

PRINCIPES DES SYSTÈMES TEMPS-RÉEL

1. Généralités sur les OS temps réel

Introduction rapide aux OS temps réel

Objectif

Masquer à l'application les particularités du matériel
=> machine virtuelle plus ou moins complexe

Classification des OS :

Généraliste (UNIX...)

Généralistes étendus temps réel (Linux, POSIX...)

Temps réel d'origine

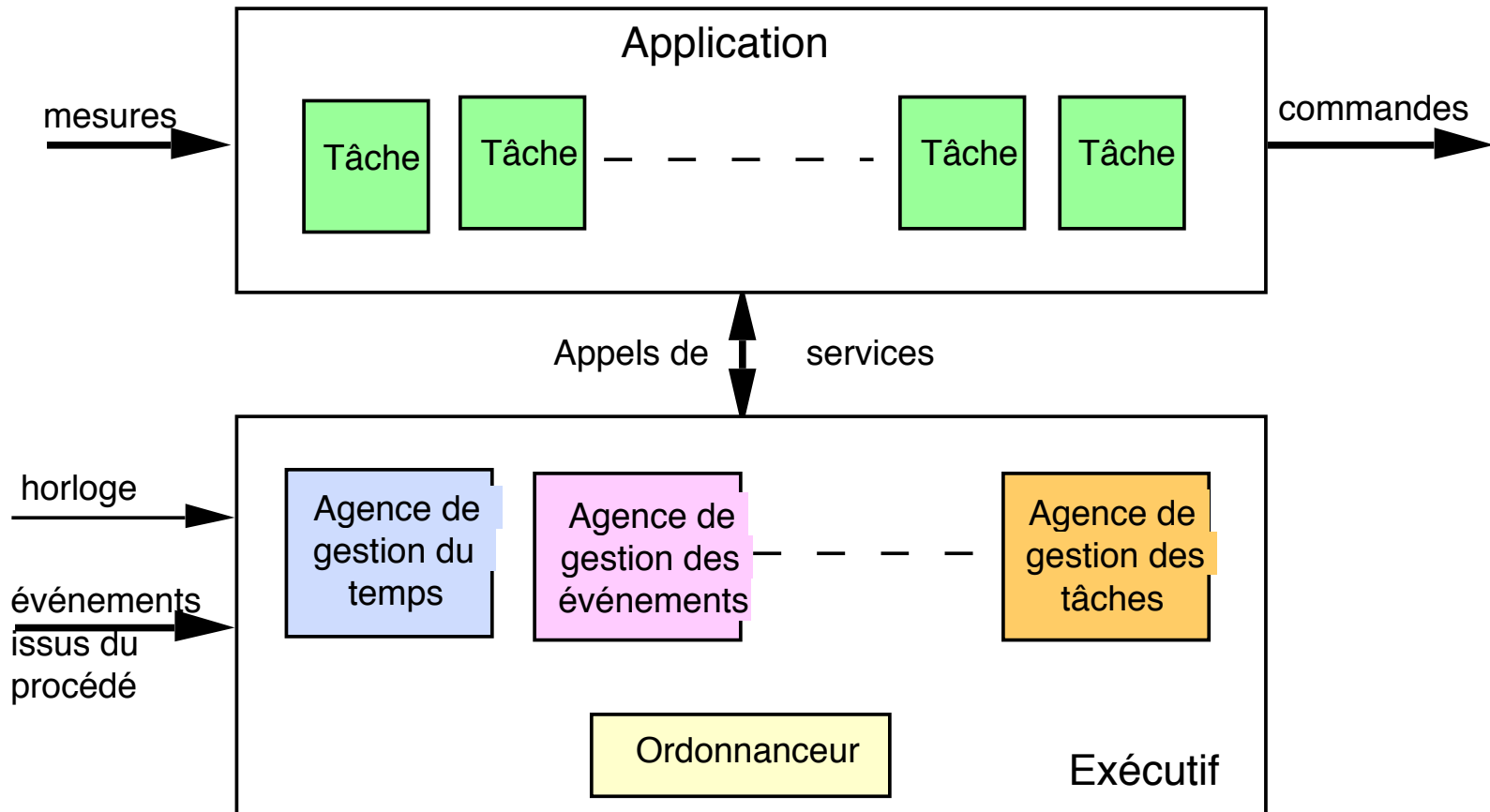
Petits noyaux pour applications embarquées limitées

=> APEX

=> OSEK

Introduction rapide aux OS temps réel

Structure générale :



Généralités sur les OS temps réel

Les principales caractéristiques des noyaux temps réel

- **Conformité à une norme** ou pseudo-norme (POSIX, projet Sceptre)
- **Compacité** (pour les applications embarquées)
- **Environnement cibles** (microprocesseurs, architecture, ...)
- **Environnement hôtes** (type OS)
- **Outils d'aide au développement** (debug, analyse en ligne, ...)
- **Primitives temps réel** (liste de tous les services fournis)
- **Caractéristiques de l'ordonnanceur** (politiques d'ordonnancement)
- **Caractéristiques temporelles :**
 - ***temps de masquage des interruptions*** (*interrupt latency*) : temps pendant lequel les interruptions sont masquées et ne peuvent donc pas être prises en compte (exécution de primitives atomiques, manipulation des structures critiques, ...)
 - ***temps de retard de l'ordonnanceur*** (*preemptive latency*) : temps maximal pendant lequel le noyau peut retarder l'ordonnanceur.
 - ***temps de réponse*** (*task response time*) : temps entre l'occurrence d'une interruption et l'exécution de la tâche réveillée

Généralités sur les OS temps réel

Deux grandes catégories d'O.S. temps réel :

- les OS Temps Réel d'origines :
 - Les OS orientés domaines (aéronautique, automobile...)
 - Les OS temps réel généraux (Tornado, QNX, ...)
- offrent une gestion fine des priorités
- offrent des primitives système rapides, en temps borné (gestion des interruptions, des sémaphores...)
- pas de mémoire virtuelle, mais verrouillage de pages en mémoire centrale
- minimisation de l' « overhead » (le temps pris par le système pour s'exécuter et se gérer lui-même)

=> la solution au Temps Réel Dur

Généralités sur les OS temps réel

Deux grandes catégories d'O.S. temps réel :

=> les O.S. classiques (Unix...) étendus pour le temps réel

- permettent le développement concurrent d'applications temps réel et non temps réel dans un environnement standard et confortable
- Mais il a fallu :
 - revoir les politiques d'ordonnancement
 - renforcer la notion de préemption
 - définir des primitives systèmes réentrantes
 - définir la notion de processus léger (pour faciliter la préemption avec sauvegarde de contexte puis la reprise avec restitution de contexte)

=> modifications importantes et complexes

LynxOS

Unix System V

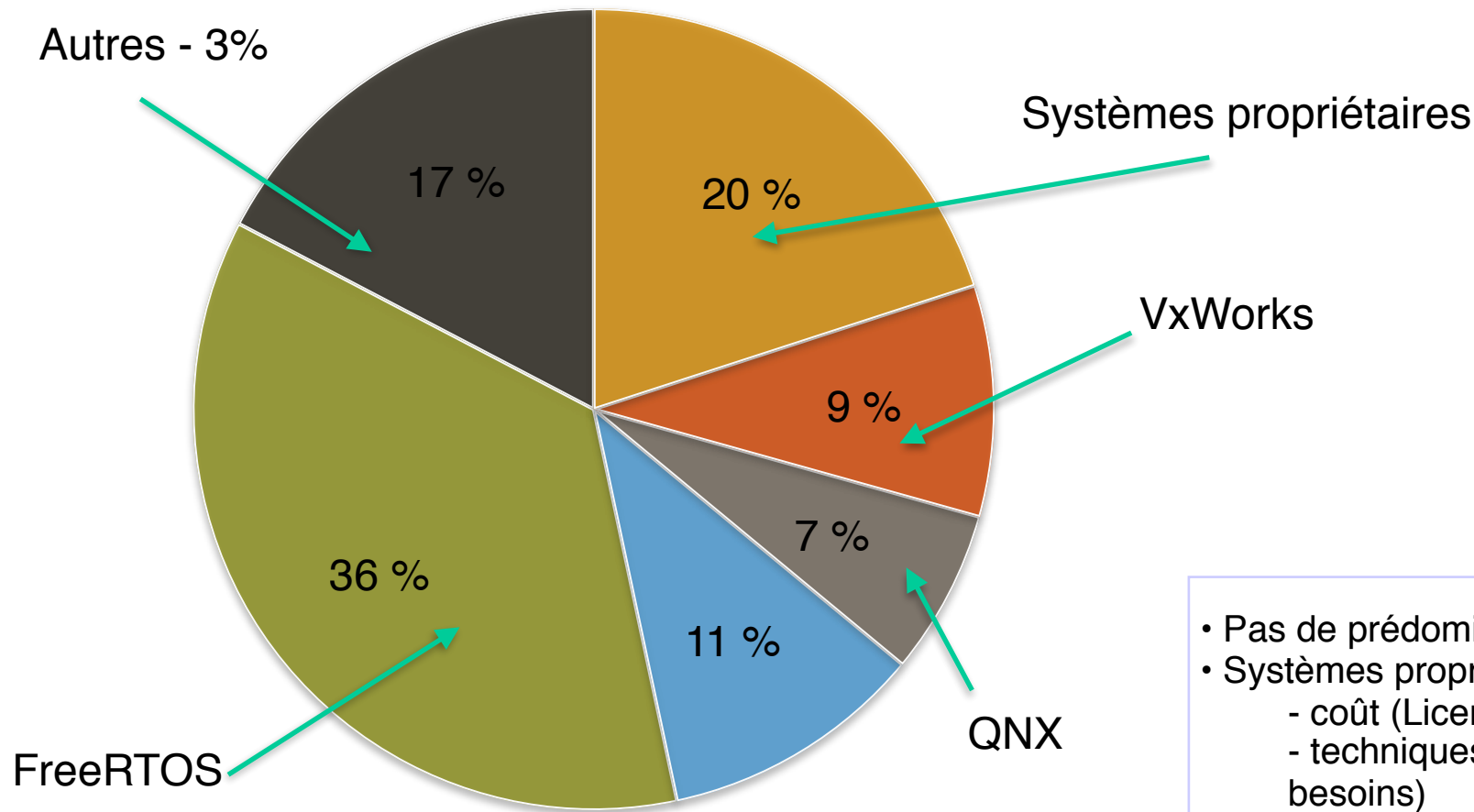
Solaris 2.4

=> pour le temps réel lâche

Généralités sur les OS temps réel

Situation de l'offre industrielle des noyaux temps réel

Embedded Market Study (USA - 2014)



