



Systemes et executifs temps-réel

Licence professionnelle « Systemes embarqués dans l'automobile »

Isabelle PUAUT (Cours + TD)

Jean-François DEVERGE et Christophe Pais (TP)



Applications temps-réel

- Systemes en **interaction** avec l'environnement
- **Contraintes de temps** à satisfaire
 - La date de livraison d'un resultat est un critere de correction
- Systemes dédiés (non généralistes)
 - Matériel (processeur, DSP, FPGA, ASICs, **capteurs**, **actionneurs**) + Logiciel
- Contraintes industrielles
 - Systemes embarqués : contraintes de poids, taille, consommation énergétique, coût, surface
 - Time-to-market, sûreté de fonctionnement



Notion de contraintes de temps

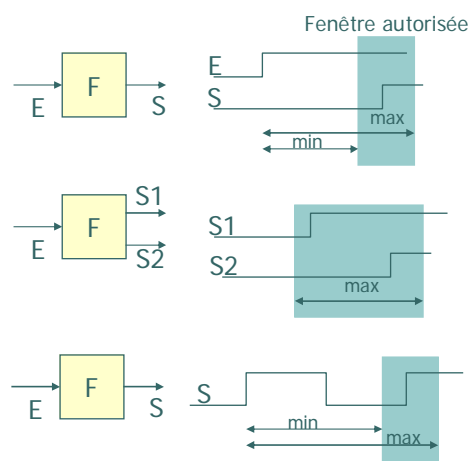
- Définition
 - Limite **quantifiée** (en rapport avec le **temps réel**) sur le temps séparant deux événements
 - Exemple : échéance de terminaison au plus tard (deadline) : limite maximale entre arrivée d'un calcul (tâche) et sa terminaison
- Source des contraintes de temps
 - Délai de réaction du système avant dysfonctionnement (airbag, ABS, injection électronique)
 - Stabilité du processus physique contrôlé (ex: pendule inversé)

3



Exemples de contraintes de temps

- Échéance de terminaison au plus tôt ou au plus tard (deadline)
- Cohérence entre instants de production des résultats
 - Ex : synchronisation son-image
- Cadence de production des sorties
 - Ex : régularité de présentation des images dans une vidéo



4



Classes de systèmes temps-réel

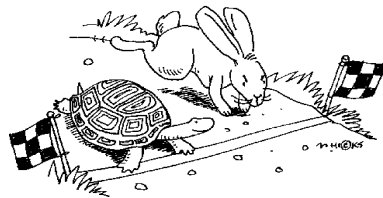
- Temps-réel **strict/dur** (**hard** real-time) : le non respect d'une contrainte de temps a des conséquences graves (humaines, économiques, écologiques)
 - Besoin de garanties
- Temps-réel **souple/mou** (**soft** real-time) : on peut tolérer le non respect occasionnel d'une contrainte de temps

5



Mauvaises interprétations de la notion de temps-réel

- « Real-time is not real-fast » ou « rien ne sert de courir, il faut partir à point »



- Temps-réel ne veut pas dire rapide

6



Domaines d'application

- Electronique grand public
 - Caméras numériques, appareils photo numérique
 - Multimédia, téléphonie
 - décodeurs vidéo
 - téléphones portable
 - PDA
 - consoles de jeu



7



Domaines d'application

- Automobile
 - Systèmes anti-blocage de freins, contrôle moteur, informatique de confort
 - Contrôle de procédés industriels
 - Avionique et spatial
 - Périphériques informatique
 - FAX
 - imprimantes
- Grande **diversité des besoins** (puissance, fiabilité, criticité)

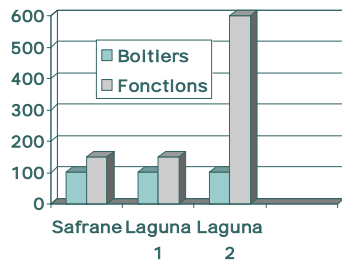


8

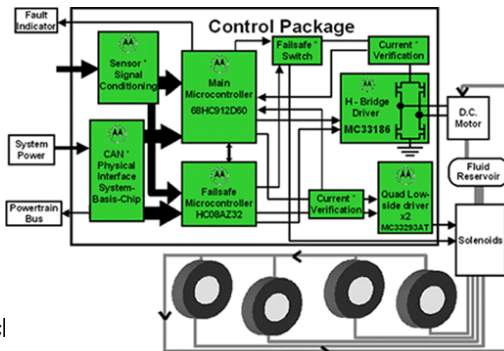


Domaines d'application : automobile

- Evolutions des fonctions logicielles dans l'automobile
- Système antiblocage (ABS)



(C. Balle, Journée Illiatec
octobre 2001)

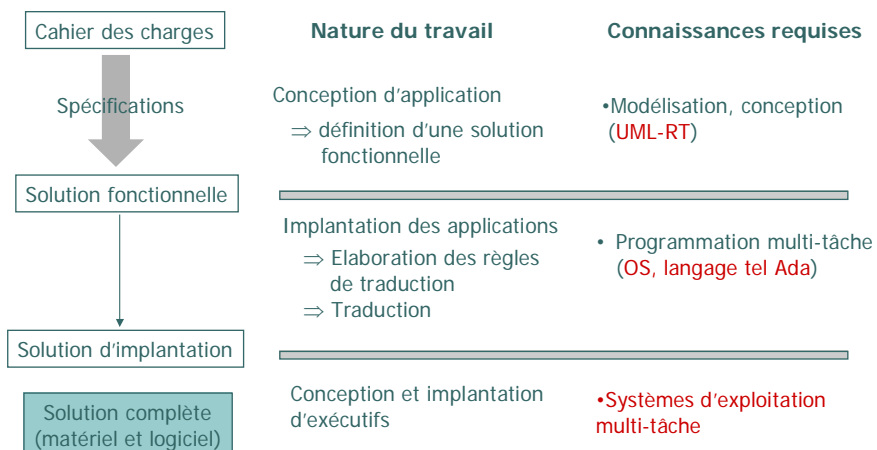


(site Web motorola :
www.motorola.com)

9



Processus de développement



10



Contenu du cours

- Introduction aux temps-réel
- Systèmes d'exploitation temps-réel
 - Présentation par type de fonction
 - Concepts généraux
 - Illustration sur OSEK/VDX
 - Exercices
- Vérification d'ordonnançabilité
- Exposé industriel (Autosar, TNI-Software)

11



Systèmes d'exploitation temps-réel

Illustration sur OSEK/VDX



Systèmes d'exploitation temps-réel

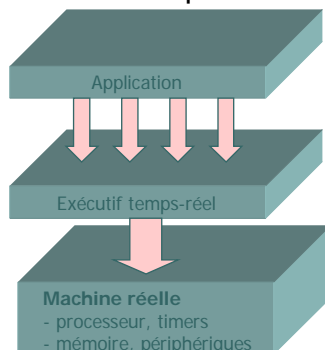
- Terminologie
 - Système d'exploitation = exécuteur = environnement d'exécution
- Objectif
 - Faire coexister plusieurs traitements (calculs) sur le même processeur (partage du processeur)
 - Interface de programmation normalisée (API) : **portabilité**

13



Rôle d'un exécuteur temps-réel

- Objectif : masquer à l'application les particularités du matériel : **machine virtuelle (API)** plus ou moins complexe



Appels systèmes (primitives)

Ex : gestion de tâches (création, arrêt)
Ex : gestion du temps (attente d'un délai)

Gestion du matériel

Ex : sauvegarde de registres pour préemption
Ex : écriture registre timer pour délai

⇒ Plus besoin de manipulations du matériel

14



Types de services fournis

- Gestion des **tâches** (création, ordonnancement)
- Gestion des **synchronisations** (précédences + partage de ressources)
- Gestion des **communications** (ex : messages)
- Gestion de la **mémoire**
- Gestion du **temps**: délai, activation périodique
- Gestion des **périphériques**

15



Plan du chapitre (1/2)

- Gestion des tâches
 - Concepts généraux de gestion
 - Vue d'ensemble d'OSEK/VDX
 - Gestion des tâches dans OSEK/VDX
 - Fonctions de mise au point d'OSEK/VDX (Hooks)
- Gestion du temps
 - Concepts généraux
 - Alarmes OSEK/VDX

16



Plan du chapitre (2/2)

- Synchronisation
 - Synchronisation de type « attente »
 - Concepts généraux
 - Sémaphores à compteurs
 - Synchronisation par événements OSEK/VDX
 - Synchronisation de type « exclusion mutuelle »
 - Sémaphores d'exclusion mutuelle (mutex)
 - Ressources partagées OSEK/VDX
 - Inversion et héritage de priorités
- Gestion d'interruptions
- Gestion mémoire
- Classes de conformité OSEK/VDX

