Chapitre I

INTRODUCTION AUX SYSTÈMES D'EXPLOITATION EMBARQUÉS

Imene Sghaier

PLAN DU CHAPITRE

- Définition
- Champs d'application
- Propriétés des systèmes embarqués
- Contraintes des systèmes embarqués
- Architecture typique d'un système embarqué
- Software pour les SEMs
- Systèmes d'exploitation pour l'embarqué
- Processus, programmes et tâches
- Types et caractéristiques de tâche
- Threads
- Gestion de processus

Définition

- **Un système embarqué est un système électronique et informatique autonome qui est dédié à une tâche particulière et qui est contenue dans un système englobant. Il n'est «généralement» pas programmable.**
  - Matériel et application intimement liés
  - Logiciel enfoui … noyé dans le matériel … pas facilement discernable comme dans un PC.
  - Pas d'E/S standards
  - Matériel et application intimement liés

CHAMPS D'APPLICATION: 4 CLASSES

- **Calcul généraliste** :
  - Similaire aux applications bureau mais embarqué (assistant personnel, téléphone portable, etc.)
  - Consoles de jeux vidéo,
- **Contrôle de systèmes :** Contrôle de systèmes en Temps Réel (Moteur d'automobile, processus chimique, processus nucléaire, système de navigation aérien)
  - Moteur, voiture, avion, processus chimique, nucléaire, navigation, etc.
- **Traitement du signal :** Calcul sur de grosses quantités de données (Radar, Sonar, compression vidéo.
  - Compression vidéo, radar, flux de données, etc.
- **Réseaux et communications**
  - Transmission de données, commutation, routage, téléphone, Internet, etc.

PROPRIÉTÉS DES SEM

L'autonomie

- Les systèmes embarqués doivent en général être autonomes.
- Cette autonomie est nécessaire lorsque l'intervention humaine est impossible, mais aussi lorsque la réaction humaine est trop lente ou insuffisamment fiable.
- On peut dégager deux types d'autonomie :
  - **Autonomie de fonctionnement :**
    - remplir leur mission sans intervention humaine.
  - **Autonomie fonctionnelle (Totale ou Partielle) :**
    - exécution des services sans sollicitation à d'autres systèmes leur mission sans intervention humaine.
  - **Autonomie énergetique :**
    - avoir sa propre source d'alimentation capable de tenir pendant de périodes plus ou moins longues selon l'application.

PROPRIÉTÉS DES SEMS

La spécificité

- Les systèmes embarqués sont dédiés pour une ou plusieurs applications spécifiques pour lesquelles ils ont été crées.
- Ils doivent être dotés d'un ensemble de composants matériels et logiciels spécialisés afin de réaliser ses taches requises.

La complexité / l'intelligence

- La cohabitation de l'électronique analogique, numérique, des composantes RF (RadioFréquence) sans interférences est une tâche difficile.
- La complexité d'un SEMs augmente avec le nombre de ses composants et le nombre des ses applications spécifiques.
- Intelligence c'est le moyen d'améliorer la qualité du système
- Exemple : Auto-diagnostic, Auto-configuration, Adaptabilité, Evaluation des risques…

PROPRIÉTÉS DES SEMS

La réactivité

- Un SEM doit interagir avec son environnement à une vitesse qui est imposée par ce dernier. Ceci induit donc des impératifs de temps de réponse. C'est pour cette raison que le Système embarqué est souvent basé sur un système temps réel.
- L'autonomie fonctionnelle diminue avec la croissance de la réctivité

La Mobilité

- C'est la capacité ou la possibilité de déplacement du système.
- Le but est de concevoir des solutions mobiles qui facilitent le déplacement du système.
- Un système mobile contient des outils qui peuvent faciliter certains processus et répondre à des besoins réels.
  - *Exemple : la géolocalisation, …*

PROPRIÉTÉS DES SEMS

L'exécution en temps réel

- L'exécution des opérations doivent être faites avec des échéances (deadlines) précises au-delà de laquelles les résultats ne sont plus valides.
  - *Exemple critique : le contrôleur de frein d'une voiture*
- « Real-Time is not Real-Fast » : un système temps réel n'est pas forcement un système rapide. Il existe des contraintes strictes qui ne sont pas courtes (dépend de l'environnement).
- Il ne s'agit pas de rendre le résultat le plus rapidement possible, mais simplement à temps.
- La puissance de calcul : condition nécessaire mais pas suffisante.
- Vitesse moyenne ou vitesse maximale n'est pas importante, le pire-cas importe.

