

# Fonctionnement des ordinateurs. Aspects matériels et logiciels.<sup>1</sup>

© 2002, par Sangwo Yves

---

<sup>1</sup>Ceci est une version  $\beta$ , fctord02 $\beta$ .pdf

## Introduction

Ce document ne peut en aucun cas être considéré comme étant un syllabus, cours ou même un simple résumé du cours INFO 018 “*Fonctionnement des ordinateurs. Aspects matériels et logiciels*” dispensé par M. Raymond Devillers en première candidature d’informatique (ULB).

Le but de ce document est d’aider les étudiants concernés par ce cours à se préparer pour leur examen oral. Dès lors, il est fortement recommandé de n’avoir recours à celui-ci qu’après avoir vu et suffisamment compris la matière du cours.

A l’examen, il y a toujours une multitude de questions pièges, qui soit n’ont pas été vu au cours, soit n’ont pas fait l’objet d’une analyse profonde. C’est exactement le genre de la plupart de questions rassemblées sur ces pages. Ces questions sont réparties en trois grandes parties, selon les trois grandes questions dites “de base” (on tombe toujours sur l’une d’entre elles) :

- Le linker
- Les interruptions
- La pagination

Une approche critique est indispensable vis-à-vis de ce document, car les réponses à toutes ces questions ont été recueillies auprès des étudiants <sup>2</sup> et peuvent dès lors contenir des erreurs ou être incomplètes, même si on a tâché de ne prendre que des réponses “sûres”, vérifiées par plusieurs générations d’étudiants (voir crédits).

Pour tout ajout de question/réponse, report d’erreurs ou autre, veuillez vous adresser soit à moi par l’e-mail indiquée en bas des pages, soit aux administrateurs du forum [www.candiulb.be](http://www.candiulb.be), lieu d’hébergement de ce document.

Yves, Sangwo

---

<sup>2</sup>Ce document est un upgrade de “CyberTuyaux 1996 - The Next Generation” (fctord96.doc par Tipon)

## Question I. “Le Linker”

1. Faire le schéma d'un linker avec gestion de bibliothèque + hypothèses.

*Voir cours et l'organigramme d'un linker pur sur ftp.candiulb.be (linker.pdf)*

2. Pourquoi les corrections d'adresse (relocation, ...) se font-elles toujours en fin de mot ?

*Parce que sinon, il faut considérer les cas où une adresse d'un demi-mot est en plein milieu, et donc sauver la position en bit à l'intérieur du mot dans une table supplémentaire, le cas où une adresse chevauche 2 mots, ...*

3. Quelle est le nombre d'entrées de la TSG ?

*La TSG (contient tous les symboles des TE) contient tous les symboles et leurs adresses finales, c'ad que le nombre d'entrées est égale à la somme des nombres d'entrées des TE (tables des entrées).*

4. Comment reconnaître la Table des entrées (TE) de la Table des externes (TX) ?

*Dans la Table des externes (TX), un symbole peut apparaître plus d'une fois (à chaque fois qu'il est référencé et rien n'empêche de référencer plusieurs fois un symbole dans du code) mais il ne doit être défini qu'une fois dans une Table des entrées (TE).*

5. A quoi les adresses sont-elles relatives ?

*L'assembleur produit des adresses relatives par rapport au début du module. Ces adresses sont relocatées lors de la 1ère passe du linker et mémorisées dans la TRG (Table de relocation finale/globale). A cause de cette relocation, elles sont maintenant relatives à "d", c'ad au début du programme.*

**Remarque :** le nombre d'entrées de la TRG est égale à la somme des nombres d'entrées des TR (table de relocation) et des TX (table des externes).

6. Ce schéma était celui d'un chargeur link-relocate avec gestion de bibliothèque. Quelles sont les modifications à faire pour un linker pur ? (schéma + hypothèses!!)

*La différence entre un linker pur et un chargeur link-relocate est qu'à la fin de la 2ème passe, on n'exécute pas le programme mais on le met sur fichier avec une table de relocation finale (TRG). La relocation effectuée lors de la 2ème passe n'est donc que partielle et la relocation finale sera effectuée lors du chargement du programme binaire relocatable par un chargeur relocate-É-go. Il y a par conséquent 2 relocations.*

*Dans la 1ère passe, le seul changement c'est qu'on a  $D = 0$  au début (on veut que les adresses des symboles globaux que l'on a placées dans la TSG soient relatives au début du programme).*

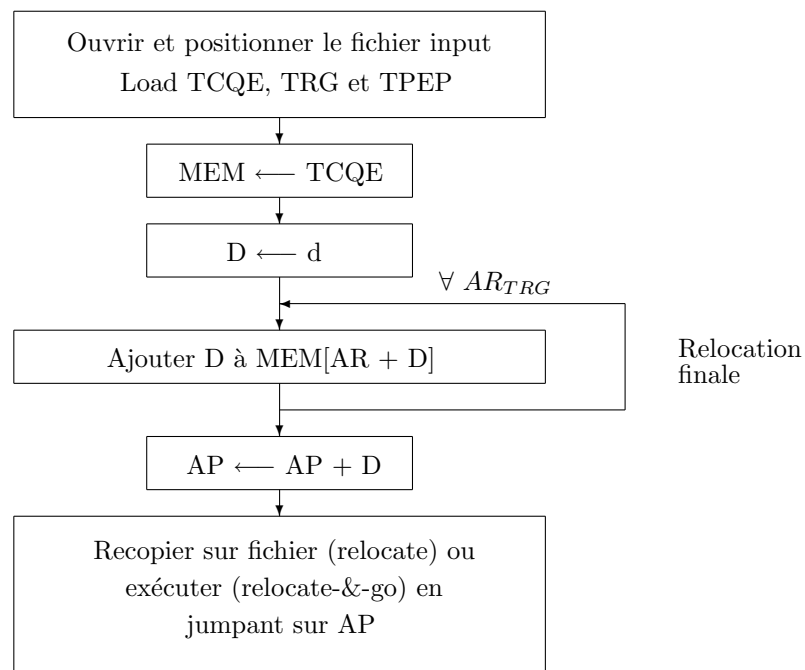
Les différences avec le schéma du link-relocate-&-go sont qu'à la 2ème passe :

