

## Partie machine

(durée estimée : 1h15)

### Consignes pour la soumission des exercices sur machine

Cet examen sur machine comporte 2 exercices.

Dans votre répertoire privé (celui dans lequel vous êtes positionné juste après votre connexion) vous devez créer un répertoire de nom **os302VOTRENOM** (« VOTRENOM » doit être remplacé par votre nom de famille en majuscules, sans accent, sans tiret, sans espace...).

Dans le répertoire **os302VOTRENOM**, créez un répertoire **EXO1** et un autre **EXO2**.


Le(s) source(s) de l'exercice 1 doivent se trouver dans le répertoire EXO1 et ceux de l'exercice 2 doivent être dans le répertoire EXO2.

Les codes sources doivent être déposés en suivant les étapes illustrées ci-dessous :

- Créer une archive avec l'ensemble de vos codes sources
- Il faut se connecter sur <http://192.168.130.150:8080>
- Ensuite il faut cliquer sur **Sélection du fichier**, sélectionner l'archive avec les codes sources, et enfin cliquer sur **Envoie** (voir la figure ci-dessous).

[Accueil](#) [Liste des fichiers disponible](#) [Déconnexion](#) [Changement de mot de passe](#)

**Envoie de votre travail**

 Sélection du fichier

Pas de fichier

Envoie

Il était 2022-05-17 10:55:59 quand la page fut affichée.

Filesender

**Exercice 1 : création de processus (4 points)**

L'objectif de cet exercice est d'écrire un programme qui permet à un processus de créer 3 processus fils, puis de se synchroniser sur la terminaison de ses processus fils. A la fin de la synchronisation, le processus père affiche l'ordre de terminaison de ses fils. Le fichier source aura pour nom `ex01.c`.

**Question 1.**

Ecrire une fonction qui permet de créer 3 processus fils et de placer les identifiants (PID) correspondants dans un tableau de 3 éléments. Le prototype de cette fonction doit être :

```
int creation (pid_t tab[3]);
```

Cette fonction renvoie le nombre de processus créés en cas de succès et -1 en cas d'échec. Elle place les PID des processus créés dans le tableau `tab` dans l'ordre de leur création.

**Question 2.**

Ecrire une fonction qui permet au processus père de se synchroniser sur la terminaison des 3 processus fils créés. Cette fonction doit avoir le prototype suivant :

```
int attente (pid_t tab[3], pid_t fin[3]);
```

La fonction `attente()` renvoie 0 en cas de succès et -1 en cas d'échec. Les PID des processus fils sont transmis à la fonction `attente()` au moyen du paramètre `tab`. Le paramètre `fin` permet de récupérer les PID des processus dans l'ordre de leur terminaison.

**Exercice 2 : gestion des signaux et communication par tube (8 points)**

L'objectif de cet exercice est de permettre à un processus père de compter le nombre de signaux de type SIGINT qu'il reçoit et d'afficher à chaque fois un message indiquant si ce nombre est pair ou impair. Une fois le nombre de signaux attendus (6) reçus, le processus père en informe son processus fils par un message envoyé sur un tube « J'ai fini de recevoir des signaux ! ».

Cet exercice sera traité en 2 versions : la première avec la fonction `signal()`, la seconde avec la fonction `sigaction()`. Il vous est demandé d'effectuer un *test d'erreur* lors de l'appel de ces 2 fonctions. La version utilisant la fonction `signal()` sera placée dans un fichier source de nom `exo2v1.c`. La version utilisant la fonction `sigaction()` sera placée dans un fichier source de nom `exo2v2.c`.

Lorsque le nombre de signaux reçus atteint 6, le programme se termine.

Pour la réalisation de cet exercice, on considère que le nombre de signaux SIGINT reçus est représenté dans une *variable globale* de nom `nb_sig`.

**Question 1.**

Ecrire une fonction permettant de tester si le nombre de signaux SIGINT actuellement reçus est pair et d'afficher le message « Nombre pair ! ». Cette fonction a le prototype suivant :

```
void affichage_pair (int sig) ;
```

**Question 2.**

Ecrire une fonction permettant de tester si le nombre de signaux SIGINT actuellement reçus est impair et d'afficher le message « Nombre impair ! ». Cette fonction a le prototype suivant :

```
void affichage_impair (int sig) ;
```

**Question 3.**

Ecrire la fonction `main()` qui permet à un processus père de créer un processus fils. Le fils temporise en attendant que son père termine de compter les signaux reçus. Le processus père gère la réception des signaux SIGINT de sorte à adapter l'affichage à la parité du nombre de signaux SIGINT actuellement reçus. Il sera peut-être nécessaire de compléter le code des 2 fonctions d'affichage précédemment codées.

**Rappel :** deux versions du programme sont attendues : une avec `signal()` et une avec `sigaction()`. Pensez aux tests d'erreurs pour ces 2 fonctions !