  - ***Attention :*** *Ne pas confondre temps réel et rapidité*
  - *Temps réel veut dire prédictibilité et non rapidité*

PROPRIÉTÉS DES SEMS

L'exécution en temps réel

- **Temps réel dur** (**hard real-time**) : le non respect des contraintes temporelles entraîne la faute du système.
  - Les échéances ne doivent jamais être dépassées.
  - *Exemple : contrôle de trafic aérien, système de conduite de missile, ...*
- **Temps réel souple** (**soft real-time**) :
  - le respect des échéances est important mais le non respect des échéance n'a pas de graves conséquences.
  - Le système doit répondre au mieux, le plus rapidement possible.
  - *Exemple : système d'acquisition de données pour affichage, …*
- **Temps réel ferme** (**firm real-time**) : temps réel souple, mais si l'échéance est dépassée le résultat obtenu n'a plus de valeur (il est donc écarté).
  - *Exemple : projection vidéo, …*

PROPRIÉTÉS DES SEMS

L'exécution en temps réel

- La plupart des systèmes embarqués sont des systèmes "**mutlirate**" ou "**multipériode**"
  - Les données sont capturées à un certain rythme.
  - Les traitement sur ces données ne sont pas forcément à la même granularité.
  - Les actionneurs fonctionnent à une fréquence différentes.
- Un STR est généralement géré par un système d'exploitation ou un noyau Temps Réel (Real Time Operating System, RTOS).
- L'échelle de temps de l'échéance peut varier d'une application à l'autre
  - microseconde en contrôle radar
  - milliseconde pour la synchronisation image/son (mpeg)
  - minute pour les distributeurs automatiques

PROPRIÉTÉS DES SEMS

La robustesse, fiabilité et sécurité

- L'environnement est souvent hostile, pour des raisons physiques (chocs, variations de température, impact d'ions lourds dans les systèmes spatiaux, ...) ou humaines (malveillance). C'est pour cela que la sécurité - au sens de résistance aux malveillances - et la fiabilité - au sens de continuité de service - sont souvent rattachées à la problématique des systèmes embarqués.
- Le système doit être opérationnel même lorsqu'un composant est défaillant.

PROPRIÉTÉS DES SEMS

La consommation d'énergie

- Le Système Embarqué Mobile doit être à faible consommation car il est alimenté par des batteries.
- Une consommation excessive augmente le prix du système car il faut alors des batteries de plus forte capacité.

La flexibilité

- La flexibilité des SEMs se résume par la capacité de changer quelques propiétés ou fonctionnalités du système.
- Ces changements peuvent être réalisés par :
  - la reprogrammation de son calculateur
  - la reconfiguration matérielle de ses circuits logiques programmables (FPGA, CPLD, PLD)
  - la réutilisation de ses composants virtuels(IP)

PROPRIÉTÉS DES SEMS

Le volume / le poids

- La miniaturisation des SEMs est un facteur important qui affecte directement sa mobilité.
- Réduction de l'encombrement grâce à la miniaturisation des composants électroniques.
- La miniaturisation du système nécessite plus de temps de conception et réduit le délai de vie du système.

Le coût

- Beaucoup de systèmes embarqués sont fabriqués en grande série et doivent avoir des prix extrêmement faibles pour faire face à la concurrence.