- Il faut initialiser la TRG.
- Pour chaque AR de la TR et de la TX, on ajoute  $AR+(D-d)$  à la TRG. **Rem :**  $D-d$  = adresse relative par rapport au début du programme,  $d$  = adresse du début du programme lors du link,  $D$  = adresse de travail actuelle.
- A la fin de la 2ème passe, au lieu de démarrer l'exécution, on produit un fichier exécutable qui contient une TCQE, une TRG et une TPEP (qui contient un AP relatif au début du programme).

Voir le schéma de la 2<sup>ème</sup> passe

7. Dans le cas d'un linker pur, il faut ensuite un chargeur relocate-&-go pour pouvoir effectuer la relocation finale et exécuter le programme. Faire le schéma de ce chargeur.

Schéma d'un chargeur relocate-&-go :



8. Qu'est-ce qu'on met comme valeur d'adresse lorsqu'on remarque qu'il s'agit d'un symbole externe à l'étape d'assemblage du module ?

Quand on rencontre un externe, dans le code on met en general 0 et c'est l'adresse de ce 0 qui est enregistrée dans la TX. Donc, quand on aura l'adresse vraie on la rajoutera bien à l'endroit renseigné dans la TX.

9. Dans le schéma du chargeur link-relocate avec gestion de bibliothèque, à la deuxième passe, après la boucle  $\alpha$ , pourquoi si le symbole appartient à l'annuaire de librairie, on

ne retire pas le symbole de la TENS (table des externes non-satisfaites) ? Pourtant on devrait puisque l'externe est satisfait maintenant.

*En fait, on le retire de la TENS mais pas à cet endroit là. En effet, juste en-dessous, on applique la boucle  $\alpha$  au module bibliothèque, et c'est dans cette boucle qu'on retirera le symbole en parcourant la table des entrées.*

10. Dans le schéma d'un chargeur de modules, à la 1<sup>ère</sup> passe, qui détermine "d" ( $D \leftarrow d$ ) ?

*"d" est choisi par l'utilisateur ou par l'OS (le chargeur de modules demande à l'OS d'allouer de la mémoire (gestionnaire mémoire)).*

11. Quelles sont les modifications à faire dans le schéma du linker pur pour la pagination à la demande (v. chapitre 10, gestion de la mémoire centrale) ?

*Il ne faut pas créer de table de relocation finale (TRG) car on a un hardware qui travaille avec des adresses relatives au début du programme.*

12. Pourquoi la TMBUse doit-elle être linéaire (càd pourquoi l'ordre des modules bibliothèques de la 1<sup>ère</sup> passe doit-il être conservé lors de la 2<sup>ème</sup>) ?

*Parce que pour chaque module bibliothèque, D doit être identique lors de la 1<sup>ère</sup> passe et de la 2<sup>ème</sup>, sinon les infos créées lors de la 1<sup>ère</sup> passe mémorisées dans la TSG et tenant compte de D à ce moment-là seront devenues incorrectes lors de la 2<sup>ème</sup>.*

13. Que se passe-t-il si, à l'exécution du programme, on n'avait pas trouvé un symbole dans l'annuaire ?

*Interruption interne. C'est une erreur de violation mémoire. En effet, à la première passe, on avait introduit une adresse assez grande pour qu'il y ait éjection.*

14. Peut-on ne charger qu'un seul module à la fois lors de la relocation ? A quoi ça servirait ?

*Oui. En travaillant toujours dans le même buffer mémoire, on économiserait de la place ; il suffirait de sauvegarder sur fichier disk notre buffer de travail après chaque relocation, à la suite des buffers sauvegardés précédemment. De plus, ne charger qu'un module à la fois permettrait de compiler un programme plus grand que la mémoire (ce qui peut servir pour produire des binaires destinés à être exécutés dans des environnements à mémoire virtuelle : pagination/segmentation à la demande (v. ch 10)).*