17

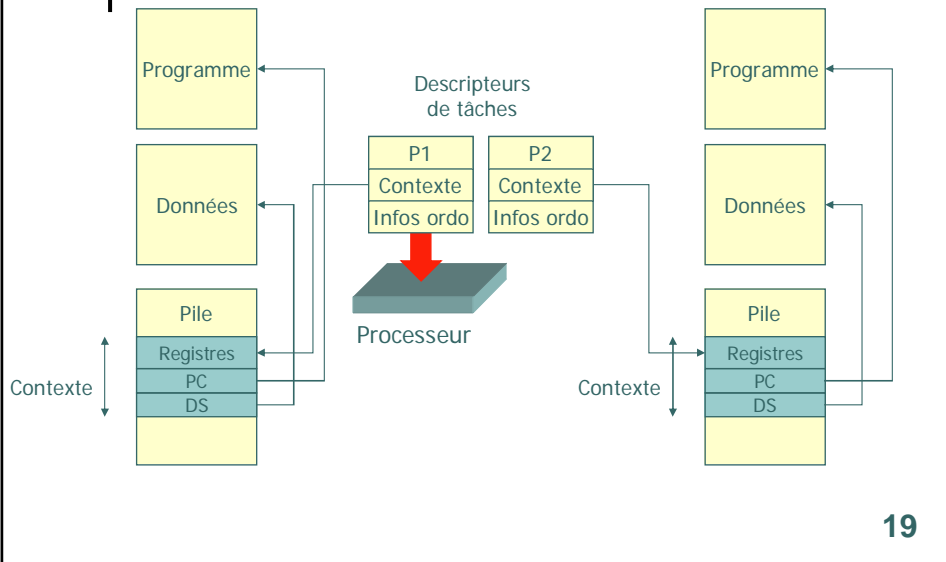


Gestion des tâches Terminologie (1/2)

- **Programme**
 - Instructions à dérouler (statique)
- **Processus, tâche**
 - Entité dynamique composée de :
 - Programme (composante statique)
 - Données, pile d'exécution, contexte (composante dynamique)
 - **Descripteur** : structure de donnée regroupant les informations sur une tâche (nom, contexte d'exécution, ...)
- Exemple : lecture périodique d'un capteur de température : 1 programme + 1 tâche périodique

18

Gestion de tâches Terminologie (2/2)



19

Gestion de tâches Types de tâches

- Arrivées des tâches
 - **Périodiques** : arrivée à intervalles réguliers (P_i)
 - Date d'activation initiale, offset O_i
 - Si pour tout i, j $O_i = O_j$, tâches synchrones
 - Si $D_i = P_i$, tâche à échéance sur requête
 - **Sporadiques** : on connaît une borne minimale sur l'intervalle entre deux arrivées
 - **Apériodiques** : tout le reste
- Remarque : exécution de tâches périodiques cyclique sur l'intervalle
 - $[0, P_{PCM}(P_i)]$ pour les tâches synchrones
 - $[\min(O_i), \max(O_i, O_j + D_j) + 2 * P_{PCM}(P_i)]$ dans le cas contraire

20



Gestion de tâches

Niveaux de parallélisme (1/2)

- Parallélisme réel
 - Systèmes mutiprocresseurs uniquement
- Pseudo-parallélisme : le processeur exécute successivement plusieurs traitements
 - Traitements exécutés en séquence : ordonnancement (**non préemptif**)
 - Un traitement peut être interrompu par un autre traitement : ordonnancement (**préemptif**)

21



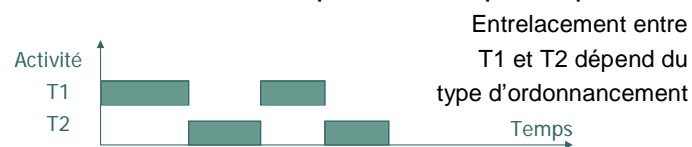
Gestion de tâches

Niveaux de parallélisme (2/2)

- Système multitâche monoprocesseur non préemptif



- Système multitâche monoprocesseur préemptif



- Système multitâche multiprocesseurs

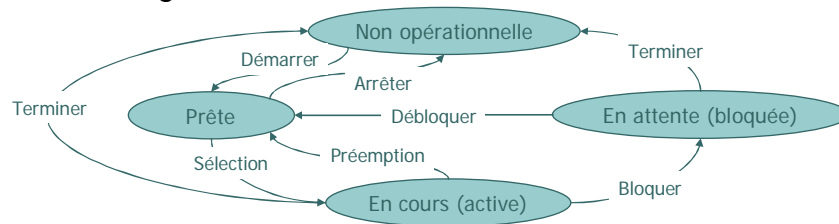


22

Gestion de tâches

Etats d'une tâche

- Partage du processeur par plusieurs tâches
 - Notions de tâche **prête** et tâche **active**
- Primitives de synchronisation : **blocage** des tâches (⇒ état **bloquée**)
- Diagramme de transition entre états



Terminer, Arrêter, Démarrer : primitives (⇒ appelées par les tâches) de gestion de tâches
Bloquer, Débloquer : primitives de synchronisation
Sélection, Prémption : décisions internes de l'ordonnanceur

23

Gestion de tâches

Fonctions du gestionnaire de tâches

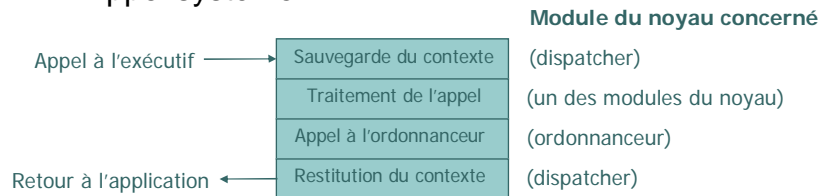
- Ordonnanceur : **choix de la tâche à exécuter** selon une politique d'ordonnancement
 - Structure de données adaptée (exemple, liste chaînée par priorités)
- Allocation du processeur (dispatcher) : **gestion du contexte d'exécution**
 - Objectif : conserver l'état du processeur (registres) quand une tâche est suspendue
 - Sauvegarde contexte (pile)
 - Restauration contexte depuis le descripteur de tâche

24

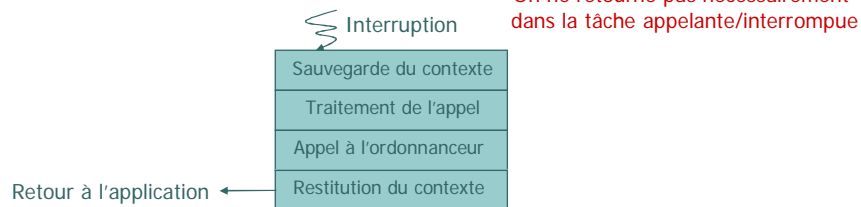
Gestion de tâches

Appels système et interruptions

o Appel système



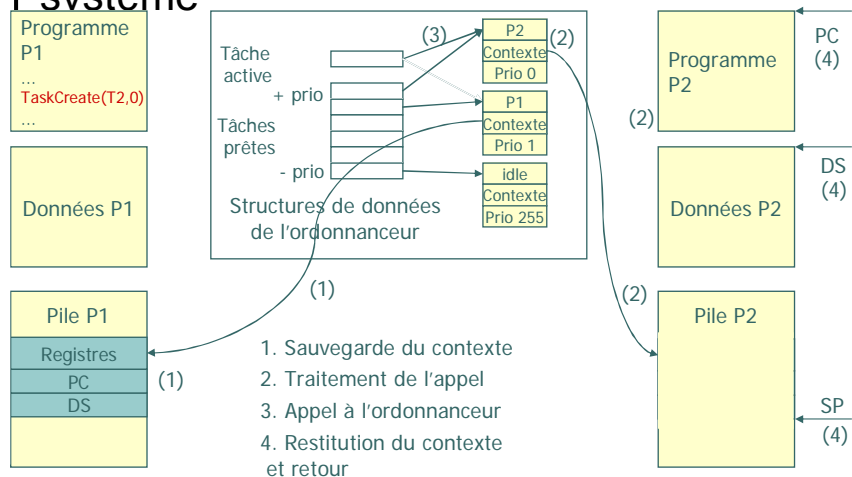
o Gestion d'interruption



25

Gestion de tâches

Exemple de déroulement d'un appel système



26

Gestion de tâches

Ordonnancement de tâches

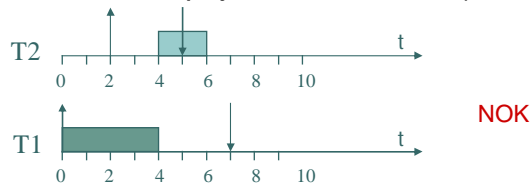
- Problème
 - Calculs à exécuter + contraintes de temps = dans quel ordre exécuter ?
- Définition
 - Ensemble des règles définissant l'ordre d'exécution des calculs sur le processeur
- Problème important : impact sur le respect des contraintes de temps
 - Exemple
 - Tâche T1 : arrivée en 0, temps d'exécution 4, échéance 7
 - Tâche T2 : arrivée en 2, temps d'exécution 2, échéance 5

27

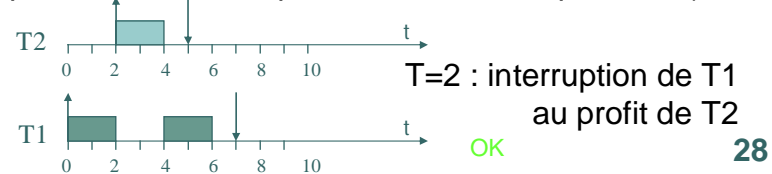
Gestion de tâches

Ordonnancement de tâches

- Ordonnancement O1 (premier arrivé, premier servi, on n'interrompt jamais une tâche)



- Ordonnancement O2 (priorité, T2 plus prioritaire que T1, on interrompt une tâche moins prioritaire)



28

Gestion de tâches

Classification des ordonnancements

- Critères de classification
 - Instant ou l'ordre d'exécution est décidé : **hors-ligne / en-ligne**
 - Possibilité d'interruption d'une tâche par une autre : **préemptif / non préemptif**