Optimisation

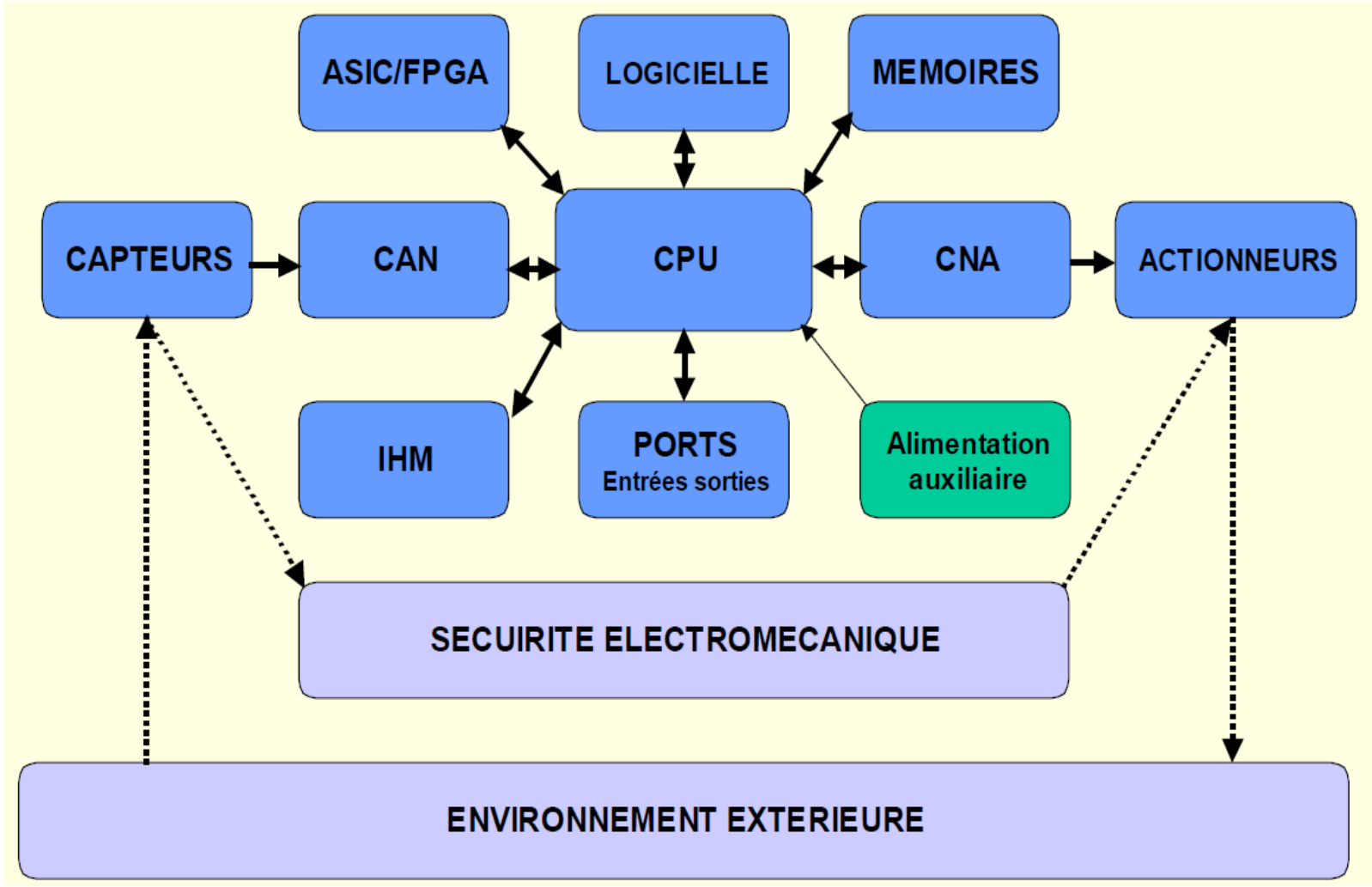
- Généralement logicielles de petite taille car plus c'est grand, plus il y a de chance d'avoir des bugs. Ce sont des logiciels produits à grande échelle, le moindre centime compte.

Tolérance aux fautes

Contrainte des systèmes embarqués

- contrainte de taille mémoire: l'estimation la plus précise possible de l'utilisation mémoire
- Contrainte de puissance de calcul: la sélection du processeur d'après la charge de travail a effectuer et d'après la largeur des registres.
- contrainte de temps de réponse: le système doit réagir vite
- contrainte de fiabilité
- contrainte de sécurité
- contrainte de ressource d'énergie/autonomie : : la conception hardware (dispositifs utilisés) et software (complexités des algorithmes implémentés) doit prendre en compte la consommation.
- contrainte d'architecture matérielle
- contrainte de prix de développement (pur et licences)
- contrainte de prix de vente (amortissement et royalties)
- contraintes... juridiques !

