Programmation temps-réel, introduction

Aline Huf. <alinehuf@ai.univ-paris8.fr>

2016-2017

Table des matières

1 Le temps réel

- programmation système
- programmation parallèle
- programmation concurrente
- communications MPI/OpenMP
- mémoire partagée/répartie
- Message Passing Interface (norme définissant une bibliothèque de fonctions)
- threads POSIX
- POSIX (famille de normes, standardisation des API (Application Programming Interface))

1.1 Mais qu'est-ce que le temps réel ?

Discussion : citer des applications/systèmes temps réel

Applications : surveillance de centrales nucléaires, météo, aviation, automobile, bourse, jeux vidéos, traitement d'image/du son, visio-conférence.
Systèmes sans contrainte temporelle : on veut un résultat juste peu importe quand
Systèmes temps réel : contrainte de temps et/ou de qualité de service.

1.2 Le temps

Temps réel souple : échéance souple (soft deadline)
- s'accommode de dépassements des contraintes temporelles
- après la limite : devient pénible ou inutilisable
- ex : visio-conférence, jeux en réseau

Temps réel strict ou dur (hard real-time) : échéance stricte (hard deadline)
- doit respecter des limites temporelles données même dans la pire des situations d'exécution possibles
- une réponse en retard ne vaut pas mieux qu'une réponse fausse
- ex : pilote automatique d'avion, surveillance de centrale nucléaire, voitures autonomes

1.3 La qualité de service

Temps réel strict
- tâche critique : une tâche doit absolument être réalisée correctement et dans un temps donné

➥ risque de coût financier ou humain (exploration spatiale, alerte dans une centrale nucléaire)

Temps réel souple
- qualité de service : on peut la diminuer pour tenir les délais

➥ renoncer à l'affichage de certaines images, baisser la définition, la qualité de la prévision (météo : indice de confiance).

1.4 Évaluation d'un système temps réel

Respect des contraintes temporelles.

Temps réel strict : fonction booléenne
- respect de l'échéance de chaque activité
- différents services et algorithmes utilisés :
  - ➤ exécutables en un temps borné
  - ➤ peuvent être interrompus par un processus plus prioritaire

Temps réel souple : fonction complexe dépendant de l'application
- compromis entre qualité des résultats et échéance moyenne
- marge ou le résultat est de moins en moins bon puis inacceptable.

1.4.1 Caractéristiques d'un système temps réel

Prévisibilité (predictability)

- interruption d'une activité en cours : temps de réaction plus ou moins long et prévisible
- prévoir le temps d'exécution et la périodicité de chaque tâche
- politique d'ordonnancement : prévoir l'enchaînement des tâches

Déterminisme (determinism)

- pouvoir déterminer le comportement temporel du système (même situation = même comportement)
- éliminer toute incertitude (= conserver la prévisibilité)
  - ➤ sur le comportement de chaque tâche individuellement
  - ➤ Sur le comportement des activités groupées dans le contexte d'exécution
  - ➤ (+ordonnancement)
- facteurs de variation :
  - ➤ charge de traitement : variation des durées de traitement
  - ➤ entrées/sorties : durée des communications et temps de réaction
  - ➤ interruptions : temps de réponse du système
  - ➤ exceptions : erreurs et exceptions matérielles et logicielles =fiabilité

Fiabilité (reliability)

- système tolérant aux fautes (fault tolerant)
- garantir le comportement du système et de ses composants
  (prévisibilité ⇒ déterminisme ⇒ fiabilité)
- Systèmes embarqués (embedded systems) : pas d'intervention humaine possible
- Systèmes dédié (dedicated systems) : conçus du point de vue matériel et logiciel pour une application spécifique pour éliminer au maximum toute incertitude et risque d'erreur fatale.

1.4.2 Test d'acceptabilité

Test d'acceptabilité = analyse de faisabilité, contrôle d'admission.