29

Gestion de tâches : classification (1/3)

« hors-ligne » / « en-ligne »

- Hors-ligne
 - La séquence d'ordonnement est **pré-calculée** avant l'exécution effective (on dit aussi « **time-driven** » scheduling)
 - A l'exécution, l'ordonneur est un simple **séquenceur** (« cyclic scheduler ») : pas besoin d'exécutif multitâche



30



Gestion de tâches : classification (2/3) « hors-ligne » / « en-ligne »

- Ordonnancements « en-ligne »
 - Les décisions d'ordonnancement sont prise **au cours** de l'exécution
 - A l'exécution, l'ordonnanceur implante un **algorithme d'ordonnancement** permettant de savoir à tout instant quel tâche exécuter : besoin d'un exécutif multitâche
 - Généralement, ordonnancements conduits par la **priorité**

31



Gestion de tâches : classification (3/3) Non préemptif / préemptif

- Non préemptif
 - On n'interrompt **jamais** l'exécution d'une tâche en cours au profit d'une autre tâche
- Préemptif
 - La tâche en cours **peut perdre** involontairement le processeur au profit d'une autre tâche (jugée plus urgente)
 - Préemptif ⇒ besoin d'un exécutif multitâche

32

Gestion de tâches

Ordonnements représentatifs

- Ordonnement préemptif **par priorité**
 - Algorithme d'ordonnement : sélectionner la **tâche la plus prioritaire** à tout instant
 - $prio(T3) > prio(T2) > prio(T1)$



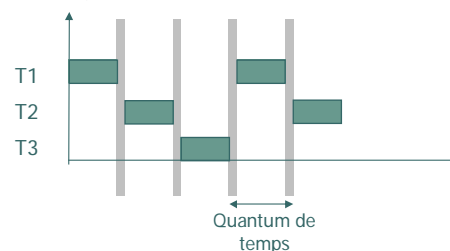
- Sous-classification
 - Priorités **fixes** (statiques) : indépendantes du temps
 - Priorités **dynamiques** : évoluent avec le temps
- Quasi tous les exécutifs du marché
- C'est l'utilisateur qui fixe les priorités (comment ?)

33

Gestion de tâches : classification (5/5)

Ordonnements représentatifs

- Temps partagé avec politique du **tourniquet** (round-robin)
 - Tranche de temps de taille fixe (quantum)
 - Pas de prise en compte des contraintes de temps
 - Souvent utilisé en complément du mécanisme de priorité pour les tâches de priorités égales (équité entre ces tâches)



34



Gestion de tâches : ordonnancement

Ordonnements représentatifs

- Hors-ligne non préemptifs (séquenceurs)
 - Jouent une séquence d'exécution générée statiquement
 - OsekTime (automobile)

35



Gestion de tâches : ordonnancement

Priorité fixe: Rate Monotonic (1/2)

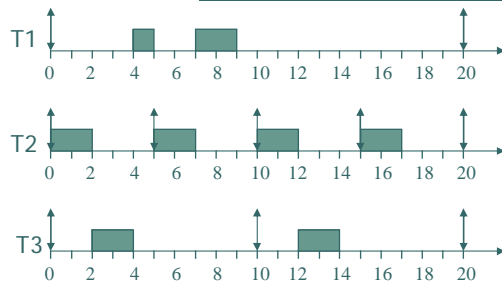
- Pour tâches périodiques seulement
- Définition [Liu & Layland, 1973]
 - La priorité d'une tâche est inversement proportionnelle à sa période d'activation (conflits résolus arbitrairement)
- Propriété
 - Optimal dans la classe des algorithmes à priorités fixes pour des tâches périodiques indépendantes à échéance sur requête ($D_i = P_i$).

36

Gestion de tâches : ordonnancement

Priorité fixe: Rate Monotonic (2/2)

Tâche	Pi	Ci	Di
T1	20	3	20
T2	5	2	5
T3	10	2	10



Prio(T2) > Prio(T3) > Prio(T1)
Exécution cyclique
(PPCM des périodes)

37

Gestion de tâches : ordonnancement

Priorité fixe: Deadline Monotonic

- Pour tâches périodiques
- Définition [Leung & Whitehead, 1985]
 - La priorité d'une tâche est inversement proportionnelle à son échéance relative (conflits résolus arbitrairement)
- Propriété
 - Optimal dans la classe des algorithmes à priorités fixes pour des tâches périodiques indépendantes à échéance \leq période

38

Gestion de tâches : ordonnancement Priorité dynamique: EDF (1/2)

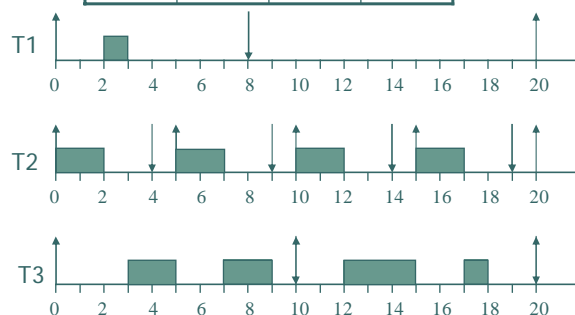
- Earliest Deadline First (EDF)
- Applicable pour tâches périodiques **et non périodiques**
- Définition [Liu & Layland, 1973]
 - A tout instant, la tâche la plus prioritaire (parmi les tâches prêtes) est celle dont **l'échéance absolue est la plus proche**
 - Préemptif
- Propriété
 - Optimal dans la classe des algorithmes préemptifs pour des configurations de tâches périodiques indépendantes avec échéance \leq période

39

Gestion de tâches : ordonnancement Priorité dynamique: EDF (2/2)

Tâche	Pi	Ci	Di
T1	20	1	8
T2	5	2	4
T3	10	4	10

(2 préemptions en 5 et 15)



40



Gestion de tâches : ordonnancement

Ordonnancement à priorités : bilan

- Utilisateur
 - Choix des priorités
- Système d'exploitation
 - Attribution du processeur selon la priorité des tâches prêtes
 - Primitive de changement de priorités (sur certains systèmes)
- Remarque
 - EDF n'est pas un ordonnanceur de base des exécutifs temps-réel => priorités à changer par l'utilisateur

41




Gestion de tâches dans OSEK/VDX

Introduction à OSEK/VDX

- Créé dans les années 90 par un consortium de sociétés automobiles (BMW, Bosch, DaimlerChrysler, Opel, Siemens, VW, Renault, PSA) et d'universitaires
- Objectif : standard d'architecture ouverte pour l'électronique embarquée dans les véhicules
- Trois parties dans l'architecture
 - Système d'exploitation temps-réel (ce cours)
 - Gestion du réseau
 - Communication (échanges de données entre U.C.)
- Standard **d'interface**, diverses implantations

42



Gestion de tâches dans OSEK/VDX

Introduction à OSEK/VDX

- Noyau statique
 - Objets du système définis **statiquement** à la compilation (tâches, messages, sémaphores, etc ...)
 - Langage de définition des services : OIL (OSEK Implementation Language)
 - Faible consommation mémoire
- Classes de conformité: différentes « versions » du noyau par rapport aux appels système disponibles

43




Gestion de tâches dans OSEK/VDX

Tâches OSEK/VDX

- Tâches **basiques**
 - N'ont pas accès aux services de synchronisation
 - Trois états : **running**, **ready**, **suspended**
- Tâches **étendues**
 - Ont accès aux services de synchronisation
 - Un état supplémentaire : **waiting**
- Attributs (statiques, liste non exhaustive)
 - PRIORITY (grande valeur = grande priorité)
 - SCHEDULE (NON ou FULL) : préemptif ou non
 - ACTIVATION : nombre max. d'exéc. simultanées
 - AUTOSTART : défini si activé au démarrage de l'OS

44




Gestion de tâches dans OSEK/VDX

Politique d'ordonnancement (1/2)

- Ordonnancement non préemptif
 - Pour tâches non préemptibles (SCHEDULE = NON)
 - Conservation du processeur tant qu'une de ces tâches s'exécute
- Ordonnancement non préemptif
 - Pour tâches préemptibles (SCHEDULE = FULL)
 - **En-ligne, préemptif**
 - Par **priorités** (valeur la plus haute = tâche plus prioritaire)
 - Priorités **statiques**

45



Gestion de tâches dans OSEK/VDX

Politique d'ordonnancement (2/2)

- Ordonnancement mixte préemptif – non préemptif
 - Moyen : obtention ressource spéciale RES_SCHEDULER
 - Effet : on inhibe le mécanisme d'ordonnancement, le processeur n'est plus réattribué
 - Appel système : getResource(RES_SCHEDULER)
 - Contrôle de l'attribution du processeur : appels système ChainTask et Schedule
- Pas disponible dans toutes les réalisations (pas en TP)

46



Gestion de tâches dans OSEK/VDX

Politique d'ordonnancement : API

- `StatusType ActivateTask(TaskType tid);`
Activation de la tâche (passage dans l'état READY, puis un jour RUNNING quand elle deviendra la plus prioritaire)
- `StatusType TerminateTask(void);`
Terminaison de la tâche (retour à l'état SUSPENDED). Doit impérativement être appelé avant de sortir d'une tâche.
- `StatusType ChainTask(TaskType tid);`
Terminaison de la tâche appelante, combinée avec l'activation de la tâche passée en paramètre (combinaison des appels système `TerminateTask` et `ChainTask` en un seul appel.