ARCHITECTURE TYPIQUE D'UN SEM



Architecture typique d'un SEM

- Capteurs (interrupteurs, etc) couplés à des convertisseurs analogique/numérique.
- Actionneurs (LED, etc) couplés à des convertisseurs numérique/analogique.
- Calculateur (processeur embarqué et ses E/S).
- Possibilité d'avoir un/des FPGAset (Field programmable gate array : circuits intégrés)/ou ASICs pour jouer le rôle de coprocesseurs (accélération matérielle)
- Les SEMs utilisent généralement des microprocesseurs à basse consommation d'énergie (niveau logique +5V) ou des microcontrôleurs, dont la partie logicielle est en partie ou entièrement programmée dans le matériel, généralement en mémoire dans une mémoire morte (ROM), EPROM, EEPROM, FLASH, etc.

SOFTWARE POUR LES SEMS

- Un Logiciel embarqué: programme/application utilisé dans un équipement et complètement intégré dans ce dernier
- Les logiciels créés pour les systèmes embarqués sont appelés firmwares.
- Stockés généralement dans la mémoire en lecture seule ou dans la mémoire flash

Système embarqué: Matériel(s) + logiciel(s) (+ OS)

- Ils fonctionnent le plus souvent avec des ressources matérielles limitées
- **2 types de systèmes embarqués:**
  - Systèmes embarqués destinés à l'utilisateur (high-end): généralement une version dégradée d'un OS existant (ex: Linux). Ex: routeurs, PDA, etc.
  - Systèmes embarqués profondément enfouis: peu de fonctions, très petite empreinte mémoire, généralement construit fromscratch.

    Ex: Appareil photo numérique, téléphones portables, etc.

Systèmes d'exploitation pour l'embarqué

Pourquoi un système d'exploitation?

- Les systèmes d'exploitation permettent de gérer les ressources matérielles en assurant leurs partages entre les différents utilisateurs.
- De présenter une interface homogène et générique(en abstrayant la complexité matérielle): une machine virtuelle mieux adaptée aux utilisateurs.

Systèmes d'exploitation pour l'embarqué

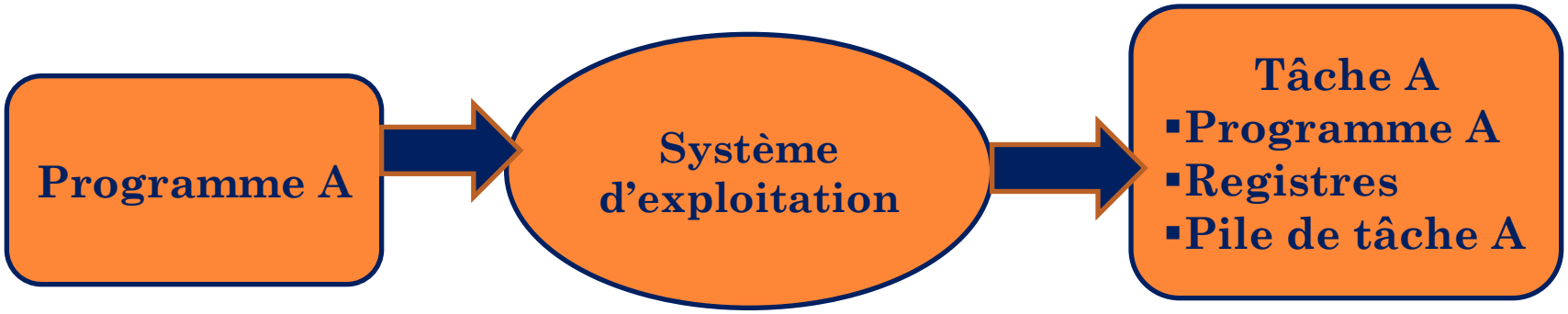
Pourquoi un système d'exploitation pour l'embarqué ?

