UNIVERSITE DE DOUALA

Ecole Nationale Supérieure polytechnique de Douala

COURS DE SYSTEMES TEMPS REEL

CHAPITRE 2 : Gestion des tâches dans un STR

1- Définition :

Pour un système, une tâche est une opération faisant intervenir un nombre réduit des éléments de ce système.

2- Taxonomie

- Tâche immédiate : générées par une interruption sur le μp (en général par une E/S), elles doivent être traitées rapidement et sont toujours plus prioritaires que les tâches différées. Elles sont traitées par le système d'interruption ;
  Exemple : suite à une INT horloge, le SE incrémente son compteur de temps horloge et décrémente le temps restant à attendre pour des tâches mises en pause pour une durée fixe ;
- Tâche différée : générées par une application logicielle. Elles peuvent faire l'objet d'une programmation d'ordonnancement spécifique ;
- Tâche indépendante : l'exécution d'une telle tâche ne requiert pas l'intervention ou l'autorisation d'une autre tâche ;
- Tâche interdépendante : une telle tâche a son exécution liée d'une manière ou d'une autre à une tâche tierce (autorisation à recevoir ou à envoyer, service en attente ou à rendre, etc.) ;

3- Segmentation

Un processus se compose de plusieurs zones mémoires ou segments.

- Segment de code (code segment) : contient les instructions codées par le programmeur (langage machine) ;
- Segment de données (data segment) : contient les données modifiées, stockées et générées par le programme ;
- Segment de pile (stack segment) : contient des données temporaires (accessibles plus facilement) ;

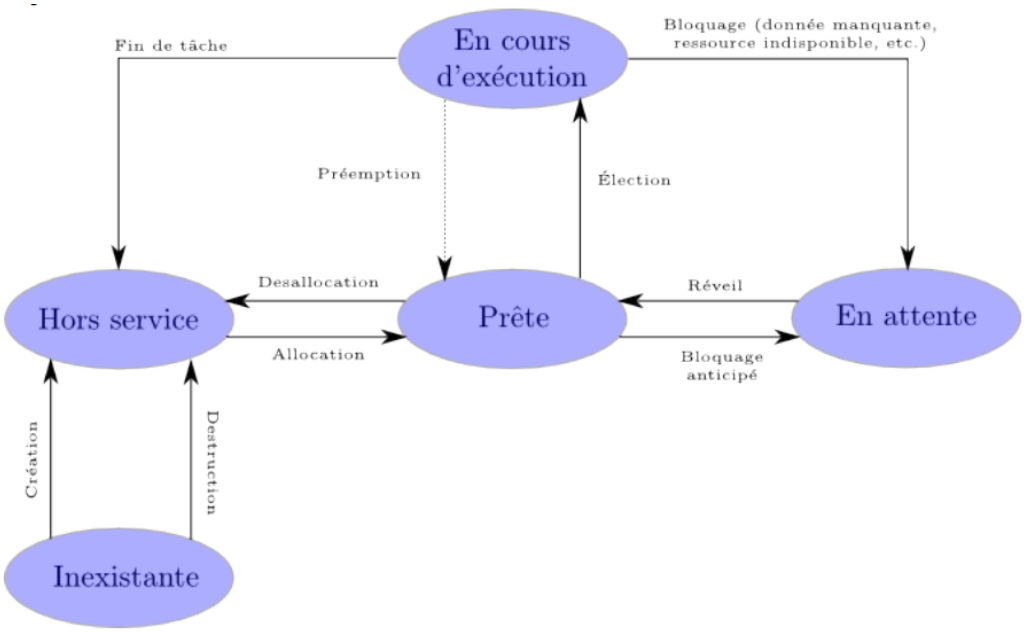
- Le tas (heap) : allocation dynamique à la suite du data segment (Non-systématique).

4- Contexte des tâches

Le contexte d'une tâche se définit comme l'ensemble des éléments permettant la poursuite de l'exécution `a l'instant où une tâche est interrompue. Ce sont :

- 3 pointeurs vers les segments ;
- Nom (id) ;
- Droits d'accès (ressources matérielles et logicielles accessibles) ;
- Identificateur de l'état courant ;
- Identité de l'utilisateur (admin, user, SE) auquel est rattachée la tâche.