- Pas de prédominance d'un système
- Systèmes propriétaires :
 - coût (Licence)
 - techniques (adaptation aux besoins)
 - stratégie (maîtrise)

Généralités sur les OS temps réel

Comparaison des noyaux temps réel du marché (1998)

Nom	Société	Env. cible	Env. hôte	Compacité	Norme	Ordo.	Application
RTC Real Time Craft	GSI-Tecsi (France)	- 680x0 - Intel x86, Pentium - NS - multiprocesseur	- Windows - Unix	4 Ko.	Sceptre	- 65532 niv. prio. - tourniquet	- automobile, - etc.
pSOS+	Integrated Systems Inc. (USA)	- 680x0 - PowerPC - Intel x86, Pentium - i960 - multiprocesseur	- Windows - Unix	10 Ko. (PSOSelect : 1,8 Ko)	÷ Posix	- 255 niv. prio. - tourniquet	- console de jeu (SEGA), - etc.
VRTXsa Versatile Real Time eXecutive	Integrated Systems Inc. (USA)	- Motorola - Intel	- Unix	13 Ko. (VRTXmc : 4 Ko)	÷ Posix	- 255 niv. prio. - tourniquet - hritage de priorit	- t l phonie - etc.
VxWORKS	Wind River Systems (USA)	- Motorola - Intel - Sparc - RSx000 - É	- Windows - Unix	12 Ko.	Posix	- 255 niv. prio. - tourniquet - hritage de priorit	- automobile - espace (NASA) - etc.
LynxOS	Lynx Real-Time System (USA)	- Motorola - Intel - RSx000 - É	- Windows - Unix	-	Posix	- 255 niv. prio. - tourniquet - hritage de priorit avec priorit plafond	- imprimante (Xerox) - espace (NASA) - etc.
QNX	Inforange ou QNX Systems (Canada)	- Intel	- Windows	-	Posix	- 16 niv. prio. - tourniquet	- tunnel sous la Manche - etc.
CHORUS	Chorus (France)	- Motorola - Intel	- Unix	-	Posix	- 256 niv. prio. - tourniquet	
OS9	Microware Systems (USA)	- Motorola - Intel	- Windows - Unix	-		- 65535 niv. prio. - tourniquet - vieillissement	- radio-t l phone - CD interactif - etc.

2. Le noyau temps réel OSEK/VDX

Le contexte OSEK

Contexte : « l'électronique » embarquée dans les véhicules

contraintes temps réel (dures et souples)

sûreté de fonctionnement élevée

support matériel minimal (peu de RAM, ECU 8 et 16 bits)

architecture distribuée autour de \neq réseaux (CAN, VAN, LIN ...)

fonctions transversales (inter-opérabilité des sous-systèmes)

flexibilité de l'architecture (ajout de fonctions, portabilité et réutilisabilité des fonctions logicielles)

OSEK / VDX : historique

- Proposition du groupe OSEK, constitué :
 - de constructeurs (BMW, DaimlerChrysler, Renault, PSA, etc.)
 - d'équipementiers (Bosch, Siemens, etc.)
 - d'universitaires (Univ. Karlsruhe)
- Fusion des projets OSEK (Allemand) et VDX (GIE PSA-Renault)
- Englobe le projet Européen MODISTARC (processus de certification des implémentations conformes)
- Travaux commencés en 1995 et encore en cours

Le site Web de référence : <http://www.osek-vdx.org>

Principaux service d'OSEK OS

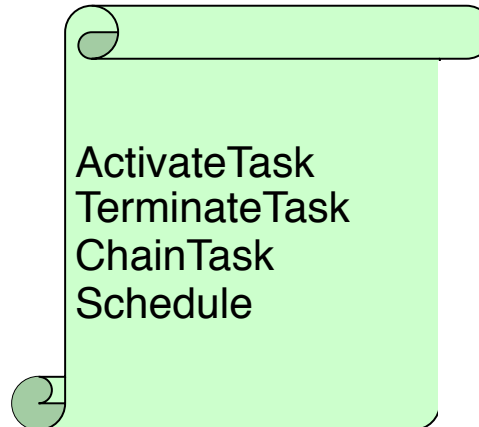
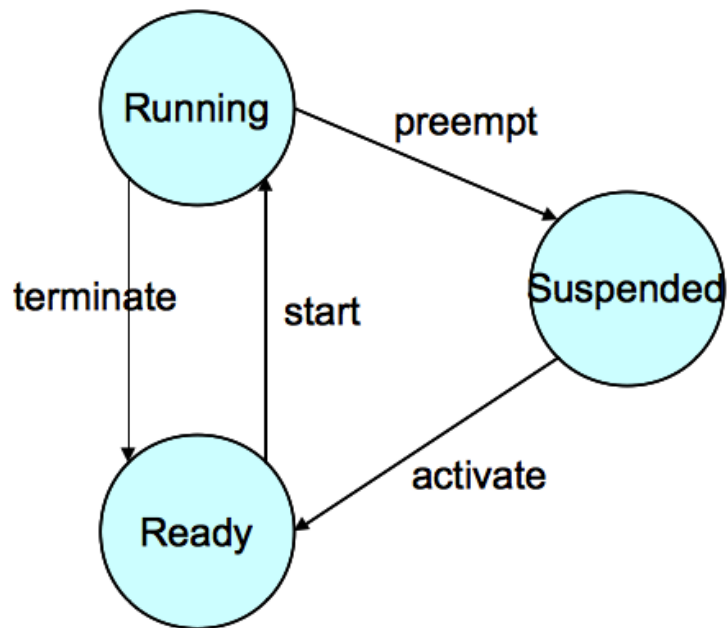
- Services pour les tâches
- Services de synchronisation (événements)
- Services d'exclusion mutuelle
- Services pour les phénomènes récurrents (compteurs et alarmes)
- Service pour la communication (dans OSEK/VDX COM)
- Services pour la gestion des interruptions
- Services systèmes et gestion des erreurs

=> Tous les objets sont statiques

Les tâches

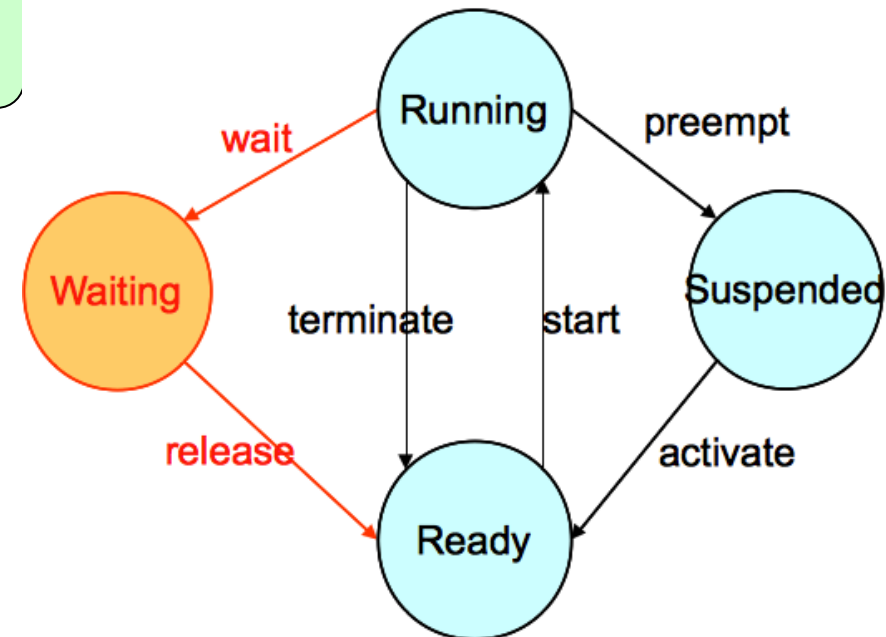
tâches basiques

= tâche sans
points bloquants



tâches étendues

= tâche basique comportant
des points bloquants



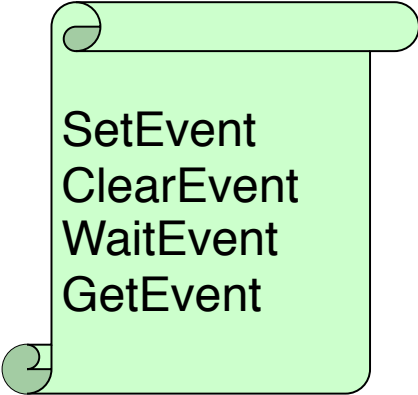
=> Possibilité de mémoriser les requêtes d'activations

L'ordonnancement

- Priorité fixe non modifiable
- 3 modes d'ordonnancement :
 - préemptif
 - non préemptif
 - mixte (préemptif pour certaines tâches et non préemptif pour d'autres)
- Gestion des ressources partagées avec « Priority Ceiling Protocol »
(héritage de priorité + évitement des inter blocages par attribution de priorité aux ressources et règle d'allocation de ressource)

Synchronisation par événements

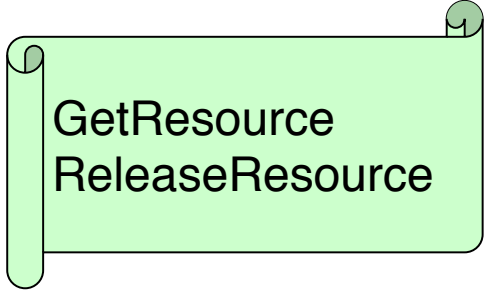
- Événements privés : 1 événement est la propriété d'une tâche (étendue)
- Modèle n producteurs / 1 consommateur
- Attente explicite des consommateurs (synchrone)
- Occurrences mémorisées, effacement explicite
- Attente possible en OU sur une liste d'événements



