ED 2 Systèmes

Exercice 1 : Mécanisme de gestion d'une mémoire virtuelle paginée

Question 1

bit V sert à générer un défaut de page si la page n'est pas présente en mémoire

bit M sert lors d'un remplacement de page, à éviter la recopie sur disque, d'une page qui n'a pas été modifiée

bit A sert à l'algorithme de remplacement de page type algorithme de l'horloge ou seconde chance.

Protec indique les droits d'accès à la page : lecture, écriture, exécution.

N° de case indique le numéro de case de mémoire physique si la page est présente en mémoire centrale.

Question 2

Voir également le cours

Structure des informations

```
type adresse : article
                npage : entier;
                depl : entier;
        fin article;
type defpage : article
                V : booleen;
                M : booleen;
                A : booleen;
                -- P :  protection
                NC : entier; numero de case
        fin article;
type instruction : (ecriture, autre);
type Tabpage : array (0..N-1) de defpage;  -- N taille d'une table de pages dans l'exemple 2**24
procedure decodage (AR : out adresse; AV : in adresse; I : in instruction)
P : defpage;
debut
        P:= Tabpage(AV.npage);
```

si non P.V alors activer le processus remplacement de page; attendre; fin si;

P.A:=1; si I=ecriture alors P.M:=vrai; fin si;

AR.depl := AV.depl; AR.npage:=P.NC;

fin decodage;

Application :

Un processus référence l'adresse virtuelle 10/3000
l'entrée 10 de la table des pages indique V=1, N°de case=20
l'adresse physique correspondant à l'adresse virtuelle 10/3000 est 20/3000.

Le processus référence maintenant l'adresse virtuelle 15/2500
l'entrée 15 de la table des pages indique V=0, N°de case=12.

Pour l'adresse virtuelle 15/2500, la page n'est pas présente en mémoire : il y a défaut de page et le N°de case n' pas de signification. Le système doit rechercher une case disponible et charger la page en défaut dans cette case.

Traitement d'un défaut de page

1 Recherche de l'adresse sur disque de la page en défaut

2 Trouver une case libre

a Si case libre alors l'utiliser

b Sinon, utiliser un algorithme de remplacement de page pour sélectionner une page victime

c Ecrire la page victime sur disque; modifier les tables de pages et de cases. Amélioration : ne ré-écrire la page que si elle a été modifiée depuis son chargement.

3 Charger en mémoire la page en défaut dans la case libre.

Question 3

La taille de la table des pages par processus est a priori 2**24 prohibitif !

Quelques solutions pour réduire la table des pages :

- Pagination de la table des pages

- Séparer l'espace virtuel de l'utilisateur en espace propre et espace système : toutes les procédures système utilisées par le processus sont dans l'espace du système et partagées. Enfin, la taille de la table des pages peut être calculée, en fin d'édition de liens par exemple, et on n'alloue que la place nécessaire...

- Utiliser une table des pages inverses.

- Enfin, dans tous les cas on peut utiliser une mémoire associative.

On pourra éventuellement étudier un compromis entre taille d'une page et nombre de pages.

Exercice 2 : Gestion de fichiers UNIX

1. Lors de la première demande de lecture, le descripteur de fichier contient en TABDIRECT[0] le numéro du premier bloc de données. Il faut donc le lire, et transférer les 256 premiers octets (disons ceux de numéro 0 à 255) de ce bloc dans une zone du programme. Lors de la deuxième demande, il s'agit du même bloc, mais comme il n'y a pas de conservation dans un tampon des blocs, il faut le relire, et délivrer les octets 256 à 511. Pour la cinquième demande, il s'agit des octets 0 à 255 du deuxième bloc de données, c'est-à-dire, le bloc de numéro TABDIRECT[1].

2. Les octets de la 41^{ième} demande sont situés dans le 11^{ième} bloc de données. Il faut donc lire le bloc INDIRECT_1 pour connaître le numéro du bloc de données qui nous intéresse (premier numéro de bloc dans ce bloc), et pouvoir lire celui-ci. Les octets de la 45^{ième} demande sont situés dans le bloc de données. Il faut donc aussi lire le bloc INDIRECT_1 pour connaître le numéro du bloc 12 de données qui nous intéresse (deuxième numéro de bloc dans ce bloc), et pouvoir lire celui-ci. Il s'ensuit que chaque demande de lecture, à partir de maintenant entraînera deux accès disque.

3. Les octets de la 1065^{ième} demande sont situés dans le 266^{ième} bloc de données. Il faut donc lire le bloc INDIRECT_2, puis celui dont le numéro est le premier dans le bloc venant d'être lu, pour avoir le numéro du bloc de données qui nous intéresse (premier numéro de bloc dans celui venant d'être lu), et lire enfin le bloc de données. Les octets de la 1066^{ième} demande sont situés également dans le 266^{ième} bloc de données. En l'absence de mémorisation, il faudra relire les mêmes blocs que précédemment pour satisfaire la demande. On peut donc en conclure que dorénavant, 3 accès disque seront nécessaires pour satisfaire chaque demande.

4. Les octets de la 2089^{ième} demande sont situés dans le 522^{ième} bloc de données. Il faut donc lire le bloc INDIRECT_2, puis celui dont le numéro est le deuxième dans le bloc venant d'être lu, pour avoir le numéro du bloc de données qui nous intéresse (premier numéro de bloc dans celui venant d'être lu), et lire enfin le bloc de données. Les octets de la 2090^{ième} demande sont situés également dans le 522^{ième} bloc de données. En l'absence de mémorisation, il faudra relire les mêmes blocs que précédemment pour satisfaire la demande. On peut donc en conclure que 3 accès disque sont toujours nécessaires pour satisfaire chaque demande.

5. Les 40 premières demandes de lecture demanderont chacune un accès disque. Les 1024 suivantes en demanderont chacune deux. Les demandes restantes (32768 - 1024 - 40 = 31704) en demanderont chacune trois. On a donc un total de 97200 accès disque, et un temps d'attente d'entrées-sorties de 3888 secondes, soit un peu plus d'une heure. Notons que l'indirection de niveau 3 n'est pas utilisée puisque 2 niveaux permettent de mémoriser 65 Mo, alors que nous n'en avons que 8 Mo.