INTRODUCTION AUX SYSTEMES D'EXPLOITATION

TD6
Gestion de la mémoire

S O M M A I R E

1. Gestion de mémoire par va-et-vient

1.1. Présentation générale

Il s'agit de mettre en œuvre un mécanisme de va-et-vient (swapping) sur une machine disposant d'un adressage permettant la réimplantation dynamique de programme (par exemple une machine dotée de registres de base).
Il s'agit d'un cas d'école et non pas un cas réel.

Espace mémoire (ou espace d'adressage) d'un processus.

On fait les hypothèse suivantes :

- La code, les données ainsi que la pile d'exécution d'un processus sont supposés être implantés dans une seule zone mémoire physique. Ceci permet de faciliter l'opération d'entrée/sortie lors du va-et-vient.
- Le code des processus n'est pas partagé.
- La taille d'un processus ne change pas au cours de sa vie.

Représentation et synchronisation des processus.

Les états possibles pour un processus sont les suivants :

- Vis-à-vis de l'unité centrale : ELU, PRET ou BLOQUE.
- Vis-à-vis de la mémoire : RESIDENT ou NON RESIDENT.

Une entrée dans la table des processus correspond à la structure suivante :

```
typedef struct processus {
        char    *pt_mot_etat ;  /* Zone de sauvegarde des registres */
        int     pid ;           /* Identifiant du processus */
        int     taille ;        /* Taille du processus en octets */
        int     etat_uc ;       /* Etat du processus vis-à-vis de l'UC */
        int     etat_mem ;      /* Etat du processus vis-à-vis de la mémoire */
        char    *pt_mem ;       /* Adresse mémoire du processus */
        char    *pt_disque ;    /* Adresse de l'image du processus sur disque */
        time    d_swap ;        /* Date de dernier swapin ou swapout */
        int     priorité ;      /* Priorité du processus */
        int     proc_suiv ;     /* Indice du processus suivant en terme d'éligibilité */
} PROCESSUS, * PT_PROCESSUS ;
```

Le noyau contient les données globales suivantes :

```
PROCESSUS   processus [NB_MAX_PROC] ; /* La table des processus */
int         proc_elu ;                /* indice du processus élu */
```

Un processus utilisateur peut perdre l'UC :

- à la suite d'un P bloquant,
- à la fin de sa tranche de temps,
- à l'activation d'un processus plus prioritaire.

On suppose que tous les processus existent au démarrage du système et qu'il ne meurent pas.

1.2. Exercice N°1

Dans quelles conditions logiques l'image mémoire (ou espace d'adressage) d'un processus peut-elle être rentrée en mémoire physique (swapin) ?

Dans quelles conditions logiques l'image mémoire (ou espace d'adressage) d'un processus peut-elle être vidée sur disque (swapout) ?

1.3. Exercice N°2

Quels sont les autres critères dont on doit tenir compte dans le choix des processus à vider ou à charger ?

1.4. Présentation du mécanisme de va-et_vient

Les échanges entre la mémoire physique et la mémoire secondaire sont assurés par un processus système le swappeur toujours résident en mémoire.
Son principe de fonctionnement est le suivant.

```
main () {
        int     noProcIn, noProcOut ;
        sema    reveil_swap = 0 ;
        int     place_en_memoire = 1 ;
        while (1) {
                while (place_en_memoire) {
                        noProcIn = rechercherProcessusArentrer () ;
                        if (noProcIn < 0) {
                                etat_swap = INACTIF ;
                                P (reveil_swap) ;
                        }
                        else {
                                if (swapin (noProcIn) != 0 ) {
                                        place_en_memoire = 0 ;
                                }
                        }
                }
                while ( !place_en_memoire) {
                        noProcOut = rechercherProcessusAretirer () ;
                        if (noProcOut < 0) {
                                etat_swap = INACTIF ;
                                P (reveil_swap) ;
                        }
                        else {
                                swapout (noProcOut) ;
                                place_en_memoire = 1 ;
                        }
                }
        }
}
```

On considère que celui-ci est réveillé à intervalles réguliers grâce à une interruption d'horloge dont le traitement est le suivant.

```
Interuption_horloge_swap :

        if (etat_swap = INACTIF) {
                etat_swap = ACTIF ;
                V (reveil_swap) ;
        }
```

1.5. Exercice N°3

Décrire le comportement du swappeur.

