TD 3
Mémoire virtuelle

EXERCICE 1 ▶ **Tables de pages**

On considère un système de mémoire virtuelle équipé d'une MMU. La capacité d'adressage du processeur est 32 bits. La taille des pages est 4 Kio.

1) De combien de pages est fait un espace virtuel s'il est utilisé intégralement ? Quelle est la taille des pointeurs ?

2) Quelle est la taille de la table des pages si elle est représentée de façon linéaire et si chaque entrée contient le numéro d'un cadre physique sur 48 bits, des droits sur 3 bits, un bit VALID, un bit DIRTY et 11 bits AGE ?

3) Quelle information permet de savoir si un accès à une page est valide ? Comment savoir quelles pages risquent d'être beaucoup utilisées dans le futur ? Quelle information permet de savoir si une page est présente en mémoire ? Quelle est la taille des adresses physiques ? Qu'en pensez-vous ?

4) Est-il envisageable de stocker la table de pages dans la MMU ? Est-il envisageable de la stocker en mémoire physique ? Que se passe-t-il si on a 200 processus ?

On considère maintenant une table des pages à 3 niveaux. Les niveaux 1 et 2 contiennent des tables de 4 Kio remplies de pointeurs (32bits) vers les niveaux suivants. Le niveau 3 contient des tableaux de 4 Kio contenant les entrées de la table linéaire sus-citée.

1) Comparez l'occupation de cette table à celle de la table linéaire sus-cité quand un processus n'utilise qu'1 Kio, 100 Kio, 10 Mio, 1 Gio puis tout l'espace disponible. Quid d'une table couvrant 1000 fois 1 Kio dispersés dans tout l'espace virtuel ?

EXERCICE 2 ▶ **Etats des entrées de la table de pages**

1) Décrivez l'état des bits de droits, VALID et DIRTY d'une entrée de la table de pages dans les cas suivants :

- Adresse invalide
- Adresse valide pointant vers page en lecture seule, présente en mémoire
- Adresse valide pointant vers page en écriture seule, présente en mémoire, et récemment modifiée
- Adresse valide pointant vers page en lecture-écriture, mais actuellement en copy-on-write
- Adresse valide pointant vers page absente en mémoire car swappée sur le disque

2) Dans quel cas le système va-t-il devoir rapatrier une page depuis le disque vers la mémoire physique ? Dans quel contexte l'opération est-elle effectuée ? Où sont stockées les informations nécessaires à localiser la page sur le disque ?

EXERCICE 3 ▶ **Mémoire paginée « à la Pentium »**

Exercice proposé par F. Rico

De façon similaire à la mémoire des processeurs INTEL PENTIUM, la table 3.1 représente une mémoire paginée :

- Chaque adresse est codée sur 9 bits.
- Une adresse de 9 bits correspond à 1 mot mémoire de 32 bits.
- Chaque page contient 8 mots mémoires (1 ligne du tableau 3.1).
- Il y a 2 niveaux d'indirection (table de répertoires de pages et table de pages).
- Une adresse logique est composée de 9 bits, de gauche à droite : 3 pour le répertoire de pages, 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
  3. pour signaler le droit d'écriture
  4. pour signaler le droit d'exécution
  5. pour signaler le « copy-on-write »
  6. pour le bit d'accès
  7. pour le dirty bit

— Dans les tables de répertoire de pages, seul le premier bit d'information est utilisé (celui de l'existence du répertoire correspondant).

On considère 2 processus, la table des répertoires de pages du processus 1 est à l'adresse `0o01` et celle du processus 2 à l'adresse `0o17`.

Dans cet exercice et dans le cadre du cours, on ignore les informations apportées par le bit "copy on write" et par le "dirty bit".

1) Quel est la capacité d'adressage ?

— "Chaque page contient 8 mots mémoire", donc 1 page = 8*32 bits = 32octets

— 1 table de pages = 8 pages (car 3 bits pour coder une page), cela nous donne une table occupe 256o

— la table de répertoire de pages = 8 tables de pages(même argument) = 2048o = 2kio

La taille de l'espace d'adressage est donc 2kio.

2) Pour chacun des deux processus, que contient la case d'adresse `0b101111001` ?

— Pour le proc 1, c'est l'adresse 23.1 et cela contient 42.

