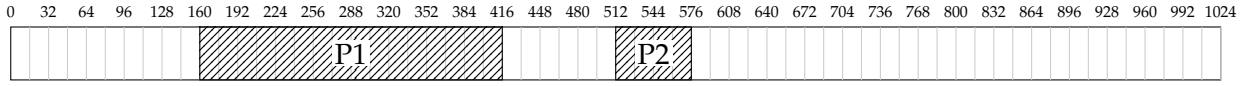
TD 05 – Allocation de la mémoire (RAM)

Exercice 1. *système à partitions variables*

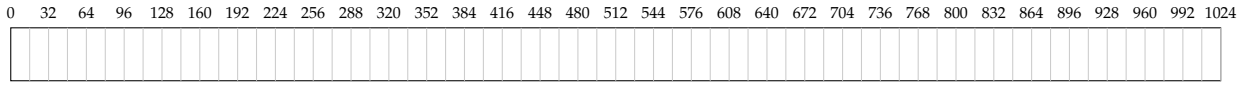
Nous allons gérer l'allocation d'une mémoire principale de 1 Gio (262 144 emplacements de 4 kio) avec le système des partitions variables. La mémoire est initialement dans l'état suivant :



1. Combien de bits sont nécessaires pour adresser les emplacements ?

Les réglettes sont graduées par 16 Mio, et les adresses seront codées sur 20 bits.

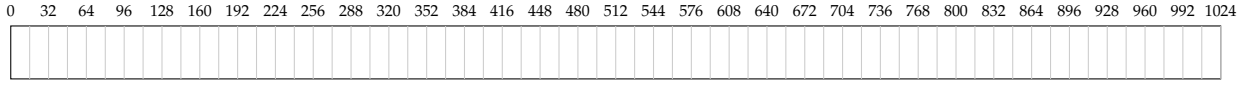
2. Parmi ces 20 bits, lesquels indiquent la graduation ? le décalage dans la graduation ?
3. Quelles sont les valeurs du registre de base et du registre limite pour P1 ?
4. Quelle est l'adresse physique correspondant à l'adresse logique `0x0CE83` de P1 ?
5. Quelle est l'adresse physique correspondant à l'adresse logique `0x01ADF` de P2 ?
6. Allouer 64 Mio de mémoire pour le programme P3, selon l'algorithme *best-fit*.



7. Quelle est l'adresse physique correspondant à l'adresse logique `0x0F2B7` de P3 ?

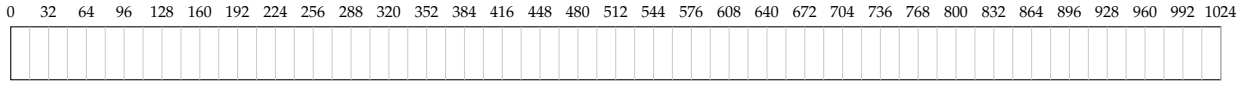
Le programme P1 est copié en ROM et enlevé de la RAM.

8. Allouer 416 Mio de mémoire pour le programme P4, selon l'algorithme *first-fit*.



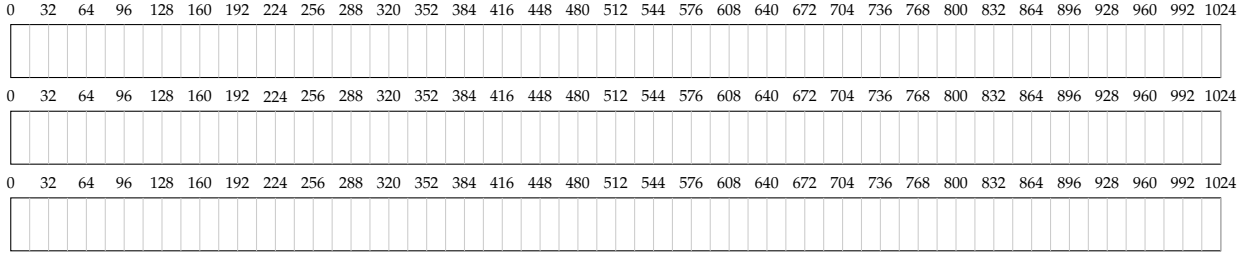
Le programme P5 demande 464 Mio de RAM. Pour les lui allouer, on veut déplacer P2…

9. Le système à partitions variables est-il assez flexible pour que l'algorithme de gestion des partitions soit autorisé à déplacer P2 (en cours d'exécution) à l'intérieur de la RAM ?
10. Si oui, déplacer P2 de 32 Mio sur la gauche et allouer P5.