- Affranchir le développeur de logiciel embarqué de bien connaître le matériel => gain en temps de développement
  - Les applications doivent avoir un accès aux services de l'OS via des APIs (réutilisabilité du code, interopérabilité, portabilité, maintenance aisée)
- Environnement de développement plus performant :
  - Maximiser la capacité de traitement (débit) & utilisation des ressources
  - Contrôle de processus sans contrainte temporelle avec un systèmes à temps partagé
  - Contrôle de processus avec contrainte temps réel avec garantie des temps de réponse
    - Systèmes à contraintes souples/molles: systèmes acceptant des variations minimes de temps de réponse (systèmes multimédias)
    - Systèmes à contraintes dures: gestion stricte du temps pour conserver l'intégrité du système (déterminisme logique et temporel et fiabilité)

    => Garantie d'un temps de réponse bon au pire cas WCET

PROCESSUS, PROGRAMMES, TÂCHES

- **Un programme** est une séquence d'instructions
- **Un processus/tâche** est l'entité dynamique qui nait de l'exécution d'un programme.



TYPES ET CARACTÉRISTIQUES DE TÂCHE

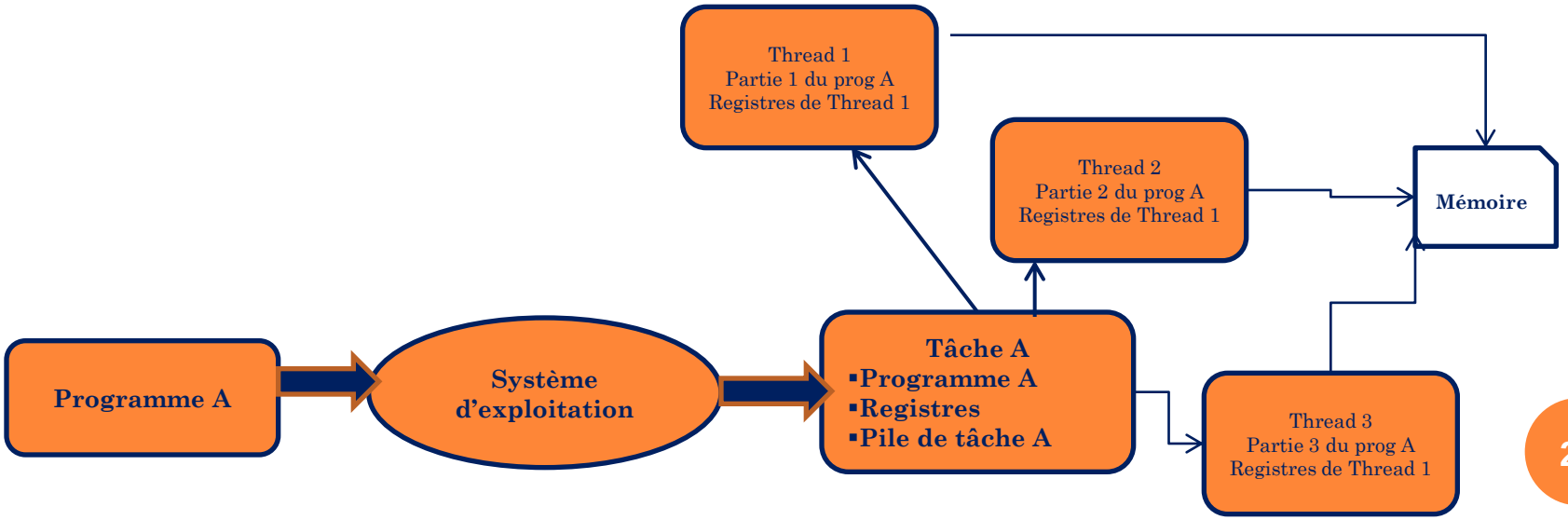
- **Caractéristiques**: Une tâche est caractérisée par ses contraintes temporelles :
  - date de réveil
  - durée d'exécution
  - délai critique (échéance)
- **Types de tâches**
  - **Tâches périodiques**: la tâche est activée à intervalles réguliers avec une durée *Ci et une période Ti* .
  - **Tâches sporadiques**: la tâche s'active à un instant quelconque, néanmoins il existe un temps minimal entre deux activations
  - **Tâches apériodiques**: on ne peut faire aucune hypothèse sur l'instant d'activation des tâches

THREAD

- **La tâche atomique ou processus légér** (thread) est le plus petit composant d'une tâche ou processus
  - Chaque processus peut contenir un ou plusieurs thread (applications dites multi-threadées) s'exécutant en quasi-simultanéité ou simultanément sur les processeurs multi-coeurs.
  - Les «petits» OS pour l'embarqué utilisent seulement les threads alors que les «grands» OS peuvent utiliser plusieurs modèles processus/threads.