SetEvent
ClearEvent
WaitEvent
GetEvent

L'exclusion mutuelle

- Services pour prendre et relâcher explicitement une ressource
- Utilisation du protocole PCP pour éviter :
 - les inversions de priorités
 - les blocages inter-tâches
- Une ressource standard : Res_scheduler (passage en mode non-préemptif)



GetResource
ReleaseResource

Alarmes et compteurs

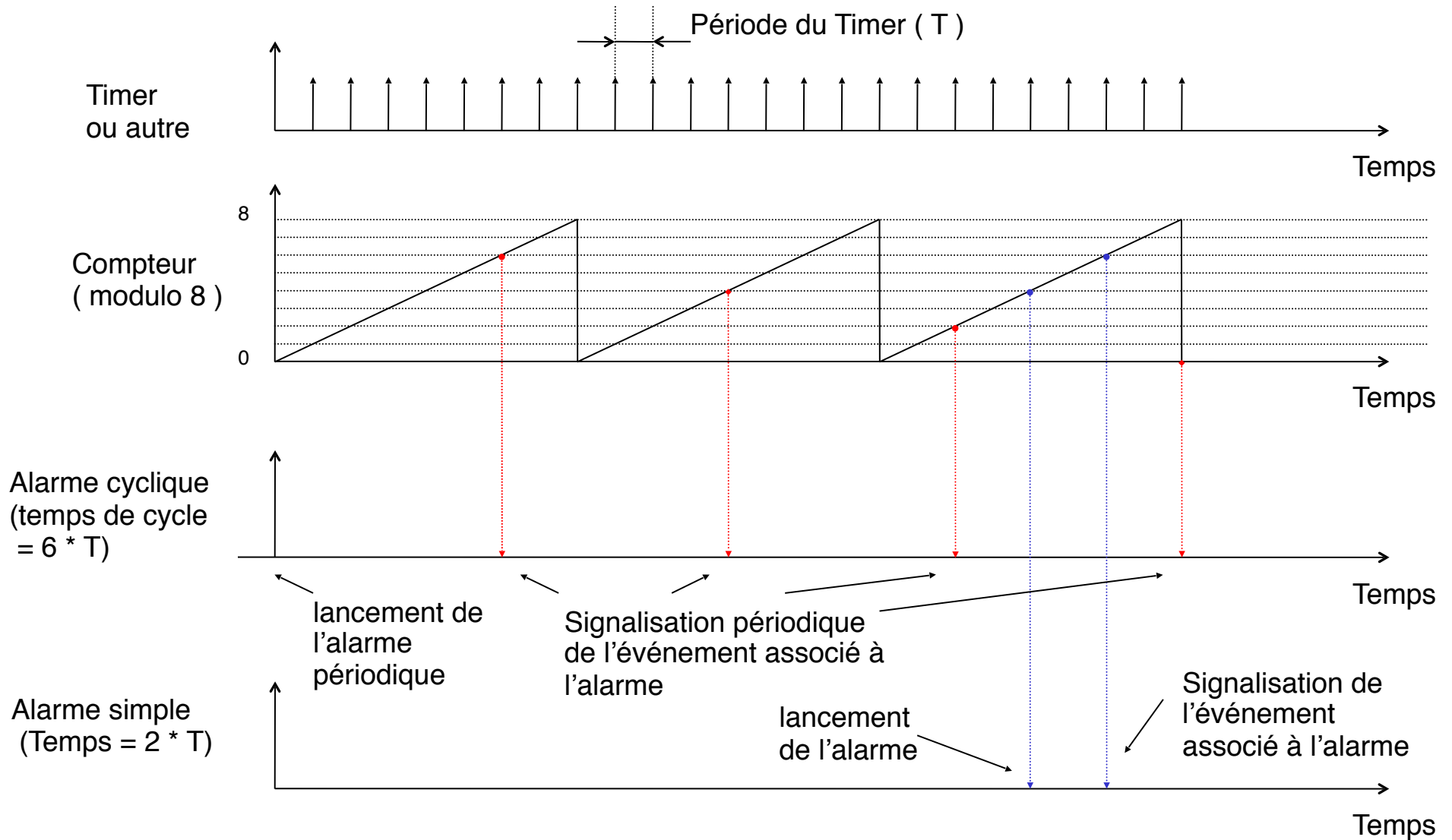
Les compteurs

- enregistrement de « ticks » externes, éventuelle pré-division
- compteur fini avec RAZ automatique

Les alarmes

- attachée à 1 compteur et 1 tâche
- unique ou cyclique, déclenchement absolu ou relatif

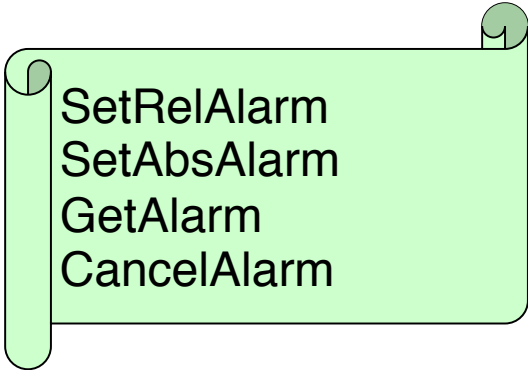
Alarmes et compteurs



Alarmes et compteurs

Applications :

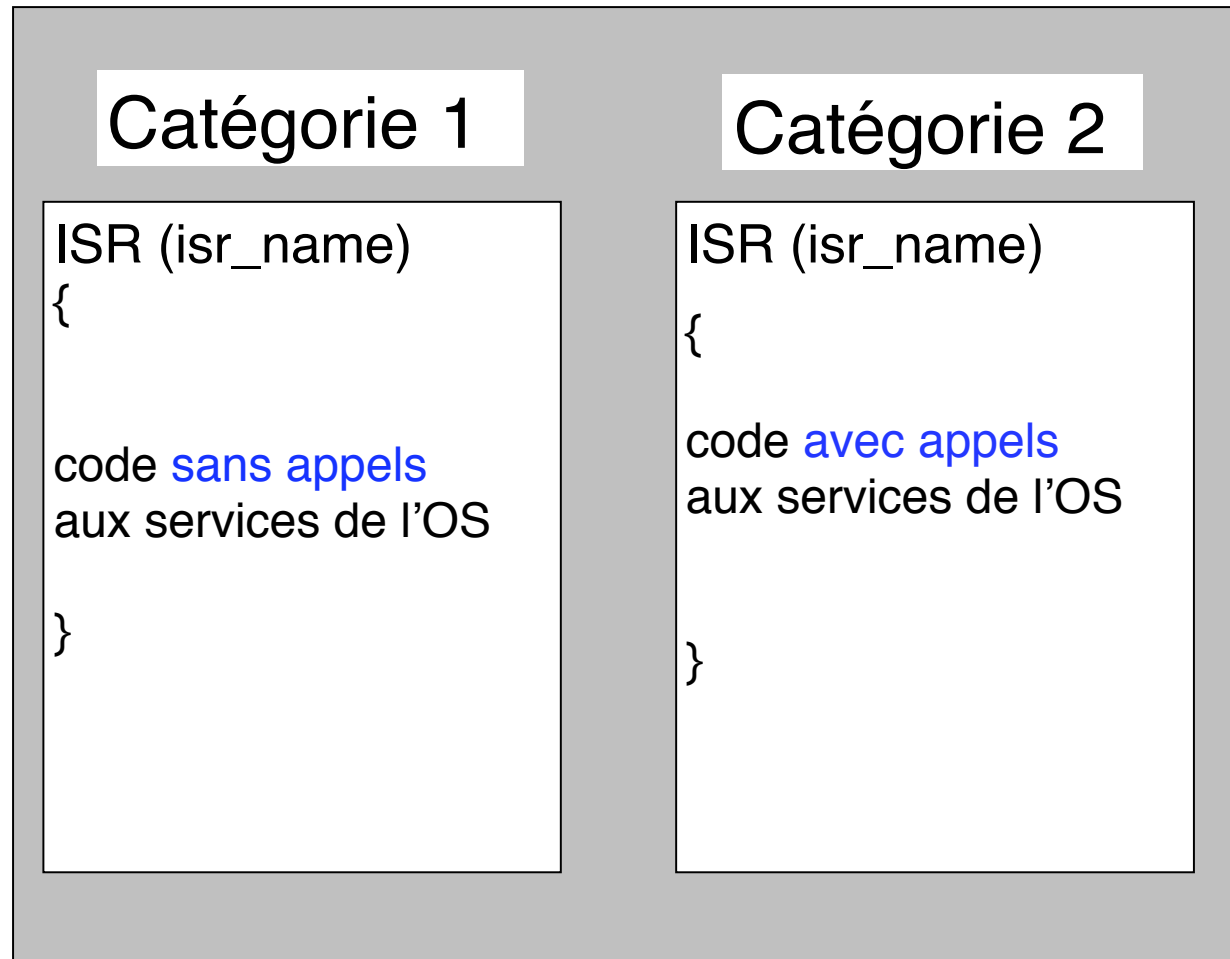
- activation de tâches périodiques ou non
- signalisation d'occurrences d'événement périodiques ou non périodiques
- garde temporelle



- SetRelAlarm
- SetAbsAlarm
- GetAlarm
- CancelAlarm

Interruption

Deux trames de prise en compte des interruptions



Les aspects communication

- Communication par messages
 - 1 message = 1 nom + 1 type de données + des attributs
- Modèle de communication asynchrone :
 - SendMessage : écriture d'une nouvelle valeur
 - ReceiveMessage : lecture de la valeur courante
- Transmission effective en fonction des attributs
- ➔ Resynchronisation nécessaire :
 - polling de l'état du message, activation de la tâche ou occurrence d'év. sur fin d'envoi ou de réception...

