Systèmes d'exploitation centralisés

1re année Informatique et Réseaux

juin 2018

Documents autorisés. Les exercices sont indépendants, l'ordre de ceux-ci est contraint par la mise en page.

Les réponses indiquées sont des éléments de correction, parfois sommaires. D'autres réponses correctes peuvent (parfois) exister. Il n'est pas exclu qu'une erreur se soit glissée dans cette correction.

I Système de fichiers (4 points)

La taille des blocs logiques influe directement sur la taille d'une inode et la nécessité de recourir à des blocs d'indirection de niveau 1, 2 ou 3. Elle influe ainsi sur le nombre d'ordre de lecture/écriture nécessaires pour lire/écrire tout le fichier. On suppose un rangement de type UFS, avec 5 blocs directs, un bloc d'indirection de niveau 1 et un bloc d'indirection de niveau 2. On suppose enfin qu'un numéro de bloc prend 8 octets.

1. Si la taille d'un bloc logique est 1 ko (1024 octets), un bloc d'indirection de niveau 1 contient les numéros d'au plus 128 blocs et un bloc d'indirection de niveau 2 référence au plus $128 * 128 = 16384$ blocs. Un fichier de taille 520 ko nécessite 520 blocs de données, soit $5 + 128 + (128 + 128 + 128 + 8)$ blocs. Combien de blocs en tout (y compris les blocs indirects) ce fichier occupe-t-il ?

   5 blocs directs + 1 bloc indirect niveau un + 128 blocs indirects niveau 1 + 1 bloc indirect niveau 2 + 4 sous-blocs indirects + 128+128+128+8
   = 520 + 1 + 5 = 526 blocs (= 526 ko utilisés)

2. Même question si on prend des blocs logiques de 256 ko.

   3 blocs directs de données (= 768 ko utilisés)

3. Compte tenu du surcoût dû aux blocs d'indirection, il est tentant de choisir une grande taille de bloc. Utiliser les questions précédentes pour montrer que ce n'est pas aussi simple.

   dernier bloc peu rempli → perte d'espace

4. Avec une grande taille de bloc, le dernier bloc de données d'un fichier est souvent peu rempli (cas 2, où le dernier bloc n'est rempli que de 8 ko). L'idée est alors d'utiliser un tel bloc pour stocker les "queues" de plusieurs fichiers. Quelles difficultés cela soulève-t-il ?

   — *Allocation de bloc : trouver un bloc peu plein ?*
   — *Déallocation d'un bloc : pas nécessairement à la destruction d'une inode → connaître toutes les inodes utilisatrices*

— *Offset au sein du bloc : le ranger dans l'inode, ou dans le bloc lui-même ?*
— *Fichier qui grossit → prévoir un peu de marge entre les "queues".*

II Noyau – Coroutines (5 points)

1. Qu'affiche le programme ci-dessous ?
2. Comment se termine-t-il ?

```
#include <stdio.h>
#include <stdlib.h>
#include "coroutines.h"

coroutine_t c1,c2,c3;

void foo() {
    int a = 1;
    printf("F0 %d\n", a);
    cor_transferer(c1,c2);
    a++;
    printf("F1 %d\n", a);
    cor_transferer(c2,c3);
    printf("F2 %d\n", a);
}

void codeA (void *unused) {
    printf("A  1\n");
    foo();
    printf("A  fini\n");
}

void codeB (void *unused) {
    printf("B  1\n");
    cor_transferer(c2, c3);
    printf("B  2\n");
    cor_transferer(c3, c1);
    printf("B  3\n");
    cor_transferer(c3, c2);
    printf("B  fini\n");
}

void codeC (void *unused) {
    printf("C  1\n");
    foo();
    printf("C  fini\n");
}

int main() {
    c1 = cor_creer("C1", codeA, NULL);
    c2 = cor_creer("C2", codeB, NULL);
    c3 = cor_creer("C3", codeC, NULL);
    cor_transferer(c1,c1);
    printf ("main fini\n");
}
```

```
A  1
F0 1
B  1
C  1
F0 1
B  2
F1 2
B  3
F2 2
C  fini
COROUTINES: fin sale.
```

III Noyau – Processus (6 points)

On souhaite enrichir notre noyau avec deux primitives : `proc_kill(processus_t qui)` qui permet de tuer un processus, et `proc_exit()` qui permet à un processus de se suicider. Observer que `proc_kill(proc_self())` et `proc_exit()` sont identiques.

1. Expliquer informellement ce que doit faire `proc_kill` selon l'état du processus tué (prêt, élu, suspendu, mort).
   — *prêt : marquer MORT et retirer de la file des prêts*
   — *élu : appeler `proc_exit`*
   — *suspendu : marquer MORT et c'est tout*
   — *mort : erreur ou ignoré*
2. Écrire le pseudo-code de `proc_kill` (en utilisant éventuellement `proc_exit`).
3. Le suicide via `proc_exit` consiste à "renvoyer" le processus dans son enveloppe et à finir l'exécution de celle-ci comme si le code du processus se terminait. Il faut donc savoir si le processus est en cours de suicide et être capable de restaurer le contexte de l'enveloppe.
   (a) Où peut-on ranger ces deux informations ?
   Dans le descripteur de processus (processus_t) ou en TSD
   (b) En utilisant la couche contexte (`getcontext` et `setcontext`), donner le pseudo-code de `proc_exit` et de l'`enveloppe` de la couche processus.

```
proc_exit() {
    (proc_self())->suicide = 1;
    setcontext(proc_self()->retour);
}
enveloppe() {
    p->suicide = 0;
    getcontext(p->retour);
    if (! p->suicide) { code(arg); }
    // comme avant après code
}
```

IV Mémoire virtuelle (5 points)

1. Dans le TP sur la mémoire virtuelle, la version présentée écrit systématiquement la page sur disque (fichier swap) lorsqu'elle est expulsée (`vider_page`). En fait, il n'est pas toujours obligatoire de l'écrire.
   (a) Donner un scénario où l'écriture lors de l'expulsion est inutile.
   Si la page n'a pas été modifiée après avoir été chargée. Ou cas particulier, une page première fois accédée et non modifiée ensuite.
   (b) Expliquer comment modifier l'implantation présentée en TD pour éliminer ces écritures inutiles.
   Cf TP avec validé/modifié

2. On souhaite maintenant faire varier le nombre de pages de swap. Dans la description des opérations, il n'est pas demandé du code C mais une sommaire description algorithmique de haut niveau.

   (a) On ajoute une page supplémentaire en fin du fichier de swap. Indiquer quelle(s) structure(s) de données il faut mettre à jour et quelles sont la ou les difficultés, et décrire les opérations à effectuer.

   Pas de difficultés : agrandir `infos_swap` *d'une case, incrémenter* `nbpagesswap`*, et marquer la nouvelle page (descripteur_page_swap) comme non utilisée.*

   (b) On supprime la dernière page du fichier de swap. Indiquer quelle(s) structure(s) de données il faut mettre à jour et quelles sont la ou les difficultés, et décrire les opérations à effectuer.

   Difficulté : si la page en question avait un contenu → trouver une autre page de swap libre (et refus de suppression s'il n'y a en a plus), déplacer le contenu de la page à supprimer, mettre à jour la table du processus (`virt_2_swap` */* `infos_proc`*) avec le nouveau numéro de page de swap.*
   Ensuite, il suffit de réduire `infos_swap` *d'une case et décrémenter* `nbpagesswap`*.*