47



Gestion de tâches dans OSEK/VDX

Politique d'ordonnancement : API

- `StatusType Schedule (void);`
Appel explicite à l'ordonnanceur. Cet appel n'a pas d'impact lorsque l'on utilise l'ordonnancement préemptif. En mode non préemptif, relâche la ressource interne acquise, et lance la tâche prête la plus prioritaire. La ressource interne acquise doit être re-demandée après le `Schedule`.
- `StatusType GetTaskID(TaskRefType tid);`
Retourne une référence sur la tâche en cours d'exécution.
- `StatusType GetTaskState(TaskType tid, TaskStateRefType state);`
Retourne l'état de la tâche tid au moment de l'appel (RUNNING, WAITING, READY, SUSPENDED, INVALID_TASK).

48



Fonctions de mise au point OSEK/VDX Hooks (crochets)

- Hooks : définitions
 - Fonctions écrites par l'utilisateur
 - Appelées par le système d'exploitation à des points clé de son exécution (ex: démarrage de l'OS, changement de contexte). Interface standardisée
- Propriétés
 - Plus haute priorité que les tâches
 - Non interruptibles par interruptions de catégorie 2
 - Appels système disponibles restreints

49



Fonctions de mise au point OSEK/VDX Hooks (crochets)

- Points d'appels des hooks
 - **Démarrage du système** (fin du démarrage du système, avant exécution de l'ordonnanceur, qui lance les tâches) : *StartupHook*
 - **Arrêt du système** : *ShutdownHook*
 - **Erreur** lors d'un appel système : *ErrorHook*
 - **Changements de contexte** : *PreTaskHook* (appelée avant de donner le processeur à une tâche) et *PostTaskHook* (appelée avant d'oter le processeur à une tâche)

50



Gestion du temps

Services classiques (1/3)

- Lecture / mise à jour du temps
 - Par **matériel** : horloge temps-réel (précision dépend du matériel)
 - Par **logiciel** : horloge système incrémentée périodiquement par routine de traitement d'interruption d'horloge (tick)
 - Granularité peut être importante (par défaut, généralement 10ms)
 - Certains systèmes permettent de changer la granularité (attention au temps passé en traitement IT horloge)

51



Gestion du temps

Services classiques (2/3)

- Actions à des dates **absolues**
 - Types d'action : signal événement, lancement tâche
 - Récurrences : une seule fois ou périodiquement
- Actions à des dates **relatives** au temps courant
 - Chiens de garde lors des primitives bloquantes (synchronisation, communications)
 - **Signal événement / lancement de tâche** après un certain délai

52



Gestion du temps

Services classiques (3/3)

- Fonctions d'instrumentation de l'exécution
 - Exemple VxWorks
 - timex(fn,args); mesure du temps pour exécuter une fonction
 - timexN(fn,args); mesure du temps pour exécuter N occurrences de la fonction avec exécutions en boucle
 - Pièges à éviter
 - Sans boucle : granularité de l'horloge
 - Tests en boucle : impact des caches

53



Gestion du temps dans OSEK/VDX

Alarmes

- Alarme = association entre
 - Un compteur (ex: tick d'horloge)
 - Une tâche
 - Une action sur la tâche (**activation** / envoi d'un **événement**).
 - Attributs de l'alarme (tâche, action) fixés statiquement (init. noyau)
- Types de déclenchement de l'alarme
 - **Cyclique** (récurrent) ou **unique**
 - Valeur **relative** ou **absolue** du compteur (pour le premier déclenchement seulement dans le cas d'alarmes cycliques)

54



Gestion du temps dans OSEK/VDX



Alarmes : API

- `SetRelAlarm(id_alarm,incr,cycle);`
Programme le déclenchement de l'alarme id_alarm. Un premier déclenchement aura lieu dans incr ticks (relatif), puis tous les cycle ticks si cycle est non nul.
- `SetAbsAlarm(id_alarm,start,cycle);`
Programme le déclenchement de l'alarme id_alarm. Un premier déclenchement aura lieu au tick start, puis tous les cycle ticks si cycle est non nul.
- `CancelAlarm(id_alarm);`
Annule le déclenchement de l'alarme id_alarm.
- `GetAlarm(id_alarm,&tick);`
Retourne la valeur en ticks jusqu'à ce que l'alarme se déclenche. Sert par exemple à vérifier si un CancelAlarm est encore utile ou non.

55



Synchronisation

- **Précédence** (synchronisation de type « attente »)
 - Notion d'ordre entre les traitements 
- **Partage de variables / ressources** (synchronisation de type « exclusion mutuelle »)
 - Echanges d'informations ou de résultats 

56



Synchronisation de type « attente »

- Rôle
 - Attendre un événement avant d'exécuter un calcul
 - Attente « passive »
- Origine des événements
 - Matériel : capteurs
 - Logiciel : synchronisation interne à une application
- Plan du chapitre
 - Classification
 - Un outil généraliste : les sémaphores à compteurs
 - Synchronisation de type « attente » dans OSEK/VDX (événements)

57



Synchronisation de type « attente »

- Classification des primitives
 - Mémorisation
 - Sans mémorisation (événement fugace, prise en compte sur niveau)
 - Si tâche en attente, alors on la réveille, sinon l'évt est perdu
 - Avec mémorisation (événement persistant, prise en compte sur état)
 - sans compte : une occurrence est mémorisée jusqu'à sa consommation : événements
 - à compte : compteur d'occurrences non effacées : sémaphores

58



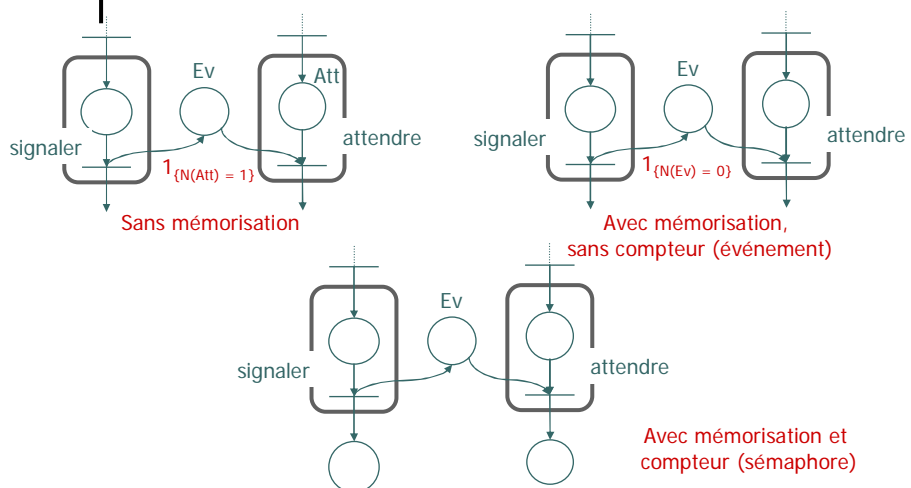
Synchronisation de type « attente »

- Classification des primitives de synchronisation (suite)
 - Directe / indirecte
 - Directe : on signale à une tâche précise (signal(Ev,T);)
 - Indirecte : global (signal(Ev);)
 - Synchrone / asynchrone
 - Synchrone : la tâche se met **explicitement** en attente de l'événement (wait(Ev);)
 - Asynchrone : on associe à une tâche un gérant d'événement qui est exécuté lors de l'occurrence de l'événement (connecter_Gérant(Ev, fonction);)

59



Synchronisation de type « attente »



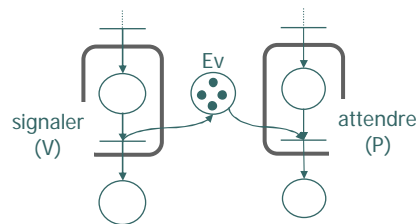
60

Synchronisation de type « attente » Sémaphores à compteurs

o Primitives classiques

- sid = create_sema(cpt); /* cpt >= 0 */
- P (sid); /* Puis-je ? */
- V (sid); /* Vas - y */

o Comportement



Valeur initiale = nombre de jetons initialement dans Ev (nombre de P possibles avant blocage)

61

Synchronisation de type « attente » Sémaphores à compteurs

```
typedef struct {
    int cpt;      /* Compteur
                 ≥ 0 = nombre de P avant blocage (nb evts non consommés)
                 < 0 = nombre d'éléments dans la file d'attente */
    file *F;      /* Tâches en attente sur le sémaphore */
} t_sema;      /* Descripteur de sémaphore */
```

```
V (t_sema *s) {
    Sauvegarde contexte;
    (s->cpt) ++;
    if (s->cpt ≤ 0) {
        T = retirer(s->F);
        T->état = Prêt;
    }
    Appel à l'ordonnançeur;
    Restitution du contexte;
}
```

```
P (t_sema *s) {
    Sauvegarde contexte;
    (s->cpt) --;
    if (s->cpt < 0) {
        Active.état = Bloqué;
        insérer(Active,s->F);
    }
    Appel à l'ordonnançeur;
    Restitution du contexte;
}
```

62

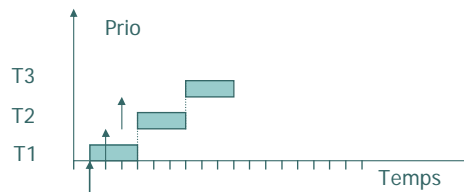
Synchronisation de type « attente » Impact gestion des files d'attente (1/2)