Les aspects communication

Quelques attributs des messages :

- UnqueuedMessage : gestion type « tableau noir »
- QueuedMessage : gestion type « boîte aux lettres »
- Transmission « à la demande »
- Transmission périodique
- Transmission mixte
 - ➡ périodique + émission en cas de changement

Les aspects communication

Quelques attributs des messages :

- UnqueuedMessage : gestion type « tableau noir »
- QueuedMessage : gestion type « boîte aux lettres »
- Transmission « à la demande »
- Transmission périodique
- Transmission mixte
=> périodique + émission en cas de changement

3 types de communications :

- 1 : 1 (1 destinataire statique)
- 1 : 1 parmi n (choix dynamique dans une liste statique)
- 1 : n (diffusion à n destinataires)

2 protocoles (non acquittés) :

- UUDT : non segmenté
- USDT : segmenté

Configuration des tâches OSEK : OIL

- Langage OIL : OSEK Implementation Language
- Notion de Hook Routine pour contrôler, de façon temporaire, le système
- Configuration des types de tâches : conformity classes
 - BCC 1 : Tâches basiques uniquement
 - 1 seule tâche active et 1 seule tâche par niveau de priorité
 - BCC 2 : Tâches basiques uniquement
 - +sieurs tâches actives et +sieurs tâches par niveau
 - ECC 1 : BCC 1 + Tâches étendues
 - ECC 2 : BCC 2 + Tâches étendues

Configuration des tâches OSEK : OIL

- Possibilité de découper en plusieurs fichiers OIL

Ex: #include « implementation.oil »

- Les objets :
 - définissent les paramètres de configuration
 - un objet principal : CPU

```
CPU ATMEL_AT91SAM7S256
{
    OS LEJOS_OSEK {
        ...
    };
    APPMODE appmode1;
    TASK task1 {
        ...
    };
    ...
}
```

Quelques objets OIL

- Objet OS

- Un seul OS par CPU
- Définit la configuration d'OSEK (hook routines, scheduler)

```
OS LEJOS_OSEK {  
    STATUS                = EXTENDED;  
    STARTUPHOOK            = FALSE;  
    ERRORHOOK              = FALSE;  
    SHUTDOWNHOOK           = FALSE;  
    PRETASKHOOK            = FALSE;  
    POSTTASKHOOK           = FALSE;  
    USEGETSERVICEID        = FALSE;  
    USEPARAMETERACCESS     = FALSE;  
    USERESSCHEDULER        = FALSE;  
};
```

- Objet APPMODE (= appmode1 {};

- Définit le mode d'application des tâches

Définition des tâches

- Objet Task

```
TASK taskname {  
    AUTOSTART = FALSE /* la tâche ne se lance pas  
                        au démarrage du systèmes */  
    PRIORITY = 3; /* priorité de la tâche,  
                  1 = priorité la plus faible */  
    ACTIVATION = 1; /* nombre de fois qu'une tâche  
                    peut être présente dans la  
                    liste des tâches prêtes */  
    SCHEDULE = FULL; /* NONE = pas de préemption  
                     FULL = préemption possible*/  
    STACKSIZE = 512; /* Taille de la pile, ici 512*/  
};
```

- Autres champs

- RESSOURCE = ressource1 (accès à la ressource n°1)
- EVENT = event1 (attente d'un événement)
- MESSAGE = message1 (synchronisation par message)

Définition des tâches

- Définition de la durée d'un tick : objet COUNTER

```
COUNTER SysTimerCnt {  
    MINCYCLE = 1; /* au min, période d'un tick */  
    MAXALLOWEDVALUE = 100; /* val. max. du cpt */  
    TICKPERBASE = 1; /* pas du cpt */  
};
```

- Réveil des tâches périodiques :

- Une alarme par tâche
- Un seul compteur associé à une alarme

```
ALARM cyclic_alarm1 {  
    ACTION = ACTIVATETASK {  
        TASK = taskname;  
    };  
    AUTOSTART = TRUE {  
        ALARMTIME = 1; /* 1ère instance d'alarme */  
        CYCLETIME = 1; /* période, ici 1 tick */  
        APPMODE = appmodel;  
    };  
};
```

Conclusion

- Une proposition mature et complète
- Des produits industriels
- Un pas important vers la portabilité des applications, la réutilisation de composants dans les ECU ...
- Variantes : OSEKtime OS
 - ➡Tâches de type TT (time triggered) pour applications critiques
 - ➡rejoint le modèle APEX pour le niveau partition

3. APEX : le standard ARINC 653

Introduction

Spécifier une interface entre un OS d'une ressource
avionique et des logiciels d'application : interface APEX
(APplication / EXecutive)

Début des travaux : 1991

Publication du premier standard ARINC 653 : été 1996

Définitions

Application :

- Réalisation logicielle d'une fonction avionique
- constituée de une ou plusieurs partitions

Exemple : Pilote Automatique

Partition :

- Entité d'exécution d'une application
- S'exécute sur un seul processeur
- Allocation déterministe des ressources aux partitions
- Unité de ségrégation spatiale (mémoire) et temporelle (CPU)
- Constituée de un ou plusieurs processus

Exemple : FM1, FM2, FE1-COM, FE1-MON, FE2-COM, FE2-MON, FG1-COM, FG1-MON, FG2-COM, FG2-MON

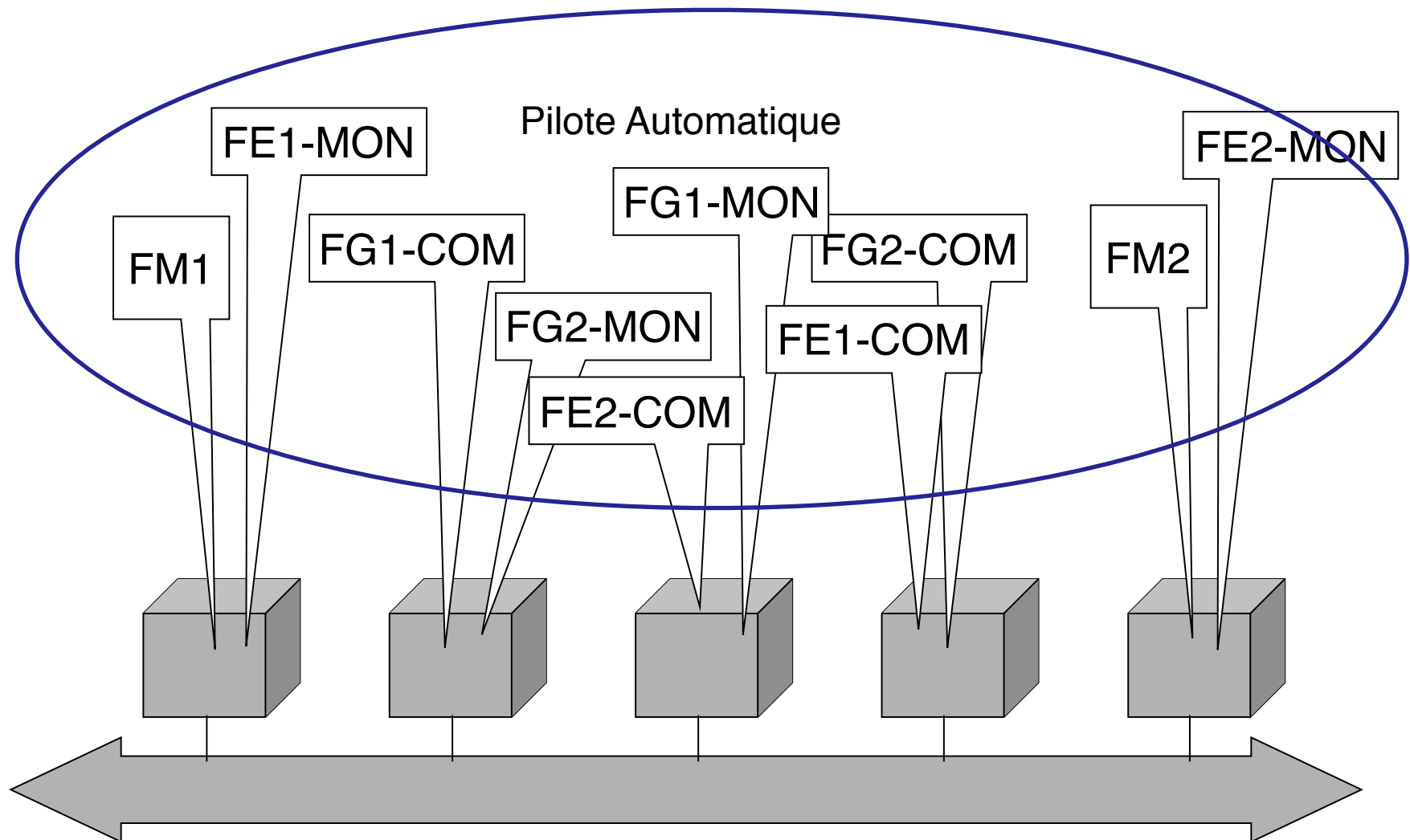
Définitions

Processus :

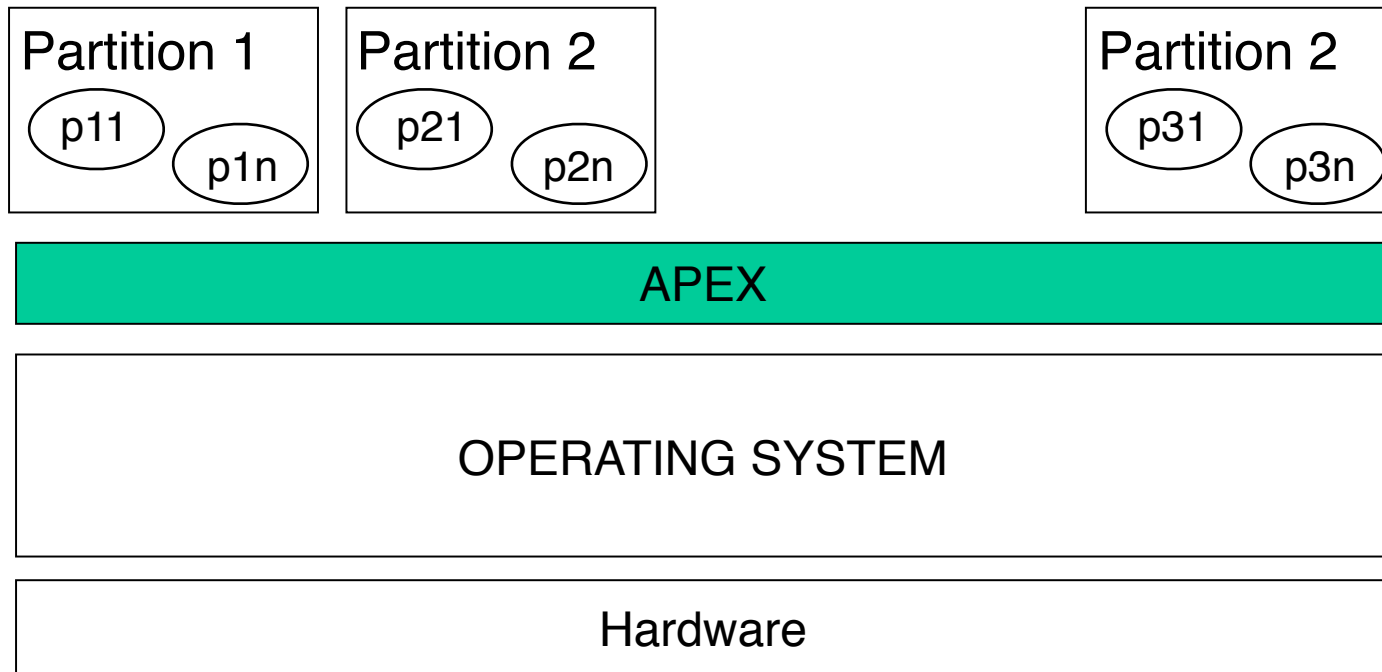
- Unité de programme qui s'exécute dans l'environnement de la partition
- Exécution concurrente des processus pour réaliser la fonction avionique

Exemple : une loi de pilotage, une logique de contrôle...

