Segmentation

Attention : Dans ce TD, toutes les *tailles*, *adresses* et *numéros de segments* sont donnés en hexadécimal.

Exercice 1. Questions de cours Les questions de cours sont à destinées à vous permettre de vérifier votre compréhension du cours. Elles sont à travailler à l'avance et ne seront pas traitées en TD ou TP.

1. Expliquez ce qu'est l'édition de liens (en quelques mots).
2. Quelle est la meilleure stratégie d'allocation dans le cas d'une allocation contigüe ?
3. Donnez deux avantages de la segmentation.
4. Comment est déterminée une erreur de segmentation ?

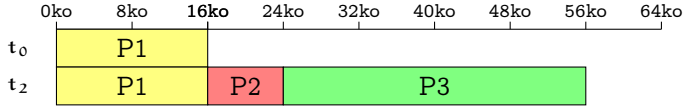
Exercice 2. Mémoire contiguë On considère un système muni de 64ko de mémoire et capable de gérer plusieurs processus en mémoire simultanement.

1. Quelle est la taille de l'adresse physique ?
   Correction: *$64ko = 2^{16}o$ donc l'adresse est sur 16bits.*
2. Quelle est la taille de l'adresse logique ?
   Correction: *16bits aussi puisqu'il n'est pas indiqué de restrictions particulière telle que le fait qu'un processus ne puisse utiliser qu'une partie de la mémoire. On suppose donc qu'il peut utiliser l'intégralité de la mémoire s'il en as besoin, auquel cas aucun autre processus ne pourra s'exécuter en même temps.*

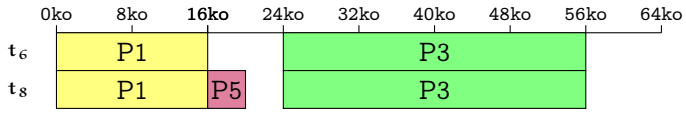
On décide d'utiliser un algorithme d'allocation contiguë qui met en attente les processus lorsqu'il n'y a plus de bloc libre de taille suffisante. On considère l'exécution suivante, pour laquelle nous donnons pour chaque processus sa date d'arrivée, son besoin en mémoire (donné à la création du processus) et sa durée d'exécution totale :

Processus	Arrivée	Mémoire	Durée
P1	0	16ko	10
P2	2	8ko	4
P3	2	32ko	12
P4	4	12ko	12
P5	8	4ko	6
P6	12	32ko	4

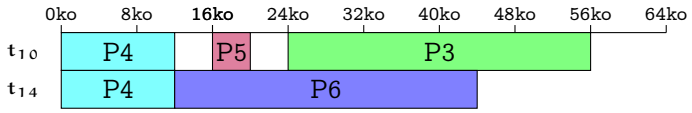
3. On suppose que l'allocation se fait suivant un algorithme de type "First Fit". Décrivez précisément l'allocation mémoire à chaque pas de temps, en expliquant au fur et à mesure comment sont positionnés les processus.
   Correction: *Aux temps t_0 et t_2 tout se passe simplement, les processus arrivent et sont mis les uns à la suite des autres en mémoire.*



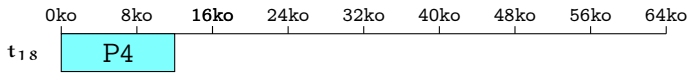
Au temps t_4, le processus P4 arrive mais il n'y a pas suffisament de place pour l'exécuter, il est donc mis en attente.



Au temps t_6 le processus P2 se termine créant un trou de mémoire libre trop petit pour y placer P4 toujours en attente. Si le système défragmentait la mémoire il pourrait entrer mais ce n'est ici pas le cas. Au temps t_8 le processus P5 arrive et est suffisament petit pour trouver une place.



Au temps t_{10} le processus P1 se termine libérant suffisament d'espace pour que P4 puisse arriver, alors que P6 qui arrive au temps t_{12} doit lui partir en attente car il est trop gros. Il attend jusqu'au temps t_{14} lorsque les processus P3 et P5 se terminent libérant suffisament de mémoire.