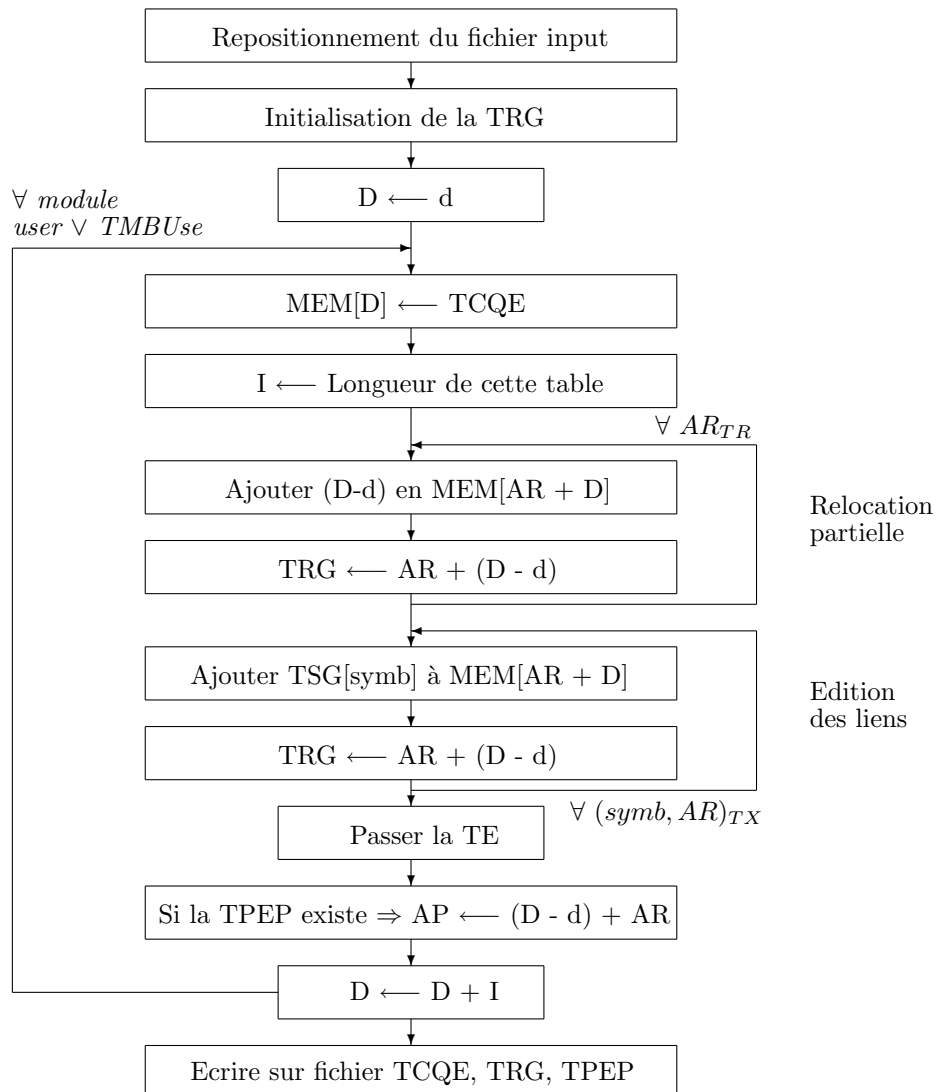
**Rem :** *Soyez capable de faire la modification sur votre schéma!!!*

15. Modifier le schéma du linker pour une gestion mémoire à MMU (v. ch. 10).

*Comme dit à la question 10, il n'y a pas besoin de TRG. Donc le schéma ressemblera au schéma normal. La différence est qu'au lieu de faire  $D \leftarrow d$ , on fait  $D \leftarrow 0$  (toujours dans le cas d'un linker pur, car on construit des adresses relatives au début du*

programme que l'on construit).

### Schéma de la 2<sup>ème</sup> passe



## Question II. “Les Interruptions”

### 1. Interruptions ?

- Définir.
- Donner un mécanisme d'interruption simple.  
(Citer les hypothèses + expliquer les schémas)
- Définir interruption interne/externe + citer tous les exemples.

Voir cours.

**Attention :** Autre exemple d'interruption interne vu plus tard : page fault (ch. 10, pagination à la demande).

### 2. Bien connaître les remarques.

Voir cours.

### 3. Dans le schéma de la boucle hardware, on a $MG \leftarrow 0$ . Est-ce qu'on ne pourrait pas demander au SV de faire ça ?

*Au départ,  $MG \leftarrow 0$  est un circuit câblé dans le processeur. Si on demandait au SV de la faire, cela voudrait dire qu'on en fait une instruction et cela pose des problèmes de protection puisque tout le monde pourrait alors switcher en mode maître.*

### 4. Dans le schéma du mécanisme avec les 2 triangles, on parle à un certain moment d'évènement non masqué mais momentanément différé, ça veut dire quoi ?

Voir cours.

**Remarque :** Si une autre interruption survient, elle sera traitée lorsqu'on sera de nouveau en mode interruptible, c'est à dire à la fin de l'interruption en cours, au prochain  $DI[i] \cdot MS[i]$ .

### 5. Comment forcer une interruption interne "instruction non identifié" ?

*Il y a minimum trois moyen :*

- On chipote sur le fichier binaire
- On chipote avec du code automodifiant
- On utilise un vieux processeur

**Remarque :** Si on écrit une mauvaise instruction dans le code-source - ca ne marchera pas, car la faute sera détectée par le compilateur/traducteur.

### 6. C'est quoi le vecteur $DI[]$ ? A quoi sert-il ?

Il sert à 2 choses :

- Voir quelles sont les interruptions non encore traitées.
- Fixer une priorité de demande d'interruption.

En effet, si 2 interruptions ont lieu,  $\sum_{i=1}^n (DI[i] \cdot MS[i]) \neq 0$ , mais quelle sera l'interruption déclenchée par le SV ? Ce sera celle qui est la plus à gauche dans le vecteur  $DI[]$  (en supposant un parcours de gauche à droite). Les interruptions sont donc rangées en ordre de priorité décroissante.

+ voir aussi remarque D (paragraphe 5).