11. Quelle est l'adresse physique correspondant à l'adresse logique `0x1CAFF` de P5 ?
12. Quelle est l'adresse physique correspondant à l'adresse logique `0x01A9D` de P2 ?

Quelques réglettes bonus…



Exercice 2. *pagination (de Fabien Rico)*

De façon similaire à la mémoire des INTEL PENTIUM, la Table 1 (en dernière page) représente une mémoire paginée :

- Une adresse correspond à un mot mémoire de 32 bits.
- Chaque page contient 8 mots mémoires (1 ligne).
- Il y a 2 niveaux d'indirection (table de répertoires de page et table de pages).
- Une adresse logique est adressée sur 9 bits, 3 pour le répertoire de page, 3 pour la page et 3 pour le décalage dans la page.
- Dans les tables de pages, pour chaque page, 7 bits sont utilisés pour détailler les propriétés de la page :
  1. pour signaler l'existence de la page
  2. pour signaler la présence de la page en mémoire principale
  3. pour signaler le droit d'écriture
  4. pour signaler le droit d'exécution
  5. pour signaler le « copy-on-write » [1]
  6. pour le bit d'accès
  7. pour le dirty bit
- Dans les tables de répertoire de page, seul le premier bit d'information est utilisé (celui de l'existence du répertoire correspondant).

On considère deux processus, la table des répertoires de pages du processus 1 est à l'adresse `0x01` et celle du processus 2 à l'adresse `0x0F`.

1. Quelle est la capacité d'adressage ?

2. Pour chacun des deux processus, que contient la case d'adresse `101111001` ?

3. Que se passe-t-il si le processus 1 essaye de lire la valeur de la case `101101101` ?

4. Même question si le processus 2 tente d'exécuter le code de l'adresse `000001011` ?

5. Que remarque-t-on pour l'adresse `101010000` du processus 1 et l'adresse `000101000` du processus 2 ? Dans quelle cas cela peut-il arriver ?

6. Mêmes questions pour l'adresse `000011111` des 2 processus. Que se passe-t'il si l'un des processus écrit à cette adresse ?

Exercice 3. *FIFO surprant (de Fabien Rico)*

Soit la chaîne de références $0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4$ et l'algorithme de remplacement FIFO.

1. Combien de défauts de page rencontrera-t-on si l'on a 3 cadres ?

2. Combien de défauts de page rencontrera-t-on si l'on a 4 cadres ?

Exercice 4. *Algorithme de remplacement (de Fabien Rico)*

1. Si l'algorithme FIFO est utilisé avec 4 cases mémoire et 8 pages, combien de défauts de page se produiront avec la chaîne de référence $0, 1, 7, 2, 3, 2, 7, 1, 0, 3$ si les 4 cases sont initialement vides ? Et avec l'algorithme LRU ? (Remplir le tableau suivant)

1. Mécanisme très intéressant que nous n'avons pas (encore) abordé en cours : quand un processus demande à copier une page, elle ne l'est physiquement que lorsqu'elle est modifiée pour la première fois. L'avantage : si au final elle n'est pas modifiée, elle ne sera jamais copiée.

Demandes de page	0	1	7	2	3	2	7	1	0	3
Pages chargées FIFO										
Défaut de page ?										
Pages chargées LRU										
Défaut de page ?										