Architecture



Architecture



Operating system :

Ordonnance les partitions du module

Ordonnance les processus dans la partition

Assure la ségrégation (partitionnement) spatiale et temporelle

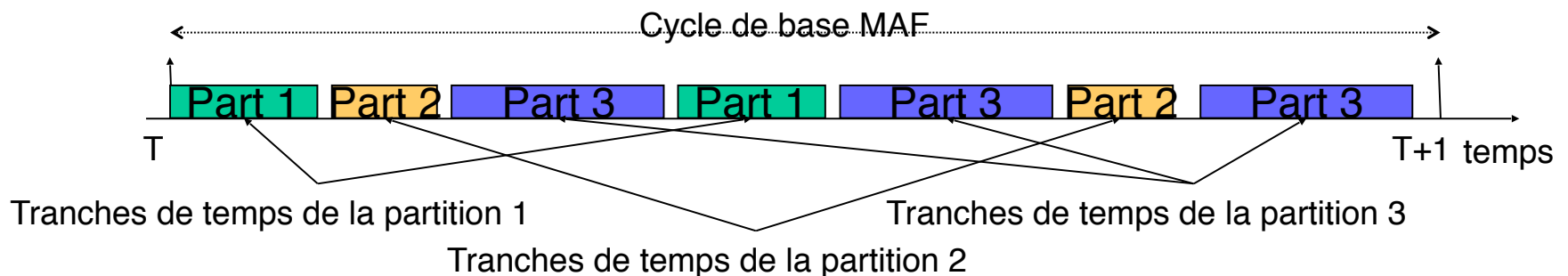
Partage des ressources

Partage spatial :

Zone mémoire prédéterminée pour chaque partition

Partage temporel (CPU) :

- Une partition n'a pas de priorité
- Allocation déterministe et cyclique du processeur aux partitions :
 - => L'OS répète un cycle de base (MAJOR time Frame: MAF) de durée fixe
 - => Attribution à chaque partition d'une ou plusieurs tranches de temps dans la MAF



=> L'allocation des ressources est définie par configuration et n'est pas modifiable dynamiquement

Gestion des partitions

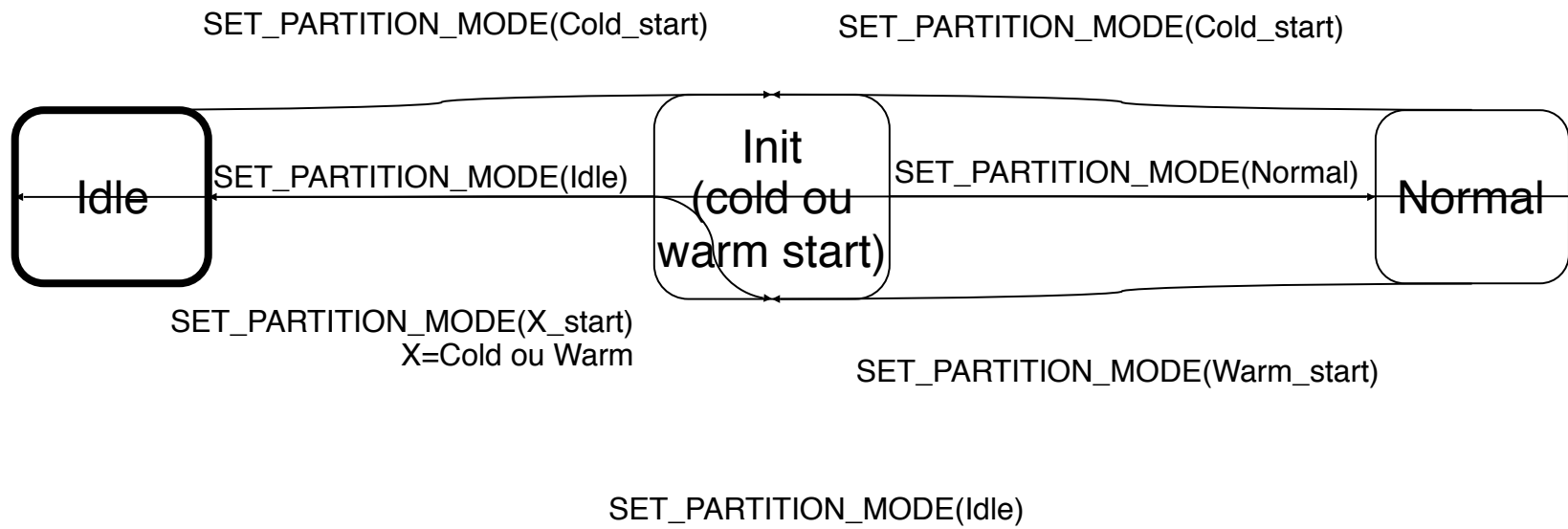
- L'OS démarre l'ordonnanceur de partition en fin d'initialisation de module
- Pendant ses tranches de temps, une partition a les états suivants
 - Idle :
 - La partition est au repos, aucun processus de s'exécute
 - Cold_start ou Warm_start : initialisation de la partition
 - Démarrage à froid : abstraction du contexte précédent de la partition
 - Démarrage à chaud : récupération du contexte précédent de la partition
 - Normal : mise en route de l'ordonnancement des processus
- Hors de ses tranches de temps une partition est suspendue
- La partition crée tous ses objets (ports, process, sémaphores...) pendant sa phase d'initialisation
- La partition démarre l'ordonnancement des processus à la fin de sa phase d'initialisation

Gestion des partitions

Service APEX des partitions

SET_PARTITION_MODE : changement de mode (Idle, Cold ou Warm start, Normal)

GET_PARTITION_STATUS : donne le mode de la partition



Gestion des processus

- Deux types de processus
 - Périodique : exécution à intervalles réguliers
 - Apériodique : exécution sur occurrence d'événement
- Pas de ségrégation entre les processus d'une même application
- Les processus d'une partition ne sont pas visibles des autres processus des autres partitions
- Chaque processus a une priorité
- L'ordonnancement des processus est basé sur un algorithme de préemption par niveau de priorité et état courant de processus
- Un seul processus par partition est « running » à chaque instant

Gestion des processus

Service APEX des processus

CREATE_PROCESS :

Création d'un lien entre le nom du processus et sa réservation à l'initialisation de la partition

Mise à l'état *Dormant* du processus et retour d'un identificateur (id)

START :

Initialisation du processus

Passage de l'état *Dormant* à l'état *Ready*

STOP :

arrêt d'un processus par suppression des droit d'accès au processeur

Passage de l'état *Ready* ou *Waiting* à l'état *Dormant*

STOP_SELF :

Auto-arrêt du processus en cours d'exécution

Passage de l'état *Running* à l'état *Dormant*

SUSPEND :

Suspension du processus jusqu'à reprise par un autre processus

Passage de l'état *Running* à l'état *Waiting*

SUSPEND_SELF :

auto-suspension du processus

Passage de l'état *Running* à l'état *Waiting*

Gestion des processus

Service APEX des processus

RESUME :

Relance d'un processus suspendu

Si le processus est déjà en attente d'une ressource, il reste à l'état *Waiting*,
sinon passage à l'état *Ready*

SET_PRIORITY :

Changement de la priorité d'un processus

LOCK_PREEMPTION :

Incrémentation du compteur de préemption de la partition et blocage du
mécanisme de préemption

UNLOCK_PREEMPTION :

Décrémentation du compteur de préemption de la partition et déblocage du
mécanisme de préemption si ce compteur est nul

GET_PROCESS_ID :

retourne l'identificateur d'un processus en donnant son nom

GET_PROCESS_STATUS :

retourne des informations d'état sur un processus.