**D.5.** On peut s'arranger pour pouvoir être interrompu même si on est déjà dans une routine de gestion par une autre interruption de priorité plus grande.

Pour ce faire, on range les demandes d'interruption par ordre décroissant de priorité et on joue sur le vecteur de masques. Il faut aussi dissocier le mode du masque général d'interruption (on peut être dans l'OS en mode interruptible et en mode maître) : on sauvera le mode dans un registre SM. La routine de gestion de l'interruption  $i$  est alors :

- Sauver registres, SP, MS ( $\Rightarrow$  stack)
- SM := mode (sauve le mode dans un reg pour savoir si mode maître ?)
- $DI[i] \leftarrow 0$
- $MS[j] \leftarrow 0, j \geq i$  (interruption  $i$  interrompue par interruption  $j$ )
- $MG \leftarrow 1$
- Gérer l'évènement correspondant (seule partie interruptible)
- $MG \leftarrow 0$
- Restaurer registres, MS, P, mode ( $\Leftarrow$  stack)
- $MG \leftarrow 1$

7. Pourquoi doit-on nécessairement effacer  $DI(i)$  ?

Facile - voir cours.

**Remarque :** Si on l'efface pas, on n'aura pas forcément une interruption non prise en compte à la fin du SV, car il se peut qu'on aie mis le  $DI(i)$  à 0 dans le code de l'OS, SV ou routine (on est dans le mode maître).

8. MG indique le mode interruptible mais quoi encore ?

MG indique l'état interruptible et le mode (privilégié/non-privilégié).

9. Définir l'appel superviseur et expliquer à quoi il sert. (+ exemples!!)

Voir cours.

**Remarque :** le SV et les routines de gestion d'interruptions - c'est du soft.

10. Dans le schéma du SV, au retour, on a

- Restaurer les registres.
- $MG \leftarrow 1$  (Rendre la machine de nouveau interruptible).
- $P \leftarrow SP$  (Revenir au processus interrompu).

Est-il gênant de permuter  $MG \leftarrow 1$  et  $P \leftarrow SP$  ?

*Il ne s'agit pas là de circuits cablés, mais d'instructions (on est dans le SV). Une interruption peut donc survenir dès le moment où on fait  $MG \leftarrow 1$ .*

*Considérons les 2 cas :*

1<sup>er</sup> cas :  $MG \leftarrow 1$   
 $P \leftarrow SP$

*Supposons qu'il y ait une interruption entre les 2 (possible puisque  $MG=1$ ), dans la boucle hardware, on fera de nouveau  $P \leftarrow SP$ . On vient d'écraser l'ancien  $P$  avec l'adresse de retour au processus utilisateur et on ne pourra jamais y revenir.*

**Precisions :** Dans ce cas on va cycliser sur **une** instruction, à savoir la dernière du SV.

2<sup>eme</sup> cas :  $P \leftarrow SP$   
 $MG \leftarrow 1$

*$P \leftarrow SP$  correspond en fait à un branchement et donc  $MG \leftarrow 1$  ne sera jamais exécuté. On sera donc parachuté en mode non-interruptible et en mode privilégié dans le processus utilisateur (le process utilisateur ne serait dans ce cas plus jamais interrompu et pourrait même effectuer des instructions privilégiées, ce qui fouerait en l'air toute la protection du système)!!!*

**Solution :** Faire les 2 en même temps (instruction atomique). On y est obligé. Cela signifie qu'une seule instruction fait les 2. Par exemple sur PC, il y a le IRET qui restaure le flag d'interruption et retourne au processus interrompu en rechargeant l'adresse du processus-utilisateur dans  $P$ ).

11. Pour que le mécanisme fonctionne bien, que doit faire le SV ?

*Sauver et restaurer les contextes, ... (voir cours).*

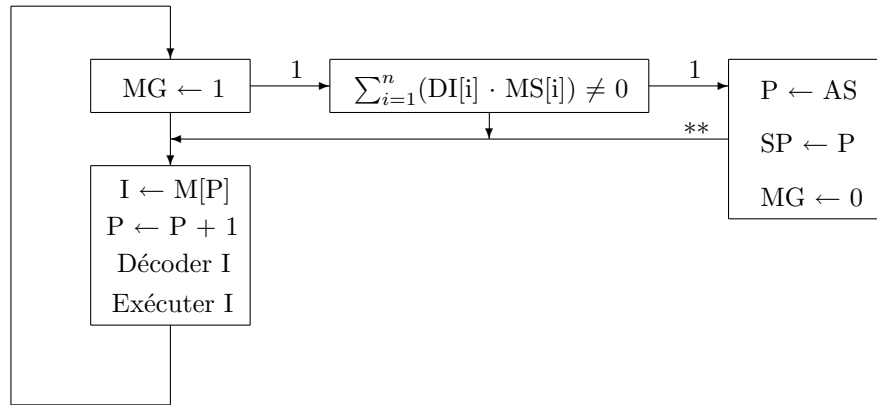
12. Dans les tâches du SV après une interruption, lorsque celui-ci doit effacer le flag de prise en charge  $DI[i]$ , est-ce dangereux ?