Illustration sur la synchronisation par sémaphores

- Code des tâches

```
s = create_sema(1);
T1 : P(s); A1; V(s);
T2 : P(s); A2; V(s);
T3 : P(s); A3; V(s);
```

Arrivée de T1 en 1, de T2 en 2, de T3 en 3. $\text{prio}(T1) < \text{prio}(T2) < \text{prio}(T3)$; Durée des A_i de 3;
- Cas d'une file gérée en FIFO

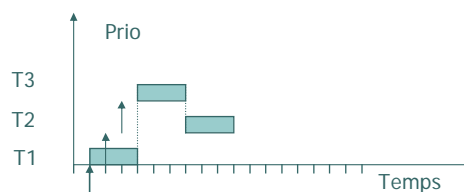


63

Synchronisation de type « attente » Impact gestion des files d'attente (2/2)

Illustration sur la synchronisation par sémaphores

- Cas d'une file gérée par priorité décroissante



- File FIFO : équité, pas de famine
- File par priorité : moins équitable, mais favorise les tâches plus prioritaires → plus adapté aux systèmes temps-réel

64



Synchronisation de type « attente » Les événements OSEK/VDX

- Types de synchronisation
 - Événements avec **mémorisation sans compte**
 - Effacement **explicite**
 - Événements **privés** (nombre limité d'événements par tâches, pas d'appel de création d'événement)
 - Signal **direct** des événements
 - **Synchrone** : appel explicite à la routine d'attente d'un événement

65



Synchronisation de type « attente » Les événements OSEK/VDX : API

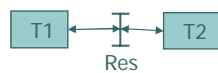
- SetEvent (tid,mask)
Positionne l'ensemble d'événements de la tâche tid désignés par le masque mask (bitmap: 1 = set, 0 = clear)
- ClearEvent (mask)
Effacement des événements désignés par mask
- WaitEvent (mask)
Attente (sans délai) des événements désignés par mask (attente sur un ensemble d'événements)
- GetEvent(tid,mask)
Retourne les événements à 1 du masque mask concernant la tâche tid (pas de relation avec événements attendus)

66

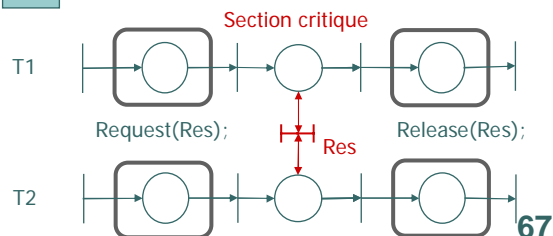


Partage de ressources

- Variable partagée → deux tâches ne doivent pas y accéder en lecture et écriture simultanément
 - Notion de **section critique**, accès en **exclusion mutuelle**
- Modèle fonctionnel



- Comportement



Partage de ressources

Implantation d'une section critique (1/4)

- Suppression/masquage des interruptions
 - Faible coût à l'exécution (cli/sti)
 - Latence (gigue) dans le traitement des interruptions, voire perte d'interruptions
 - Non valable en multiprocesseurs
 - Utilisable uniquement sur des sections critiques courtes, de taille bornée
- Augmentation temporaire de la priorité pendant la section critique
 - Retard dans le traitement des tâches non concernées par la ressource

68



Partage de ressources Implantation d'une section critique (2/4)

- Implantation par attente active

```
void Request (char *V)                void Release (char *V)
{
    while (*V) ; /* Attente ressource */
    *V = 1;
}
                                        {
                                        *V = 0; /* Libération */
                                        }
```

- Problème

- Monopolisation du processeur
- Situation de famine possible : **ne fonctionne pas !**

69



Partage de ressources Implantation d'une section critique (3/4)

- Utilisation de sémaphores (attente passive)

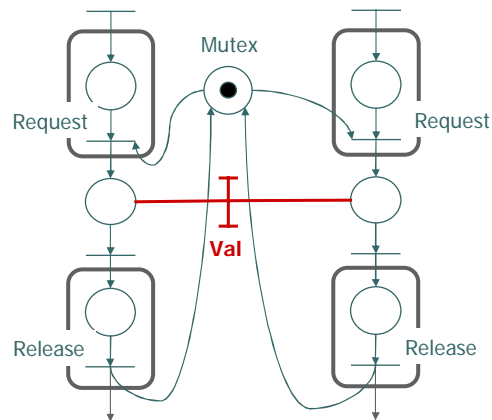
- Request(v) → P(s_v);
Release(v) → V(s_v);
- Sémaphores à **compteurs** tels que ceux présentés avant
- Sémaphores **d'exclusion mutuelle (mutex)** : cas particuliers de sémaphores avec en plus les propriétés suivantes :
 - Valeur initiale de 1 (pas de compteur en paramètre)
 - Restriction des appels système pour que le P et le V soient faits par la même tâche
 - Possibilité d'emboîter dans la même tâche (sans blocage) des accès à la même ressource

70

Partage de ressources

Implantation d'une section critique (4/4)

- Comportement des mutex



71

Partage de ressources

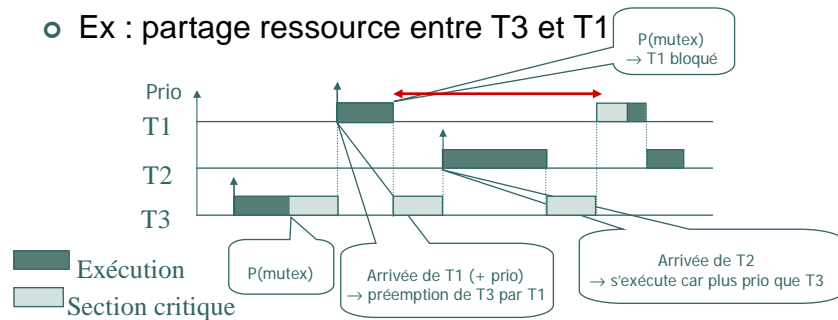
Ressources OSEK/VDX

- Equivalent à des sémaphores **d'exclusion mutuelle** (hormis la 3ème propriété de gestion des emboîtements). Gestion de file par priorités
- Interface
 - GetResource (R);
Acquisition de la ressource R. Blocage de la tâche appelante jusqu'à ce que la ressource soit disponible.
 - ReleaseResource (R);
Libération de la ressource R et déblocage des éventuelles tâches en attente de la ressource.

72

Partage de ressources Problème d'inversion de priorité

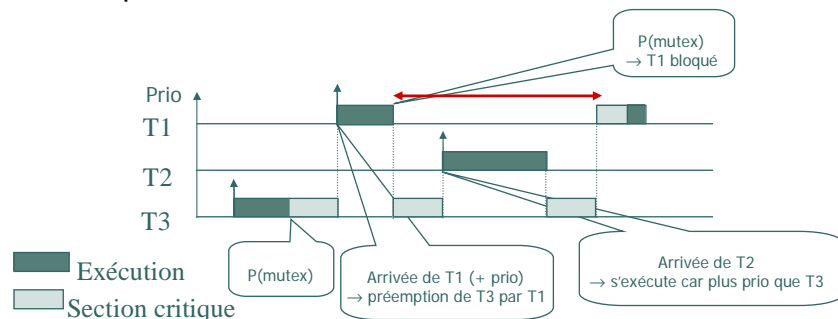
- Définition de l'inversion de priorité
 - une tâche prioritaire est bloquée alors qu'une tâche moins prioritaire qu'elle s'exécute
- Ex : partage ressource entre T3 et T1



T1 bloqué pendant toute la durée de T2. Durée blocage potentiellement infinie (multiples tâches de prio. interm).

Partage de ressources Problème d'inversion de priorité

- On voit sur l'exemple précédent qu'une gestion des files d'attente par priorité n'élimine pas l'inversion de priorité





Partage de ressources

Solution à l'inversion de priorité : PIP (1/4)

- Protocole d'héritage de priorité (PIP = Priority Inheritance Protocol)
- Prérequis
 - Ordonnancement à **priorité fixe**, priorité nominale vs priorité courante
 - Mutex pour sections critiques, section critiques « properly nested » (pas d'intersection entre deux sections critiques, ou bien emboîtement complet)

75



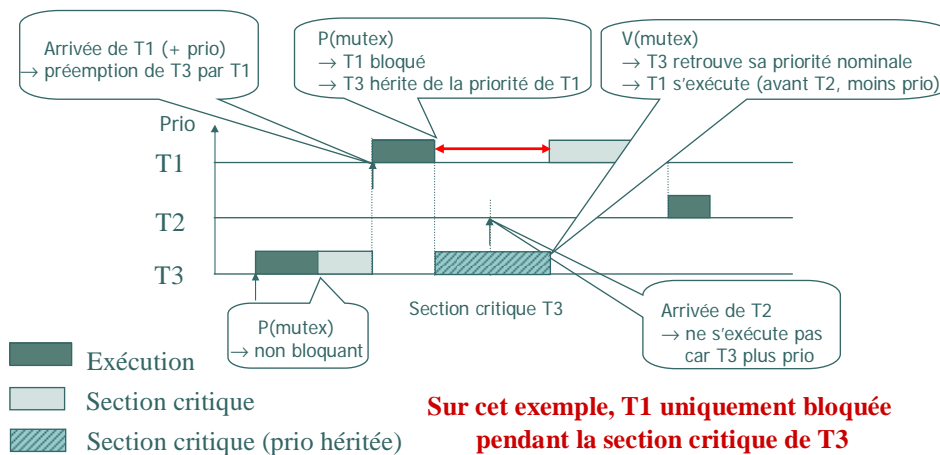
Partage de ressources

PIP (2/4)

- Définition du protocole
 - Règle d'ordonnancement: Exécution selon priorité nominale, sauf règle 3
 - Règle d'allocation de ressource
 - Ressource libre : acquisition de la ressource
 - Ressource occupée : blocage
 - Règle d'héritage de priorité
 - Quand blocage d'une tâche T lors de l'accès à R, la tâche bloquante TI hérite de la priorité de T;
 - Utilisation par TI de la priorité héritée jusqu'à libération de R, où elle retrouve la priorité qu'elle avait lors de son acquisition.

