Département Informatique

Nom :	Année scolaire : 2004–2005
Prénom :	Date : 22 juin 2005

Module ISI423
Session de juin

—

Systèmes d'exploitation et leur support d'exécution

—

Contrôle de connaissance[1] de 45 minutes

MERCI de répondre (au moins) dans les blancs.
Lire tout le sujet en entier du début à la fin, en commençant à la première page et jusqu'à la dernière page, avant de commencer à répondre : cela peut vous donner de l'inspiration et vous permettre de mieux allouer votre temps en fonction de vos compétences.

Chaque question sera notée entre 0 et 10 et la note globale sera calculée par une fonction des notes élémentaires. La fonction définitive sera choisie après correction des copies.
Attention : tout ce que vous écrirez sur cette copie pourra être retenu contre vous, voire avoir une influence sur la note d'ISI423.

1 Généralités système

Question 1 : Vous êtes impliqué dans un projet de robotique. Pour contrôler tout votre robot, votre équipe hésite entre un ordinateur portable avec un système d'exploitation ou de nombreux microcontrôleurs sans système d'exploitation. Comparer les 2 approches en argumentant. □

→
→

[1] Avec document, avec calculatrice, sans triche, sans copie sur les voisins, sans micro-ordinateur portable ou non, sans macro-ordinateur, sans téléphone portable ou non, sans oreillette de téléphone ni de dictaphone, sans talkie-walkie, sans télépathie, sans métempsycose, sans pompe. Sont tolérés : anti-sèche, tatouage ou vêtement imprimé en rapport avec le sujet, mouchoir de poche pré-imprimé, piercing ou scarifi cation en rapport avec l'ISI423, bronzage à code barre ou 2D...

→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→

Question 2 : Citez les différences générales entre la notion de processus en Unix et en Windows.

On s'intéressera particulièrement à la gestion mémoire des processus ayant terminé leur vie (durée de réponse estimée : 7 mn). □

→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→

→
→
→
→
→
→
→
→
→

2 Sécurité

Question 3 : Parmi les opérations suivantes, lesquelles ne devraient être autorisées qu'en mode noyau (cocher devant les phrases) ?

- ❏ masquer les interruptions ;
- ❏ lire l'horloge donnant la date ;
- ❏ écrire (modifier) l'horloge donnant la date ;
- ❏ modifier la table des pages de la MMU ;
- ❏ accéder à la quantité de mémoire disponible.

□

3 Temps partagé & ordonnancement

Question 4 : On a envie de faire un robot pour la coupe de la robotique Eurobot/E=M6 qui soit commandé avec un PC portable sous Linux, y compris le moteur pas à pas.

Le moteur pas à pas est à 4 phases et lorsqu'on envoie successivement les 4 commandes de phase, le moteur fait avancer le robot de 0,01 m.

Linux 2.6 sur PC a une horloge système permettant de faire des changements de contexte à une fréquence de 1 kHz par défaut. Un changement de contexte prend 7 μs de temps processeur[2].

Sachant qu'un programme simple peut à chaque coup d'horloge système s'exécuter via un changement de contexte et envoyer en 13 μs avec le port parallèle de l'ordinateur une commande de phase au moteur, quelle est la vitesse maximale d'avancement du robot ? □

→
→
→

[2] Pour plus d'information les curieux iront voir par exemple
`http ://linuxdevices.com/articles/AT3479098230.html`.

→
→

Question 5 : Quelle proportion de processeur reste disponible pour faire fonctionner le programme qui tourne dans un autre processus pour contrôler la stratégie du robot ? □

→
→
→
→
→

Question 6 : Si, maintenant, pour simplifier l'architecture logicielle du robot on décide de scinder le logiciel en de nombreux processus qui se traduisent par 3 changements de contextes en moyenne liés à une exécution puis blocage sur une attente quelconque au sein d'un seul créneau de 1 kHz. On considère que le contrôle du moteur pas à pas entre dans ces 3 changements de contextes.

Quelle proportion de processeur reste disponible pour faire d'autres choses utiles, c'est à dire hors du temps perdu à faire des changements de contextes ? □

→
→
→
→
→

Question 7 : Quel est le nombre maximal de processus différents qui peuvent être exécutés dans un créneau de 1 kHz ?

Quelle est alors le travail utile qu'il peuvent exécuter ? □

→
→
→
→
→
→
→

4 Processus

Question 8 : La fonction `system()` permet d'exécuter facilement une commande Unix à partir d'un processus sans avoir à gérer soi-même l'exécution. La page du manuel de référence de cette fonction est fournie en annexe de ce sujet de CC. Nous vous demandons de donner votre propre version de `system()`. En d'autres termes, si cette fonction n'existait pas, vous en auriez

sûrement besoin et vous devriez la ré-écrire vous-même et c'est ce que nous vous demandons ici. □

→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→

Date de naissance (s)	0	2	10	13	30	32	40
Position de destination (m)	10	30	20	12	5	25	2

TAB. 1 – Date de naissance et position des humains à placer.