*Oui, car une interruption de même type peut avoir eu lieu, et en effaçant le flag on la perdrait (on détruirait sa demande et on ne la prendrait donc pas en compte).*

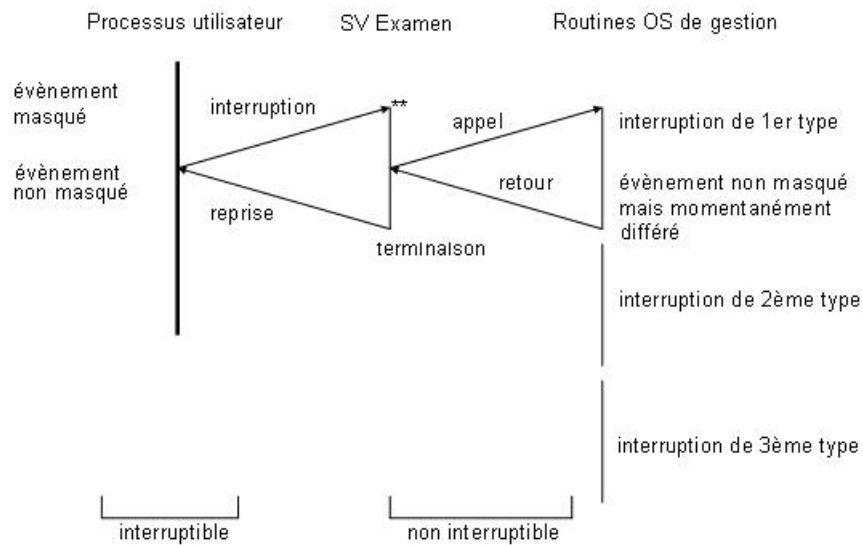
13. Dans le schéma de la boucle hardware, l'endroit  $P \leftarrow AS$  correspond à quel endroit dans le schéma avec les 2 triangles ?

**(Question piège :** si vous vous plantez, votre cote tombe directement à 7/20!!!)

Voir les 2 étoiles \*\* dans les 2 schémas.



Précisons ici que  $\sum_{i=1}^n (DI[i] \cdot MS[i]) \neq 0$  est fait entièrement par des opérations logiques : la somme est ici un OR et le "." est un AND (produit logique).



Le programme se déroule normalement. Une interruption arrive et on passe dans le SV qui sauve les contextes et appelle l'OS. Celui-ci gère l'interruption et rend le contrôle au SV. Le SV restaure les contextes et rend le contrôle au processus utilisateur.

Ces deux schémas sont intimement liés. Les endroits marqués \*\* se correspondent. Sur le schéma de la boucle du processeur, les étoiles sont situées juste après  $P = AS$ , c'est-à-dire que  $P$  pointe sur la 1<sup>ère</sup> instruction du SV. Sur le schéma software, les deux étoiles sont situées au sommet du triangle représentant l'examen du SV. De même, les endroits où la machine est interruptible (processus utilisateur) et non interruptible (SV & OS).

*Si une interruption survient lorsque la machine est non interruptible (SV & OS), l'interruption ne peut être prise en compte tout de suite (événement non masqué mais momentanément différé). L'interruption suivante attendra : à la fin de l'interruption du 1<sup>er</sup> type, le SV rendra le contrôle au processus utilisateur qui sera tout de suite de nouveau interrompu pour laisser place à l'interruption du 2<sup>ème</sup> type. Il est toutefois possible dans des cas particuliers qu'une interruption prenne en compte les demandes de même type survenant pendant sa gestion (si l'interruption dure longtemps : accès disque, ...).*

14. Dans le schéma de la boucle hardware, on a

- $MG \leftarrow 0$
- $SP \leftarrow P$
- $P \leftarrow AS$

Est-il gênant d'exécuter ces instructions dans un autre ordre ?

*Il est évident que  $P \leftarrow AS$  doit être exécuté après avoir sauvé l'adresse de  $P$  dans  $SP$ , sinon on sauverait l'adresse de  $AS$ , ce qui ne sert à rien. Par contre,  $MG \leftarrow 0$  peut être placé n'importe où, car cela se fait par hardware et il n'y a donc pas d'interruption possible à ce moment-là.*

15. Ajuster  $P$ ... Ca signifie quoi ?

*Ca signifie qu'on incrémente  $P$  pour qu'il pointe sur l'instruction suivant celle qui a été interrompue*

### Question III. “La Pagination”

1. Expliquer le principe de la pagination.

*Voir cours*

2. Où est-ce qu'on utilise la technique de la pagination ?

- *Mémoire centrale*
- *Mémoire secondaire*
- *La cache*

3. Donner un exemple de technique où un processus est divisé en pages mais qui n'est pas une pagination.

*Recouvrement minimal.*

4. Expliquer le principe de la pagination à la demande.  
(+ page fault + avantages/inconvénients)

*Voir cours*

5. Pourquoi mettre la longueur de la table des pages dans des registres de contrôle ?  
Comment accède-t-on à cette table des pages ?

*La longueur de la table des pages est dans des registres de contrôle parce qu'on en a besoin à chaque accès mémoire (protection mémoire : vérifier si le processus travaille bien dans l'espace mémoire prévu). Ca doit donc être rapide, càd hardware.*

*La table des pages doit être accessible par l'OS (pour le swapping en particulier). Une instruction privilégiée devra donc en donner l'accès. La table des pages sera en MC (mémoire centrale, reg MMU pointant vers cette zone) ou dans des registres spéciaux du MMU (ou un peu des 2...) Par ex. les descripteurs des dernières pages accédées sont dans MMU et les autres en MC (cf. le fonctionnement d'un cache).*

**Rem :** *la partie de la table des pages concernant le swapping (càd la position disk où se trouve la page) ne concerne que l'OS et n'a donc aucune raison de se trouver dans le hardware MMU, on la trouvera donc en MC dans une table de l'OS.*

6. Comparer les avantages de la pagination à la demande avec les autres techniques (notamment partitions relocatables et multiples).