Au temps t_{18} P6 se termine et seul P4 reste. Il se terminera au temps t_{22}.

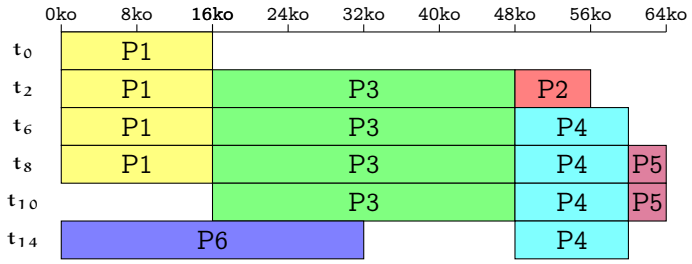
4. Quel est le temps total d'exécution ?
Correction: *P4 est de le dernier processus à s'exécuter et il se termine au temps 22.*

5. Le taux de fragmentation est la quantité de mémoire dans les "trous" de mémoire libre ramenée à la quantité de mémoire actuellement allouée. Quel est le taux de fragmentation au temps t_8, t_{10} et t_{12} ?
Correction: *Attention, on ne doit compter que les "trous", le bloc de mémoire libre à la fin de la mémoire n'en est pas un et ne doit donc pas être compté.*

- t_8 *Il n'y a qu'un seul trou de 4ko pour 52ko de mémoire utilisée, ce qui donne un taux de fragmentation d'environ 7.7%.*
- t_{10} *Le même calcul donne ici 16.7% de fragmentation.*
- t_{12} *Le taux de fragmentation ne change pas par rapport au précédent mais on peut remarquer qu'il n'y a pas de vrai problème de fragmentation ici, en effet, même si le système défragmentait la mémoire le processus P6 ne pourrait entrer car il n'y a pas suffisament de mémoire libre au total.*

6. L'exécution serait-elle meilleure avec un algorithme de type "Best Fit" ou "Worst Fit" ?
Correction: *Il est inutile de refaire toute l'exécution : il faut regarder les moments où des processus sont alloués ou mis en attente. À t_4, pour P4, nous n'aurions rien pu faire. À t_6, si nous avions mis P3 avant P2, nous aurions pu placer P4 mais ni Best Fit, ni Worst Fit ne permettent cela (ils prennent les processus dans l'ordre). Ensuite, P5 ne gène personne puisqu'il part au moment où P6 arrive. Donc nous n'aurions pas fait mieux avec ces deux algorithmes.*

7. Quelle serait l'allocation optimale afin de minimiser le temps de calcul sur cet exemple ?
Correction: *En allouant P3 avant p2, on peut placer P4 dès t_6. Cela donne :*



Et un temps total d'exécution de 18.

Exercice 3. Segmentation simple On considère un système de gestion de la mémoire utilisant la segmentation avec des adresses logiques sur 16bits dont les 4bits de poids fort indiquent le numéro de segment. Les premières lignes de la table des segments d'un processus sont données dans le tableau suivant :

Segment	Base	Limite
0	02B9	5FF
1	2A00	014
2	0090	100
3	C3A7	5D0
4	1D52	09F

1. Quelle quantité de mémoire un processus peut-il utiliser au maximum ?
Correction: *Les adresses logiques étant sur 16bits, un processus peut utiliser au maximum* 2^{16} *octets de mémoire, soit 64ko.*

2. Quelle est la taille maximale d'un segment ?
Correction: *Les adresses logiques font 16bits dont 4 sont utilisés pour le numéro de segment, le décalage est donc codé sur 12bits ce qui donne des segments de* $2^{12} = 4ko$.