THREAD

- A la différence des processus, les threads partagent les mêmes ressources et le même espace mémoire.
- Les threads d'une même tâche partagent le même répertoire de travail, fichiers, dispositifs d'E/S, données globales, espace d'adressage, code du programme, etc. mais ne partagent pas: le compteur de programme, la pile, les informations d'ordonnancement, etc.



GESTION DES PROCESSUS

- Mono-tâche :CPU dédié à un et un seul processus
- Multi-tâche :les processus se partagent le ou les CPUs
  - Multi-tâches à temps-partagé ou préemptif: un ordonnanceur gère les processus.E
- Exemples
  - **DOS-C**: OS embarqué mono tâche.
  - **VxWorks**(WindRiver vous devez payer pour l 'avoir ^_^): un unique type de thread/tâche: chaque thread/tâche implémente un thread d'exécution
  - **Embedded Linux** (TimeSys): 2 types de threads le fork de Linux et les tâches périodiques

GESTION DES PROCESSUS

- L'objectif principal de la notion de multitâche est d'exploiter le plus efficacement possible toutes les ressources d'un système.
- Le système d'exploitation doit alors mettre en oeuvre les règles de possession et de partage de ressources.
- Il doit aussi définir les compétiteurs (processus) et les contrôler tout au long de leurs exécutions. Donc, les programmes susceptibles de partager les différentes ressources doivent être indépendantes les uns des autres. Une telle approche caractérise le multitâche.

GESTION DES PROCESSUS

Création de processus

- Il existe 2 modèles utilisés lors de la création de processus :
  - ***fork/exec*** qui est dérivé du standard IEEE/ISO POSIX 1003.1
  - ***spawn*** qui est dérivé du fork/exec
- La différence entre fork/exec et spawn réside dans la façon dont la mémoire est allouée avec chacun des 2 modèles.
  - Avec fork/exec l'espace mémoire du processus père est hérité directement ou recopié (ex: code du programme et variables) ce qui donne une certaine flexibilité avec le changement des caractéristiques du processus.
  - Avec spawn la création d'un nouvel processus consiste en l'allocation d'un nouvel espace d'adressage donc pas de duplication ce qui engendrera la destruction de l'espace d'adressage du nouveau processus.

GESTION DES PROCESSUS

```
//Exemple avec fork
#include <sys/types.h>
#include <unistd.h>

void program(void)
{
processId child_processId;
/* dupliquer le prcessus: processus
fils*/
child_processId = fork();
if (child_processId == -1) {
ERROR;
}
else if (child_processId == 0) {
run_childProcess();
}
else {
run_parentParent();
}
```

```
/**Exemple avec Spawn en
utilisant la fonction execvp **/
#include <sys/types.h>
#include <unistd.h>

int program (char* program, char**
arg_list){
processed child_processId;
/* Duplication du processus*/
child_processId = fork ();
if (child_pId != 0)
/* This is the parent process */
return child_processId;
else{
/* Executer PROGRAM en le recherchant
dans
le PATH*/
execvp (program, arg_list);
/* execvp ne retourne qu'en cas d'erreur*/
fprintf (stderr, "Error in execvp\n");
abort (); } }
```

Suppression de processus

- Plusieurs raisons:
  - Terminaison normale
  - Problème matériel: manque de mémoire, etc.
  - Problème logiciel: instruction invalide, etc.
- Suppression d'une tâche:
  - L'OS libère toute la mémoire allouée: PCB, variables, code, etc.
  - Si parent supprimé:
    - Les fils sont adoptés par un autre processus
    - OU supprimés eux aussi avec libération des ressources partagées.

GESTION DES PROCESSUS

Ordonnancement

- On appelle ordonnancement la stratégie d'attribution des ressources aux processus qui en font la demande.
- L'ordonnanceur est le composant de l'OS qui détermine l'ordre et la durée des tâches qui s'exécutent sur le CPU.
- L'ordonnancement dans un domaine embarqué est un compromis entre la prédictibilité des contraintes temps réel, la complexité d'implémentation, et le délai d'exécution.