Un ordinateur possède 4 cadres. Nous donnons ci-après le moment du chargement, le temps du dernier accès et les bits `R` (accès ou *read*) et `M` (modification ou *dirty bit*) pour chaque page (les temps sont donnés en tops d'horloge) :

Page	Chargement	Dernière référence	R	M
0	126	280	0	0
1	230	265	0	1
2	140	270	0	0
3	110	285	1	1

2. Quelle sera la page qui sera remplacée avec l'algorithme NRU [2] ? FIFO ? LRU ? Deuxième chance ?

Supposons que, pour un certain programme, la suite des références à la mémoire contienne des répétitions de longues séquences de numéros de pages suivies de temps à autre par un numéro aléatoire. Par exemple $0, 1, \cdots, 511, 431, 0, 1, \cdots, 511, 332, 0, 1, \cdots$ contient la répétition de la suite $0, 1, \cdots, 511$ suivie d'une référence aléatoire aux pages 431 et 332.

3. Pourquoi, dans le cas où l'allocation de pages est inférieure à la longueur de la séquence, les algorithmes de remplacement classique (LRU, FIFO, ...) fonctionnent-ils mal ?

4. Si on alloue à ce programme 500 cadres en mémoire principale, trouver un algorithme de remplacement qui fonctionnerait mieux que LRU et FIFO.

Exercice 5. *Ordre de lecture (de Fabien Rico)*

On utilise un tableau à 2 dimensions :

```
#define N 256
int X[N][N];
```

Sur une machine 32 bits, supposons qu'un système utilise des pages de 512 octets, et qu'il fourni à un processus 4 cadres pour assurer son fonctionnement. Le premier cadre contient le code qui se trouve toujours en mémoire. Les données vont et viennent sur les 3 autres cadres. En C les données d'un tableau à deux dimensions sont rangées suivant les lignes (`X[3][1]` suit `X[3][0]` en mémoire).

1. Pour initialiser le tableau, on peut utiliser l'un des deux codes suivants. Compter le nombre de défaut de pages générés par les 2 algorithmes.

```
for (int j = 0; j < N; j++)
  for (int i = 0; i < N; i++)
    X[i][j] = 0;
```

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    X[i][j] = 0;
```

2. Not Recently Used : la victime est choisie suivant les priorités suivantes : `R`=0 `M`=0 avant `R`=1 `M`=0 avant `R`=0 `M`=1 avant `R`=1 `M`=1.

mem	00	01	02	03	04	05	06	07
00	78	240	95	99	44	173	90	178
01	1100000.09	0000000.0B	0000000.07	0000000.02	0000000.11	1100000.1B	0000000.13	0000000.09
02	97	199	233	32	109	120	39	125
03	29	191	65	231	216	171	107	159
04	192	11	148	100	61	234	88	67
05	200	245	134	225	11	247	27	105
06	7	207	83	15	94	59	120	43
07	59	117	38	47	113	100	160	51
08	25	170	208	238	31	98	76	249
09	1101010.05	1101000.06	1100110.15	1110110.1F	0010100.F7	0010010.FF	0001100.66	0010000.BC
0A	211	225	15	216	15	98	192	204
0B	0000000.0E	0010100.1F	0001111.67	0000000.1F	0000000.03	0000000.1C	0000000.1C	1110000.10
0C	181	183	246	78	39	60	173	89
0D	11	216	239	140	21	196	202	116
0E	1101010.05	1101000.06	1100110.15	1110110.1F	0011000.6C	1110000.0C	0001010.06	1110000.17
0F	1100000.0E	0000000.07	0000000.15	0000000.10	0000000.15	1100000.0B	0000000.07	0000000.06
10	187	17	12	178	51	114	23	213
11	222	169	139	255	47	191	2	123
12	108	58	44	251	202	101	200	119
13	108	42	21	173	74	82	230	164
14	168	31	6	29	62	28	204	81
15	124	255	22	94	40	2	89	248
16	199	66	232	64	153	134	178	222
17	182	223	41	38	207	145	18	249
18	123	102	191	205	63	238	208	75
19	209	196	228	25	9	52	60	182
1A	201	163	201	157	95	81	182	64
1B	0001010.5F	0010100.1A	1110010.0C	0011000.A8	0000000.A7	0001111.19	0001010.06	1110011.13
1C	191	157	23	110	19	41	144	58
1D	247	210	33	31	167	113	70	94
1E	199	133	82	214	2	155	112	171
1F	0	0	0	0	0	0	0	0

Tab. 1 – Exemple de mémoire