3. Quelles sont les adresses physiques correspondant aux adresses logiques suivantes : 0x0430, 0x1010, 0x2B0F, 0x34FF et 0x4100 ?
Correction: *Le décodage est ici très simple, le premier chiffre hexadécimal donne le numéro du segment et les trois suivants indiquent le décalage. Il suffit donc de vérifier que le décalage est bien inferieur à la limite du segment et si c'est le cas additioner le décalage et la base du segment. Cela donne donc :*

0x430 < 0x5FF, *donc* 0x06E9
0x010 < 0x014, *donc* 0x2A10
0xB0F > 0x100, *donc erreur de segmentation*
0x4FF < 0x5D0, *donc* 0xC8A6
0x100 > 0x09F, *donc erreur de segmentation*

4. En supposant que toutes les autres lignes de la table ont les valeurs base et limit à zéro, quelle quantité de mémoire est allouée à ce processus.
Correction: *Il suffit de sommer la taille des différents segments :*

$$5FF + 014 + 100 + 5D0 + 09F = D82$$

Soit 3458 octets alloués à ce processus. Il faut noter que en pratique le processus utilise un peu plus car sa table des segments doit aussi être stockée en mémoire. Il n'est toutefois pas possible de la calculer ici car on ne connait pas la taille des adresses physique et la base des segments est une adresse physique.

Exercice 4. Mini segmentation On considère un système de gestion de la mémoire pour un micro-controleur, c'est-à-dire un très petit processeur consommant très peu d'énergie dont le prix est très bas. Ce système étant très simple, ses capacitées sont limitées. Il dispose quand même d'une gestion de la mémoire segmenté mais un peu particulière.

Ce système dispose de 16ko de mémoire physique. Afin de réduire la consomation mémoire, une seule table des segments est utilisée pour l'ensemble des processus. Cette table permet de décrire 128 segments, chaque ligne est composé de :

- 3bits indiquant le numéro du processus à qui appartient le segment décris (le numéro 7 indiquant un segment partagé par tous les processus) ;
- 9bits indiquant l'adresse de base divisée par 32 (les adresses de base sont des multiples de 32 la division est donc exacte).
- 4bits donnant la taille du segment divisée par 64 (les segments ont forcément une taille multiple de 64 la division est donc exacte) ;

Cette manière de faire la segmentation permet une gestion de la mémoire offrant une protection et flexibilité tout à fait satisfaisantes pour ce type de processeur en ne consomant que 256o de la faible quantité de mémoire disponible.

Les multiplications par 32 et 64 sont très simples à réaliser en binaire simplement en ajoutant repectivement 5 et 6 zéros à la fin du nombre.

1. Quelle est la taille de l'adresse physique ?

Correction: 16ko $= 2^{14}$ *donc les adresses physiques sont codées sur* 14*bits. On peut aussi partir du fait que les adresses de base sont des adresses physiques et que* $2^9 \times 32 = 2^9 \times 2^5 = 2^{14}$.

2. Quelle est la taille maximale d'un segment ?
Correction: $2^4 \times 64 = 1$*ko au maximum.*

3. Quelle est la taille et la composition de l'adresse logique ?
Correction: *L'adresse logique doit contenir le selecteur de segment et le décalage. Il y a* 128 *segments, il faut donc* 7*bits pour le sélecteur, et les segments font* 1ko $= 2^{10}$o *au maximum donc on a besoin de* 10*bits pour le décalage. Ce qui nous donne des adresses logiques sur* 17*bits.*
Un extrait de la table des segments est donné ci-dessous ;

Segment	Processus	Base	Limite
C7	3	103	2
03	1	1CC	3
17	1	0E0	1
E5	2	000	A
06	7	0E5	4
2A	4	1B0	3

4. Donnez, pour le processus 3, les adresses physiques correspondant aux adresses logiques suivantes : `A8C7` et `1A2C`.
Correction: `A8C7` *: On commence par convertir l'adresse en binaire* `1010 1000 1100 0111` *ce qui permet d'isoler les* 6*bits du selecteur :* `2A` *et les* 10*bits du décalage :* `0C7`*. On commence par vérifier que le processus est bien le propriétaire de ce segment ce qui est le cas ici. On calcul ensuite la taille du segment en multipliant la limite donnée dans la table par* 64 *ce qui nous donne une taille de* `D0` *octets que l'on peut comparer au décalage pour voir que l'accès est bien valide. On calcul ensuite l'adresse de base du segment en multipliant la valeur donnée dans la table par* 32 *pour obtenir l'adresse* 3600 *à laquelle on ajoute le décalage qui nous donne l'adresse physique* `36C7`*.*