GESTION DES PROCESSUS

- **Ordonnancement sur un processeur**:
  - Plusieurs tâches s'exécutent sur un seul processeur
  - Un ordonnanceur, passe d'une tâche à l'autre pour simuler une exécution concurrente.
  - le temps est partagé entre les tâches de manière *équitable*
- **Ordonnancement sur multi-processeur**:
  - Plusieurs tâches s'exécutent sur plusieurs processeurs, en parallèle.
- **Temps réel**
  - Dans les systèmes temps réel, les tâches doivent être exécutées dans les délais impartis
  - D'où les stratégies d'ordonnancement utilisées sont différentes : préemption et commutation de contexte
  - Dans un système temps réel préemptif: une tâche est sélectionnée par l'ordonnanceur pour obtenir le processeur, cette tâche peut interrompre une autre tâche moins prioritaire
- **Objectifs**
  - Comment répartir le temps processeur(s) parmi un groupe de tâches ?
  - Peut-on garantir qu'aucune tâche ne ratera une échéance ?

GESTION DES PROCESSUS

- Différents critères d'ordonnancement peuvent être pris en compte dans un système embarqué :
  - Temps de réponse de l'ordonnanceur: temps pris par l'ordonnanceur pour effectuer le changement de contexte pour une tâche prête, ce temps inclut le temps d'attente de la tâche dans la file des tâches prêtes. Il faut généralement effectuer une action avant son échéance (freinage pour un système de contrôle de vitesse) ou avec un temps moyen fixé (décompresseur video, 24 images par secondes environ).
  - Le temps pris par le processus pour compléter son exécution (Turnaround time).
  - Débit ou rendement: nombre de tâches traitées par unité de temps (robot de production dans une usine)
  - Équité: quels sont les facteurs déterminant quant au choix de la tâche à exécuter.
  - Famine: un ordonnanceur doit (dans certains cas) assurer que ce problème ne se produise pas…
  - Préemptivité et non préemptivité
  - Si le système est en plus temps-réel ce qui est généralement le cas pour les systèmes embarqués il faut guarantir la précision: il faut effectuer une action à un moment précis dans le temps (horloge électronique Horloge Temps-réel).

GESTION DES PROCESSUS

Ordonnancement et multitâche

- Plusieurs problèmes se posent lorsque plusieurs tâches s'exécutent simultanément :
  - accès au processeur
  - accès concurrent à la mémoire
  - accès aux périphériques
- Il faut prévoir un ordonnancement permettant au système de remplir son rôle.
- Temps partagé et temps réel
  - Dans les systèmes classiques, le temps est partagé entre les tâches de manière é*quitable*
- Dans les systèmes temps réel, les tâches doivent être exécutées dans les délais impartis
  - D'où les stratégies d'ordonnancement utilisées sont différentes : préemption et commutation de contexte
- Dans un système temps réel préemptif: une tâche est sélectionnée par l'ordonnanceur pour obtenir le processeur, cette tâche peut interrompre une autre tâche moins prioritaire
- Un processus peut être composé de plusieurs threads: le passage d'un thread à un autre est un changement de contexte rapide car les threads partagent le contexte du processus auquel ils sont attachées

GESTION DES PROCESSUS

Quand ordonnancer?

- Un ordonnanceur intervient dans les cas suivants:
  - Fork (père ou fils)
  - Fin d'un processus et en sélectionner un autre Si aucun, un processus spécial est exécuté (idle ou Processus inactif du système)
  - Quand un processus devient bloqué c'est-à-dire en état d'attente d'une ressource non disponible ou d'une condition
  - Interruption E/S: quand une ressource attendue par un processus bloqué redevient disponible
  - Ordonnancement non préemptif
  - Ordonnancement préemptif

GESTION DES PROCESSUS

Procédure d'ordonnancement

- Passer d'un processus à un autre est coûteux

1. Sauvegarde de l'état du processus
2. Chargement du nouvel état (registres, mémoire…)
3. Démarrage du nouveau processus
4. Invalidation probable du cache mémoire

