QCM2: Contrôle continu IF-3-SYS d'avril 2024

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

- [x] À chaque appel système `fork()` (cf chap 1) le noyau crée une nouvelle table de pagination (cf chap 3). ✔
- [x] En termes d'ordonnancement (cf chap 2) une "faute de page" (cf chap 3) est équivalente à une "IO burst" ✔
- [x] En termes d'ordonnancement (cf chap 2) un appel système `sleep()` (cf chap 2) est équivalent à une "IO burst" ✔
- [ ] Les opérations de memory-mapped input/output (cf chap 1) utilisent des adresses virtuelles (cf chap 3)

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 `Pa, Pb` les deux nombres affichés par le parent, et `Ea, Eb` ceux affichés par l'enfant. Pour chaque affirmation ci-dessous, indiquez si elle est correcte.

- [ ] Pa == Pb
- [ ] Pa == Ea + 5
- [x] Pb == Eb ✔
- [x] Pa == Ea + 10 ✔

On suppose dans cette question un système avec des adresses virtuelles sur 16 bits, dont 7 bits pour le numéro de page. Si l'utilisateur fait un appel `mmap(5000)`, combien de pages lui seront allouées par le noyau ?

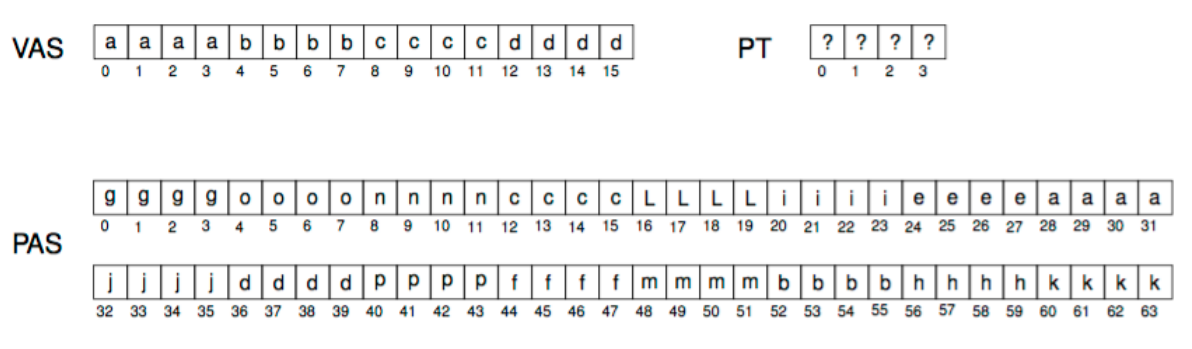
Réponse : 10 ✔

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

- [x] L'adresse physique d'une même page virtuelle peut changer au cours du temps. ✔
- [x] En général, l'espace d'adressage virtuel comporte de nombreuses pages invalides. ✔
- [x] La taille des pages virtuelles est toujours la même que la taille des pages physiques. ✔
- [ ] En général, l'espace d'adressage virtuel est plus petit que la mémoire physique.

Le diagramme ci-dessous illustre le VAS d'un processus (16 octets) ainsi que le contenu de la mémoire principale (64 octets). La taille de page est de 4 octets. Remplissez la table de pagination de ce processus (i.e. les points d'interrogation sur le diagramme) pour que la traduction d'adresses soit correctement réalisée:

PT 0: 7 ✔ 1: 13 ✔ 2: 3 ✔ 3: 9 ✔



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

- [x] Lorsqu'on appelle `free()`, l'allocateur a le droit de fusionner le bloc de mémoire désalloué avec d'autres blocs libres. ✔
- [ ] Lorsqu'on appelle `free()`, l'allocateur restitue le bloc de mémoire désalloué pour qu'il soit disponible pour les autres applications.
- [ ] Lorsqu'on appelle `malloc()` pour une variable locale, le nouveau bloc de mémoire est alloué sur la pile.
- [ ] Lorsqu'on appelle `malloc()`, l'allocateur efface le contenu du bloc de mémoire choisi avant de nous le retourner.

On s'intéresse dans cette question à la notion de "pile d'exécution" d'un programme. Pour chaque affirmation ci-dessous, indiquez si elle est correcte.

- [x] La pile d'exécution est utile pour le stockage des variables locales du programme. ✔
- [x] La pile d'exécution est utile pour offrir une forme restreinte d'allocation dynamique. ✔
- [ ] La pile d'exécution est utile pour le stockage des instructions du programme.
- [ ] Les accès mémoire vers la pile d'exécution sont plus performants car ils n'impliquent pas la MMU.

Dans cette question on s'intéresse à un gestionnaire de mémoire dynamique avec une *freelist* composée initialement de trois blocs libres de taille 250, 200, et
150 (chaînés dans cet ordre). Pour simplifier, on suppose ici 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 fait trois requêtes d'allocation successives, pour des tailles de 200, puis 150 puis 50. Ces trois requêtes réussissent sans qu'il soit nécessaire d'agrandir le tas.

Mais quelle stratégie cet allocateur applique-t-il pour choisir les blocs ? Pour chaque proposition ci-dessous, indiquez si elle est compatible avec les hypothèses.

- [x] stratégie d'allocation "first-fit" ✔
- [x] stratégie d'allocation "best-fit" ✔
- [x] stratégie d'allocation "worst-fit" ✔

Les affirmations ci-dessous portent sur notre TP4 (allocation dynamique). Pour chaque proposition, indiquez si elle est correcte.

- [x] Dans le TP4, les tailles mémoire sont toujours exprimées en nombre d'octets. ✔
- [x] Dans le TP4, trois blocs de mémoire dont les en-têtes auraient pour valeurs 0x50, 0x53, et 0x57 seraient tous les trois de même taille. ✔
- [x] Dans le TP4, les adresses retournées par `mem_alloc()` sont toujours des adresses virtuelles. ✔
- [ ] Dans le TP4, tant qu'on n'a pas implémenté la découpe des blocs (partie 3) il ne peut pas y avoir de problème de fragmentation du tas.

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.

- [x] `0xC + 3 == 0xC | ~0xC` ✔
- [x] `0xB & ~3 == ~7` ✔
- [ ] `7 | 3 == 5 + 5`
- [x] `~0xA + ~0xA == 0xA` ✔