*Avantages :*

- *Capacité d'exécuter un programme plus grand que la mémoire centrale.*
- *Aussi rapide que les autres malgré le calcul d'adresses grâce au MMU hardware.*

- Permet la préemption.
- Pas besoin de segmenter le programme.
- ...

7. Quand est-ce qu'une pagination à la demande est infaisable ?

- Si le programme a une longueur supérieure à l'espace d'adressage du processeur
- Si la mémoire secondaire n'est pas disponible
- Si la machine n'a pas d'MMU
- Si il n'y a pas assez de places en MC pour la TP

8. Comment modifier les descripteurs de pages en pagination pour diminuer leur taille ?

*Pour accélérer le calcul d'adresse et diminuer la taille des descripteurs, on ne met pas l'adresse de début de la page dans la table des pages. On met plutôt le numéro de page, qu'il suffit de multiplier par  $l$  (suffit d'un simple masque pour extraire les bits + shift car  $l$  est une puissance de 2) pour obtenir l'adresse réelle de la page.*

9. La traduction automatique des adresses en pagination se fait par hard ou par soft ? Est-il possible d'avoir une pagination software ? Autrement dit : Est-ce qu'on peut utiliser la technique de pagination sur toutes les machines ? Pourquoi ?

*En pagination, la traduction se fait par hardware (MMU). Cela permet une grande rapidité. Sur les anciennes machines, le MMU n'existait pas. On peut se demander si la pagination à la demande est possible par software sur de telles machines.*

*On ne peut pas vraiment dire que la pagination software soit applicable au niveau du CPU, car cela serait bien trop lent, mais elle est souvent utilisée pour la pagination de fichiers disk par exemple. Si on l'appliquait au niveau du CPU, ça serait plutôt dans le cadre de simulations, car elle imposerait d'intercepter chaque instruction, ... On aurait en tout cas de fameux problèmes de rapidité dûs au manque de MMU, il n'y aurait pas de registres de contrôle, pas de protection-mémoire (une protection mémoire soft n'est pas sûre!!!). Il faudrait en fait que l'OS contrôle chaque accès mémoire. C'est à cause de ce besoin de rapidité que le MMU est souvent intégré au circuit du CPU. On aurait également des problèmes au niveau des périphériques DMA qui travailleraient avec des adresses physiques et nous logiques... (avec un MMU, les adresses utilisées par un périphérique sont traduites en adresses physiques par le MMU, mais ici on serait incapable de faire son boulot!!!).*

10. Comment minimiser la perte de mémoire due à la gestion de la pagination ?

*On peut minimiser la perte de gestion de la pagination en choisissant bien la taille des pages (voir calcul dans chapitre sur la pagination statique).*

**Fragmentation interne.** *En moyenne, la dernière page d'un processus est à moitié vide. C'est-à-dire qu'en moyenne, on a une perte de  $\frac{l}{2}$  mots,  $l$  étant la taille d'une page. La solution à cette fragmentation est un compromis entre la taille des pages (frames)  $l$  et le nombre de pages d'un processus. **Rappel :** une page frame est le contenant, et*

une page est le contenu... Une diminution de  $l$  diminue la fragmentation interne mais augmente la taille des tables de pages, puisque dans ce cas, il faut plus de pages pour couvrir le même espace mémoire, ce qui entraîne :

- Il faudra un MMU capable de gérer des tables de pages de taille importante, et donc être capable d'intégrer cela dans un circuit, mais il y a des limites à l'intégration.
- De moindres performances lors des context-switches, car il faudra plus de temps pour charger et décharger cette table chaque fois qu'un processus est redémarré ou suspendu (plus d'entrées à mettre à jour).

Si  $L$  est la longueur moyenne d'un processus, le nombre de pages requises est :

$\lceil \frac{L}{l} \rceil \Rightarrow$  nombre de pages pour le processus..

$\Rightarrow$  2 cas possibles :

- Soit ça tombe juste, ça veut dire qu'il n'y a pas fragmentation interne et ce n'est pas le cas qui nous intéresse ici (on s'intéresse au cas défavorable)...
- Soit ça vaut  $(L \text{ div } l) + 1$  (1 page supplémentaire pour faire rentrer notre processus, et il y a fragmentation interne!).

**Rem :** div est la division entière.

Si  $k$  est la taille d'un descripteur de page, la taille de la table des pages est

**nombre\_de\_pages · k**

La perte totale de place due à la pagination et à sa gestion est donc égale à la place occupée par la table des pages et la place inutilisée dans la dernière page du processus :

$$P(l) = (L \text{ div } l + 1) \cdot k + l \text{ div } 2$$

Le but est de minimiser cette perte. Intuitivement, on annulerait la dérivée pour trouver un extremum :  $P'(l) = 0$ . Mais cette fonction n'est pas réelle... C'est une fonction en escalier, c'est-à-dire non continue et donc non dérivable. De plus, il s'agit d'une fonction entière, et dériver n'a de sens que si la variable est réelle. On va approcher cette fonction par la fonction réelle suivante :

$$P(l) = \left(\frac{L}{l} + 1\right) \cdot k + \frac{l}{2}$$

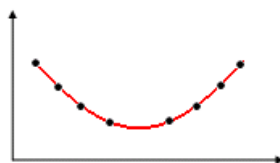
qui est une fonction réelle concave. Si on trouve un extremum, ce sera donc un minimum...

$$P'(l) = \frac{1}{2} - k \cdot \frac{L}{l^2} = 0 \implies l \sim \pm \sqrt{2kL}$$

On rejette bien sûr la solution  $-\sqrt{2kL}$ , et le minimum qui nous intéresse est  $\sqrt{2kL}$ .