- Les processus alternent souvent des phases de calcul avec des phases d'E/S □

GESTION DES PROCESSUS

Types d'ordonnanceurs

- Les systèmes ont des propriétés et des besoins différents
- Il faut des ordonnanceurs adaptés
  - Batch :
    - Pas d'utilisateurs devant des écrans,
    - ordonnanceurs non préemptifs ou presque…
    - Changements de processus réduits
    - Ex: compilateur de langage, recherche dans des BDs, calcul scientifique, etc.
  - Interactif : Préemptif pour maintenir la réactivité
    - Ex: éditeur de texte, shell, etc.
  - Temps réel: préemptif
    - Ex: Application vidéo, audio, collecte de données à partir d'un capteur
  - Hors ligne/ en ligne (statique/dynamique)
  - Conduits par la notion de priorité (fixe ou dynamique)

Gestion de processus

Algorithmes d'ordonnancement

- Algorithmes statiques: table d'exécution ou analyse « rate monotonic »:
  - RM
  - DM
- Algorithmes dynamiques:
  - EDF
  - LLF

GESTION DE PROCESSUS

Table d'exécution

- Avantages :
  - l'ordre d'exécution des tâches est connu
  - l'accès aux ressources est centralisé
- Inconvénients :
  - les événements entrants ne peuvent pas être traités rapidement
  - les caractéristiques du système doivent toutes être connues à l'avance

GESTION DE PROCESSUS

- **Rate Monotonic**: les tâches se voient assignées une priorité relative au nombre de fois où elle s'exécutent (période).
- Etant donné un ensemble n de processus indépendants qui s'exécutent périodiquement: Plus la périodicité d'une tâche est petite, plus on lui accorde une priorité importante.
- Toutes les dates limites d'exécution de tâches seront satisfaites si:

$$\sum E_i / T_i \leq n(2^{1/n} - 1)$$

Avec:

- i: tâche périodique
- n: nombre de tâches périodiques
- Ti: la période d'exécution de la tâche i
- Ei: le pire temps d'exécution de la tâche i pour une période.
- Ei/Ti : la fraction de temps CPU requise pour exécuter la tâche i.

- **Avantages:**
  - Très simple à implémenter
  - Prédictible
  - Priorités statiques
- **Inconvénients:**
  - Modèle restrictif
  - Utilisation du processeur sous-optimale
  - le rythme d'arrivée des tâches et leur temps d'exécution doivent être bien définis

GESTION DE PROCESSUS

- **L'algorithme d'ordonnancement Deadline Monotonic** (DM)
  - Algorithme d'ordonnancement préemptif à priorités constantes affectées selon le principe suivant : la tâche la plus prioritaire est celle dont le délai critique est le plus faible (délai maximum acceptable pour l'exécution d'une tâche).

L'algorithme Least Laxity First

- le travail à qui il reste le moins de marge s'exécute d'abord. Donc, il faut estimer le temps nécessaire pour chaque travail

GESTION DE PROCESSUS

L'algorithme Earliest Deadline First:

- le travail dont le résultat est nécessaire le plus rapidement est exécuté d'abord. À tout instant *t, le travail le plus prioritaire est celui dont l'échéance* est la plus courte.
- Priorité dynamique contrairement à RM
- Trois paramètres à prendre en compte et connaître pour donner des priorités aux processus:
  - Fréquence: nombre de fois que le processus est exécuté
  - Date limite: quand le processus doit avoir terminé son exécution.
  - Durée: temps d'exécution du processus.
- **Avantages:**
  - Simple à implémenter
  - Bonne utilisation du processeur
  - Réactif pour les tâches à échéances courtes
- **Inconvénients:**
  - Mauvais comportement en cas de surcharge (effet domino)
  - Moins prédictible
  - Une échéance manquée provoque une avalanche de retards d'échéances

Gestion de processus

- Si l'ordonnanceur respecte ses dates limites:
  - Il s'agit bien d'un système d'exploitation temps-réel (RTOS)
  - Sinon, il est juste embarqué
- RTOS: Ordonnancement doit être préemptif (la tâche de plus haute priorité doit préempter celle en cours d'exécution)
- Exemple d'implémentation de RTOS:
  - VxWorks: basé sur la priorité et tourniquet
  - Jbed: EDF
  - Linux (TimeSys): basé sur la priorité.

RÉFÉRENCES

- http://www.pace.ch/cours/cours2/procos1.htm
- http://syst.univ-brest.fr/boukhobza/index.php/systemes-dexploitation-pour-lembarque
- Dr. Mohamed Wassim youssef, Cours de « Systèmes d'exploitation évolués »