Gestion des processus

Gestion du temps

Le temps est exprimé en temps réel (absolu)

L'OS assure la ségrégation temporelle entre les partitions

Les services APEX pour gérer le temps :

TIMED_WAIT :

- Suspension d'un processus pour un temps minimal donné

- Passage de l'état Running à Waiting

- Un temps égal à zéro effectue un "Round Robin" sur les processus de même priorité

PERIODIC_WAIT

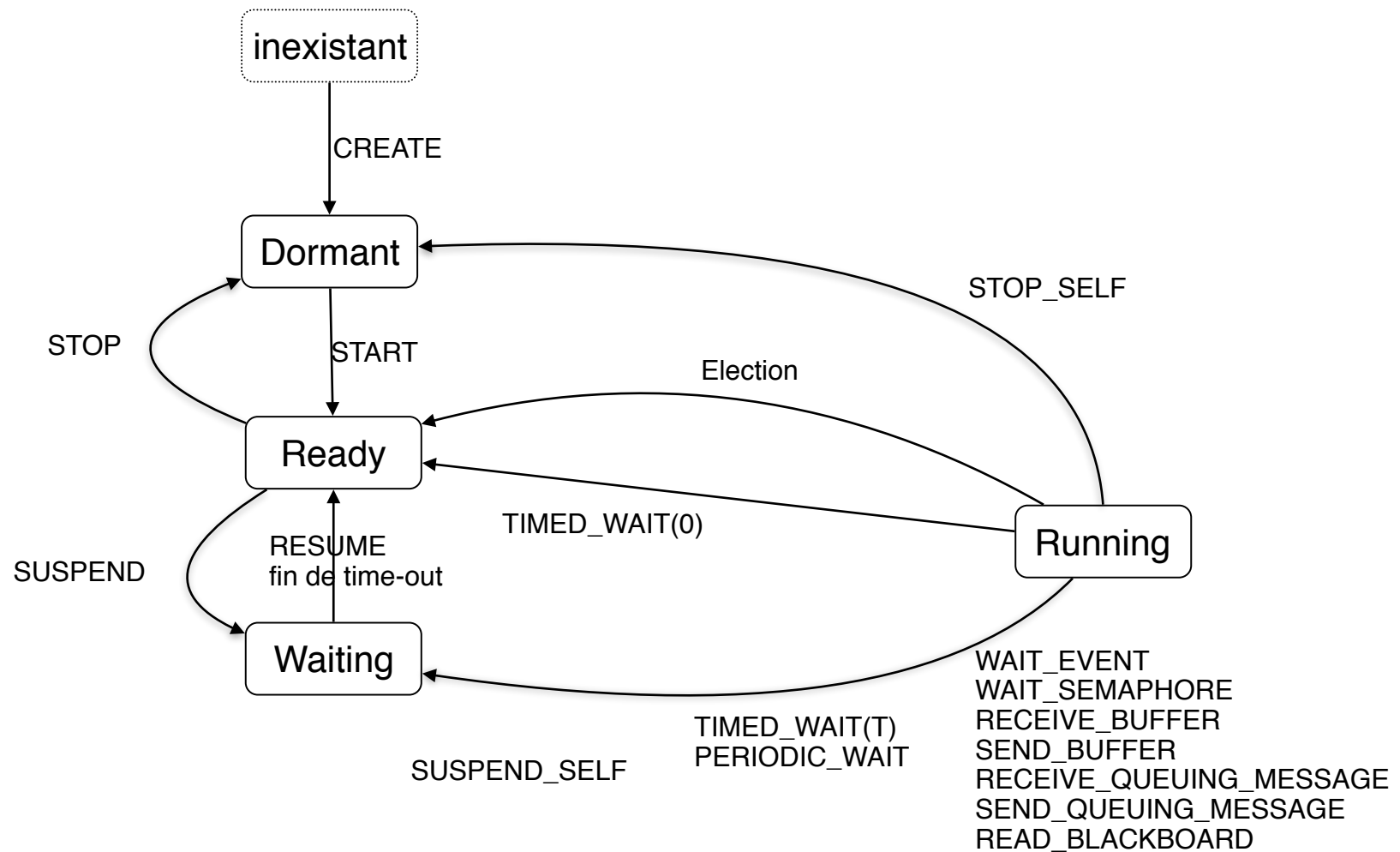
- Suspension d'un processus périodique jusqu'à la prochaine période
- réallocation du budget temps du processus au début de la période

GET_TIME

- retourne l'heure (absolue) et l'heure locale au module

Gestion des processus

Graphe d'état d'un processus



Communication entre processus

Les processus d'une même partition peuvent communiquer et se synchroniser par

- communication par boîtes aux lettres du type Buffer ou Blackboard
- utilisation de variables globales à la partition
- synchronisation par mécanismes de sémaphore et d'événement

Une boîte aux lettres permet l'échange de message entre plusieurs processus sans indiquer le noms des processus émetteurs et récepteurs

Une zone mémoire est réservée à l'initialisation de la partition pour ces objets de communication / synchronisation

Les processus peuvent créer autant d'objet de communication intra-partition que le permet la place mémoire réservée à l'initialisation

Communication entre processus

Les boîtes aux lettres de type "Buffer" :

communication par message pouvant transporter des données différentes

une occurrence de message n'écrase pas les occurrences précédentes

les messages sont stockés dans des queues de messages de type FIFO

les queues de message de sont longueurs bornées

une queue de message peut être pleine

les processus cherchant à lire dans une queue de messages vide sont mis en attente

les processus cherchant à émettre dans une queue de message pleine sont mis en attente

Services APEX correspondants :

CREATE_BUFFER

SEND_BUFFER

RECEIVE_BUFFER

GET_BUFFER_ID

GET_BUFFER_STATUS



possibilité de spécifier un time out

Communication entre processus

Les boîtes aux lettres de type "Blackboard" :

communication écriture de données

une écriture de donnée écrase la donnée précédente

une donnée écrite reste affichée jusqu'à l'écriture de la prochaine donnée ou demande d'effacement

tous les processus en attente sur un "Blackboard" vide sont réveillés lors de l'écriture d'une donnée (passage de l'état Waiting à Ready)

Services APEX correspondants :

CREATE_BLACKBOARD

DISPLAY_BLACKBOARD

READ_BLACKBOARD

CLEAR_BLACKBOARD → possibilité de spécifier un time out

GET_BLACKBOARD_ID

GET_BLACKBOARD_STATUS

Communication entre processus

Les Sémaphores :

=> sémaphores à compte

permet de contrôler l'accès des processus aux ressources de la partition

un processus attend le sémaphore pour accéder à la ressource et signale le sémaphore quand il libère la ressource

si compteur > 0 : la valeur représente le nombre de processus pouvant accéder à la ressource

si compteur $= 0$: ressource indisponible

les processus en attente sur un sémaphore sont rangés soit par FIFO soit par priorité

Services APEX correspondants :

CREATE_SEMAPHORE

WAIT_SEMAPHORE

SIGNAL_SEMAPHORE —————> possibilité de spécifier un time out

GET_SEMAPHORE_ID

GET_SEMAPHORE_STATUS

Communication entre processus

Les Événements :

=> événements à niveau (UP et DOWN)

permet de synchroniser des processus de la partition

lorsqu'un événement est positionné à UP, tous les processus en attente sur cet événement passent à l'état *Ready* et un nouvel ordonnancement des processus a lieu

lorsqu'un processus est en attente d'un événement positionné à DOWN, il passe à l'état *Waiting* avec éventuellement attente d'un time out

Services APEX correspondants :

CREATE_EVENT

SET_EVENT

RESET_EVENT

WAIT_EVENT

GET_EVENT_ID

GET_EVENT_STATUS

—————> possibilité de spécifier un time out

Communication entre partitions

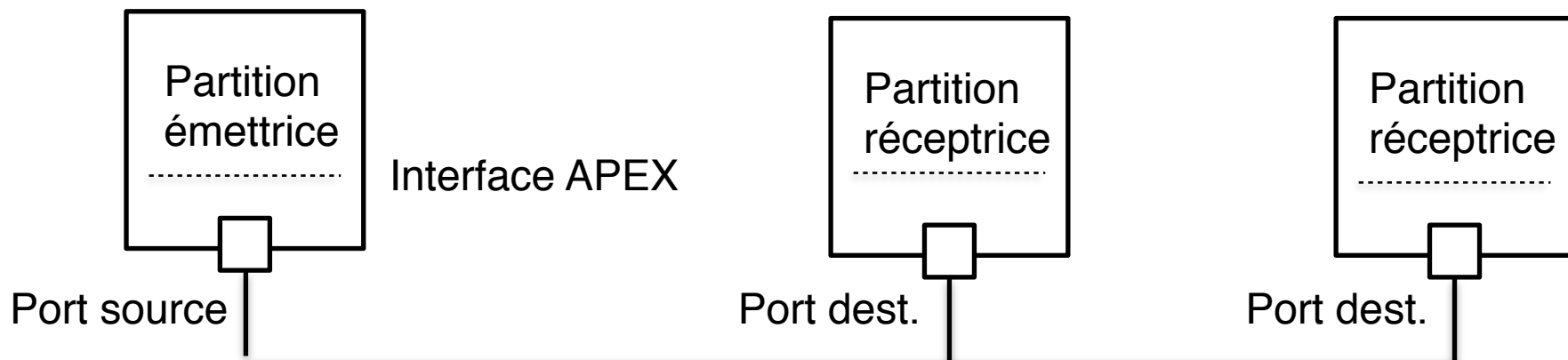
Les communications entre partitions (d'un même module ou non) se font uniquement par échange de message

Un émetteur est relié à un ou plusieurs récepteur par un canal logique appelé "Channel"

Les entités d'émission ou de réception correspondent aux partitions

Une partition accède à un Channel via un "Port"

=> Service APEX d'émission et de réception via des ports



Communication entre partitions

- La partition émettrice ne connaît pas le nom et la localisation des partitions réceptrices
- La connexion physique entre le port d'émission et les ports de réception est établie par configuration statique du réseau
- Le protocole de communication entre émetteur et récepteurs doit être le même
(même format des message, même politique d'acquittement...)
- L'appel périodique d'un service d'émission APEX génère un message périodique, et inversement, un appel apériodique génère un message apériodique
=> la périodicité n'est pas un attribut du port

Communication entre partitions

Protocole de transfert

- Sampling mode :
 - le message véhicule toujours la même donnée mise à jour
 - chaque occurrence d'un message écrase le précédent
 - les messages sont de taille fixe
 - les messages sont non segmentables par l'OS

- Queuing mode :
 - le message peut véhiculer des données différentes
 - les occurrences sont sauvegardées dans une FIFO
 - les messages peuvent être de longueur variable
 - les messages sont segmentables par l'OS

Communication entre partitions

Attributs des ports

- Identifier (id) : valeur retournée lors de la création du port
- Nom
- Protocole de transfert : sampling ou queuing
- Sens du transfert : input ou output
- Taille maximale des messages
- Taille du port
 - identique à la taille des messages pour les ports sampling
 - profondeur de la FIFO pour les ports queuing
- Période de rafraîchissement pour les ports sampling
 - permet de surveiller la fraîcheur du dernier message reçu
- Politique d'attente en mode queuing :
 - FIFO ou priorité

Communication entre partitions

Service APEX sampling

CREATE_ SAMPLING_PORT
WRITE_ SAMPLING_MESSAGE
READ_ SAMPLING_MESSAGE

lecture du dernier message reçu dans le port et indique si l'age du message est cohérent avec l'attribut Refresh period du port

GET_ SAMPLING _ID
GET_ SAMPLING _STATUS

Service APEX queuing

CREATE_ QUEUING_PORT
SEND_ QUEUING _MESSAGE
RECEIVE_ QUEUING _MESSAGE
GET_ QUEUING _ID
GET_ QUEUING _STATUS

—————>possibilité de spécifier un time out

Conclusion sur APEX

Synthèse :

- Un OS à deux niveaux :
 - partition + ordonnancement cyclique statique
 - processus + ordonnancement dynamique par priorité
- Objectif intuitif : émuler des architectures fédérées classiques où chaque fonction dispose de ses propres ressources et gère ses propres processus comme elle l'entend
- Niveau partition rendu nécessaire pour des questions de certification
 - => garantir qu'une fonction défaillante ne pourra jamais en perturber une autre
 - => notion de partitionnement stricte

Interrogation : n'aurait-il pas été plus simple de supprimer le niveau partition et de multiplier les ressources physiques ?