- temps réel strict :
  - ➤ le concepteur doit prouver que les limites temporelles ne seront jamais dépassées quelle que soit la situation.
  - ➥ comportement dans le pire des cas
  - ➤ difficile si le système n'est pas mono-processeur, mono-tâche, sans interruption
  - ➥ solution : garantir une limite max en se ménageant une marge
- temps réel souple :
  - ➤ on peut se contenter de mesures statistiques obtenues sur un prototype
  - ➥ comportement moyen (+ limite basse et haute)
  - ➥ écart type

Moyenne

$$Moyenne = \frac{x_1 + x_2 + ... + x_N}{N} = \frac{\sum_{i=1}^{n} x_i}{N} = \bar{x} \tag{1}$$

Variance

$$Variance = \frac{(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + ... + (x_N - \bar{x})^2}{N} = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{N} = V \tag{2}$$

Ecart type

$$Ecarttype = \sqrt{V} = \sigma(sigma) \tag{3}$$

1.4.3 Compromis et limites

Compromis hardware/software :

- Capacités, propriétés natives (rapidité), fiabilité du hardware
  - Mémoire, cpu, communications
- Poids des différents traitements pour mener à bien une tâche donnée
  - Algo, complexité
- Gestion du hardware pour permettre l'exécution des traitements
  - noyau temps réel
  - ordonnancement (organisation des tâches), gestion de la mémoire, préemption (une tâche en interrompt une autre), etc.

Limites des systèmes classiques :

- Ordonnancement : temps partagé
- Gestion des entrées sorties et des interruptions sous optimales
- Gestion de la mémoire virtuelle très souple : engendre des fluctuations des temps d'exécution des activités d'un système
- Résolution temporelle pas assez fine

1.5 Origines de la latence d'un système

Pour comprendre l'origine de la latence d'un système, il faut comprendre les mécanismes fondamentaux d'un OS.
Latence : délai entre un évènement et le début de la réaction du système

- Les délais de scrutation du système (ex : surveillance des entrées sorties)
- Les délais dus à l'OS (ordonnancement, interruptions)
- Les délais du calcul applicatif
- Les délais de transmission d'un message

2 Multitâche : organisation des tâches

2.1 Système d'exploitation multitâche

- Aspect multitâches organisé sur une structure monoprocesseur ou multiprocesseurs.
- But : utiliser au maximum le processeur. Pendant les accès aux périphériques (entrées/sorties) le processeur peut rester inutilisé.
- Nécessite la capacité d'organiser plusieurs tâches en leur attribuant tour à tour :
  - ➤ du temps processeur (éventuellement en concurrence : ordonnancement)
  - ➤ l'accès à la mémoire
  - ➤ l'accès aux périphériques (resources)
  - ➤ gérer la synchronisation, la communication entre les tâches (IPC Inter Processus Communication).
  - ➤ gérer le temps, l'horloge
- Deux implémentations possibles : coopératif / préemptif

2.1.1 Multitâche coopératif

- les tâches s'exécutent tour à tour
- file d'attente
- si une tâche se bloque, tout se bloque

2.1.2 Multitâche préemptif (ou avec réquisition)

- une tâche utilise le processeur pendant un temps donné
- un ordonnanceur (scheduler) attribue le processeur à une tâche en fonction de sa priorité
- temps partagé entre les tâches (time sharing) : simulation de fonctionnement quasi-parallèle
- temps partagé entre les utilisateurs : unix - systèmes multitâche multi-utilisateur

2.1.3 Multiprocessing

- multiprocessing asymétrique – ASMP :
  - ➤ un processeur pour le système, les autres pour les applications
- multiprocessing symétrique – SMP :
  - ➤ multiprocesseur symétrique (à mémoire partagée) = symmetric shared memory multiprocessor (SMP)
  - ➤ répartition de tous les processus sur les processeurs
  - ➤ le système dispose toujours d'un pourcentage du temps processeur

- Utilisation de plusieurs processeurs
  - ➤ permet un véritable traitement en parallèle
  - ➤ nécessite de gérer :
    - – communication ;
    - – synchronisation ;
    - – et partage des ressources.

2.2 Programme vs. tâche/processus

- **Programme :** Objet statique, suite d'instructions stockées en mémoire agissant sur un ensemble de données.
- **Tâches / processus (programme exécuté) :** Objet dynamique géré par l'OS, tâche ou process.

