- Une primitive de destruction
  - Libération et mise à jour de la liste
- Un jeton par objet
  - identifiant de création
    - handle sous NT
    - entier sous VxWorks
  - Peut être lôindice dans le tableau

jean-philippe.babau@univ-brest.fr

UBO

La tâche

- Rôle et principes
  - Exécution des actions
    - procédure en général sans paramètre et sans retour
    - en général en boucle infinie avec attente
- Données pour gérer un fil dœexécution
  - identifiant
  - fonction à exécuter
  - priorité
  - état
    - prête, bloquée, en-cours
  - temps dôattente
  - Systèmes réentrants : pile
  - Système préemptif : contexte dœexécution
    - valeurs des registres, compteur de programme

jean-philippe.babau@univ-brest.fr

UBO

Tâche : les primitives

- Appel dôune autre tâches
  - création (prête ou bloquée)
  - activation
  - réactivation
- Appel dôune autre tâche ou dans la tâche
  - destruction
  - suspension
  - demande du niveau de priorité
  - changement du niveau de priorité
- Appel dans la tâche
  - mise en sommeil
  - appels de fonctions

UBO

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(é );                    taskSafe ();
```

- iRMX

```
static TOKEN rq_create_task(priority, (void far *) taskId, ...);
```

- Win32

```
CreateProcess( NULL|appliName, "MonProcessFils", é )
HANDLE CreateThread(lpThreadAttributes, dwStackSize, lpStartAddress, lpParameter, dwCreationFlags
    CREATE_SUSPENDED, lpThreadId );
```

- POSIX

```
int pthread_create(pthread_t, pthread_attr_t , void *(*start_routine)(void*), void *arg);
typedef struct { é  int attr_priority é  } 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);
```

- NXC

```
start taskName;
```

UBO

Le sémaphore

- Rôle et principes
  - synchronisation entre tâches
  - exclusion mutuelle
  - Gestion d'unités partagées
    - unité définie par l'utilisateur
      - une unité = 1 Ko
      - une unité = ressource
- Données pour un sémaphore
  - sémaphore binaire
    - booléen
  - sémaphore à compte
    - valeur initiale / valeur maximum
  - Politique de gestion de la file d'attente
    - FIFO ou à priorité
    - Héritage ou non des priorités

jean-philippe.babau@univ-brest.fr

UBO

Sémaphore : les primitives

- Création
  - Initialisation des données
- Destruction
  - Vérification de l'impact sur l'application
    - Si la liste d'attente n'est pas vide
- Dépôt
  - nombre d'unité
    - Le contenu maximum peut bloquer le dépôt (temps infini / fini /nul)
  - dépôt multiple dans un sémaphore à compte
- Retrait
  - Mise à jour du nombre d'unité
  - Blocage si vide avec temps d'attente (infini / fini /nul)

jean-philippe.babau@univ-brest.fr

UBO

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 : événement */
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
```

- NXC

```
mutex mutex1,mutex2;          é .          Acquire(mutex1);          é          Release(mutex1);
```

- TIM micro-kernel : pas de « sémaphore »

jean-philippe.babau@univ-brest.fr

UBO

L'événement

- Rôle et principes
  - Signalisation d'un instant
    - Information temporelle
  - synchronisation entre tâches
  - rendez-vous
- Données pour un événement
  - Politique de mémorisation de l'événement
  - Politique de comptage des événements
  - Politique de gestion de la file d'attente
    - FIFO ou à priorité
  - événement composite : opérateurs
    - et, ou
  - Politique de consommation si plusieurs événements composites

jean-philippe.babau@univ-brest.fr

UBO

Evénement : les primitives

- Création
  - Initialisation des données
- Destruction
  - Vérification de l'impact sur l'application
    - Si la liste d'attente n'est pas vide
- Envoi
- Attente
  - Acquittement de l'événement
    - Implicite ou explicite par les tâches
  - Blocage si non arrivé avec temps d'attente (infini / fini /nul)
- Interrogation
  - Opération *testAndSet*

UBO

Exemples (signal ~ événement)