- => Avionique Modulaire non intégrée
- => Seul partage de ressource = le réseau

Présentation de VxWorks

- OS temps réel d'origine généraliste

=> Cible : applications embarquées (plateforme NASA...)

- temps de latence faible pour le traitement d'It et le changement de contexte
- Surcharge « appel system » faible
- Modulable
- Offre la possibilité de développement croisé à partir d'une station Unix
- Adhère à POSIX
- Programmation en C, C++, Java, ...
- Disponible sur bus VME & PCI, sur les cibles Motorola 680x0 & 683xx, Intel ix86 & i960, MIPS, SPARC, PowerPC, ...

Présentation de VxWorks

- Caractéristiques :

- Multitâche avec ordonnancement préemptif basé sur priorité, différents moyens de synchronisation/communication intertâche, supporte l'interception d'interruptions, les chiens de garde, et la gestion mémoire.

- Compatible POSIX :

- la plupart des interfaces sont spécifiées par le standard 1003.1b
⇒ portabilité

- Système d'ES :

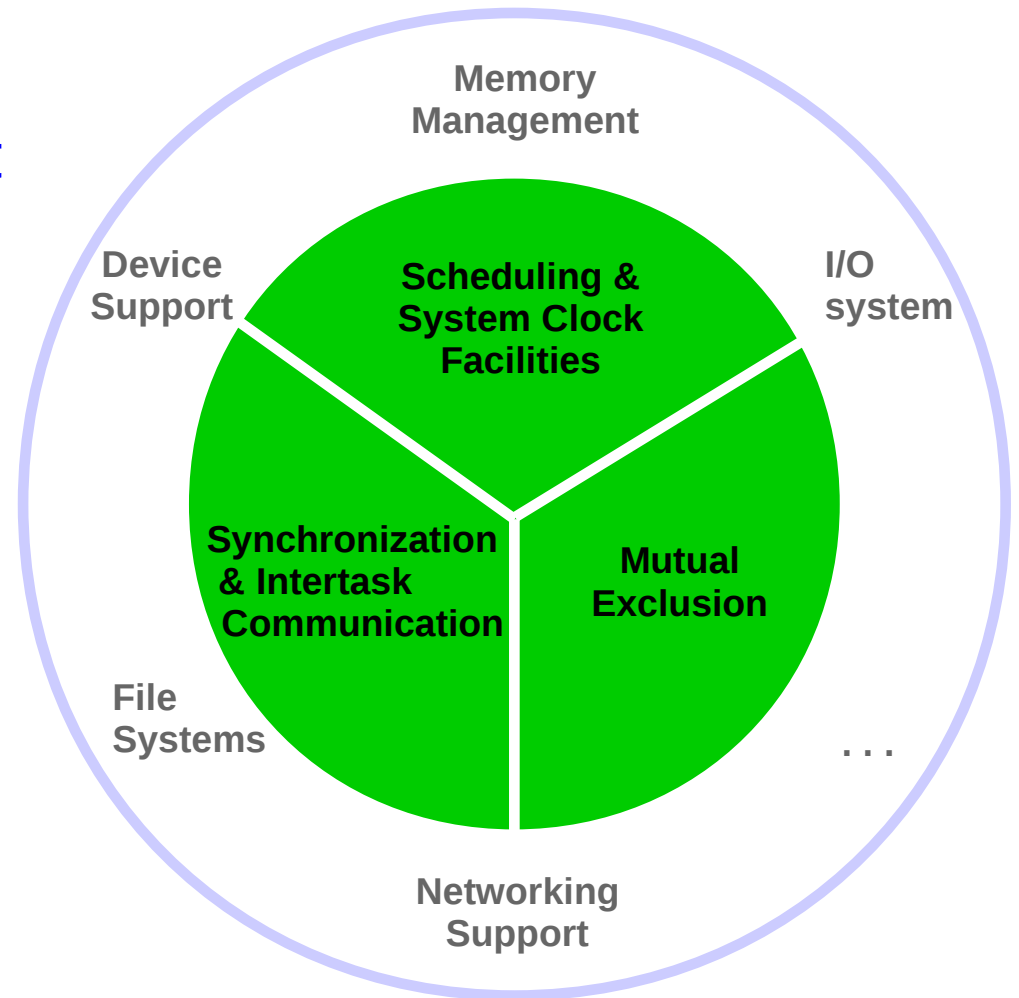
- rapide et flexible, compatible ANSI C
- inclut les E/S bufferisées standard Unix et les E/S asynchrones standard POSIX

Présentation de VxWorks

- Consiste en un micro-noyau et des utilitaires périphériques

➡Modulable

➡Adaptable au besoin

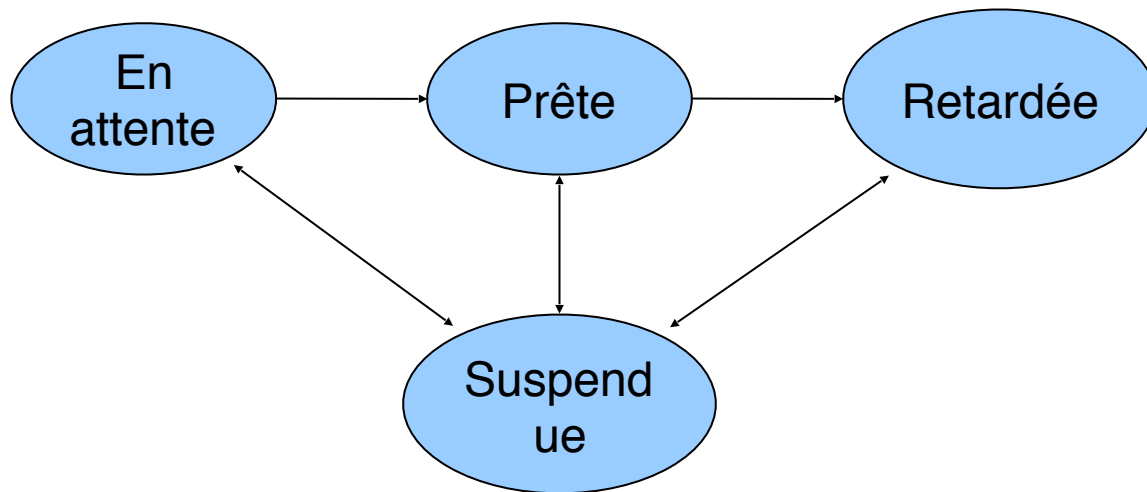


Qu'est-ce qu'une tâche VxWorks

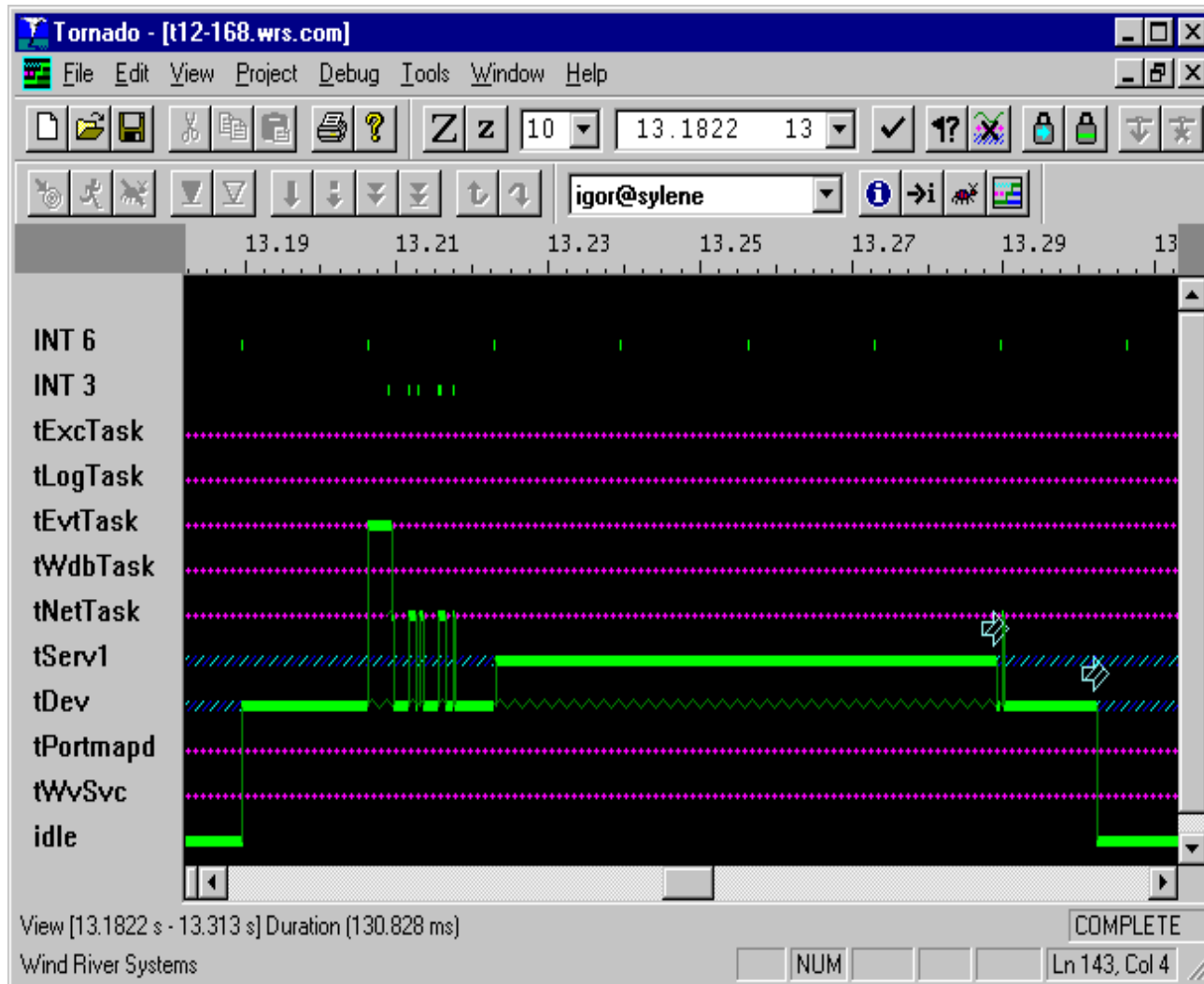
- Une tâche a un contexte d'exécution.
Elle possède :
 - un pointeur d'instruction (position courante de l'exécution)
 - une pile pour les variables locales, les arguments de fonctions,...
- Seule une tâche peut être en cours d'exécution à un moment donné.
 - Quand une tâche ne s'exécute pas, son contexte est stocké dans son Task Control Block (TCB).
 - Le TCB est la structure de données que le noyau utilise pour représenter et contrôler les tâches.