Les processus gérés par le système d'exploitation ont les caractéristiques suivantes :
- Espaces mémoires séparés
- Communiquent par des canaux spéciaux
- Création coûteuse en temps
- Associés à des mécanismes de protection très coûteux
  (interactions qui viendrait perturber le code/les données)
- Passage d'un processus à l'autre : coûteux transferts de mémoire

Un processus manipule différentes zones mémoire :
- Zone programme
- Zone de données
- Zone de pile (données temporaires)

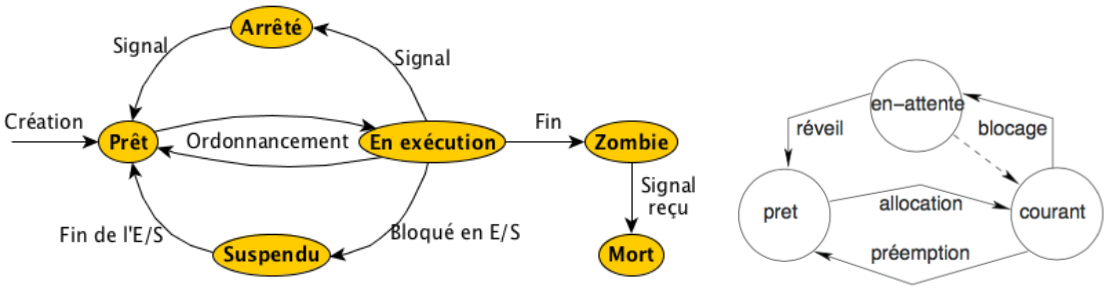
Le système utilise des registres particuliers pour identifier le processus courant :
- un compteur ordinal (program counter) : position dans le code du programme
- un pointeur de pile (stack pointer) : position dans la pile
- un registre d'état : droits, adresses, priorités
- un certain nombre de registres généraux : registres non spécialisés
  (entiers, flottants, adresses, indexes, etc...)

Ensemble des informations caractérisant l'exécution d'un processus : **contexte du processus**
- Processus lourds (**fork**) (contexte important) : nom, état des registres, état du processus + mécanismes de protection
- Processus légers (**thread**) (contexte partagé/propre) (lightweight process)
  - ➤ exécutés au sein du même contexte
  - ➤ contexte propre réduit aux registre : compteur ordinal, pointeur de pile
  - ➤ reste du contexte partagé : contexte global

Temps réel : trouver un équilibre entre processus légers et processus lourds.

2.3 Gestionnaire de tâches et ordonnancement

L'ordonnanceur est une procédure de service de l'OS chargée de choisir quelle tâche peut utiliser le processeur.
étapes de création d'une tâche : non-existante -> non-opérationnelle -> opérationnelle



Plusieurs états existent pour une tâche :
- Courant (en exécution : tâche élue, qui possède le processeur
- Prêt : tâche éligible, demande le processeur
- En attente (suspendu ou arrêté) : tâche bloquée en attente d'un événement

Transition d'un état à un autre :
- Réveil : en attente -> prêt
- Allocation : prêt -> courant
- Préemption : courant -> prêt
- Blocage : courant -> en attente

L'OS représente les tâches à travers des structures de données :
- descripteur de tâche (task control block)
  - ➤ état de la tâche
  - ➤ contexte d'exécution
  - ➤ chaînage entre tâches (permet de gérer les files "en attente" et "prêt")

Ordonnanceur (scheduler) :
- files de tâches en-attente / file de tâches prêtes.
- fait passer une tâche d'une file à l'autre en fonction de l'évolution
- applique une **politique d'ordonnancement** pour choisir la tâche courante parmi les tâches prêtes

2.4 Les politiques d'ordonnancement

Ordonnanceur en ligne : agit en direct pendant le fonctionnement du système
Ordonnanceur hors ligne : choisit à l'avance l'ordre des tâches.

Commutation de contexte (context switch) : s'effectue chaque fois que le système change de tâche courante.