5- Cycle de vie d'une tâche



6- Les états

Pour une tâche donnée, nous distinguons les états suivants :

- En cours d'exécution (courante) : une seule tâche peut être dans cet état. C'est elle qui accède à l'UC, "elle a la main" ;
- Prête : elle demande à être exécutée ;
- En attente : elle attend un événement, une ressource non-partageable ou un délai ;
- Hors service : elle existe et est présente en mémoire mais ne demande pas encore l'UC ;
- Inexistante : non créée, non initialisée ou morte.

7- Les transitions

Le passage d'une tâche à une autre est appelé 'transition'. On en distingue :

- exec -> HS : la tâche vient de se terminer ;

- exec -> prête : la tâche est interrompue par le SE (préemption) ;
- exec -> attente : la tâche demande au SE d'être mise en attente, car elle a besoin de qqch qui n'est pas encore là ;
- Prête -> HS : à la demande d'une autre tâche qui supervise ;
- HS -> prête : à la demande d'une autre tâche qui supervise ;
- Prête ->attente : à la demande d'une autre tâche qui supervise ;
- En attente -> prête (réveil) : délai de mise en attente terminée OU événement survenu OU ressource libérée OU sur demande d'une autre tâche qui supervise ;
- HS -> inexistant : libération mémoire ;
- Inexistant -> HS : allocation mémoire.

8- Qualificatif des tâches

Tâches et fréquences d'activation :

- Tâche périodique : elle est activée à intervalles de temps réguliers et connus du programmeur. Son exécution doit être achevée avant la fin de la période.
- Tâche non-périodique :
  - Tâche sporadique : un temps minimal a entre deux activations est défini et connu du programmeur. L'exécution doit être terminée avant a (contrainte forte),
  - Tâche apériodique : la tâche apparait de façon non-prévisible, on se contente de contraintes souples sur le temps de réponse moyen.

9- Descripteurs (éventuels) de tâches

Possibles attributs quantificatifs d'une tâche P_i :

- La période T_i (si tâche périodique),
- Le temps de traitement maximal ou capacité C_i,
- La date d'activation ou de réveil r_i,
- Le délai critique D_i (délai maximum admissible pour l'exécution de P_i),
- L'échéance $R_i = r_i + D_i$;
- La laxité $L_i = D_i - C_i$.
- Pour une tâche périodique la k^{eme} date d'activation r_i^k vaut $r_i + k * T_i$.
- Quand $D_i = T_i$, on parle de tâche à échéance sur requête. Il va de soi que la logique veut que : *pour tout i*, $0 < C_i < D_i < T_i$. Connaître ces paramètres est quasiment indispensable pour faire la programmation temps réel.
- Une tâche est dite faisable si toutes ses instances peuvent se terminer avant leurs échéances.

10- Conséquences sur l'UC

- Facteur d'occupation de l'UC par la tâche *Pi* : $u_i = T_i/C_i$,
- Facteur de charge de l'UC par la tâche *Pi* : $ch_i = D_i/C_i$,
- On définit la charge globale comme : $U = \sum_i u_i = \sum_i P_i\ T_i/C_i$.

11- Interruption

Il existe deux manières principales d'interagir avec l'environnement :

- Par scrutation cyclique des périphériques (les périphériques sont interrogés à intervalles de temps réguliers sur la présence d'un événement à traiter),
- Par système d'interruptions.

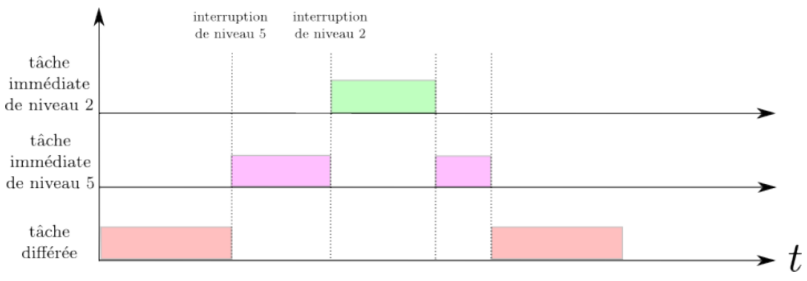
On ne peut espérer concevoir un système temps réel efficace sans utiliser d'interruptions. Elles permettent de réagir aux événements et donc d'être en phase avec l'environnement du système.

Il existe deux grandes catégories d'interruptions :

- Matérielle : signal physique câblé sur le processeur. Un changement d'état du signal provoque une rupture de séquence sur le μp. Elle génère une tâche immédiate.
- Logicielle (trap, exception) : elle est générée par une application ou par le système.

