

LAS - Examen Juin

Vu la situation sanitaire actuelle et la décision de la HE d'organiser les examens exclusivement à distance, l'examen de l'UE « BINV2180 : Linux : appels système » prendra la forme d'un travail à réaliser à partir du mercredi 10 juin 2020 à 8h30 jusqu'au jeudi 11 juin 2020 à 11h30.

L'examen contient 3 questions :

1. Appels systèmes relatifs aux signaux
2. Appels systèmes relatifs à la mémoire partagé et aux sémaphores
3. Appels systèmes : socket - exec – open – close – dup - read – write - fork - wait

Pour chacune de ces questions, nous vous demandons de créer un programme distribué en C. Vos programmes doivent utiliser les appels système Linux vus durant les cours.

1) Planning

Pour la réalisation de ce travail, nous suivrons le planning ci-dessous :

1. **Mercredi 10 juin 2020 à 8h30** : publication de l'examen sur <https://evalmoodle.vinci.be/>.
2. **Mercredi 10 juin 2020 à 8h30** : ouverture sur <https://evalmoodle.vinci.be/> d'un forum que vous pouvez utiliser pour poser vos questions.
3. **Mercredi 10 juin 2020 à 8h30** : ouverture d'un devoir sur <https://evalmoodle.vinci.be/> permettant de remettre vos travaux.
4. **Mercredi 10 juin 2020 de 11h à 12h** : séance de questions/réponses « live » et facultative à l'aide de l'outil Microsoft Teams : <https://bit.ly/2AgpyoO>.
5. **Jeudi 11 juin 2020 de 9h à 10h** : séance de questions/réponses « live » et facultative à l'aide de l'outil Microsoft Teams : <https://bit.ly/2AgpyoO>.
6. **Avant le jeudi 11 juin 2020 de 11h30** : remise de vos travaux via le devoir ouvert sur <https://evalmoodle.vinci.be/>. Les travaux remis en retard seront pénalisés.
7. **Jeudi 11 juin 2020 de 11h30** : fin de l'examen, fermeture du forum et fermeture du devoir.

2) Travaux à remettre

Nous vous demandons de soumettre un fichier zip contenant un répertoire par question. Voici la liste des fichiers à soumettre :

L'examen contient 3 questions :

1. Appels systèmes relatifs aux signaux (répertoire robot_signaux) :
 - a. robot.c,
 - b. Makefile
2. Appels systèmes relatifs à la mémoire partagé et aux sémaphores (répertoire robot_shm) :
 - a. controller.c
 - b. robot_create_ipc.c
 - c. robot_delete_ipc.h
 - d. Makefile
3. Appels systèmes : socket - exec – open – close – dup - read – write - fork - wait (répertoire ftp) :
 - a. server.c
 - b. client.c
 - c. Makefile
 - d. communication.h
 - e. communication.c
 - f. utils.h
 - g. utils.c
 - h. Le répertoire ftpdir avec les fichiers html

3) Canal de communication

En cas d'urgence, les étudiants peuvent contacter :

- 1) Monsieur José Vander Meulen par téléphone en utilisant le numéro suivant 0460245194, par mail à l'adresse suivante : jose.vandermeulen@vinci.be, en envoyant un message dans l'application Microsoft Teams, ou en démarrant une conversation orale dans l'application Microsoft Teams. Durant l'examen, il sera disponible pendant les « heures de bureau ». **Monsieur Vander Meulen s'occupe des 2 premières questions (signaux, mémoire partagée et sémaphores)**
- 2) Monsieur Olivier Choquet par mail à l'adresse suivante : olivier.choquet@vinci.be, en envoyant un message dans l'application Microsoft Teams, ou en démarrant une conversation orale dans l'application Microsoft Teams. Durant l'examen, il sera disponible pendant les « heures de bureau ». **Monsieur Choquet s'occupe de la dernière question (socket).**

4) Appels systèmes relatifs aux signaux

Y/X	0	1	2	3	4	5	6	7	8	9
9										
8										
7										
6										
5										
4	R									
3										
2										
1										
0										

Tableau 1 : Une grille de 10 X 10 cellules



Imaginons un robot se trouvant dans une grille de 10 X 10 cellules (cf Tableau 1). Chaque cellule possède une coordonnée X et une coordonnée Y. Dans le Tableau 1, le robot (R) se trouve dans la cellule X = 0 et Y = 4. **Initialement, le robot se trouve en X = 0 et Y = 0.**

Le robot est télécommandé par une manette qui possède 4 boutons :

1. Le bouton « U » qui permet au robot d'aller vers le nord, qui incrémente Y.
2. Le bouton « D » qui permet au robot d'aller vers le sud, qui décrémente Y.
3. Le bouton « L » qui permet au robot d'aller vers l'ouest, qui décrémente X.
4. Le bouton « R » qui permet au robot d'aller vers l'est, qui incrémente X.