`1A2C` *: De la même manière on obtient* `06` *comme sélecteur et* `22C` *comme décalage. Le processus a bien le droit d'accéder à ce segment car il est marqué comme global (valeur 7 dans le champs processus) mais le décalage est superieur à la limite (une limite de* 3 *donne une taille de* `C0` *après la multiplication). On a donc ici une erreur de segmentation.*

Exercice 5. Segmentation avancée On considère un système doté de 1Mo de mémoire physique utilisant la segmentation. Chaque processus peut utiliser un maximum de 256 segments d'une taille maximale de 64ko. 128 de ces segments sont locaux au processus et sont décris par une table des segments propre à ce processus. Les 128 autres segments sont commun à tous les processus et décris par une table des segment commune à tous les processus du système. Le bit de poids fort de l'adresse logique indique, quand il vaut 1, que le segment référencés par cette adresse logique est un segment global.

Un segment ne peut commencer qu'à une adresse multiple de 256, cela implique que le dernier octet de son adresse de base sera toujours 0 et qu'il n'est donc pas nécessaire de le stocker dans la table des segments.

Tous les segments, qu'ils soient locaux ou globaux, sont décris dans une des deux tables par leur adresse de base, leur limite ainsi que par 3bits de protection R, W et X, indiquant si un accès en lecture, en écriture ou en exécution est autorisé, ainsi qu'un bit S indiquant si l'accès au segment est réservé au mode superviseur du système.

1. Quelle est la taille des adresses physique ?
Correction: *Il y a* 1*Mo de mémoire physique, c'est-à-dire* 2^{20} *octets, donc les adresses physique font* 20*bits.*

2. Quelle est la taille de l'adresse logique ?
Correction: *L'adresse est ici composée d'un bit indiquant si l'on accède à un segment global, de* 7*bits codant le numéro de segment (*$128 = 2^7$*) et de* 16*bits pour le décalage (*64*ko=* 2^{16}*) soit un total de* 24*bits pour l'adresse logique. Un processus peut donc en théorie utiliser un maximum de* 16*Mo de mémoire même s'il n'y a en réalité que* 1*Mo de mémoire dans le système.*

3. Quelle est la taille de la table des segment locale d'un processus ?
Correction: *Chaque ligne est composée de :*

1 *2bits pour la base. C'est un adresse physique sur* 20*bits mais les* 8*bits de poids faible sont forcément* 0 *et ne sont pas stockés ;*

1 6bits pour la limite, pour des segments de 64ko maximum ;
4 bits pour les bits de protection.
donc 32bits ou 4o par ligne. Il y a 128 lignes dans une table donc un total de 512o pour la table.

On suppose qu'un processus P1 est présent dans le système, on donne ci-dessous des extraits des tables des segments. Vous pouvez considérer que toutes les lignes qui ne sont pas présentes sont enièrement à zéro. (la colonne RWXS indique la valeur des 4bits en notation binaire)

Table locale de P1

Seg.	Base	Limite	RWXS
00	000	0000	0000
0C	020	4000	1010
13	1D4	0800	1000
21	000	0400	0000
2A	C70	D704	1100
48	1D4	0800	1101

Table globale

Seg.	Base	Limite	RWXS
21	000	0200	1100
2A	A5C	0100	1100
6C	032	1000	0010

4. Que se passe-t-il lorsque le processus réalise des accès en écriture aux adresses suivantes :2A04E5, A10386 et 0c1F5E ?

Correction: 2A04E5 *: Le bit de poids fort est à zéro, on utilisera donc la table locale, les 7bits suivants nous indiquent que l'on doit utiliser le segment* 2A *dont le bit de protection W est bien à 1 et le bit S est bien à 0. Le décalage est de* 04E5 *ce qui est bien inférieur à la limite du segment, l'accès est donc valide. Le calcul de l'adresse physique correspondante ce fait en prennant l'adresse de base à laquelle on concatène un octet nul :* C7000 *et on ajoute le décalage pour obtenir* C74E5.

