Contrôle PRS

Déroulement du Contrôle

ATTENTION LE SUJET EST CONSTITUÉ DE **DEUX PARTIES**
CE QUI DONNERA LIEU A **DEUX COMPOSITIONS DIFFÉRENTES**

LA PREMIERE CONCERNE LA PARTIE THEORIQUE
ELLE SE FAIT SUR PAPIER ET **SE TERMINE AU BOUT D'UNE HEURE !**

LA DEUXIÈME CONCERNE LA PARTIE PRATIQUE
ELLE SE FAIT SOUS UNIX AVEC UN DÉPÔT QUI
FERME À LA FIN DU CONTRÔLE

LE CONTRÔLE **DURE 4H** EN TOUT

1. Après avoir démarré votre ordinateur sous `Ubuntu`.
2. Et après avoir **téléchargé le ZIP du contrôle** et vérifié qu'il contient les éléments suivants :
   a. Le **sujet** du contrôle **en PDF** et une **archive** des codes C utilisés.
   b. Le **support du cours en PDF** (**c'est le seul document autorisé avec l'aide en ligne**).
3. **Débranchez** physiquement le **câble réseau** de votre machine (**aucune communication — quelque soit sa forme — n'est autorisée**).
4. **Prenez** une **feuille** et composez la 1° partie (théorique) :
   Dès que vous avez fini cette partie, vous rendez votre composition et vous passez à la partie pratique. De toutes les manières vous serez préempté au bout d'1h (durée max**).**
5. Vous êtes prêt à composer, la 2° partie (pratique : le reste du temps), en respectant les règles qui suivent :
   a. **Créez** un **répertoire** de travail avec un nom de la forme `PRS_MON_NOM_CTP.`
   b. **Copiez** l'**archive** des codes fournis dans ce **répertoire**.
   c. **Créez** un fichier dans votre **répertoire** pour y rédiger **vos réponses**. **Le fichier à rendre doit être au format PDF !**
   d. **Commencez par lire le sujet en entier pour en avoir une vue d'ensemble**.
   e. Composez.
6. A la fin de la composition de la partie pratique, **compressez** votre répertoire contenant le fichier de **réponses** portant votre **NOM**, ainsi que les fichiers source **C** et tout autre fichier utile.
7. **Branchez** le **câble réseau** de votre machine.
8. **Déposez** votre travail sur la plateforme moodle, en utilisant le lien prévu à cet effet.

Attention à l'heure de dépôt de votre travail, le retard de remise sera sanctionné

Bon travail à tous, Thomas, vous dirait : « Enjoy ! »

Partie 1 :
Epreuve théorique de programmation système (2018/2019)
Questions de cours & de compréhension

Dans vos réponses, ne vous contentez pas de recopier le poly !

Question 1

Quelles sont les politiques d'ordonnancement dites « temps réel » implémentées par le système Linux. Décrire le fonctionnement de chacune d'elles ainsi que l'ordre de priorité entre-elles.
Donner les échelles POSIX ainsi que l'échelle utilisée par Linux pour fusionner les différentes échelles.

Question 2

a) Donner une définition de la préemption et d'un noyau préemptif.

b) Le noyau Linux est-il préemptif ?

Question 3

a) Donner le modèle d'états d'un processus.

b) Ce modèle utilise 3 états et pourrait avoir six transitions au lieu de quatre pour être complet.
Existe-t-il des circonstances dans lesquelles l'une ou l'autre des transitions manquantes pourraient se produire ?

c) Définir l'état Zombie d'un processus et ce qui le provoque. Décrire ce qui se passe (s'est passé) (se passera).

d) Comment un processus devient orphelin et que lui arrive t-il par la suite ?

Question 4

Citer deux avantages à l'utilisation des threads au lieu des processus.

Question 5

a) Que se passe t-il si un processus prêt reçoit plusieurs occurrences d'un même signal ?

b) Est ce que le traitement d'un signal peut être interrompu à son tour par un autre signal ?

c) Peut-on redéfinir le traitement par défaut du signal `SIGKILL` ?

d) Comment redémarrer un appel système bloquant cassé par un `CTRL-C` ?

Question 6

Comment garantir le partage d'une variable en mode écriture ?

Partie 2 :
Epreuve pratique de programmation système (2019/2020)

Travail préliminaire

Dans le ZIP téléchargé sur Moodle, vous trouverez une archive compressée nommée `PRS_CTP_Codes.tar.gz`.

Cette archive est composée d'un fichier source `karaOK_v0.c` et d'un fichier d'entête `common.h`.

Question 1

a. Quelle est la commande permettant de décompresser cette archive au format tar.gz ?

b. Donner la commande permettant de renommer le dossier ainsi obtenu, en ajoutant « votre nom », et réaliser cette opération.

Question 2

Quelle est la commande permettant de générer un exécutable nommé `karaOK` ?

Question 3

La compilation signale des avertissements concernant des déclarations implicites ainsi que des erreurs. Pourtant une grande partie des informations manquantes sont présentes dans le fichier `common.h`, comment corriger ces problèmes ?

Question 4

Le début de la fonction principale récupère sur la ligne de commande le nombre de processus fils qui seront créés. Cette valeur doit être comprise entre `1` (*valeur par défaut*) et `7`.

a. Donner un exemple de demande d'exécution de ce programme.

b. Quel est le rôle de la fonction `atoi()` ?

c. Dans quel volume du manuel se trouve la description de cette fonction ?

d. Quel fichier d'entête faut-il inclure dans le fichier source pour pouvoir l'utiliser ?

e. Mêmes questions pour `amoi()` et `alui()`.