Ordonnanceur (scheduler) :
- **préemptif ou avec réquisition** : une tâche peut être interrompue pour traiter une autre plus urgente, désallouée par décision de l'ordonnanceur suite à une interruption par exemple.
- **non préemptif ou sans réquisition** : chaque tâche doit s'exécuter complètement avant de passer à la suivante
  - plus simple pour prévoir les interactions entre tâches
  - pas réactif, pas interactif

2.4.1 Ordonnancement statique

Statiques pilotés par table :

- Hors ligne
- L'application va être découpée en séquences élémentaires qui ne seront jamais interrompues
- Une séquence est une procédure définie par l'utilisateur, c'est l'unité de base
- Un processus est une suite ordonnée de séquences
- L'ordonnancement des processus est régit par un calendrier (référence de temps interne)
- C'est une table spécifiant le liste des processus à activer
- Application systématique par ordonnancement cyclique, ex : ABCABCABC
- Les processus sont indépendants les uns des autres
- L'ordonnancement des processus est régi par une horloge
- Il n'existe pas de préemption
- Le découpage en séquences non interruptibles assure la protection des données partagées entre les processus
- Il est aisé de certifier le comportement de l'application

Statiques préemptifs basés sur les priorités :

- Hors ligne
- Permet d'assigner les priorités aux tâches
- Une fois en cours d'exécution, le système utilise un ordonnancement préemptif basée sur la notion de priorité fixée apriori.
- Algorithme le plus connu : Rate-Monotonic Scheduling (RMS), la priorité la plus forte est donné à la tâche qui possède la plus petite période.

2.4.2 FIFO

FIFO (first-in first-out) : premier arrivé, premier servi

- la tâche réveillée ou préemptée est placée à la fin de la file des tâches prêtes
- la tâche courante qui se bloque en attente est placée à la fin de la file
- la tâche en tête de file devient la tâche courante

2.4.3 Round Robin

Round Robin / tourniquet : à tour de rôle (quantum de temps)

- les tâches sont placées dans une file FIFO et sont activées périodiquement
- la tâche courante est exécutée pendant un quantum de temps T puis placée en attente à la fin de la file
- la tâche en tête de file devient la tâche courante, exécutée pendant un temps T, etc...
- égalité de traitement pour toutes les tâches (pas de notion d'urgence) : utilisé pour gérer le temps de traitement accordé à plusieurs utilisateurs.

exemple : trois tâches T1, T2 et T3 de durées 6, 3 et 4. Les tâches démarrent à t0, ordre d'arrivée T1, T2 et T3. Le quantum de temps utilisé par l'ordonnanceur est de 1 :

t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
T_1	T_2	T_3	T_1	T_2	T_3	T_1	T_2	T_3	T_1	T_3	T_1	T_1

2.4.4 Ordonnancement par priorité

- préemptif : si tâche plus prioritaire devient prête, elle remplace la tâche courante
- non-préemptif : la tâche la plus prioritaire devient la nouvelle tâche courante quand la tâche courante s'achève ou se met en attente

- Temps réel : choix de la priorité en fonction de critères d'analyse
  - ➤ HPF : Highest Priority First
  - ➤ EDF : Earliest Deadline First : échéance la plus proche
  - ➤ LLF : Least Laxity First = EDF + notion de durée de travail
- Prise de décision quand le système est actif
- Les critères peuvent varier durant l'exécution

exemple : trois tâches T1, T2 et T3 de durées 6, 3 et 4 et de priorités 1, 2 (plus prioritaire) et 1. Les tâches démarrent à t0, ordre d'arrivée T1, T2 et T3 :

t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
T_2^2	T_2^2	T_2^2	T_1^1	T_1^1	T_1^1	T_1^1	T_1^1	T_1^1	T_3^1	T_3^1	T_3^1	T_3^1

2.4.5 Ordonnancement mixte : priorité avec tourniquet

Unix (systèmes classiques) : priorités et partage du temps
- Traitement des tâches de priorité les plus hautes
- Tourniquet parmi les tâches de même priorité
- Traitement d'interruptions au détriment de tâches de fortes priorités

exemple : trois tâches T1, T2 et T3 de durées 6, 3 et 4 et de priorités 1, 2 (plus prioritaire) et 1. Les tâches démarrent à t0, ordre d'arrivée T1, T2 et T3. Le quantum de temps utilisé par l'ordonnanceur est de 1 :

t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
T_2^2	T_2^2	T_2^2	T_1^1	T_3^1	T_1^1	T_3^1	T_1^1	T_3^1	T_1^1	T_3^1	T_1^1	T_1^1

