Prénom Nom : _____ Groupe : _____

R3_05 - Programmation système - **1h45**

Formulaire A4 manuscrit recto-verso autorisé

1. Pour chacune des affirmations ci-dessous, entourez V si elle est correcte ou entourez F si elle est incorrecte et/ou absurde.

   V F L'appel `exec()` permet de créer un nouveau processus pour exécuter un nouveau programme.

   V F À chaque appel système `fork()` le noyau crée une nouvelle table de pagination.

   V F L'adresse physique d'une même page virtuelle peut changer au cours du temps.

   V F La pile d'exécution est utile pour le stockage des variables locales du programme.

   V F Les accès mémoire vers la pile d'exécution sont plus rapides car ils n'impliquent pas la MMU.

   V F L'appel `mmap()` renvoie une adresse physique.

   V F Chaque inoeud (`ext2`) contient le nom du fichier correspondant.

   V F Chaque inoeud (`ext2`) contient les numéros des blocs de données du fichier correspondant.

2. On s'intéresse dans cette partie à un système de gestion de fichiers basé sur une *File Allocation Table*. La FAT est un tableau d'entiers avec une case pour chaque bloc du disque :
   - Si le bloc $n°i$ est libre, alors `fat[i] = FAT_FREE = -2`
   - Si le bloc $n°i$ est le dernier bloc d'un fichier, alors `fat[i] = FAT_EOF = -1`.
   - Sinon, `fat[i]` est positif ou nul, et indique le numéro du bloc suivant dans le même fichier.

   Dans toutes les questions suivantes, vous pourrez compter sur les déclarations ci-dessous :

```
enum { FAT_EOF = -1, FAT_FREE = -2};
#define NB_BLOCKS ...
int fat[NB_BLOCKS] = { ... };
```

   (a) Voici un exemple de FAT avec un disque formaté en 16 blocs.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
fat[i]	-1	-2	-2	-2	-1	15	-2	-2	10	12	-1	-2	4	-2	-2	8

   Dans cet exemple, combien y a-t-il de fichiers, et quels sont les blocs qui composent chaque fichier ?

   (b) Écrivez une fonction `int count_free_blocks_at(int pos)` qui parcourt la FAT et renvoie le nombre de blocs libres consécutifs à partir du bloc $n°pos$ (inclus). Par exemple, avec les valeurs de la question précédente, on aura `count_free_blocs_at(1) = 3`, `count_free_blocs_at(7) = 1`, `count_free_blocs_at(9) = 0`

```
int count_free_blocks_at(int pos)
{




}
```

(c) En principe, les différents blocs qui composent un fichier peuvent être situés n'importe où sur le disque, puisqu'ils sont chaînés par la FAT. En pratique cependant, placer ces blocs à la suite les uns des autres sur le disque donnera de meilleures performances. Écrivez une fonction `int find_hole_ff(int len)` qui parcourt la FAT, cherche un «trou» d'au moins `len` blocs libres consécutifs, et renvoie le premier numéro de bloc du trou choisi, ou -1 en cas d'échec. Vous appliquerez une stratégie de recherche first-fit, **s'arrêtant au premier trou suffisamment grand**.

```
int find_hole_ff(int len)
{









}
```

(d) Écrivez une fonction `int find_hole_bf(int len)` qui fait la même recherche mais en suivant une stratégie best-fit, en renvoyant un trou de taille la plus petite possible.

```
int find_hole_bf(int len)
{









}
```

3. On s'intéresse dans cette question à un système de mémoire virtuelle paginée. Au cours de l'exécution d'un programme, on observe la MMU faire les traductions d'adresses suivantes (en notation hexadécimale) :
   $3E1F \rightarrow 7E1F$, 5454 $\rightarrow$ 9454, $2A8D \rightarrow 6A8D$, $E127 \rightarrow 2127$, $C8BA \rightarrow 08BA$, 9762 $\rightarrow$ 5762, $72CF \rightarrow A2CF$.
   Complétez (si les informations ne permettent pas de répondre, indiquez inconnue) :

   (a) La taille des pages virtuelles est : _____

   (b) La taille des pages physiques est : _____

   (c) La taille maximale de la mémoire d'un processus est : _____

   (d) La quantité de RAM disponible sur la machine est : _____

4. Si on exécute le programme ci-dessous, combien affichera-t-il de lignes, et avec combien de numéros distincts ?

```
int main(int argc, char *argv[])
{
  int n = 2;
  while(n > 0) {
    fork();
    n = n-1;
    printf("%d \n", getpid());
    fork();
  }
  pause();
  return 0;
}
```

Ce programme affichera :

☐ numéros en tout, dont
☐ numéros différents.

5. Soit un cache contenant 4 places. Dans la suite, chaque lettre de la séquence indique un accès à un bloc.

E D C B A E D A A A B C D E

(a) Remplir le tableau ci-dessous montrant l'évolution du cache en LRU. Quel est le taux de succès ?

	E	D	C	B	A	E	D	A	A	A	B	C	D	E
1														
2														
3														
4														

Taux de succès : ☐

(b) Quel serait le meilleur taux de succès si on pouvait voir dans l'avenir ?

	E	D	C	B	A	E	D	A	A	A	B	C	D	E
1														
2														
3														
4														

Taux de succès : ☐

6. On dispose d'un cache à correspondance directe de 16 lignes de 16 octets.

(a) Sachant qu'une adresse est sur 16 bits, et que l'unité d'adressage est l'octet, calculez, en bits, la taille de l'étiquette (tag) ☐ l'index ☐ l'offset ☐

(b) On lit, successsivement, aux adresses suivantes :

`0x0400, 0x0832, 0x0408, 0x0503, 0x0836, 0x0400`

Inidquez, à chaque fois l'entrée du cache concernée, et s'il s'agit d'un défaut ou succès. Le cache est initialement vide.

7. Depuis le bash, on a exécuté la commande `echo foo | wc -c`. Voici la sortie de `strace` pendant l'exécution de la commande :

```
58797 pipe([3, 4])                      = 0
58797 clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f892b19ea10) = 58798
58797 close(4)                          = 0
58798 close(3)                          = 0
58797 clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f892b19ea10) = 58799
58798 dup2(4, 1)                        = 1
58798 close(4)                          = 0
58798 execve("/bin/echo", ["/bin/echo", "foo"], 0x7ffdea26fe98 /* 57 vars */ <... detached ...>
58797 close(3)                          = 0
58797 wait4(58798,  <... unfinished ...>
58799 dup2(3, 0)                        = 0
58799 close(3)                          = 0
58799 execve("/usr/bin/wc", ["/usr/bin/wc", "-c"], 0x7ffdea26fe98 /* 57 vars */ <... detached ...>
58797 <... wait4 resumed> NULL, 0, NULL) = 58798
58797 --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=58798, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
58797 wait4(58799, NULL, 0, NULL)       = 58799
58797 --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=58799, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
58797 exit_group(0)                     = ?
58797 +++ exited with 0 +++
```

(a) Quel est le pid du processus bash ? pid = ☐

(b) Expliquez précisément ce que fait le bash pour exécuter la commande. Justifiez vos réponses.