- 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 */
```

UBO

La boîte aux lettres

- Rôle et principes
  - échange d'information synchronisées entre tâches
  - objet de communication entre lecteurs/écrivains
    - pas de propriétaire
    - un ou plusieurs écrivains
    - un ou plusieurs lecteurs
- Données pour la boîte aux lettres
  - Taille et nombre maximal de messages
  - Communication de données ou de pointeurs sur données
  - Politique de gestion de la file d'attente
    - FIFO ou à priorité
  - Politique de gestion de la file des messages
    - FIFO ou à priorité (message urgent)
- 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@univ-brest.fr

UBO

Boîte aux lettres : les primitives

- Création
  - Initialisation des données
- Destruction
  - Vérification de l'impact sur l'application
    - Si la liste d'attente n'est pas vide
- Dépôt
  - Blocage si plein avec temps d'attente (infini / fini / nul)
    - Parfois écrasement si plein
- Retrait
  - Blocage si vide avec temps d'attente (infini / fini / nul)
- Interrogation
  - Opération *testAndSet*
- Création/destruction du message
  - Allocation et libération mémoire
  - Primitives spécifiques

jean-philippe.babau@univ-brest.fr

UBO

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)
```


UBO

La communication par pipe

- Rôle et principe
  - échange d'information synchronisées entre tâches de processus distinct
  - Boîte aux lettre nommée
  - objet de communication entre lecteurs/écrivains
    - pas de propriétaire
    - un ou plusieurs écrivains
    - un ou plusieurs lecteurs
- Paramètres du pipe
  - nombre maximum de message
  - taille des messages
  - Politique de gestion de la file d'attente
    - FIFO ou à priorité
  - Politique de gestion de la file des messages
    - FIFO ou à priorité (message urgent)

UBO

Pipe : les primitives

- Création/Destruction
  - Initialisation des données
  - Enregistrement auprès des processus
- Ouverture/Fermeture
  - accès en R / W / RW
- Ecriture
  - Blocage si plein avec temps d'attente (infini / fini / nul)
    - Parfois écrasement si plein
- Lecture
  - Blocage si vide avec temps d'attente (infini / fini / nul)
- Interrogation
  - Opération *testAndSet*

UBO

Les ressources

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

UBO

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
- masquage/démasquage des interruptions
  - « pas trop longtemps »
- pas de protection
  - opération atomique (affectation, comparaison)
    - attention à bien vérifier l'atomicité de l'opération
  - un seul utilisateur

jean-philippe.babau@univ-brest.fr

UBO

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@univ-brest.fr

UBO

Interruption : les primitives

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

UBO

Exemple

- Windows CE

- une interruption IRQ1 est liée à une pin (front montant ou descendant, selon le matériel) et un niveau logique NL1
- lôarrivée dôune IRQ1 masque toutes les autres interruptions
- dès que le niveau logique de lôIRQ1 est reconnu, la routine dôinterruption correspondante est lancée. Durant son exécution, les autres interruptions sont mémorisées (selon la politique du matériel) sauf celle liées à NL1 (perte)
- la routine dôinterruption 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 lôévénement
- ce thread a en charge de lôacquittement explicite de lôinterruption (primitive InterruptDone(dwSysIntr))

UBO

Lâalarme

- Rôle et principes
  - activation à date fixe
  - attente
  - mise en place dûun time-out actif
  - interruption programmée
- Données de lâalarme
  - Source
    - Généralement gérée au niveau de lâexécutif
    - logicielle ou matérielle
    - interne ou externe
  - Temporisation
    - Unité de temps en ticks ou durée (ms, ns, ...)
  - mono ou multi réveil
  - données applicatives

UBO

Alarme : les primitives

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

UBO

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@univ-brest.fr

UBO

Gestionnaire de temps

- Base de temps unique pour le système
  - Mise en place d'une interruption périodique : compteur de ticks
    - Activation d'un timer
    - intervalle de x ms (10, 200, ...)
  - Définie par défaut à 1/60 seconde en VxWoks
    - peut être modifiée
  - Gestion du dépassement de capacité
    - Sur 16 bits, l'instant d'après 65535, c'est l'instant 0
    - une attente de 10 ticks après l'instant 65530 et avant l'instant 65531 abouti à l'instant 4
- Utilisation
  - Attente
    - en ticks : taskDelay(3) => temps d'attente = [2, 3]
  - Alarme
    - Activation périodique
      - À programmer
    - Contrôle d'échéance, de timeout

jean-philippe.babau@univ-brest.fr

UBO

Gestionnaire de temps

Å Exemples

ï VxWorks

Respect de la norme IEEE standard, POSIX 1003.1b pour la gestion de l'horloge temps réel

```
int clock_gettime(
clockid_t clock_id, /* clock ID (always CLOCK_REALTIME) */
struct timespec * tp /* where to store current time */)
DESCRIPTION This routine gets the current value tp for the clock.
RETURNS 0 (OK), or -1 (ERROR) if clock_id is invalid or tp is NULL.
  struct timespec {
            time_t tv_sec; /* seconds */
            long tv_nsec; /* nanoseconds (0 -1,000,000,000) */}
```