Notez que le monde du robot est circulaire :

1. Si le robot est en « Y = 9 » et qu'on appuie sur le bouton « U », le robot ira en « Y = 0 ».
2. Si le robot est en « Y = 0 » et qu'on appuie sur le bouton « D », le robot ira en « Y = 9 ».
3. Si le robot est en « X = 0 » et qu'on appuie sur le bouton « L », le robot ira en « X = 9 ».
4. Si le robot est en « X = 9 » et qu'on appuie sur le bouton « R », le robot ira en « X = 0 ».

Dans cette question, le robot ne peut se déplacer **qu'au nord et au sud**. Dans la suivante, il sera amélioré et pourra aller dans les 4 directions.

Ecrivez un programme « **robot.c** » qui crée un fils. Le père représentera la manette et le fils le robot.



1. Le père affiche un prompt à l'écran (>>>) et lit des lignes au clavier. Les lignes simulent le fait d'appuyer sur un bouton. Elles peuvent prendre les trois formes suivantes :
 - a. « U\n » : dans ce la cas le père envoie un signal « SIGUSR1 » à son fils
 - b. « D\n » : dans ce la cas le père envoie un signal « SIGUSR2 » à son fils
 - c. « Q\n » : dans ce la cas le père envoie un signal « SIGKILL » à son fils
2. Le fils affiche la position initiale du robot. De plus, il gère le traitement des signaux « SIGUSR1, SIGUSR2 et SIGKILL » et affiche, à chaque déplacement du robot, la position de celui-ci. Il gère les signaux de la manière suivante :
 - a. Lorsqu'il reçoit un signal « SIGUSR1 » le robot se déplace vers le nord.
 - b. Lorsqu'il reçoit un signal « SIGUSR2 » le robot se déplace vers le sud.
 - c. Lorsqu'il reçoit un signal « SIGKILL » le fils est tué.

Voici une exécution possible :

```
ec2-user:~/environment/2020_LAS_EXAMEN_JUIN_ROBOT $ ./robot
>>>X = 0, Y = 0
U
>>>X = 0, Y = 1
U
>>>X = 0, Y = 2
U
>>>X = 0, Y = 3
D
>>>X = 0, Y = 2
D
>>>X = 0, Y = 1
D
>>>X = 0, Y = 0
D
>>>X = 0, Y = 9
D
>>>X = 0, Y = 8
U
>>>X = 0, Y = 9
Q
ec2-user:~/environment/2020_LAS_EXAMEN_JUIN_ROBOT $
```



Complétez le fichier « **robot.c** » et le fichier « **Makefile** »

5) Appels systèmes relatifs à la mémoire partagé et aux sémaphores

Dans cette question, nous allons améliorer le robot de la question précédente. Plusieurs manettes pourront maintenant le diriger. Chaque manette aura quatre boutons (« U, D, L, R »).

Notez que dans cette question, vous ne devez pas manipuler de signaux. Vous ne devez donc pas étendre votre programme de la question précédente, mais plutôt écrire un nouveau programme distribué.

Nous allons maintenant représenter les coordonnées du robot par deux « int » définis dans une mémoire partagée. L'accès à cette mémoire partagée doit être protégé par des sémaphores.

Nous vous demandons de compléter les 4 fichiers suivants :

1. **robot_create_ipc.c** : permet de générer un programme qui gère la création et l'initialisation de la mémoire partagée et du sémaphore.
2. **robot_delete_ipc.c** : permet de générer un programme qui gère la destruction de la mémoire partagée et du sémaphore.
3. **controller.c** : permet de générer un programme qui simule une manette à 4 boutons.
4. **Makefile**

Le programme « controller » affiche un prompt à l'écran (>>>) et lit des lignes au clavier. Les lignes simulent le fait d'appuyer sur un bouton. Elles peuvent prendre les cinq formes suivantes :

1. « U\n » : le robot va vers le nord
2. « D\n » : le robot va vers le sud
3. « L\n » : le robot va vers l'ouest
4. « R\n » : le robot va vers l'est
5. « Q\n » : le programme se termine

Il affiche la position initiale du robot. Après chaque ligne lue, il met à jour la mémoire partagée et affiche la position du robot. Plusieurs programmes « controller » peuvent être exécutés de manière simultanée.

6) Appels systèmes : socket - exec - open - close - dup - read - write - fork - wait

Ecrivez un programme « **server.c** » qui sera un serveur FTP basique. Ce serveur FTP peut recevoir les commandes suivantes : LS et GET_INDEX. La commande LS est matérialisée par le code 100 et la commande GET_INDEX par le code 200. Le serveur FTP écoute sur un port donné en argument, il accepte **un client à la fois**, traite sa demande puis ferme la connexion avec ce client. Le serveur ne s'arrête jamais sauf via un CTRL-C.

La commande **GET_INDEX** renvoie le fichier index.html présent dans le répertoire de base du ftp (./ftplib).

La commande **LS** effectue un « ls -l » dans le répertoire de base du ftp (./ftplib) et renvoie le résultat de ce ls. Cette commande LS sera exécutée via l'appel système **execl**.