76

Partage de ressources PIP (3/4)



77

Partage de ressources PIP (4/4)

o Propriétés

- Temps de blocage borné (sauf interblocage)
 - Possibilité de vérification a priori des échéances
- Interblocage possibles
- Pour implanter PIP, il n'est pas nécessaire de connaître **quelle tâche utilise quelle ressource** (par contre, nécessaire pour calculer les temps de blocage)

o Remarque

- **N'élimine pas l'inversion de priorité**, la rend juste de durée bornée

78

Partage de ressources

Solution à l'inversion de priorité : PCP (1/5)

- PCP = Priority Ceiling Protocol (Protocole à priorité « plafond »)
 - ex: OSEK/VDX
- Prérequis
 - Ordonnancement à priorité fixe
 - Ressources utilisées par toutes les tâches sont **connues a priori** (contrairement à PIP)

79

Partage de ressources

PCP (2/5)

- Définitions
 - **Priorité plafond** (priority ceiling) d'une ressource $\Pi(R)$ = priorité maximale des tâches qui l'utilisent
 - **Priorité plafond courante** (current priority ceiling, ou ceiling) à un instant t : $\underline{\Pi}(t)$ = maximum des priorités plafond des ressources **utilisées** en t

80



Partage de ressources PCP (3/5)

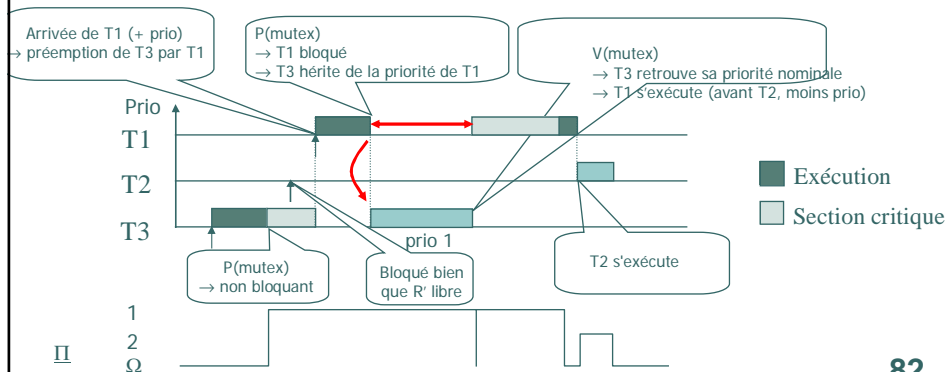
1. Ordonnancement: selon priorité nominale, sauf règle 3 (
2. Allocation de la ressource R demandée par une tâche T
 - a) Ressource occupée : blocage
 - b) Ressource libre
 - i. T strictement plus prioritaire que $\Pi(t)$, allocation de R à T
 - ii. Sinon, R allouée à T si c'est T qui possède la ressource de plafond $\Pi(t)$, sinon, T est bloquée (différent PIP)
3. Héritage de priorité
 - a) Quand blocage d'une tâche T lors de l'accès à R, la tâche bloquante T1 **hérite de la priorité de T prio(t)**,
 - b) Conservation priorité et ce jusqu'à ce que T1 libère toute ressource de plafond supérieur ou égal à prio(t). A ce moment, T1 reprend la priorité qu'il avait à quand il a acquis la ressource.

81



Partage de ressources PCP (4/5)

- o Exemple : partage de R entre T1, T3, T2 utilise une ressource R'
 - a) $\Pi(R) = \text{prio}(T1)$; $\Pi(R') = \text{prio}(T2)$; Initialement $\Pi(0) = \Omega$;



82



Partage de ressources PCP (5/5)

- Propriétés
 - **Durée de blocage bornée** : borne = durée maximale d'une section critique d'une tâche de priorité inférieure (plus faible qu'avec PIP)
 - Durée de blocage pour une tâche T_i = Durée de la plus longue section critique d'une tâche de priorité inférieure et gardée par un sémaphore s tel que $\Pi(s) \geq \text{prio}(T_i)$
 - Identifier les sections critiques des tâches de priorité inférieure qui peuvent bloquer T_i (direct + push-through)
 - Filtrer toutes les sections critiques vérifiant $\Pi(s) \geq \text{prio}(T_i)$ et prendre le maximum

83



Partage de ressources PCP (5/6)

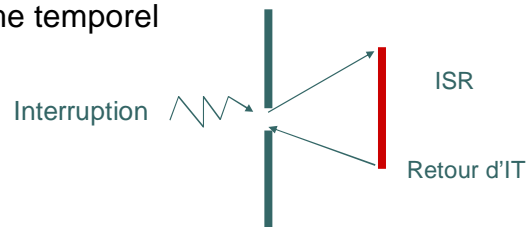
- Propriétés (suite)
 - Durée de blocage maximal B_i plus faible qu'avec PIP
 - **Pas d'interblocages** possibles
- Mais nécessite de connaître toutes les ressources partagées par toutes les tâches (**non transparent pour l'utilisateur**)
 - Interface OSEK/VDX : liste des ressources utilisées par chaque tâche
- N'élimine pas l'inversion de priorité, en rend la durée bornée

84



Gestion des interruptions

- Définitions
 - Sollicitation externe → requête d'interruption
 - Exécution d'une routine de traitement d'interruption (ISR = Interrupt Service Routine)
 - Priorités entre interruptions (selon architecture)
 - En général, appel à l'ordonnanceur au retour de l'ISR
- Diagramme temporel



85



Gestion des interruptions

- Un ISR n'a pas le statut d'une tâche
 - Pas de contexte reconnu de l'exécutif (s'exécute soit dans le contexte de la tâche interrompue, soit dans un contexte spécifique)
- Restrictions sur les ISR
 - Ne doit **pas appeler des routines bloquantes** (n'est pas une tâche)
 - Doit être **courte** : mode non préemptif en général
 - Signaler un événement pour lancer une tâche, sauf si ISR très courte (ex: mise à jour horloge système)

86



Gestion des interruptions

- Modes de gestion des périphériques
 - Sous interruption
 - pas/peu de latence dans le traitement des entrées sorties
 - peu de maîtrise des instants de demandes d'interruptions → difficulté à intégrer dans les méthodes d'analyse d'ordonnançabilité
 - Polling : examen périodique des registres de coupleurs de périphériques
 - plus de latence dans le traitement des entrées / sorties
 - plus facile à intégrer dans les méthodes d'analyse d'ordonnançabilité


87



Gestion des interruptions Primitives typiques

- Connexion d'un ISR à un niveau d'interruption
- Masquage / démasquage des interruptions
 - Utilisable pour mettre en œuvre des sections critiques de courte durée
 - Avec mémorisation de l'état précédent (imbricable, gestion d'une pile) ou non

88



Gestion des interruptions

Interruptions OSEK/VDX

- Types de routines d'interruption (ISR = Interrupt Service Routine)
 - **Catégorie 1** : aucun appel système dans l'ISR. Retour systématique à la tâche interrompue par l'ISR. Surcoût minimum.
 - **Catégorie 2** : appels système autorisés. Réordonnancement différé à la fin de l'ISR, et uniquement si pas d'autre interruption demandée
- **Priorités entre ISR**. Une ISR ne peut être interrompue que par une ISR plus prioritaire. Nombre de priorités non normalisé (dépendant matériel et implantation)
- ISR **plus prioritaires** que tâches

89



Gestion des interruptions

Interruptions OSEK/VDX : API

- void DisableAllInterrupts(void);
Désactive toutes les interruptions, en sauvant au préalable la liste des IT activées. Aucun appel système n'est autorisé en mode IT désactivées.
- void EnableAllInterrupts(void);
Restaure l'état sauvegardé par DisableAllInterrupts. Pas d'imbrication gérée
- void SuspendAllInterrupts(void)
void ResumeAllInterrupts(void)
Même chose que DisableAllInterrupts/EnableAllInterrupts sauf que l'imbrication est gérée
- void SuspendOSInterrupts(void); void ResumeOSInterrupts(void)
Même chose que / ResumeAllInterrupts, sauf que concerne uniquement les interruptions de catégorie 2.

90



Gestion de la mémoire

- Types de gestion de mémoire
 - Statique vs dynamique
 - Protégé vs non protégé
 - Quid de la pagination en contexte temps-réel ?

