Université Côte d'Azur
Faculté des Sciences - Département d'Informatique

Licence d'Informatique L2
Introduction aux Systèmes et Réseaux

TD n ° 9 (Corrigé) : Gestion de la mémoire - 2

1 Transformation des adresses paginées

Prenons un système de mémoire paginée (serveur intel 32 bits avec drapeau PAE (*physical address extension* activé) dans lequel les adresses sont codées sur 32 bits et chaque page mesure 2^{12} octets. Le système associe à chaque processus une table des pages comme dans l'exemple ci-dessous. :

	...
PTBR →	<drapeaux et protection, 100>
	<drapeaux et protection, 120>
	<drapeaux et protection, 10>
	<drapeaux et protection, 30>
	...

Chaque entrée de la table des pages est un couple `<drapeaux et protection (8 bits), numéro de page physique (24 bits)>`. Le PTBR (page-table base register) indique le début de la table de pages pour le processus considéré. Dans ces conditions,

1. quelle est la taille maximum de la mémoire logique d'un processus et la taille maximum de la mémoire physique ?
   Maximum logique = $2^{20} \times 2^{12}$ *soit 4 Go. Avec des adresses sur 32 bits ce n'est pas étonnant.*
   Maximum physique = $2^{24} \times 2^{12}$ *soit 64 Go. Dans ce type d'architecture il faut plusieurs processus pour exploiter une mémoire centrale supérieure à 4 Go.*
   Si la taille de la page était de 1 Ko, le maximum logique ne change pas et le maximum physique passerait à 16 Go.
2. quelles sont les adresses physiques qui correspondent aux adresses logiques 9000, 3000, 12288 et 4100 ?
   numéro de page = logical // pagesize
   offset = logical % pagesize
   le numéro de cadre est donné
   *adresse physique linéaire = numero de cadre * pagesize + offset*

adr. log. linéaire	couple log.	couple phy.	adr. phy. linéaire
9000	(2, 808)	(10, 808)	41768
3000	(0, 3000)	(100, 3000)	412600
12288	(3, 0)	(30, 0)	122880
4100	(1, 4)	(120, 4)	491524

2 Les algorithmes de remplacement

En cas de défaut de page dans un système paginé, les algorithmes de remplacement doivent déterminer une page physique victime qui sera sauvée sur disque (si nécessaire) et remplacée par la page virtuelle que l'on désire consulter. Il existe principalement quatre stratégies pour choisir la page victime :

1. Algorithme FIFO : la victime est la page physique la plus anciennement chargée.
2. Algorithme LRU (Least Recently Used).
3. Algorithme FINUFO (First In Not Used, First Out) ou algorithme de la 2ème chance.
4. Algorithme LFU (Least Frequently Used).

On vous donne deux listes `dpp` et `dpv` de taille `NB_PP` (nombre de pages physiques) et `NB_PV` (nombre de pages physiques). Les classes décrivant les pages physiques et virtuelles sont décrites ci-dessous.

```
class DPV:
    presente = False # page physique associée ?
    npp = None       # numéro de page physique associée

class DPP:
    libre = False  # non utilisée par un autre processus ?
    npv = None       # page virtuelle associée
    modif = False    # accédée en écriture ?
    info = 0         # usage selon l'algo choisi...
```

2.1 Transformation des adresses

À l'aide de ces déclarations, il est possible d'écrire la fonction `npv2npp(npv)` qui détermine un numéro de page physique (`npp`) à partir d'un numéro de page virtuelle (`npv`). Elle utilise la fonction `defaut_de_page(npv)` qui fera le gros du travail en traitant l'apparition d'un défaut de page sur la page virtuelle `npv` en chargeant une page physique. `npv2npp` se borne donc à tester si on doit faire un défaut de page, puis à modifier éventuellement certains champs de la page physique correspondant à la page logique.