Lois d'ordonnancement (systèmes Temps Réel) :
- Limiter le partage du temps (changement de contexte couteux)
- Limiter les préemptions (très coûteux)
- Utiliser les priorités
- Ordonner les priorités à priori lors des tests de faisabilité, en fonction des situations/charges du système possibles
- Grande importance de l'ordonnancement/traitement des préemptions

3 Interaction avec l'environnement et interruptions

Interruption : un signal qui demande l'interruption d'une tâche
- en provenance de l'utilisateur : Ctrl+C,
- en provenance d'un périphérique/capteur (signal que la lecture de données est achevée, signal de l'horloge)

Processus "préempté" : processus qui est interrompu par une interruption.

3.1 Scrutation et interruption

Exemple :
- un processeur exécutant un programme unique.
- une mémoire dans laquelle sont rangées les instructions du programme et ses données.
- une unité de gestion des entrées/sorties
- les trois reliés par un bus : canal de communication pour échanger les données.

Modèle simplifié de traitement :
- programme alternant trois phases :
  - ➤ acquisition de données
  - ➤ traitement et calcul
  - ➤ restitution du résultat
- communication avec l'unité d'entrées/sorties durant les phases 1 et 3,
- calcul pendant le reste : il sait exactement quand il doit interagir avec l'environnement.

Interaction avec l'environnement :
- le programme doit pouvoir interagir avec l'environnement (tenir compte d'une saisie clavier par exemple) alors qu'il est en train de faire autre chose : un calcul par exemple
- Plusieurs manières de gérer cette interaction :
  - ➤ scrutation cyclique (polling)
  - ➤ interaction par interruptions

3.1.1 Interaction par scrutation cyclique

Principe : on interroge régulièrement les périphériques (boucle infinie)

```
do {
    while (données non disponibles)
        vérifier entrée sur capteurs & lire capteurs
    traiter les données
    démarrer les réactions
    while (actions en cours)
        vérifier les actionneurs
} while (non arrêt du système)
```

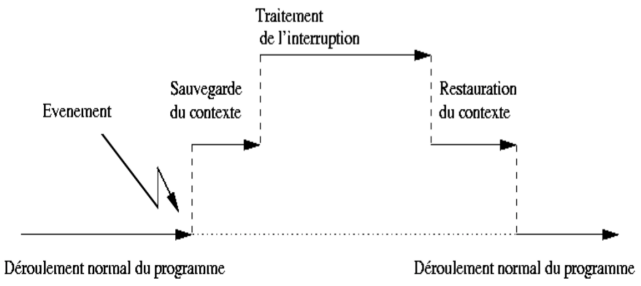
Avantages :
- Latence du système = temps de traitement d'une boucle
- Cas proche du système simple séquentiel vu précédemment
- Simple à programmer

Inconvénients :
- « Alignement » de la boucle de traitements sur le périphérique le plus lent
  - ➤ bon si peu périphériques et identiques en temps de réaction
- Programme difficile à maintenir
  - ➤ si beaucoup de périphériques, de vitesse différente.
  - ➤ Ajout d'un périphérique : remise en cause de l'existant
  - ➤ re-tester chaque élément pour vérifier les contraintes de temps

3.1.2 Interaction par interruptions

Principe : Un signal hardware associé à l'arrivée d'un événement déclenche l'activation de la fonction qui va lire la donnée d'entrée. Le cours normal de l'exécution du programme est interrompu.
- Initialement introduit pour gérer les E/S d'un processeur
- Concept étendu à la notion d'interruption interne ou exception (trap) logicielle
- interruptions désignées selon leurs sources matérielles ou logicielles :
  - ➤ interruption d'entrée/sortie ;
  - ➤ interruption d'horloge ;
  - ➤ interruption mémoire.
- sauvegarde du contexte d'exécution (compteur ordinal – program counter, état de la mémoire, registres)
- appel du gestionnaire d'interruption (interrupt handler) : traitement a exécuter en urgence

signal d'interruption :



Scrutation des interruptions :

```
do {
    while (état == données disponibles)
        lire données mémorisées
    faire un traitement
} while (non arrêt du système)
```

gestionnaire d'interruption (handler) :

```
Lire et mémoriser la donnée
positionner (état = données disponibles)
acquitter l'interruption
```

"acquitter l'interruption" : valider, le travail est fait, on peut réveiller la tâche qui attendait la donnée.