→
→

5 Entrées-sorties (Matrix — Épisode IV)

Question 9 : Dans le système d'alimentation électrique de la Matrice, l'électricité est produite par des humains vivant dans des cocons. À leur naissance, les humains sont récupérés par une pieuvre électrique qui doit les placer dans des cocons préétablis pour faire de l'équilibrage de charge[3] afin de produire correctement de l'électricité.[4]

On vous demande de proposer et comparer des algorithmes (méthodes) simples permettant d'optimiser les mouvements de la pieuvre. On argumentera en donnant le temps de placement global de vos méthodes.

Précisions : la pieuvre peut transporter jusqu'à 3 humains d'un coup, les humains naissent (arrivent) à différentes dates en position 0 (origine) et doivent être placés à différentes positions. Les dates d'arrivée et les positions de destination (en mètres linéaires depuis l'origine) sont indiquées sur le tableau **??**. Toutes les dates sont en secondes et les positions en mètres. Les déplacements de la pieuvre sont de 1 m/s et prendre un humain coûte 1 s, le placer délicatement dans un cocon 2 s. La pieuvre est en position 0 à l'origine et doit être remise en position 0 à la fin. □

→
→
→
→
→
→
→
→
→
→
→
→
→
→
→

[3] Électrique... On suppose que la matrice est en triphasé.

[4] Toute ressemblance avec l'optimisation du déplacement des têtes de lecture-écriture d'un disque dur, le démarrage et le temps de positionnement n'est pas une pure coïncidence fortuite...

→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→
→

6 Question subsidiaire

Il s'agit d'une question « bonus ». Contrairement aux autres questions elle peut rapporter des points mais pas vous en faire perdre si vous ne répondez pas.

Question 10 : Expliquer ce que Monsieur l'Ancien Premier Ministre de la France (Europe) Jean-Pierre RAFFARIN a voulu dire dans cette *raffarinade* :

« *Je n'aime pas beaucoup ne pas être dans le logiciel central de moi-même.* »

dans la perspective d'ISI423. □

→
→
→
→
→
→
→
→
→
→
→
→

NOM

system − Exécuter une commande Shell.

SYNOPSIS

#include <stdlib.h>

int system (const char * *string***);**

DESCRIPTION

La fonction **system()** exécute la commande indiquée dans *string* en appelant **/bin/sh -c** *string*, et revient après l'exécution complete de la commande. Durant cette exécution, le signal **SIGCHLD** est bloqué, et les signaux **SIGINT** et **SIGQUIT** sont ignorés.

VALEUR RENVOYÉE

La valeur renvoyée est −1 en cas d'erreur (par exemple echec de fork()) ou le code de retour de la commande sinon. Ce dernier code est dans le format indiqué dans **wait**(2). Ansi le retour de la commande sera *WEXITSTATUS(status)*. Dans le cas où **/bin/sh** ne peut pas être exécuté, le code de retour sera identique à celui d'une commande effectuant un *exit(127)*.

Si la valeur de *string* est **NULL**, **system()** renvoie une valeur non nulle si le shell est accessible, et zéro sinon.

system() n'affecte pas le statut d'attente des autres processus fils.

CONFORMITÉ

ANSI C, POSIX.2, BSD 4.3

NOTES

Comme mentionné plus haut, **system**() ignore SIGINT et SIGQUIT. Ceci peut rendre ininterruptible un programme qui l'appelle en boucle, à moins qu'il ne vérifie le code de retour du fils, par exemple

```
while(qqchose) {
    int ret = system("foo");

    if (WIFSIGNALED(ret) &&
        (WTERMSIG(ret) == SIGINT || WTERMSIG(ret) == SIGQUIT))
            break;
}
```

N'utilisez jamais **system()** dans un programme avec les privilèges Set-UID ou Set-GID. Des variables d'environnement avec des valeurs étranges peuvent être utilisées pour corrompre l'intégrité du système. Utilisez les fonctions de la famille **exec**(3) à la place, mais pas **execlp**(3) ni **execvp**(3).

system() ne fonctionnera pas correctement avec les programmes ayant des privilèges fournis par les bits Set-UID ou Set-GID sur les systèmes où **/bin/sh** est **bash** version 2, car celui-ci rejette les privilèges au démarrage. (Debian utilise une version modifiée de bash où ce comportement est abandonné si on l'invoque sous le nom **sh**).

La vérification de disponibilité du shell **/bin/sh** n'est pas réellement faite. Il est toujours supposé disponible. Le standard ISO C réclame cette vérification, mais POSIX.2 précise que le retour doit toujours être non-nul car un système sans shell n'est pas conforme. Ceci justifie le choix d'implémentation.

Il est possible qu'une commande shell renvoie 127, ainsi le code de retour n'est pas une indication sûre de l'échec de **execve()**. Verifiez *errno* pour en être sûrs.

VOIR AUSSI

sh(1), **signal**(2), **wait**(2), **exec**(3)

TRADUCTION

Christophe Blaess, 1996-2003.