Note : il ne faut pas oublier que cette opération est matérielle, donc pas de gros algorithme coûteux en temps et/ou en silicium.

```
def npv2npp(npv):
    if not dpv[npv].presente
        defaut_de_page(npv)
    npp = dpv[npv].npp
    <1> # modifier éventuellement le champ "info" de la page
    # si l'accès est en écriture alors:
     dpp[npp].modif = True
    return npp
```

La ligne indiquée `<1>` sera spécifique à l'algorithme de remplacement choisi et aura pour but de lire/écrire la champ `info` d'une page physique.

2.2 Traitement du défaut de page

Écrivez la routine `defaut_de_page(npv)` en utilisant les fonctions et variables ci-dessous.

`sauver_page(npp, npv)` # mém. centrale → disque
`charger_page(npv, npp)` # mém. centrale ← disque
`choisir_victime()` # choisir une page physique victime
`nb_pp_occupees` # variable globale

Son rôle est de :

1. libérer une page physique, ce qui revient à :
   (a) si au moins une page physique n'est pas occupée, se placer sur la 1ère page physique libre, modifier le champ `libre` de la page physique choisie, et incrémenter le nombre de pages physiques occupées.
   (b) sinon, la fonction `choisir_victime()` est appelée et retourne la page physique à évincer, le champ `presente` de la victime passe à faux. Si celle-ci a été modifiée, la sauvegarder sur disque.
2. charger la page physique associée à la page virtuelle depuis le disque.
3. initialiser/modifier les champs de la page virtuelle et de la page physique qui le nécessitent.

```
def liberer_pp():
    global nb_pp_occupees
    if nb_pp_occupees < NB_PP:
        npp = 0
        while not dpp[npp].libre:
            npp += 1
        nb_pp_occupees += 1
        dpp[npp].libre = False
    else:
        npp = choisir_victime()
        vict = dpp[npp].npv
        dpv[vict].presente = False
        if dpp[npp].modif:
            sauver_page(npp, vict)
    return npp

def defaut_de_page(npv):
    npp = liberer_pp()
    charger_page(npv, npp)
    dpv[npv].presente = True
    dpv[npv].npp = npp
    dpp[npp].npv = npv
    dpp[npp].modif = False
    dpp[npp].info = 0
```

2.3 Choix de la victime

La fonction `choisir_victime()` choisit la page physique victime en cas de défaut de page. L'implémentation de l'algorithme FIFO vous est donné. On vous demande de programmer cette fonction pour les trois autres algorithmes décrits précédemment.
Note :

- Pour la version FIFO, utiliser une variable globale victime qui donne la prochaine page physique victime.
- Pour la version FINUFO, utiliser une variable globale finufo qui donne la dernière page physique victime.

Stratégie FIFO :

```
vict_fifo = 0
def choisir_victime_FIFO():
    global vict_fifo
    return vict_fifo++ % NB_PP
```

Dans ce cas le morceau de code `<1>` est vide et le champ info de la classe DPP est inutile.
Stratégie FINUFO :

```
finufo = 0
def choisir_victime_FINUFO():
    while dpp[finufo % NB_PP].info == 1:
        dpp[finufo % NB_PP].info = 0
        finufo += 1
    return finufo % NB_PP
```

Dans ce cas le morceau de code `<1>` devient `dpp[npp].info = 1`. À chaque accès (écriture ou lecture) le bit finufo est forcé à 1.
Stratégie LRU :

```
horloge = 0
def choisir_victime_LRU():
    min = dpp[0].info
    victime = 0
    for i in range(1, NB_PP):
        if dpp[i].info < min or (dpp[i].info == min and not dpp[i].modif):
            min = dpp[i].info
            victime = i
    return victime
```

Dans ce cas le morceau de code `<1>` devient `dpp[npp].info = horloge; horloge += 1`. A chaque accès (écriture ou lecture) on note la date.
Stratégie LFU :

```
def choisir_victime_LFU():
    victime = 0
    nb_ref = dpp[0].info
```

```
    for i in range(1, NB_PP):
        if dpp[i].info < nb_ref or (dpp[i].info == nb_ref and not dpp[i].modif):
            nb_ref = dpp[i].info
            victime = i
    return victime
```

Dans ce cas le morceau de code `<1>` *devient* `dpp[npp].info += 1`*. À chaque accès (écriture ou lecture) on incrémente le nombre d'utilisations.*