

QCM Moodle IF-3-SYS de contrôle continu, avril 2022

Pour chaque affirmation ci-dessous, indiquez si elle est correcte.

- L'appel système `mmap()` permet d'éviter le problème des défauts de page.
- Quand on invoque `mmap()`, le noyau nous alloue toujours un nombre *entier* de pages.
- L'adresse retournée par `mmap()` est *toujours* une adresse virtuelle.
- Quand on invoque `mmap()`, le noyau modifie la *table de pagination* de notre processus.

On s'intéresse à un système avec mémoire virtuelle: la taille des pages est de 1024 octets, et l'adressage se fait sur 32 bits. Combien de bits sont nécessaires pour encoder un numéro de page ?

Réponse :

Comment s'appelle l'évènement qui se produit lorsque le programme accède à une donnée mais que celle-ci a été déchargée de la RAM vers le swap ?

Veuillez choisir une réponse :

- une faute de page
- une erreur de segmentation
- une commutation de contexte
- une préemption

On s'intéresse au fragment de code ci-dessous:

```
r = fork();
if ( r != 0 )
{
    v = v + 5;
}
else
{
    v = v - 5;
}
printf("%d,%d\n", v, &v);
```

À la dernière ligne, chaque processus affiche deux nombres: on notera P_a , P_b les deux nombres affichés par le parent, et E_a , E_b ceux affichés par l'enfant. Pour chaque affirmation ci-dessous, indiquez si elle est correcte.

- $P_a == E_a + 10$
- $P_a == E_a + 5$
- $P_a == P_b$
- $P_b == E_b$

Dans le TP sur la mémoire virtuelle, on a utilisé `mmap()` pour réimplémenter la commande unix `cat`. Mais avec quelle(s) option(s) faut-il invoquer `mmap()` dans ce cas d'usage ? Pour chaque proposition ci-dessous, indiquez si elle est vraiment *obligatoire* dans cet exercice. Respectivement, ne cochez pas la case si vous pensez que le programme marcherait quand même sans cette option.

- `MAP_ANONYMOUS`
- `MAP_FILE`
- `MAP_PRIVATE`
- `MAP_SHARED`

Certaines sections d'un processus ne sont pas initialisées avec le contenu du fichier exécutable, mais sont créées à la volée par le noyau. De quelles sections s'agit-il ?

- .text
- .data
- .heap
- .stack

On s'intéresse dans cette question à un allocateur dynamique. L'application fait une requête `malloc(N)`. Pour chaque affirmation ci-dessous, indiquez si elle est correcte.

- L'allocateur a le droit de réserver un bloc de taille strictement supérieure à N.
- L'allocateur a le droit de réserver plusieurs blocs distincts, pour une taille totale de N.
- L'allocateur a le droit d'attendre qu'un bloc se libère pour ensuite le réserver.
- L'allocateur a le droit de réserver un bloc de taille strictement inférieure à N.

Le programme ci-dessous alloue plusieurs tableaux. Dans quelle section de l'espace d'adressage sera placé le tableau `tabA` ?

```
int tabA[]={1,2,3,4};
int main(void)
{
    int *tabB = malloc(100 * sizeof(int));
    for(int i=0; i<100; i++)
    {
        tabB[i] = tabA[i % 4];
    }
    return 0;
}
```

- .data
- .text
- .heap
- .stack

On suppose dans cette question un allocateur dynamique best-fit avec une freelist composée initialement de deux blocs libres de taille 100 et 300 (chaînés dans cet ordre). Pour simplifier, on suppose qu'un bloc libre de taille N peut servir à allouer une zone de taille N (ce qui revient à négliger l'espace occupé par les méta-données) et que l'allocateur peut toujours "découper" les blocs sans perte de place.

L'application demande successivement à allouer des zones de tailles 50, puis 200, puis 50, puis 100.

On se demande si l'allocateur parviendra à satisfaire toutes ces requêtes avec l'espace libre initial (répondez "vrai") ou bien s'il va devoir faire appel au noyau pour demander à agrandir le tas (répondez "faux").

Sélectionnez une réponse :

- Vrai
- Faux

On suppose dans cette question un CPU calculant sur 4 bits: tous les nombres sont représentés sur 4 bits, les calculs sont tronqués à 4 bits, etc. Pour chacune des égalités ci-dessous, indiquez si elle est correcte.

- $0xC + 3 == 0xC \mid \sim 0xC$
- $\sim 0xA + \sim 0xA == 0xA$
- $0xB \& \sim 3 == \sim 7$
- $7 \mid 3 == 5 + 5$