91



Gestion de la mémoire Statique vs dynamique

- Gestion **statique** de mémoire
 - Les **nombres** de tous les objets du noyau sont connus (ou bornés) au chargement du noyau
 - Les **tailles** de tous les objets du noyau (piles, tampons de réception des messages, zones de code, de données) sont connus (ou bornés)
 - Pas de routine de création d'objet / d'allocation de mémoire
 - Avantages
 - pas de pénurie de mémoire à l'exécution (sûreté)
 - temps d'accès aux objets constant et faible (tableaux)
 - Inconvénients
 - Peu flexible

92

Gestion de la mémoire Statique vs dynamique

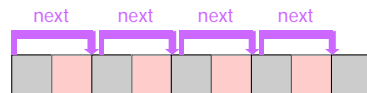
- Gestion **dynamique** de la mémoire
 - On ne connaît pas les nombres et/ou taille des objets
 - Routine de créations d'objets (tâches,sémaphores, ...) et ou d'allocation dynamique de mémoire
 - Avantages
 - flexibilité
 - Inconvénients (voir transparent suivant)
 - Temps d'accès aux objets plus élevé et plus difficile à prédire (listes)
 - Risque de pénurie de mémoire en cours d'exécution
 - Risque d'émiettement de la mémoire (fragmentation)
 - Durée de l'allocation de mémoire difficile à borner

93

Gestion de la mémoire Statique vs dynamique

- Gestion dynamique de mémoire (suite): ex, allocateur « first-fit »
 - Repérer les blocs occupés (pleins) des blocs libres (trous)
 - Chaînage de blocs libres dans les trous eux-mêmes
 - First-fit : on retourne le premier bloc libre de la liste de taille suffisante

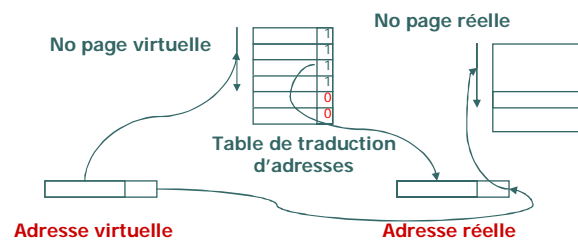
```
for (b = h->first; b != tail ; b = b->next) {  
    if ( b->size >= size ) {  
        break; /* block found */  
    }  
}
```



94

Gestion de la mémoire Protégé vs non protégé

- o Gestion **protégée** de la mémoire
 - Tâches dans des espaces mémoire séparés (espaces d'adressage)
 - **Support matériel** (MMU, mécanisme de segmentation du processeur)
 - Exemple d'utilisation d'une MMU avec table des pages linéaires




95

Gestion de la mémoire Protégé vs non protégé

- o Gestion **protégée** de la mémoire
 - Avantages
 - Résistance aux fautes de programmation
 - Inconvénients
 - Latence de traduction d'adresse (accès à la table des pages), non prévisibilité quand cache de traduction d'adresse - TLB)
 - Occupation mémoire supplémentaire pour les tables de traduction d'adresses
 - Latence de changement de contexte plus élevée (sauvegarde/restauration du contexte mémoire, rechargement TLB)

96




Gestion de la mémoire Protégé vs non protégé

- Gestion **non protégée** de la mémoire
 - Tâches dans le même espace mémoire (accès direct à la mémoire réelle)
 - Avantages
 - temps d'accès à la mémoire constants (pour une technologie mémoire donnée)
 - Inconvénients
 - Accès mémoire erronés non détectés, possibilités d'écrasement du code/données des autres tâches
 - ex:

```
int tab[100];  
for (i=0;i<100;i++) tab[2*i] = 0;
```

97



Gestion de la mémoire Pagination à la demande ?

- Définition
 - Chargement en mémoire vive « à la demande » (swap in) quand une entrée est invalide dans la table de traduction (défaut de page)
 - Recopie en stockage secondaire (disque) quand la mémoire est pleine (swap out)
 - Intérêt : utilisation de tâches ayant de besoins en mémoire supérieurs aux capacités de la mémoire physique

98



Gestion de la mémoire

Pagination à la demande ?

- Quid de la pagination en contexte temps réel ?
 - temps d'accès à la mémoire difficile à prédire et potentiellement **très** grand (latence d'accès au disque) → **à proscrire en contexte temps-réel**
 - utilisé dans les systèmes d'exploitation **généralistes** (non dédiés temps-réel)
 - extensions de systèmes généralistes au temps-réel : primitives de **verrouillage des pages** en mémoire (on interdit leur transfert sur disque), ex : extensions temps-réel de POSIX

99



Gestion de la mémoire

Gestion mémoire dans OSEK/VDX

- Gestion mémoire statique
 - Déclaration de l'ensemble des tâches, ressources, événements, ...
 - Pas de fonction d'allocation dynamique de mémoire (malloc)
- Espace mémoire protégé : non standardisé (dépend matériel)
- Pagination à la demande : absent

Aucun appel système de gestion mémoire

100



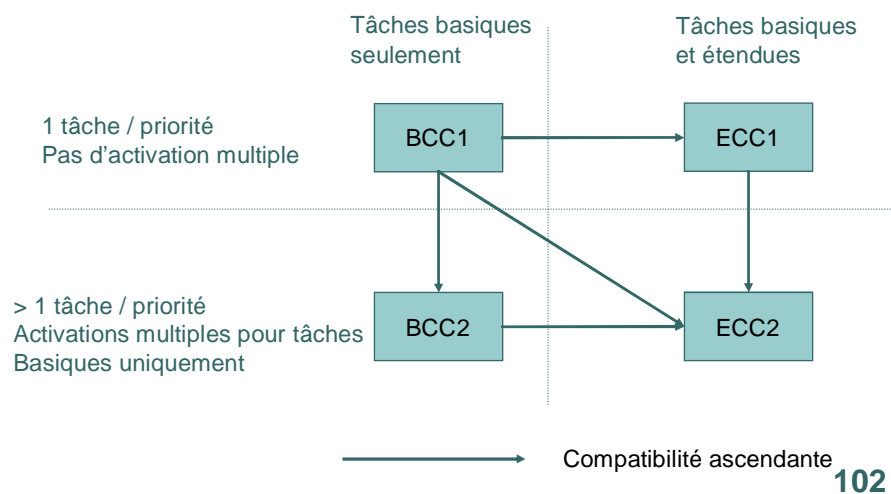
Classes de conformité

- Motivations
 - Adaptation aux caractéristiques des applications
 - Adaptation aux capacités des architectures (processeur, mémoire)
- Définition
 - Implantation d'un sous-ensemble clairement identifié des fonctionnalités OSEK
- Paramètres des classes de conformité
 - Nombre d'activations simultanées des tâches
 - Types de tâches (basiques/étendues)
 - Nombre de tâches par priorités

101



Classes de conformité



102



Classes de conformité

Besoins minimaux pour être conforme à une classe

	BCC1	BCC2	ECC1	ECC2
Exécutions multiples	Non	Oui	BT: non ET: oui	BT: oui ET: oui
Plus d'une tâche par priorité	Non	Oui	Non	oui
Nombre d'événements par tâche	---		8	
Nombre de priorités	8		16	
Ressources	RES_SCHEDULER		8 (incluant RES_SCHEDULER)	
Alarmes				1

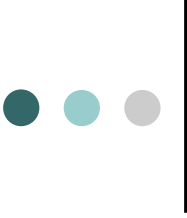
103



Quelques autres exécutifs temps-réel


- Standardisation (d'interface)
 - Extensions temps-réel de POSIX
 - OSEK/VDX (automobile) : les premiers produits sortent (e.g. OSEKWorks de WindRiver)
 - μitron : standard utilisé dans l'embarqué au Japon
- Quelques exécutifs temps-réel industriels
 - VxWorks, pSOS
 - VRTX, QNX
- Produits open-source
 - RT-Linux, RTAI
 - Nucleus, Jaluna

104




Vérification d'ordonnançabilité

Quelques bases



105



Vérification des contraintes temporelles

Introduction (1/3)

- Rôle
 - **Formules mathématiques** ou **algorithmes** permettant de vérifier (prouver) que les tâches respecteront leurs contraintes de temps
- Classification
 - Vérification **hors-ligne** (avant exécution)
 - Critères analytiques calculables (CN, CS, CNS)
 - Simulation
 - Cibles = systèmes temps-réel **strict**
 - Vérification **en-ligne** (pendant exécution)
 - **Tests d'acceptation** en-ligne pour savoir si on accepte de nouvelles tâches
 - Cibles = systèmes temps-réel **souple**

106

Vérification des contraintes temporelles

Introduction (2/3)

- Données d'entrée des méthodes de vérification : **modèle** du système
 - Connaissance des informations sur les tâches (**modèle de tâches**)
 - **Arrivée** des tâches: périodique, sporadique, aperiodique (dates d'arrivées)
 - **Synchronisations** : précédences, exclusions entre tâches
 - Temps d'exécution **au pire-cas (WCET)**
- Sorties
 - Verdict sur le respect des contraintes de temps pour **un algorithme d'ordonnement donné**

107

Vérification des contraintes temporelles

Introduction (3/3)

- Conditions nécessaires et/ou suffisantes
 - Condition **nécessaire** C
 - Il **faut que** C soit vérifiée pour que les contraintes de temps soient respectées (système faisable)
 - C non vérifiée \Rightarrow système non faisable
 - C vérifiée \Rightarrow système peut être faisable (ou pas)
 - Condition **suffisante**
 - Il **suffit que** C soit vérifiée pour que le système soit faisable
 - C vérifiée \Rightarrow système est faisable (à coup sûr)
 - C non vérifiée \Rightarrow système peut être faisable (ou pas)
 - Condition **nécessaire et suffisante**
 - C vérifiée **équivalent** à montrer la faisabilité du système