Les interruptions sont également codifiées en fonction de l'événement auquel elles correspondent (E/S, horloge, défaut de page de mémoire).

- Le système d'interruption (ou gestionnaire d'interruption/*Programmable Interrupt Controler*) est un dispositif physique qui se charge de traiter les signaux physiques d'interruption. Il travaille en collaboration avec le μp pour commuter les tâches et avec le SE pour appeler une routine de traitement de l'interruption.
- Comme pour les tâches, il faut également gérer plusieurs niveaux de prioriété pour les interruptions.
- Cette gestion s'opère à l'aide de masques applicables sur le code d'une interruption.

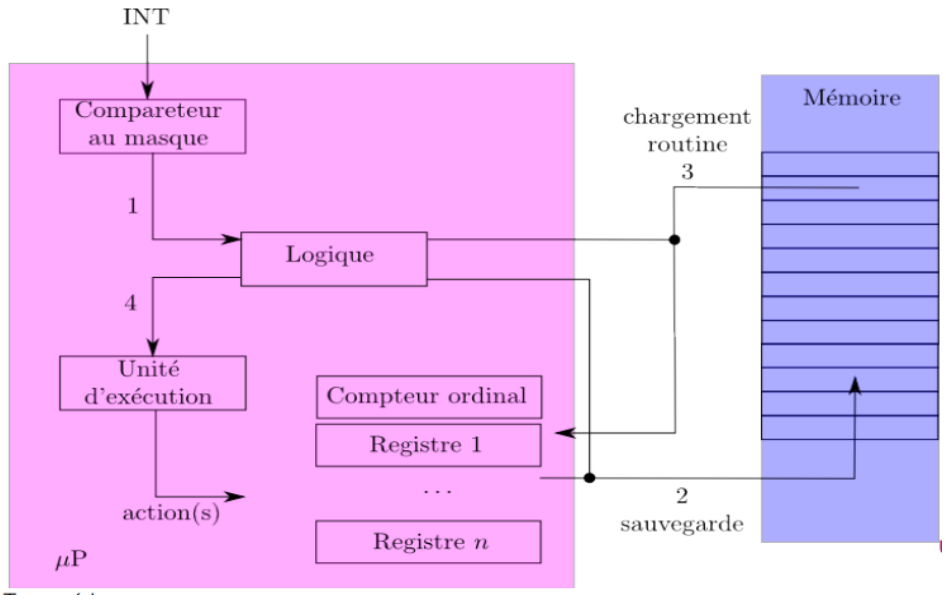


Attention à ne pas mélanger :

- Une interruption matérielle engendre la création d'une tâche immédiate (et donc prioritaire par rapport à toutes celles dans les files d'attente de l'ordonnanceur). Cette tâche correspond souvent à une routine (petit programme en dur).
- Une interruption logicielle provoque une commutation de tâches différées.
- Une tâche différée peut demander à générer une interruption logicielle si elle supervise la tâche courante.

Séquence d'interruption :

- Si non masquée, l'interruption arrive au μp. Une interruption ne peut être prise en compte qu'à la fin de l'instruction en cours (atomicité),
- Le contexte de la tâche courante est sauvegardé,
- La routine associée à l'interruption est lancée,
- Un acquittement est envoyé pour signifier la fin du traitement de l'interruption,
- Si une autre interruption de priorité inférieure est en attente, elle est traitée, sinon on appelle l'ordonnanceur pour traiter la prochaine tâche différée et charger le nouveau contexte.



Pour gérer les interruptions, il faut également être capable :

- De stocker les interruptions en file d'attente,
- De déterminer l'origine d'une interruption.

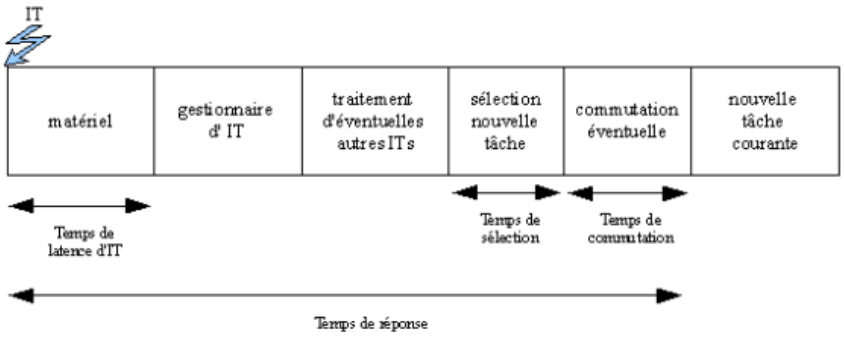
L'identification de la source de l'interruption peut être :