Question 5

a. Que réalise l'appel ci-dessous :

```
system("echo ;ps -o pid,ppid,size,cls,comm,cmd");
```

b. Préciser la signification des informations affichées en vous aidant du `man`.

Question 6

L'un des affichages obtenus, montre la présence de processus zombies.

a. Prendre une capture d'écran de cet affichage et indiquer quel(s) est(sont) le(s) processus concernés. Comment est noté leur état ?

b. Comment effectuer efficacement la recherche d'un mot-clé comme zombie sur la page de manuel concernant la commande `ps` ?

c. Copier dans le compte-rendu, le paragraphe (4 lignes) décrivant ce que sont des processus zombies.

Question 7

Sans modifier le code source du programme, quelle option de compilation faut-il spécifier pour afficher l'extrait du refrain, prévu dans la fonction `welcome()` ? Insérer dans le compte-rendu une copie d'écran présentant cet affichage.

Question 8

Donner la commande permettant d'exécuter le programme de telle sorte que les messages produits par les instructions `fprintf(stderr, …)` n'apparaissent plus dans la fenêtre d'exécution ?

Ajouts et modifications à apporter

Première étape

Commencer par créer une copie du fichier `karaOK_v0.c` en `karaOK_v1.c`.

Question 9

Donner la commande permettant de réaliser l'opération précédente.

Question 10

Dans le nouveau fichier, mettre en commentaire, les instructions

```
system("echo ;ps -o pid,ppid,size,cls,comm,cmd");
```

et

```
sleep(DUREE_VIE_PERE);
```

a. Ajouter dans la section signalée dans le code source, les opérations garantissant que le processus père attende la terminaison de tous ses fils avant de se terminer lui-même. Après chaque terminaison, on affichera sur le dispositif d'affichage des messages d'erreur, le numéro (de 1 à 7) et le pid du processus fils qui s'est terminé – on exploitera le fait qu'un fils qui se termine fournit son numéro, comme code de terminaison –

b. Insérer dans le compte-rendu une capture d'écran montrant l'absence de processus zombies, avant la terminaison du processus père.

Question 11

Modifier le code de la fonction `processusFils()` comme suit :

```
/* ---------------------------------------------------------------------- */
void processusFils(int no)
{
    struct timespec delai = { .tv_sec = 0, .tv_nsec = 100000000L};

    initialWait(no);
    fprintf(stderr, "\n\tFils %d : \tlancement effectif du processus %d, "
                    "fils de %d\n", no + 1, getpid(), getppid());

//    sleep (DUREE_VIE_FILS);

    setTerm(RED + no);
    printf("\t\"");
    for (char * p = Msg[no]; *p != '\0'; p++)
    {
        printf("%c", *p);
        fflush(stdout); /* vider le buffer d'affichage sur l'écran (stdout) */
        nanosleep(&delai, NULL);
    }
    printf("\"\n");
    resetTerm();

    fprintf(stderr, "\tFils %d : \tfin du processus %d\n", no + 1, getpid());
    exit(no);
}
```

La fonction `initialWait()` sert à produire une attente avant lancement effectif du traitement, générée de façon aléatoire.

L'affichage du message associé au numéro du fils est réalisé caractère par caractère ; un délai d'attente de 100 ms est effectué entre l'affichage de deux caractères successifs.

Chaque processus fils utilise une couleur distincte pour l'affichage de son message.

a. Compiler puis exécuter le programme de façon à créer 4 processus fils et que les messages produits par les instructions `fprintf(stderr, …)` ne soient pas affichés sur l'écran.

b. Insérer une capture d'écran du résultat obtenu et expliquer pourquoi les messages affichés sont incompréhensibles et leur couleur aussi variée.

Seconde étape

Commencer par créer une copie du fichier `karaOK_v1.c` en `karaOK_v2.c`.

Question 12

Comment garantir que lorsqu'un processus fils a commencé l'affichage de son message, il puisse aller jusqu'au bout de cet affichage.

a. Pour cette application, quelle est la ressource critique à accès exclusif, dont il faut protéger l'accès ?

b. Mettre en œuvre cette solution. On s'assurera que les objets créés ou ouverts sont correctement traités en fin de processus.

Question 13

a. Expliquer pourquoi la commande permettant de générer l'exécutable karaOK a besoin d'une option supplémentaire ?
Donner cette commande

b. Effectuer le même test qu'à la 0a
Insérer une capture d'écran présentant le résultat obtenu avec quatre fils.

c. Deux exécutions successives donnent-elles exactement le même résultat. De quoi dépend l'ordre d'affichage des messages ?

Question 14

Dans quel dossier sous Linux, le pseudo-fichier associé à la ressource critique, est-il créé ?

Troisième étape

Commencer par créer une copie du fichier `karaOK_v2.c` en `karaOK_v3.c`.

Question 15

Sans modifier la manière dont les processus fils sont créés, ni agir sur l'attente effectuée au début de chaque processus fils, comment peut-on néanmoins garantir que les messages seront affichés dans l'ordre dans lequel ils sont rangés dans le tableau `Msg[]` ?

Question 16

Mettre en œuvre cette solution et effectuer le test pour un nombre de fils égal à 7. Les noms des fichiers associés seront construits sur le principe suivant :

a. Déclaration d'une variable destinée à contenir successivement les noms de fichiers associés :

```
char synchroFilename [FILENAME_MAX + 1];
```

b. Construction et utilisation des noms des fichiers associés :

```
for (int i = 0; i < nbFils; i++)
{
    sprintf(synchroFilename, "%s_%d", SEM_SYNCHRO, i + 1);
    CHECK_IFNULL(synchro[i] = sem_open(synchroFilename, … ), …)
}
```