Gestion des ticks

```
int sysClkRateGet (void) /* renvoie le nombre de ticks par seconde */
STATUS sysClkRateSet(int ticksPerSecond)
```

ï RT-Linux

```
gethrtime() /* renvoie la précision en nanosecondes */
clock_gethrtime(); /* renvoie le temps courant en ns*/
```

ï NXC

```
CurrentTick(); /* renvoie le temps courant en ms sur 32 bits (unsigned long)*/
```

jean-philippe.babau@univ-brest.fr

41

UBO

Fonctions réentrantes

utilisables simultanément par plusieurs tâches

Contexte d'appel par tâche (pile)

```
int somme (int a , int b)
{
      return (a+b) ;
}
int coef  = 3 ;

int correction (int a)
{
      return ( a * coef ) ;
}
int coef  ;
int coefMini  = 3;

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

OK

pas OK

OK si protection par mutex de *coef*

jean-philippe.babau@univ-brest.fr

42

UBO

Fonctions non réentrantes

non utilisables simultanément par plusieurs tâches

Pas de contexte dâappel par tâche (pas de pile)

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

```
int coef  = 3 ;

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

OK si macro (recopie du code dans chaque tâche, *mot-clé inline* avec NXC)

```
int coef  ;
int coefMini  = 3;

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

pas OK

OK si protection par mutex de lâappel de la fonction, *mot-clé safecall* avec NXC et mutex de protection de coef

UBO

Gestion de la mémoire

- prédictibilité
  - taille, temps
  - Implémentation spécifique de lâallocation dynamique
    - Activation périodique de la récupération mémoire (*Garbage Collector*)
- 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)

Plan

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

Architecture d'un RTOS

- Constantes
  - Caractéristiques du processeur et de la carte
  - Types (short, int, é)
  - Intervalle de temps pour l'horloge
  - Taille de la pile
  - Nombre maximal d'éléments (tâches, sémaphores, é)
  - Niveau maximal de priorité
  - Taille maximale d'un nom de tâche, é
- Couche de portage
  - Spécifique pour chaque processeur et chaque carte
    - Initialisations, gestion des périphériques
    - Codage en assembleur pour l'accès aux registres
    - Liée au compilateur utilisé
  - Gestion des interruptions
    - Nested Interrupts
  - Gestion du tick
    - Timer périodique
    - Heure (nombre de ticks)
  - Gestion des changements de contexte

UBO

Architecture dôun RTOS

- Exécutif
  - Module indépendant dôune cible
    - API standard dôappel au port
  - Gestion mémoire
  - Gestion des tâches