- Directe : un niveau de priorité est directement associé à un périphérique (très rapide),

- Par scrutation (polling), une requête est envoyée à tous les périphériques générant l'interruption reçue. Le "coupable" se fait alors connaitre. (Un même type d'interruption partagée par plusieurs périphériques),
- Par vectorisation : le périphérique envoie sur le bus de données un numéro de vecteur, traitée comme une adresse vers la routine de la part de l'UC. (Besoin d'une table prédéfinie en mémoire vive chargée au moment du démarrage de du SE),

D'un point de vue temps réel il faut contrôler :

- Le temps de commutation (temps moyen pris par le système pour commuter entre deux tâches),
- Le temps de sélection (temps moyen pris par le système pour déterminer l'identité de la prochaine tâche),
- Le temps de latence des interruptions (temps entre la réception de l'interruption et le déclenchement de la routine correspondante),
- Le temps de latence d'entrée dans la tâche (somme de toutes les phases du traitement d'une interruption),
- Le temps de réponse (temps de latence d'entrée dans la tâche mais avec éventuellement plusieurs interruptions au milieu).



Toujours d'un point de vue temps réel il est à noter que :

- On a besoin des interruptions,
- Elles causent un aléa dans le temps de traitement d'une t^ache,
- Il est préférable de remonter un maximum d'interruption au niveau logiciel pour garder un contrôle dessus (et limiter l'aléa).

12- Le partage de ressources critiques

- Il existe certaines ressources matérielles (imprimantes) ou logicielles (variables globales) qui ne doivent être allouées que à une seule tâche à la fois
- Solutions disponibles pour régler cette question :
  - Masquer toutes les interruptions (lourd et possiblement gênant)
  - Mettre un verrou : variable booléenne V (1 accès verrouillé, 0 accès autorisé) et une file d'attente $F(V)$ pour les tâches qui demandent le verrou quand celui-ci est déjà posé (pas d'attente active)
  - Utiliser un sémaphore (généralisation du verrou à plusieurs points d'entrée dans la section)

Le sémaphore :

- Variable entière S et file d'attente $F(S)$
- Deux primitives : attendre/demander $P(S)$ et signaler/libérer $V(S)$

$P(S)$:
$S \leftarrow S - 1$
si $S \leq 0$ alors suspendre tâche appelante et l'insérer dans $F(S)$.

$V(S)$:
$S \leftarrow S + 1$
si $S \leq 0$ alors réveiller la tâche en tête de $F(S)$

Le sémaphore :

- Si $S > 0$, S est le nombre de droit d'entrée en section critique (en général S vaut 1 au maximum mais cela peut être plus),
- Si $S \leq 0$, S est le nombre de tâches en attente de la section critique.

Interblocage (deadlock) :

tâche 1		tâche 2
$P(S_1)$	1	
	2	$P(S_2)$
utiliser ressource 1	3	
	4	utiliser ressource 2
$P(S_2)$	5	
	6	$P(S_1)$
bloquée		**bloquée**

13- Synchronisation des tâches

Certaines tâches sont interdépendantes, il faut donc pouvoir s'assurer qu'une tâche A est à un tel point d'avancement avant qu'une tâche B exécute telle instruction ou qu'un événement ait eu lieu.

plusieurs types :

- Directe : les tâches en relation connaissent leurs "identités" respectives,
- Indirecte : pas besoin d'identification.

Méthode directe : tâche B supervise tâche A.

- deux variables booléennes : $ETAT$ (0 = bloquée, 1 = éveillée), $FANION$ (1 = signal d'éveil envoyé)
- deux primitives : $BLOQUER()$ et $EVEILLER(A)$

$BLOQUER()$ (appelée par A) :

Si $FANION = 1$ alors $FANION \leftarrow 0$
Sinon appel système pour passer en attente, $ETAT \leftarrow 0$

$EVEILLER(A)$ (appelée par B) :

Si $ETAT = 0$ alors appel système pour réveiller A, $ETAT \leftarrow 1$
Sinon $FANION \leftarrow 1$

Méthode par sémaphore (indirecte) : Sémaphore *S* initialisée à 0, le code de la tâche A contient un *P* (*S*) et celui de la tâche B un *V* (*S*) => A est bloquée tant que B n'a pas fait toutes les instructions précédant le *V* (*S*).