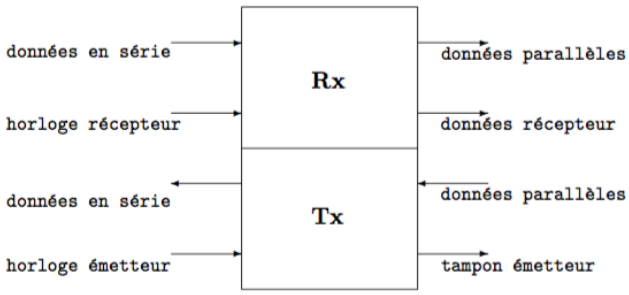
signal → gestionnaire d'interruption (tout petit bout de code) → traitement plus lourds effectué par un programme.

Avantages :
- Le traitement d'un événement externe va être partagé entre un traitement immédiat effectué par le gestionnaire d'interruption et par une gestion différée, confié à une tâche : traitement multitâche possible.

Inconvénients :
- Une grande partie du temps CPU est consacré à autre chose que l'application
- Programme difficile à maintenir

- nested interrupts = interruption qui en interrompt une autre, interruptions opérant à des niveaux de priorités différents
- queued interrupts = même niveau de priorité, interruptions mises en file

3.1.3 Exemple : UART



Considérons un programme émettant des caractères sur une liaison série via un UART (Universal Asysnchronous Receiver Transmitter).
- UART : composant électronique qui gère une ligne série.
- deux séries de registres pour transférer et recevoir des caractères
- interruptions différentes pour l'émission et l'arrivée d'un caractère
- un caractère émis toutes les milli-secondes environ

Le programme ne peut émettre le caractère suivant que quand l'UART est prêt.
- beaucoup de temps à tester s'il est autorisé à émettre le caractère suivant
- temps utilisable pour exécuter une autre tâche

Étapes du fonctionnement :
- Le programme émet un caractère (demande l'accès à l'UART) et se met en attente avec un appel à une primitive système wait().
- Lorsque l'UART est prêt à émettre un caractère, il génère une interruption TxReady
- Le gestionnaire d'interruption correspondant effectue l'opération signal()
- Le programme suspendu peut reprendre le processeur et émet le caractère suivant.

3.2 Gestion des interruptions et multitâche

Traitement des interruptions : influence sur le temps de réponse du système
- étapes qui conduisent à la prise en compte effective des interruptions
- méthodes pour en faire un bilan temporel

3.2.1 Étapes de prise en compte des interruptions

Aiguillage des interruptions (Interrupt dispatch)
- sauvegarder le contexte de la tâche courante (ensemble des registres)
- détermine ensuite la source de l'interruption
- invoquer le gestionnaire d'interruption correspondant

➥ réalisé entièrement par le matériel

un seul point d'entrée pour l'arrivée des interruptions :
- un logiciel identifie le matériel responsable de l'interruption
- le matériel associe une priorité à son interruption

On peut contrôler les interruptions avec des instructions spécifiques du processeur :
- soit interdire l'ensemble des interruptions d'un système
- soit les interdire de façon individuelle en agissant par masque sur le mot de contrôle des différentes sources d'interruption. L'interruption est dite inhibée.
- Lorsque plusieurs interruptions arrivent en même temps, la plus prioritaire est traitée immédiatement.

Relation entre gestionnaire d'interruption et tâche
- traitement court : tout est fait par le gestionnaire d'interruption

➥ système temps réel : peut traiter les interruptions de manière transparente
- si le gestionnaire d'interruption appelle une procédure dans le système d'exploitation

➥ permet de tracer les opérations : test / validation
- traitement lourd/complexe :
  - ➤ la tâche chargée du traitement se met en attente
  - ➤ quand l'interruption arrive, elle réveille la tâche qui effectue le traitement de l'interruption
  - ➤ pendant le traitement, l'interruption est masquée pour ne pas interrompre le traitement de l'interruption par la même interruption
  - ➤ à la sortie du gestionnaire d'interruption, si d'autres interruptions sont en attente, les autres gestionnaires d'interruption sont exécutés avant d'entrer dans le code de la tâche associée au premier gestionnaire.