108

Vérification de contraintes temporelles

Plan

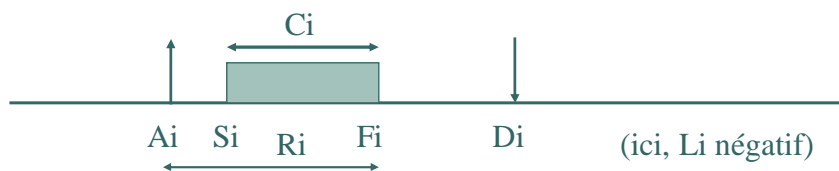
- Faisabilité pour **tâches périodiques et ordonnancements préemptifs à priorités fixes**
 - Rappel des notations et hypothèses
 - Simulation
 - Condition nécessaire d'ordonnançabilité
 - Vérification pour ordonnancements à priorités fixes
 - Condition pour Rate Monotonic (CS) et Deadline Monotonic (CS)
 - Condition générale pour algorithmes à priorités fixes : Response Time Analysis (CNS)

109


Analyse d'ordonnançabilité

Notations (1/2)

- Arrivée A_i
- Temps de calcul maximum sans préemption C_i
- Echéance (deadline) : relative ou absolue D_i
- Date de début (start time) et fin (finish time) S_i, F_i
- Temps de réponse $R_i = F_i - A_i$
- Retard (lateness) $L_i = (F_i - D_i)$



110



Analyse d'ordonnançabilité Notations (2/2)

- Arrivées des tâches
 - **Périodiques** : arrivée à intervalles réguliers (P_i)
 - Date d'activation initiale, **offset** O_i
 - Si pour tout i, j $O_i = O_j$, tâches synchrones
 - Si $D_i = P_i$, tâche à échéance sur requête
- Instant critique : date pour laquelle une arrivée de tâches produira son pire temps de réponse

111



Analyse d'ordonnançabilité en périodique Hypothèses

- Hypothèses explicites
 - Tâches périodiques de période P_i
 - Temps d'exécution pire-cas C_i constant pour toutes les instances
 - Echéance sur requête ($D_i = P_i$)
 - Tâches indépendantes (pas de synchronisation)
 - Tâches synchrones ($O_i = 0$)
- Hypothèses implicites
 - Pas de suspensions des tâches
 - Une tâche peut être lancée dès son arrivée (A_i) : pas de gigue au démarrage
 - Surcoûts du système d'exploitation (démarrage tâche, préemption) nuls

112

Analyse d'ordonnançabilité en périodique

Vérification par simulation

- Périodicité des tâches → ordonnancement cyclique (pas la peine de simuler à l'infini)
- Durée de la période d'étude (ou pseudo-période)
 - Tâches synchrones : [0,PPCM(Pi)]
 - Tâches échelonnées (au moins un offset Oi non nul)
 - [min(Oi), max(Oi,Oj+Dj) + 2 * PPCM(Pi)]
- Evaluation
 - Valide quel que soit l'algorithme d'ordonnancement
 - Période d'étude peut être très longue selon les relations entre périodes
 - Adapté aux ordonnancements hors-ligne

113

Analyse d'ordonnançabilité en périodique

Condition nécessaire d'ordonnançabilité

- Notion de charge processeur

$$U = \sum_{i=1}^n \frac{C_i}{P_i}$$

- Condition nécessaire d'ordonnançabilité : $U \leq 1$
 - Cette condition n'est pas une condition suffisante (pas pour tous les ordonnancements)

114

Analyse d'ordonnançabilité en périodique

Faisabilité pour Rate Monotonic

(1/2)

- Définition RM : la priorité d'une tâche est inversement proportionnelle à sa période (conflits résolus arbitrairement)
- Condition de faisabilité

$$\sum_{i=1}^n \frac{C_i}{P_i} \leq n(2^{\frac{1}{n}} - 1)$$

- faible complexité $O(n)$
- condition **suffisante** seulement (si $U \in [U_{\text{lub}}, 1]$, le jeu de tâches **peut** respecter ses échéances)
- s'applique aussi quand les tâches ne sont pas synchrones

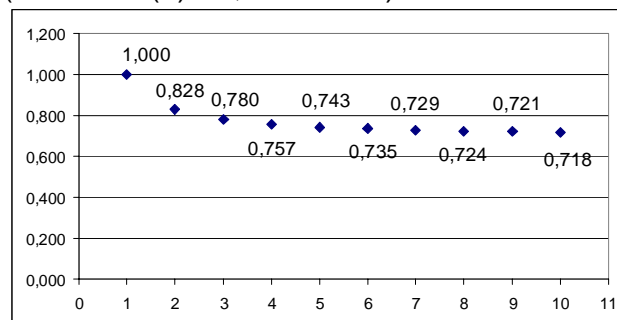
115

Analyse d'ordonnançabilité en périodique

Faisabilité pour Rate Monotonic

(2/2)

- Variation du seuil de charge en fonction de n (limite = $\ln(2) = 0,69314718$)



- Cas particulier : périodes des tâches sont des **harmoniques** (pour tout i, j tq $P_i < P_j$, $P_j = k P_i$ avec k entier) : $U_{\text{lub}} = 1$

116

Analyse d'ordonnabilité en périodique

Faisabilité pour Deadline

Monotonic

- Hypothèse précédentes sauf H3 : $D_i \leq P_i$
- Définition DM [Leung & Whitehead, 1985]
 - La priorité d'une tâche est inversement proportionnelle à son échéance relative (conflits résolus arbitrairement)
- Condition de faisabilité

$$\sum_{i=1}^n \frac{C_i}{D_i} \leq n(2^{\frac{1}{n}} - 1)$$

- faible complexité $O(n)$
- condition **suffisante** seulement (si $\sum (C_i/D_i) > U_{lub}$, le jeu de tâches **peut** respecter ses échéances)
- s'applique aussi quand les tâches ne sont pas synchrones

117

Analyse d'ordonnabilité en périodique

Response Time Analysis (1/6)

- Response time analysis (RTA, analyse de temps de réponse) [Joseph & Pandya, 1986, Audsley & al, 1993]
 - Applicable pour **tous** les ordonnancements à priorité statique, quel que soit l'assignation des priorités (RM, DM, autre)
 - Condition **nécessaire et suffisante**
 - Utilisable pour tout D_i (même si non inférieur à P_i : dans ce cas, RM et DM ne sont plus optimaux)
 - S'applique aussi quand les tâches ne sont pas synchrones, mais est alors une condition suffisante seulement

118



Analyse d'ordonnançabilité en périodique Response Time Analysis (2/6)

- Principe
 - Calcul du **temps de réponse** R_i de chaque tâche à son instant critique (calcul itératif)
 - Vérification pour tout i que $R_i \leq D_i$

119



Analyse d'ordonnançabilité en périodique Response Time Analysis (3/6)

- Introduction au calcul de temps de réponse R_i

$R_i = C_i$ (temps d'exécution pire-cas de T_i)
+ I_i (**interférences** dues à des préemptions par tâches plus prioritaires)

Nombre maximal de préemptions subies par une instance de T_i :

$$\sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil$$

Interférence correspondante
(C_j pour chaque préemption par T_j)

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil C_j$$

Temps de réponse R_i

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil C_j$$

120



Analyse d'ordonnançabilité en périodique Response Time Analysis (4/6)

- Pas de solution simple (R_i apparaît des deux cotés)
 - Solution R_i = plus petite valeur telle que l'équation est vérifiée
 - Pas besoin de vérifier l'équation $\forall R_i \leq D_i$ (seulement quand le nombre de préemptions augmente)

121



Analyse d'ordonnançabilité en périodique Response Time Analysis (5/6)

- Algorithme de calcul des R_i (itératif)

$$w_i^0 = C_i$$
$$w_i^{k+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^k}{P_j} \right\rceil C_j$$

- Quand la série converge, on a calculé le temps de réponse R_i
- Le système est ordonnançable quand $R_i \leq D_i$

122



Analyse d'ordonnancement en périodique Response Time Analysis (6/6)

o Exemple

	C _i	P _i	prio
T4	3	10	
T3	1	5	
T2	2	20	
T1	2	20	

$$w_1^0 = C_i = 2$$

$$w_1^1 = C_i + \left\lceil \frac{C_1}{P_4} \right\rceil C_4 + \left\lceil \frac{C_1}{P_3} \right\rceil C_3 + \left\lceil \frac{C_1}{P_2} \right\rceil C_2 = 2 + \left\lceil \frac{2}{10} \right\rceil 3 + \left\lceil \frac{2}{5} \right\rceil 1 + \left\lceil \frac{2}{20} \right\rceil 2 = 2 + 3 + 1 + 2 = 8$$

$$w_1^2 = C_i + \left\lceil \frac{8}{P_4} \right\rceil C_4 + \left\lceil \frac{8}{P_3} \right\rceil C_3 + \left\lceil \frac{8}{P_2} \right\rceil C_2 = 2 + \left\lceil \frac{8}{10} \right\rceil 3 + \left\lceil \frac{8}{5} \right\rceil 1 + \left\lceil \frac{8}{20} \right\rceil 2 = 2 + 3 + 2 + 2 = 9$$

$$w_1^3 = C_i + \left\lceil \frac{9}{P_4} \right\rceil C_4 + \left\lceil \frac{9}{P_3} \right\rceil C_3 + \left\lceil \frac{9}{P_2} \right\rceil C_2 = 2 + \left\lceil \frac{9}{10} \right\rceil 3 + \left\lceil \frac{9}{5} \right\rceil 1 + \left\lceil \frac{9}{20} \right\rceil 2 = 2 + 3 + 2 + 2 = 9$$

123