Exercice :

1) faire en sorte qu'une tâche A s'exécute si les tâches B OU C ont fini une certaine instruction.

2) faire en sorte qu'une tâche A s'exécute si les tâches B ET C ont fini une certaine instruction.

Synchronisation par événements :

- Exemple classique : une tâche attend un traitement effectué par un périphérique. Une fois terminé, le périphérique lève une interruption, la routine signale l'évènement, la tâche en attente peut continuer.
- L'événement peut être global (concerne toutes les tâches, ou un groupe de tâche) ou local (une tâche en particulier)
- L'événement peut être identifié par un mot de code (id)

- Une variable booléenne Ev (0 = non arrivé, 1 = arrivé)
- Une file d'attente F (Ev)
- Trois primitives SET(), RESET() et WAIT()

SET () (appelé par routine ou tâche) :
  $Ev \leftarrow 1$
  répéter
    si $F(Ev) \neq \emptyset$ **alors**
      appel système pour débloquer le processus en attente
    fin si
  jusqu'à $F(Ev) = \emptyset$

RESET () (appelé par une tâche satisfaite) :

  $Ev \leftarrow 0$
  vider $F(Ev)$

WAIT () (appelé par la ou les tâche(s) ayant besoin du signal) :

si Ev = 0 **alors**

insérer la tâche dans F (Ev)

fin si

Remarques : Attention à ne pas effacer un signal avant de l'avoir pris en compte.

Le rendez-vous à *N* points d'entrées :

- *N* tâches (au max) peuvent demander un rdv,
- Besoin d'un compteur de tâches en attente CPT,
- *B*esoin d'un système de masquage pour rendre le code gérant le rdv ininterruptible,
- Besoin de signaux pour gérer la file d'attente des tâches.

Le rendez-vous à N points d'entrées (code générique) :

RDV _init () :

mutex ←1

CPT ←0 (nbr de tâches arrivées)

Le rendez-vous à *N* points d'entrées (code générique) :

```
RDV_proc() :
  P(mutex)
  CPT ← CPT + 1
  si CPT < N alors
    V(mutex)
    wait()
  sinon
    service éventuel
    set(all)(réveil des tâches par diffusion)
    CPT ← 0
    V(mutex)
  fin si
```

Le rendez-vous à *N* points d'entrées, remarques :

- Plusieurs exemples sur le web notamment en langage Java,
- Pas forcément besoin de connaitre l'identité des tâches entrantes.

14- Communication entre tâches

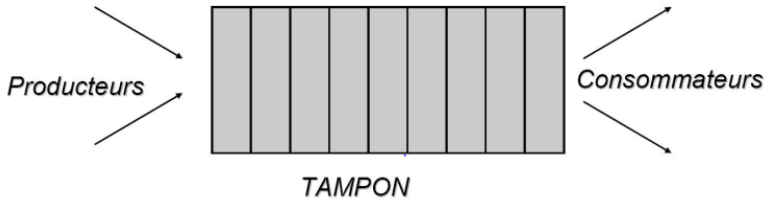
Communication = synchronisation + transmission de données.

Techniques les plus répandues :

- Une variable globale (avec éventuellement un *timestamp*),
- Le schéma producteur-consommateur,
- La boite aux lettres,
- Le tube.

Le schéma producteur-consommateur :

- Un tampon est alloué et sert à déposer des messages fournis par des producteurs.
- Les consommateurs accèdent en lecture au tampon et libèrent un emplacement par lecture.
- Le tampon a une taille limitée de *N* messages.
- Dans un tube, pas de limite sur *N*.



La boite aux lettres : idem mais tous les messages sont au même format.

- 1 sémaphore *place* correspond à des jetons de places disponibles dans le tampon (init à *N*),
- 1 sémaphore *mail* correspond à des jetons de messages disponibles dans le tampon (init à 0),
- Deux primitives : *ENVOYER*() et *RECEVOIR*(),
- Le dépôt de message doit être non interruptible,
- La primitive *RECEVOIR*() est bloquante.

Algorithme

$ENVOYER()$:

$P\ (place)$

déposer(message,destinataire)

$V\ (mail)$

$RECEVOIR()$:

$P\ (mail)$

retirer(message)

$V\ (place)$

Exercice : approvisionnement automatisé d'un site industriel.