Activation de la tâche associée à une interruption (gestionnaire d'interruption)
- Si une tâche exécute une primitive du système d'exploitation qui inhibe la préemption, à la sortie du gestionnaire d'interruption, il faut attendre la fin de cette primitive.
- L'ordonnanceur est ensuite appelé :
  - ➤ il sélectionne la tâche suivante selon la politique
  - ➤ changement de contexte si la tâche sélectionnée est différente de la tâche courante.

3.2.2 Coût temporel du traitement des interruptions

Temps de commutation, temps de sélection et temps de préemption :
- temps de sélection : temps moyen pris par le système pour choisir la prochaine tâche courante
- temps de commutation : temps moyen pris par le système pour commuter entre deux tâches
- temps de préemption = temps de sélection + temps de commutation

Temps de latence des interruptions :
- intervalle de temps qui se situe entre le moment où le processeur reçoit l'interruption et le moment où la première instruction du gestionnaire d'interruption est exécutée (dépend de l'architecture et du firmware)

Temps de latence d'entrée dans la tâche (process dispatch latency time) : cas ou une seule interruption est traitée
- délai dû au matériel : terminer l'instruction courante et se brancher sur le gestionnaire d'interruption
- délai dû au gestionnaire d'interruption : sauvegarder le contexte, exécuter l'interruption, recharger le contexte
- délai dû au système : déterminer si il faut recharger le contexte précédent et le faire si il y a lieu

task response delay : temps de réponse qui prend en compte la situation où plusieurs interruptions arrivent en même temps.

3.2.3 Les mécanismes du traitement d'interruption pour le temps réel

systèmes d'exploitation classiques : pas de garantie sur le temps de latence des interruptions
Accéder à un niveau aussi proche du matériel :

- réservé à des programmes spéciaux (gestionnaires de périphérique, device driver)
- exécutent leurs instructions dans un mode de protection réservé à l'OS lui-même (en mode noyau).

systèmes d'exploitation temps réel :

- niveaux de priorité suffisants pour répartir les degrés d'urgence des interruptions
- strict minimum dans le gestionnaire d'interruption (les traitements qui ne peuvent pas attendre)

➥ peut être évalué, paramètre fixe du système

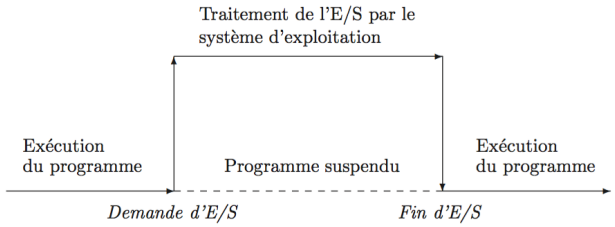
- déporter le traitement effectif dans une tâche :

➥ meilleur contrôle du comportement temporel des tâches

3.3 Gestion des entrées/sorties

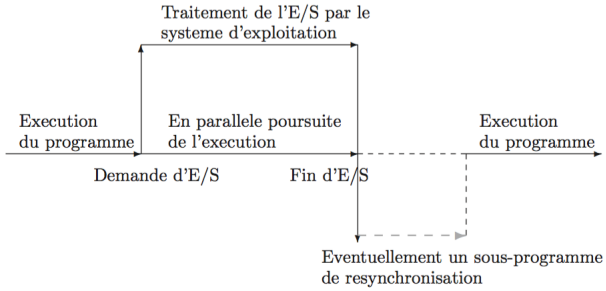
différence de vitesse entre E/S et temps de calcul : problème de première importance pour les OS.

- couche spécialisée pour le traitement des E/S
- point de passage obligatoire pour les programmes qui veulent interagir avec leur environnement
- l'OS gère lui même les E/S pour éviter qu'un programme monopolise le système



E/S synchrone :

- le programme est suspendu le temps que l'E/S soit géré par le système
- une autre tâche peut être exécutée pendant ce temps



E/S asynchrone :

- Le traitement d'E/S s'effectue en parallèle à la poursuite de l'exécution du processus
- resynchronisation à la fin de l'E/S :
  - ➤ mise à jour d'une variable qui peut être testée par le processus appelant s'il veut savoir si l'E/S est terminée ou pas
  - ➤ exécution d'un sous-programme spécifié par le programmeur (équivalent à un sous-programme de traitement d'interruption) qui se charge de la resynchronisation

E/S ont une incidence importante sur l'exécution des processus.