Pour récupérer le résultat du exec, il est nécessaire de rediriger la sortie standard vers un fichier (ls.txt). Vous pourrez ensuite lire le contenu de ce fichier et l'envoyer au client.

Ecrivez un programme « **client.c** » qui sera un client FTP basique et qui permettra d'envoyer les 2 commandes citées au serveur FTP. Le client prend 2 arguments : l'adresse IP du serveur et le port du serveur. Le client envoie une commande et se termine. Il faut donc lancer 2 fois ce client pour tester les 2 commandes. Le client affiche le menu suivant :

```
printf("Bienvenue dans le programme client FTP basique\n");
printf("Vous pouvez effectuer les commandes suivantes\n");
printf("\t 1 : lister les fichiers présents sur le serveur FTP (LS)\n");
printf("\t 2 : récupérer le fichier index.html sur le serveur FTP\n");
printf("\t 3 : quitter le client FTP\n");
```

Voici le résultat du programme attendu pour GET_INDEX :

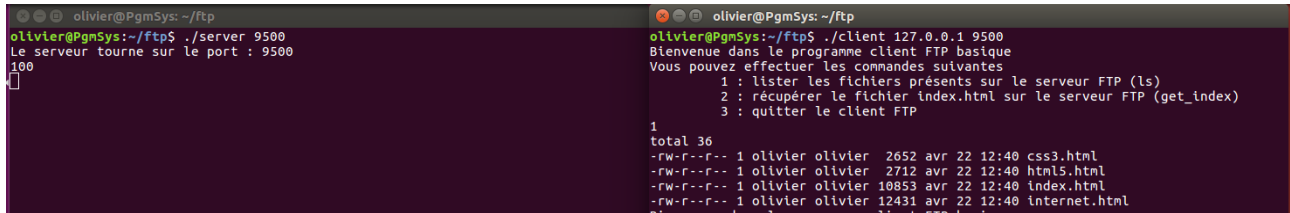
```
olivier@PgmSys: ~/ftp/solution
olivier@PgmSys:~/ftp/solution$ ./server 9500
Le serveur tourne sur le port : 9500

olivier@PgmSys:~/ftp/solution$ ./client 127.0.0.1 9500
Bienvenue dans le programme client FTP basique
Vous pouvez effectuer les commandes suivantes
1 : lister les fichiers présents sur le serveur FTP (ls)
2 : récupérer le fichier index.html sur le serveur FTP (get_index)
3 : quitter le client FTP
2
```

Ensuite après avoir effectué le choix « 2 », le client affiche la page html :

```
olivier@PgmSys: ~/ftp/solution
:jeanluc.collinet@ipl.be">Collinet
Emmeline</a><a class="mail" href="mailto:
Jean-Luc</a>
</div>
<div id="error">
<p>
Si vous détectez une erreur, une faute d'orthogr
aphe n'hésitez pas à
envoyer un <a href="mailto:olivier.choquet@ipl.b
e"> mail</a> avec le
nom de la page et l'erreur détectée.
</p>
</div>
<div id="version">
<p>
Syllabus HTML <br> version 5.0
</p>
</div>
</div>
</body>
</html>
```

Voici le résultat du programme attendu pour LS :



```
olivier@PgmSys: ~/ftp
olivier@PgmSys:~/ftp$ ./server 9500
Le serveur tourne sur le port : 9500
100
□

olivier@PgmSys: ~/ftp
olivier@PgmSys:~/ftp$ ./client 127.0.0.1 9500
Bienvenue dans le programme client FTP basique
Vous pouvez effectuer les commandes suivantes
1 : lister les fichiers présents sur le serveur FTP (ls)
2 : récupérer le fichier index.html sur le serveur FTP (get_index)
3 : quitter le client FTP
1
total 36
-rw-r--r-- 1 olivier olivier 2652 avr 22 12:40 css3.html
-rw-r--r-- 1 olivier olivier 2712 avr 22 12:40 html5.html
-rw-r--r-- 1 olivier olivier 10853 avr 22 12:40 index.html
-rw-r--r-- 1 olivier olivier 12431 avr 22 12:40 internet.html
```

Remarques importantes :

1. Vous disposez d'un répertoire « ftp » qui contient tout ce qui est nécessaire pour cet exercice.
2. Vous disposez d'un fichier « utils.h » et « communication.h », utilisez les méthodes qui s'y trouvent à bon escient. Ceci sera évalué
3. Le makefile est déjà fait !
4. Nous vous demandons d'écrire dans le fichier « communication.h » et « communication.c » la fonction « initSocketClient »
5. Les affichages à l'écran peuvent être fait via la fonction printf
6. L'utilisation de poll n'est pas nécessaire pour cet exercice
7. Commencez par faire la commande GET_INDEX qui est plus facile à réaliser
8. Ecrivez une fonction « void transfer_file (char * pathfile, int sockfd) » dans le code du serveur qui permettra de lire un fichier et de l'envoyer au client
9. La lisibilité du code est importante !