Mais ce n'est pas le minimum de la fonction réelle que l'on cherchait. Ce qu'on veut, c'est le minimum de la fonction entière. Les deux minimums ont toutes les chances d'être différents, mais on peut se demander s'ils sont proches.

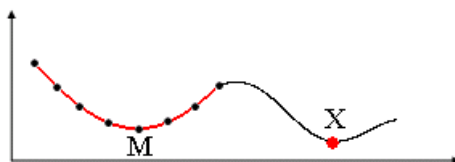
Regardons l'allure de la fonction entière :



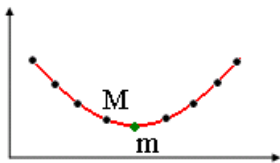
Il est évident qu'il existe une infinité de fonctions réelles passant par les points d'une fonction entière (il suffit de prendre n'importe quelle fonction biscornue ou non passant par ces points!).

Ici, la fonction n'existe qu'en des valeurs entières. En ayant choisi une fonction réelle qui passait par ces valeurs entières, on avait pris des risques car certaines de ces fonctions réelles peuvent avoir un minimum à un tout autre endroit que la fonction entière.

**Exemple :**



Le minimum de la fonction entière est  $M$ . Celui de la fonction réelle est  $X$ . On ne peut donc pas approcher la fonction entière par n'importe quelle fonction réelle. Mais la nôtre est bien choisie, car c'est une fonction réelle concave (on n'a pas plein de bosses du genre sinusoïde par exemple), et dans ce cas le minimum de la fonction réelle ( $m$ ) est proche de celui de la fonction entière ( $M$ ) :



En regardant l'équation, on constate que notre fonction réelle est une hyperbole. Le minimum recherché est donc un des 2 points de la fonction entière les plus proche du minimum trouvé en minimisant la fonction réelle. Tout ceci marche parce que la fonction réelle est concave, et le minimum de la fonction entière est proche du minimum de la fonction réelle.

Comme il faut aussi que le calcul des adresses ( $AV \text{ div } l$ ) soit simple, il convient de choisir  $l$  égal à la puissance de 2 la plus proche de la racine.

11. Pourquoi, dans le calcul de la taille des pages peut-on momentanément oublier que l'on manipule des entiers ?

Car le minimum de la fonction entière est proche de celui de la fonction réelle concave.

12. Pourquoi AR est-elle si facile à calculer si  $l = 2^x$  ?

*Il suffit d'un masque pour extraire les bits.*

13. Quelle est la taille maximale d'un programme en mémoire si les adresses sont codées sur 32 bits ?

$$T = 2^{32} \text{ bits} = 2^2 \cdot 2^{30} \text{ bits} = 4 \text{ gigas}$$

14. Pourquoi ne peut on pas exécuter des processus beaucoup plus grands que la mémoire ?

- Les adresses sont codées sur  $n$  bits  $\Rightarrow$  la + importante limitation (expliquer)
- La mémoire secondaire peut être saturée
- La table des pages peut être trop grande pour la mémoire centrale

15. Où se situe la table des pages ?

*Soit en mémoire centrale (MC), soit dans des registres spéciaux du MMU.*

16. Est-ce qu'on peut swaper la table des pages ?

*Non, car le swapper en a besoin pour travailler.*

17. La protection mémoire paraît facile ( $AR \geq 0$  et  $(AR \text{ div } l) \leq \text{taille\_de\_la\_table\_des\_pages}$ ). Il reste toutefois un petit problème... Lequel ?

*Il se peut que le programme n'occupe pas un nombre entier de pages, et il y a alors apparition de fragmentation interne. Dans la fin de la dernière page, il n'y a rien d'intéressant et le programme ne peut théoriquement pas y accéder, mais le mécanisme de protection n'empêche pas une incursion dans cette zone. On avait dû arrondir et prendre le plafond mais cela laisse un trou inutilisé (qui n'est donc pas protégé) et à ne pas utiliser.*

18. Quelles sont les restrictions de la pagination à la demande si l'on emploie des programmes plus grands que la mémoire ?

- Surveiller la place que prennent les descripteurs de page en mémoire physique (cette place grandit avec le nombre de pages et de page frames)
- La mémoire disponible est limitée par le nombre de bits pour coder l'espace d'adressage du CPU. On a donc une borne supérieure du nombre de pages que l'on peut adresser
- Possibilité d'overflow de la mémoire secondaire (MS)
- Taux de swaps élevé
- ...