    - Structure TaskControlBlock
  - Gestion des sémaphores et mutex
    - Structure SemaphoreControlBlock
  - Gestion des files dôattentes
  - Gestion de lôordonnanceur
    - Activation et élection
    - Listes de tâches
  - Initialisations

UBO

Implémentation dôune 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 de la tâche
    - compteur de programme, pointeur de pile, registres
    - données (paramètres de la fonction à exécuter, données spécifiques)
    - paramètres dôexécution (non préemptible, attente en tick, ressources utilisées, é)
  - Exemple VxWorks
    - Structure WIND_TCB
    - Primitives dôaccès 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

UBO

Allocation mémoire

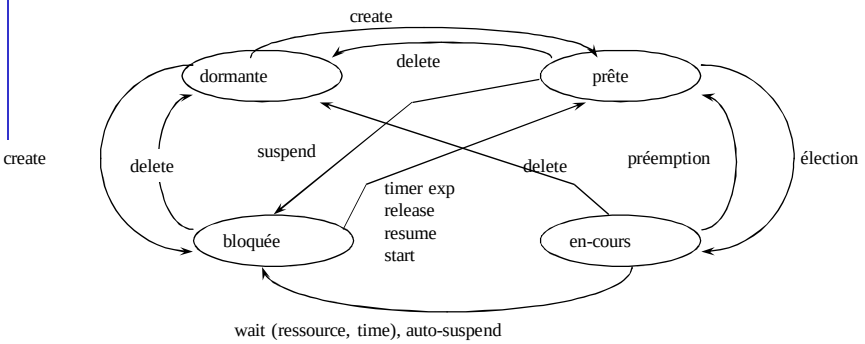
- Gestion de listes de tâches, de sémaphores, é
- Prédictibilité
  - Tableaux de taille fixe
  - MicroC-OS-II
    - 64 tâches, une tâche par niveau de priorité
- Gestion dynamique
  - Allocation des TCB selon l'occupation mémoire
  - FreeRTOS
    - Niveau maximal de priorité fixé à la compilation du noyau
    - Une liste doublement chaînée de tâches par niveau de priorité
    - Initialisation des listes de tâches lors de la création de la première tâche

jean-philippe.babau@univ-brest.fr

UBO

États d'une tâche

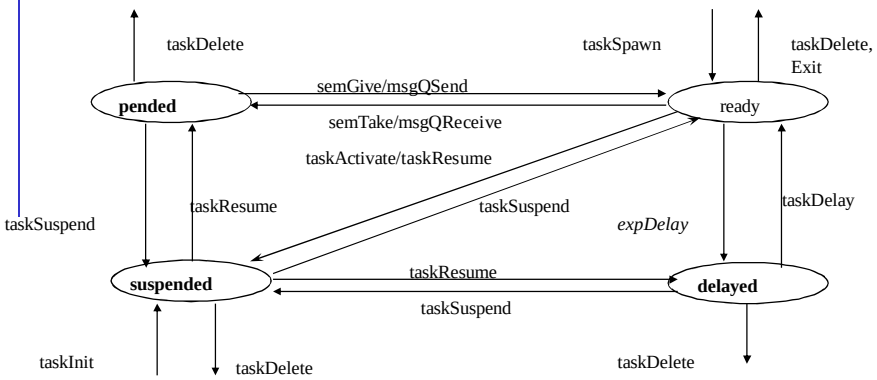
- état
  - dormante
    - en mémoire
  - en-cours
    - priorité maximum parmi les prêt
  - prête
    - zone allouées
  - bloquée
    - en attente, suspendue



jean-philippe.babau@univ-brest.fr

Etat de la tâche sous VxWorks

- états
  - prêt / **bloqué**



Gestion des tâches

- Tableau ou liste de tâches (ensemble de *TCB*)

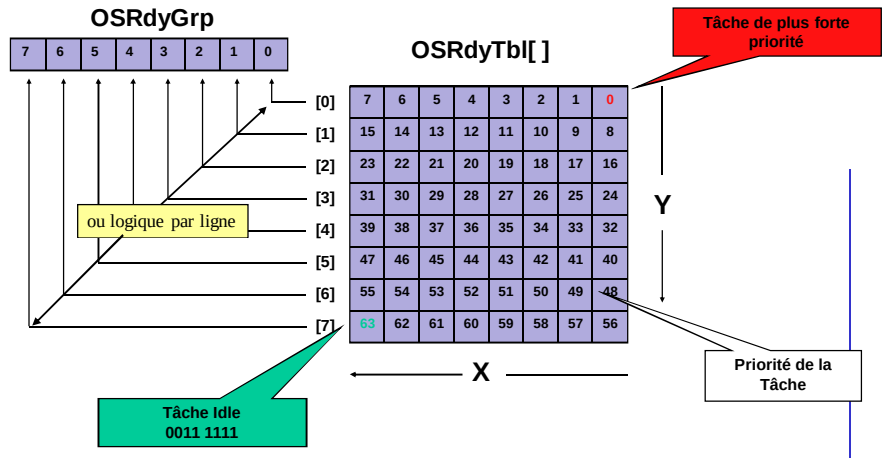
Num	Ident	Etat	Priorité
0	Acq	bloquée	10
1	Trait	prête	20
2	Cmde	prête	30

- 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

Ordonnanceur

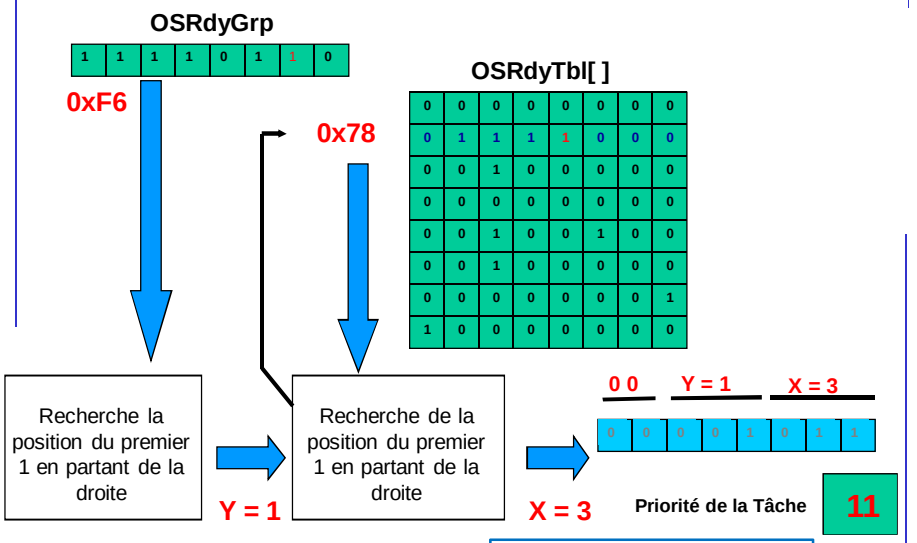
- Mode préemptif
  - Appelé après chaque changement d'état
    - Dernier appel des primitives de communication
    - Dernier appel d'une routine (si déblocage possible d'une tâche)
- Mode non préemptif
  - Appelé à la fin de tâche
    - task_suspend(), task_exit()
- Blocage de l'ordonnanceur
  - Passage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

Ordonnanceur µC/OS II : table des tâches prêtes



UBO

Ordonnanceur µC/OS II : trouver la tâche à exécuter



jean-philippe.babau@univ-brest.fr

UBO

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@univ-brest.fr

UBO

Ordonnanceur µC/OS II : les opérations

Trouver la tâche prête de plus haute priorité