A10386 *: Le bit de poids fort nous indique cette fois-ci un segment global, le* 21*. Celui-ci à une taille de* 0200 *qui est inférieure au décalage, il y a donc une erreur de segmentation. Attention, de ne pas utiliser le segment local* 21 *où l'accès aurait pus se faire car celui-ci est plus grand !*

0C1F5E *: Il s'agit ici du segment local* 0C*, le décalage est correct mais le processus n'a pas les droit en écriture sur ce segment (bit W à 0) et donc là aussi une erreur de segmentation est déclenchée.*

5. Où est probablement stocké le code du processus P1 ?

Correction: *Seuls les segments* 0C *de la table locale et* 6C *de la table globale on le bit d'exécution à 1, c'est donc nécessairement dans un de ces deux segments. Probablement dans le segment local car il est relativement peu pertinent de placer le code d'un processus dans un segment global.*

Les question suivantes sont plus générales et sont destinées à vous amener à réfléchir sur les possibilitées offertes par ce type de système de gestion de la mémoire. La conception de ces systèmes est toujours un compromis entre les contraintes pratiques et les fonctionalitées offertes.

6. Qu'y a-t-il de particulier avec les segments locaux 13 et 48 ? À quoi sert ce type de configuration ?

Correction: *Les deux segments correspondent à la même portion de la mémoire physique il est donc possible d'accéder à la même mémoire via deux adresses logiques différentes mais avec des bits de protection différents. La configuration utilisée ici permet au processus d'accéder en lecture uniquement à cette portion de la mémoire via le segment 13 tandis que l'OS peut y accéder en lecture et en écriture via le segment 48. Cela permet à l'OS de partager une partie de sa mémoire sans risque que le processus ne la modifie et sans avoir besoin dans faire une copie ailleurs en mémoire.*

7. Qu'y a-t-il de particulier avec le segment local 00 ? À quoi sert-il ?

Correction: *Il est de taille zéro et sans aucune permission. Cette configuration indique qu'un accès à l'adresse logique* 000000 *déclenchera forcément une erreur. Cette adresse est très souvent utilisée comme représentation du pointeur* NULL *et donc permet de repérer beaucoup d'erreur de programmation.*

8. En quoi ce système de gestion de la mémoire est-il particulièrement intéressant pour l'OS ?

Correction: *Il permet à l'OS de réserver quelques segments dans la table globale pour y placer son code et ses données. Les segments sont globaux donc quelque soit le processus actuellement en cours d'exécution ils sont accessibles sans qu'il soit nécessaire de reconfigurer la MMU et À l'aide du bit de protection S ils peuvent être rendus inaccessible aux processus de manière à garantir l'intégrité de l'OS.*

Lors d'une interruption aussi bien materielle que logicielle, l'OS n'est donc pas obligé de reconfigurer la MMU pour traiter l'interruption, il peut le faire avec la configuration du processus courant ce qui accélère nettement son traitement.

9. Le fabriquant de ce système souhaite produire une nouvelle version disposant de 4Mo de mémoire, quelle(s) modification(s) proposeriez-vous pour supporter cette quantité de mémoire physique ?

Correction: *Il est déjà possible pour un processus d'adresser une telle quantité de mémoire, il n'est donc pas nécessaire de faire de modification de ce côté. Le problème viens du fait que les adresses de base des segments ne permettent pas de couvrir plus de 1Mo de mémoire physique.*

Augmenter la taille de l'adresse de base dans les tables des segments est problématique car dans ce cas les ligne des table ferait plus de 32bits ce qui rendrait les accès complexes pour la MMU. Par contre, si l'on force les segments à ne commencer qu'à des adresses physiques multiples de 1024, ce sont les 10bits de poinds faible qui seront à zéro et non plus les 8bits de poids faible. Il est donc possible, dans le même espace, de stocker des adresses de base ayant 2bit de plus ce qui permet de couvrir 4Mo de mémoire physique.

Il est fort probable que ce soit d'ailleur la raison initiale de cette contrainte sur la position de départ des segments.