19. Quelle est la différence entre un linker sur une ancienne machine et sur une nouvelle ?

*Un linker sur une ancienne machine ne disposera pas de MMU. Une nouvelle machine oui, et il n'y a pas besoin de relocation finale dans ce cas.*

20. Qui déclenche le page fault ?

*Le MMU déclenche l'interruption page fault. Le processeur est alors parachuté dans l'OS qui le dirige vers la routine de gestion associée à cette interruption. Le MMU déclenche l'interruption mais c'est le processeur qui exécute la routine de gestion correspondante. L'interruption page fault peut être considérée comme une interruption interne car elle est liée à ce que fait le CPU pour le moment (une interruption externe par contre se déclenche indépendamment de ce que fait le CPU, ex : horloge, dma disk, ...)*

21. A quel pourcentage travaille le MMU ?

*Le MMU travaille à 100%. En effet, à chaque instruction, il est utilisé*

- Pour lire le code de l'instruction ( 1<sup>er</sup> accès mémoire, 1<sup>er</sup> calcul MMU).*
- Pour aller chercher le/les opérandes mémoires (autres accès mémoires, autres calculs MMU) + éventuellement indirections (pointeur de pointeur).*

*Il peut donc y avoir plusieurs accès mémoire pour une seule instruction (et donc plusieurs calculs MMU).*

22. Comment accélérer l'activité du MMU et pour quel(s) organe(s) ce MMU travaille ?

*Pour accélérer le travail de MMU, il faut avoir les TPs dans des registres spéciaux du MMU. Le MMU travaille principalement pour le processeur, mais aussi pour les périphériques DMA (Direct Memory Acces) du genre "canaux" etc.*

23. Est-ce qu'on a besoin du MMU pour exécuter un programme tout simple (p.e. on fait 10 fois INC AX en ASM) sans aucun accès mémoire ?

*Il n'existe pas de programmes sans accès mémoire, car on en fait déjà un lors du lancement de chaque programme.*

24. La traduction d'adresse ne peut pas se faire au chargement. Pourquoi ?

*En pagination (statique ou à la demande), si un vecteur a une longueur supérieure à celle d'une page, on aura beau charger son adresse de début, le problème se posera toujours à l'exécution car il sera à cheval sur deux pages (pas nécessairement contiguës). Il est donc nécessaire d'avoir un MMU.*

*Il faut également que, arrivé à la fin d'une page, le code puisse continuer à la suivante (même de type de raisonnement que pour le vecteur).*

*De plus, si on travaille en pagination à la demande, une page peut être swappée out et puis swappée in dans une autre page frame en mémoire. Il faut donc que les*

*références au code et aux données soient indépendantes de l'emplacement physique en mémoire, et il faut un mécanisme de traduction dynamique d'adresse pour que cela puisse s'exécuter.*

25. Quelles sont les différences entre pagination statique et partitions multiples ?

*La pagination statique et les partitions multiples sont toutes les deux NC, NP (non contiguës, non préemptives), **mais** en pagination, on n'a pas de découpage du programme en segments comme il y en a dans les partitions multiples.*

*Voir les différences entre pagination et segmentation (segmentation statique 3.3).*

|   | <b>Pagination</b>                 | <b>Segmentation</b>                              |
|---|-----------------------------------|--|
| <i>Technique transparente ?</i>   | Oui                               | Non  |
| <i>Taille des pages/segments ?</i>  | Fixée par l'OS                    | Variable   |
| <i>L'espace d'adressage total peut-il dépasser la taille de la mémoire physique (mémoire virtuelle) ?</i> | Oui en pagination à la demande    | Oui en segmentation à la demande                 |
| <i>Espace d'adressage virtuel...</i>  | Continu                           | Discontinu                                       |
| <i>Possibilité de protéger différemment 2 zones différentes ?</i>   | Non                               | Oui  |
| <i>Gestion facile des données dont la taille varie ?</i>  | Non                               | Oui  |
| <i>Possibilité de partager des données ?</i>  | Non                               | Oui  |
| <i>But</i>  | Avoir un grand espace d'adressage | Séparer logiquement certains espaces d'adressage |
| <i>Savoir si un programme entre en mémoire ?</i>  | Facile                            | Difficile  |
| <i>Fragmentation interne</i>  | Petite                            | Nulle  |
| <i>Fragmentation externe</i>  | Nulle                             | Grande   |
| <i>Protection mémoire</i>   | Facile                            | Difficile  |
| <i>Descripteurs</i>   | Faciles                           | Complicés  |

26. Citer les techniques de gestion de la mémoire (toutes).

*Voir cours.*

27. Dans quelles techniques n'a-t-on pas besoin de relocation finale ?

*Les techniques à relocation hardware automatique :*

- pagination/segmentation (relocation par le MMU)*
- partition relocatable (registre de base RA)*

28. Quelle est la meilleure méthode pour protéger la mémoire pour une partition multiple ? Comment ça se passe ?

*La méthode clé-serrure est assez adaptée. Il faudrait sinon une paire de registres début-fin pour chaque segment. La méthode clé-serrure impose toutefois que les segments soient en un nombre entier de blocs mémoire (gestion de mémoire par blocs + fragmentation interne).*

29. Quelle est la meilleure méthode pour utiliser le maximum de mémoire?

*Les meilleures techniques pour utiliser le plus de mémoire sont évidemment celles où il y a le moins de fragmentation :*

- Partitions relocatables*
- Pagination statique/à la demande*

*Les partitions relocatables étant lourdes à gérer (switching time), il faut avouer que la pagination est la meilleure des 2 techniques en pratique.*

## Crédits

**John'stick** (90-91)

**Baltus** (91-92)

**Vachon** (92-93)

**Cristina** (92-93)

**Rod** (94-95)

**Tipon** (92-93 et 95-96)

**Sangwo Yves** (01-02)

Merci à tous les membres de candiulb ayant posté les Q/R de leurs examens sur le forum.