```
Y = OSUnMapTbl[OSRdyGrp];

X = OSUnMapTbl[OSRdyTbl[Y]];

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];
```


UBO

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

Un sémaphore (à programmer soi même) proposé par TIM micro-kernel é

```
/** Semaphore initialization **/
void sem_init(unsigned int sem_id)
{
      if( sem_id > 31)                    32 maximum
            return BAD_ID;

      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;
      }
}
```


Un sémaphore (à programmer soi même) proposé par TIM micro-kernel é

Â Codes des tâches appelantes

```
Task 1 ()
{     /** Wait until resource 0 is available **/
      while( !sem_p(0) );
            use_resource();
      sem_v(0);                                  ! Boucle sans fin !
}

Task2()
{
      /** Try to access resource 0 and continue **/
      if( sem_p(0) )
      {
            use_resource();
            sem_v(0);
      }
}

main()
{
      sem_init(0);
      sem_v(0);
}
```

UBO

Plan

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

jean-philippe.babau@univ-brest.fr

UBO

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 ú sans fonctionnalités)
  - Royalties (royalty free)
  - Développement supplémentaires
    - Drivers, protocoles

jean-philippe.babau@univ-brest.fr

UBO

Choix vis-à-vis dôun objectif

- Optimisation des ressources
  - Critères de taille et/ou de performance
  - Le comportement de lâapplicatif est connu / estimé
  - Le comportement du support est configurable par lâapplication
- 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, é
- Maintenabilité
  - Accès au code
  - traçabilité des appels
- Portabilité du code
  - Respect de normes

jean-philippe.babau@univ-brest.fr

UBO

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, é
  - Services de haut niveau pour les exécutifs
    - Activation périodique
    - Suivi déchéances
    - cf. cours multitâches
  - Mise en place dôintergiciels pour supporter des paradigmes de haut niveau
    - Objet actifs UML, composants
  - Attention à bien assurer lâatomicité des services

jean-philippe.babau@univ-brest.fr

UBO

Conclusion

- RTOS : concepts génériques
  - Tâches, sémaphores, é
- é mais mises en î uvre spécifiques

➔ Bien lire les spécifications

- Implémentation
  - Programmation au niveau noyau
    - Attention car espace non protégé
  - Liée aux spécificités du matériel (mémoire, IT)
    - Une couche de portage spécifique par cible
  - Simple et prédictible en temps
    - Allocations statiques

➔ Maitrise du HW, prédiction a priori

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

jean-philippe.babau@univ-brest.fr