— Pour le proc 2, c'est l'adresse 20.1 et cela contient 17 (attention il y avait une erreur).

explication L'adresse en octal (3 bits donnent un chiffre de 0 à 7) est `o571`, cela signifie qu'on recherche « la case numéro 1 de la page numéro 7 du répertoire de page numéro 5 du processus ». Pour le processus 1, on lit :

— 01.5 (page physique 01, case 5) et on trouve 1100000.33 ce qui signifie que le bit d'existence (1er bit) est 1 et le répertoire de page est à l'adresse physique 33.

— 33.7 (page physique 33, case 7) = 1110011.23 (existence, présence, écriture, bit accès et ditry bit sont à 1), la page est à l'adresse physique 23.

— 23.1 = 42.

Avec un raisonnement identique, pour le second processus nous obtenons : 17.5 = 1100000.13 ↦ 13.7 = 1110000.20 ↦ 20.1 = 17 (existe, présent et modifiable).

3) Que se passe-t-il si le processus 1 essaye de lire la valeur de la case `0b101101101` ?

— $0b101101101 = 0o555$.

— 01.5 = 1100000.33 ↦ 33.5 = 0001111.31 n'existe pas (1er bit à 0).

Donc, la table de page n'existe pas, c'est un SEGFAULT.

4) Que remarque-t-on pour l'adresse 0b101010000 du processus 1 et l'adresse 0b000101000 du processus 2 ?

— Processus 1 : $0b101010000 = 0o520$ et 01.5 = 1100000.33 ↦ 33.2 = 1110010.14 ↦ 14.0 = 197

— Processus 2 : $0b000101000 = 0o050$ et 17.0 = 1100000.16 ↦ 16.5 = 1110000.14 ↦ 14.0 = 197

C'est la même adresse physique, c'est donc une page partagée et tout ce qui y sera écrit par l'un des processus pourra être lu par l'autre.

C'est un partage d'une page modifiable et non exécutable (pas de code donc). C'est donc une zone d'échange entre processus. Ce genre de partage peut venir d'une demande explicite (`shm_...`) ou découler de l'implémentation d'un canal de communication (un pipe par exemple).

adresse de la page	Décalage dans la page							
mem	0	1	2	3	4	5	6	7
00	70	48	22	142	32	126	215	128
01	1100000.11	0000000.13	0000000.07	0000000.02	0000000.21	1100000.33	0000000.23	0000000.11
02	107	63	71	241	171	91	93	223
03	29	167	27	139	154	87	59	79
04	33	54	197	180	13	112	71	168
05	130	15	233	70	8	206	76	139
06	178	227	4	159	116	23	58	200
07	204	28	123	120	61	173	51	11
10	185	63	13	130	24	18	117	87
11	1101010.05	1101000.06	1100110.25	1110110.37	0010100.57	0010010.77	0001100.66	0010000.74
12	157	115	237	19	159	219	30	187
13	0000000.16	0010100.37	0001111.67	0000000.37	0000000.03	0000000.34	0000000.34	1110000.20
14	197	248	206	145	47	28	197	249
15	129	126	122	137	205	70	220	127
16	1101010.05	1101000.06	1100110.25	1110110.37	0011000.46	1110000.14	0001010.06	1110000.27
17	1100000.16	0000000.07	0000000.25	0000000.20	0000000.25	1100000.13	0000000.07	0000000.06
20	122	17	153	217	242	151	150	209
21	205	201	173	237	183	208	31	78
22	147	148	236	235	243	118	195	249
23	145	42	235	41	148	181	30	83
24	193	36	2	240	167	207	147	192
25	69	69	216	223	163	41	58	141
26	41	93	176	16	99	10	178	207
27	223	135	55	26	205	211	248	24
30	193	26	8	3	42	191	181	232
31	165	162	122	104	67	191	78	123
32	219	55	208	116	148	253	55	143
33	0001010.57	0010100.32	1110010.14	0011000.31	0000000.63	0001111.31	0001010.06	1110011.23
34	191	142	215	171	241	121	112	180
35	1	94	177	206	225	148	86	2
36	111	5	242	120	32	44	143	201
37	0	0	0	0	0	0	0	0

FIGURE 3.1 – Exemple de mémoire