Gestion des tâches

- Nombre illimités de tâches
- 256 niveaux de priorités
- Ordonnancement préemptif + round-robin
- Temps de changement de contexte < quelques μ s



Ordonnancement préemptif basé sur les priorités



- A tout instant, la tâche *prête* de plus forte priorité s'exécute.
- Une tâche de plus forte priorité qui passe prête préempte la tâche en cours d'exécution.
- Les interruptions préemptent toute tâche.

Communication intertâche

Les tâches doivent coordonner leurs exécutions

VxWorks offre un ensemble de mécanismes de communication :

mémoire partagée : partage simple des données (entre toutes les tâches)

sémaphores : exclusion mutuelle basique et synchronisation

files de messages et **pipes** : transfert de messages intra CPU

sockets et **remote procedure calls** : transfert via réseau

signals : traitement d'exceptions

Communication intertâche : sémaphores

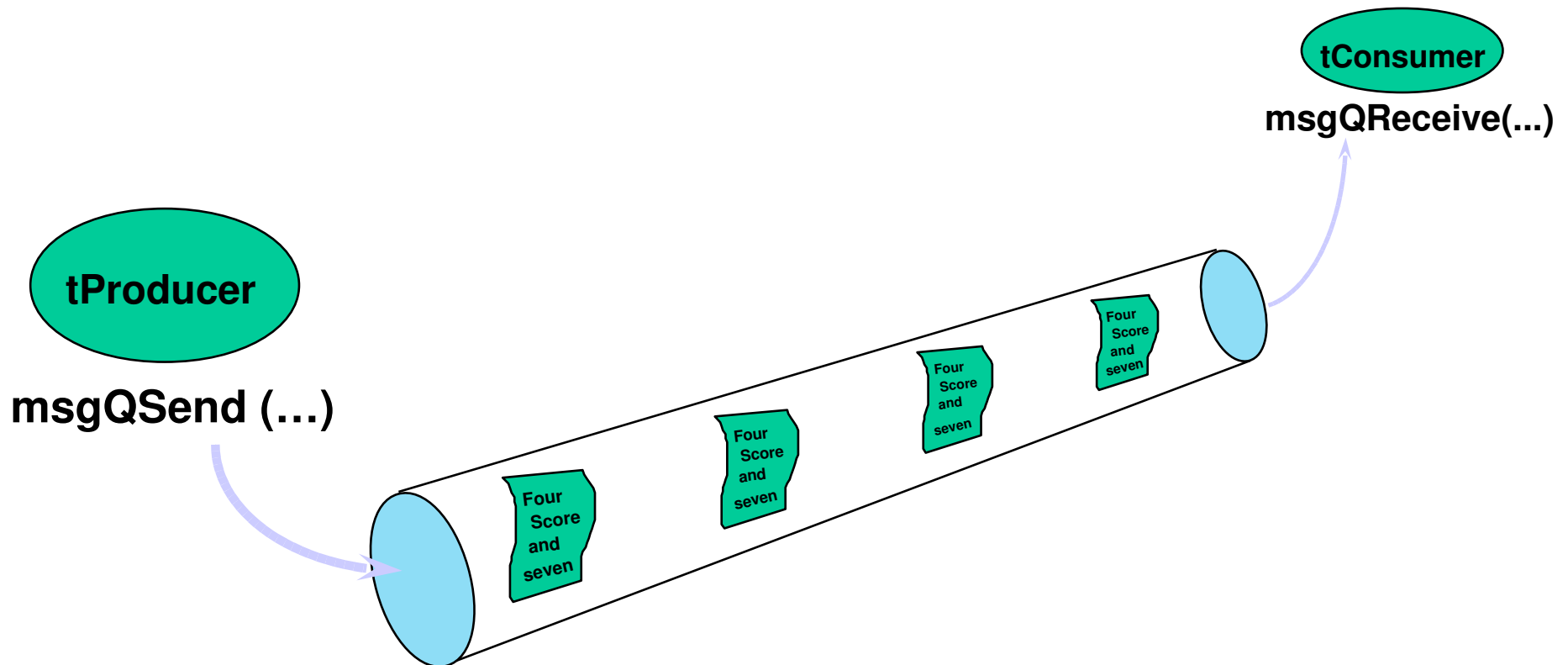
Mécanisme basique de synchronisation et d'exclusion mutuelle.

- Objets du noyau permettant de
 - bloquer/débloquer des tâches
 - coordonner leur exécution avec celles des autres tâches et les événements externes.
- VxWorks offre 3 types de sémaphores :
 - binaire : synchronisation
 - compteurs : allocation de ressources
 - mutex (exclusion mutuelle) : héritage de priorité
- Gestion de la file d'attente des tâches :
 - FIFO
 - par priorité des tâches

Communication intertâche : messages

Buffer FIFO de messages de longueur prédéfinie

une tâche se suspend en cas de lecture sur file vide ou d'écriture sur file pleine. Possibilité spécifier time-out.



Communication intertâche : pipes

Peripherals d'E/S virtuels, associés à des files de messages

- Taille max prédéfinie du pipe et des messages
- Même synchronisation que sur les files
- Les ISRs (interrupt service routines) peuvent écrire mais pas lire (idem files).
- Une caractéristique importante : ***select***().
Permet à une tâche d'attendre qu'une donnée soit disponible sur un ensemble de périphériques d'E/S (pipes, sockets, périphériques série).

Communication intertâche : signaux

VxWorks implante les signaux POSIX.

Deux utilisations :

- Avertir une tâche de l'arrivée d'un événement de façon asynchrone
- Traitement d'exceptions

Si une tâche commet une exception, elle est suspendue sauf si elle a installé un handler de signal pour le signal correspondant

⇒ le handler peut relancer la tâche ou reprendre son exécution à un état antérieur

Communication intertâche : interruptions

Pour obtenir la réponse la plus rapide possible à des interruptions externes les *interrupt service routines* (ISRs) s'exécutent dans un contexte spécial hors du contexte des tâches \Rightarrow éviter les changements de contexte.

- l'adresse de l'ISR est stockée dans la table des vecteurs d'interruptions ; elle est directement appelée par le hardware
- l'ISR démarre le travail (sauvegarde des registres, ...) et appelle ensuite la fonction C qui traite l'interruption

Communication intertâche : watchdog timers

Mécanisme qui permet à une fonction C de s'exécuter après un délai spécifié.

Utilisé pour la scrutation périodique et la détection de dépassement de deadline

Entrées / sorties

Basées sur POSIX

Tous les canaux d'entrée/sortie sont représentés par des descripteurs de fichier

Ils sont manipulés par les fonctions :

`open, read, write, ioctl, close`

Entrées/sorties bufférisées :

`fread/fwrite, fgetc/fputc`

Entrées/sorties formatées :

`fscanf/fprintf`

Programmation sous Tornado/VxWorks

Création d'une tâche

```
TASK_ID t_id;  
t_id=taskSpawn(char *nom,  
               unsigned char priorite,  
               unsigned char flags,  
               unsigned char taille_pile,  
               FUNCPTR fonction,  
               void arg1,  
               void arg2,  
               ...,  
               void arg10)
```

Programmation sous Tornado/VxWorks

Suspension / reprise

```
STATUS taskSuspend(TASK_ID t_id)
```

```
STATUS taskResume(TASK_ID t_id)
```

Verrouillage / déverrouillage d'une tâche

```
STATUS taskLock(TASK_ID t_id)
```

```
STATUS taskUnlock(TASK_ID t_id)
```

Destruction d'une tâche

```
STATUS taskDelete(TASK_ID t_id)
```


Programmation sous Tornado/VxWorks

Création d'un sémaphore

```
SEM_ID semBCreate(unsigned option,  
                  unsigned valeur)  
SEM_ID semMCreate(unsigned option)  
option = SEM_Q_PRIORITY, SEM_Q_FIFO  
valeur = SEM_FULL, SEM_EMPTY
```

Prendre le sémaphore

```
STATUS semTake(SEM_ID s_id, int timeout)
```

Rendre le sémaphore

```
STATUS semGive(SEM_ID s_id)
```

Destruction du sémaphore

```
STATUS semDelete(SEM_ID s_id)
```

Programmation sous Tornado/VxWorks

Gestion des tâches périodiques : utilisation des watchdogs

Création d'un watchdog

```
WDOG_ID wdCreate()
```

Lancement du watchdog

```
STATUS wdStart(WDOG_ID w_id, int delai,  
               FUNCPTR handler, void arg)
```

Stopper le watchdog

```
STATUS wdCancel(WDOG_ID w_id)
```

Destruction du watchdog

```
STATUS wdDelete(WDOG_ID s_id)
```