- difficile, voire impossible, de borner le temps de traitement de l'E/S et les temps de blocage et déblocage ou de resynchronisation des processus associés
- difficile de gérer des priorités entre E/S.

➥ maîtrise des temps d'exécution particulièrement difficiles dans le contexte du temps réel

3.4 Gestion de la mémoire

une des missions d'un système multitâche :
- mettre à disposition des processus les ressources mémoire nécessaires
- famille Unix : mécanisme de mémoire virtuelle paginée

Mémoire virtuelle :
- un espace mémoire associé à chaque tâche
- un espace physique pour stocker les informations qui peut être plus petit
  - ➤ mémoire physique contient les données en cours d'utilisation
  - ➤ le reste est placé sur le disque
  - ➤ swapping : système de va-et-vient, chargement de parties de l'espace mémoire de la tâche au grée de son exécution

➥ permet d'exécuter des programmes qui demandent un espace mémoire plus grand que l'espace mémoire physique.

➥ permet de partager l'espace mémoire physique entre plusieurs tâches

pagination :
- mémoire virtuelle découpée en pages
- mémoire physique conçue pour stocker un certain nombre de pages
- présence d'une table indiquant les pages chargées en mémoire
- plusieurs niveaux de mémoire possible avec un accès de plus en plus rapide
- si une tâche demande un accès à la mémoire
  - ➤ l'OS vérifie si la page virtuelle est dans la mémoire physique
  - ➤ sinon défaut de page (page fault)
    - ∗ regarder s'il a une page physique libre en mémoire physique
    - ∗ sinon il faut choisir une page à décharger :
      - politique de remplacement (la plus ancienne, la moins utilisée)
      - si la page a été modifiée depuis son chargement, il faut l'écrire sur le disque
    - ∗ lancer la lecture de la page dans laquelle se trouve l'adresse mémoire virtuelle à laquelle la tâche veut accéder
    - ∗ la tâche est bloquée en attente de la fin d'E/S correspondante
    - ∗ réveiller la tâche (basculer le contexte avec une autre tâche exécutée en attendant)
  - ➤ MMU – Memory Management Unit : fait la traduction entre les adresses en mémoire virtuelle et les adresses physiques

problèmes pour le temps réel : défaut de page
- opération lourde à gérer
- blocage de la tâche
- modifie le schéma d'ordonnancement de façon non prévisible
- inacceptable en temps réel : on ne maitrise pas l'échéance d'une application

solution des systèmes temps réel :
- "pré-réservation de ressources"
- verrouiller des pages en mémoire pour empêcher qu'elles ne soient déchargées

➥ gestion plus compliquée de la mémoire.
- MMU parfois utilisée pour gérer des espaces mémoires séparés pour assurer l'indépendance entre des parties distinctes d'un système
- plus de risque de débordement mémoire et d'accès non désiré entre les tâches
- inconvénient : temps de sauvegarde des registres du MMU à chaque changement de contexte

3.5 Horloge et gestion du temps

horloge : génère des interruptions en fonction des oscillation d'un cristal de quartz
gestionnaire d'interruption associé qui peut rendre les services suivants :
- incrémenter un compteur qui retiendra le temps écoulé
- envoyer un signal à une tâche spéciale, la tâche horloge
- servir de diviseur de fréquence :
  ex : à partir de l'interruption matérielle ne signaler un tick que toutes les N fois.

la tâche horloge : gère les réveils, les délais (shell : date, time, at, crontab, ...)
temps utile pour :

- donner une échéance : éviter qu'une tâche reste bloquée ou ne prenne trop de temps
- déclencher une tâche périodiquement

problème pour le temps réel : définition trop faible

- temps élastique
- mesure pas assez précise (ex : fonction sleep(), précision à la seconde)
- système temps réel : offre des services de gestion du temps gérés directement par le matériel