1.6. Allocation/Libération de la mémoire

On suppose réalisée une gestion de l'espace mémoire par zones de tailles quelconque.
On dispose des 2 fonctions suivantes :

char * allouerZone (int taille)
taille taille de la zone à allouer
code retour pointeur sur le premier octet de la zone mémoire allouée ou NULL si pas suffisamment de mémoire.

int libererZone (int taille, char *pt_mem)
taille taille de la zone à libérer
pt_mem pointeur sur le premier octet de la zone mémoire à libérer
code retour 0 si libération réussie ou –1 en cas d'échec.

1.7. Les Entrée/Sorties sur disque

On dispose des 2 fonctions suivantes pour réaliser des E/S sorties synchrones avec le disque :

int lireDisque (char * pt_disque, char *pt_mem, int nb_octet)
nb_octet nbre d'octets à lire
pt_disque adresse disque
pt_mem adresse mémoire
code retour 0 si lecture réussie ou –1 en cas d'échec.

int ecrireDisque (char * pt_disque, char *pt_mem, int nb_octet)
nb_octet nbre d'octets à écrire sur le disque
pt_disque adresse disque
pt_mem adresse mémoire
code retour 0 si écriture réussie ou –1 en cas d'échec.

1.8. Exercice N°4

En supposant écrites les procédures :
RechercherProcessusArentrer ()
RechercherProcessusAvider ()
écrire les fonctions **swapin** et **swapout** du swappeur à l'aide des primitives présentées précédemment.

2.La pagination

2.1. Présentation

On dispose d'un système de pagination avec les caractéristiques suivantes :

- Taille des pages : 512 octets
- Nombre de pages réelles : 16

Le processus P1 dispose de la table des pages suivantes.

N° page virtuelle	N° page réelle	Présent en mémoire
0	2	1
1	3	1
2	0	1
3		0
4	1	1
5		0

2.2. Exercice N°5

a) Quelle structure d'adressage peut-on imaginer ?

b) Quelle est la taille de l'espace d'adressage du processus P1 ?

c) Quelle est la taille de mémoire réelle occupée actuellement par P1 ?

2.3. Exercice N°6

Soit l'adresse virtuelle 2058 du processus P1.

a) Exprimer celle-ci sous la forme (N° de page, Déplacement) dans l'espace d'adressage virtuelle de P1.

b) Exprimer l'adresse réelle correspondante sous la forme (N° de page, Déplacement).

c) Quelle est la valeur de cette adresse réelle ?

3.La segmentation

On considère la table de segments suivante :

Segments	Base	longueurs
0	540	234
1	1254	128
2	54	328
3	2048	1024
4	976	200

Calculez les adresses réelles correspondantes aux adresses relatives au segments plus haut : (0, 128), (1, 99), (1, 100), (2, 465), (3, 888), (4, 100), (4, 344) .

4.L’écroulement (Thrashing)

La mémoire centrale comprend 25 cadres. 5 processus s’exécutent simultanément. Chaque processus dispose de 5 cadres et produit la suite de références (en numéros de pages) :

Processus i :
0+5*i, 1+5*i, 2+5*i, 3+5*i, 4+5*i, 0+5*i, 1+5*i… pour i=0..4.

L’algorithme de remplacement utilisé est LRU et la mémoire est initialement vide.

1- Quel est le taux de défauts de page provoqué par cette exécution ?
2- On introduit un 6ème processus, analogue aux 5 autres quant à la suite de références produites (pour i=5) ce qui fait que chaque processus dispose maintenant de 4 cadres. Calculer le nouveau taux de défaut de page.