Université Grenoble Alpes **Année 2018/2019**
L3 MIAGE

Introduction au Système
Planche pour les TP n°4 et 5

Exercice 1.

1. Placez-vous ailleurs que dans votre répertoire de login et exécutez la commande :

```
( who ; cd ; ls -l )
```

et observer attentivement le résultat - *en particulier quelle différence observez-vous par rapport à l'exécution de la commande :* `who ; cd ; ls -l` *?*

2. Écrire le programme C qui réalise l'équivalent de l'exécution de cette commande particulière.

Exercice 2.

1. Exécuter dans votre shell les commandes :

`ps ax | grep bash` et `ps ax | grep bash | wc -l`

et assurez-vous de comprendre l'objectif de ces commandes.

2. Écrire le programme C qui réalise l'équivalent de l'exécution de la commande

```
ps ax | grep bash | wc -l
```

Exercice 3.

Cet exercice constitue le projet à rendre (par 2). Vous devez :

- **commencer à l'étudier en séance le 5 octobre,**
- **le travailler d'ici la séance suivante**,
- et **présenter son état** courant - qui **devra être quasi-terminal** - **à cette dernière séance**, le 12 octobre. Le binôme devra être au complet.

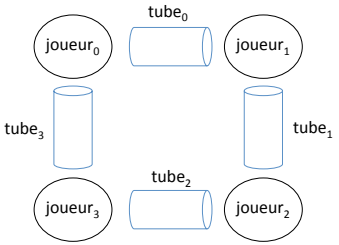
Les dernières finitions, et le rapport, pourront être réalisés jusqu'au **19 octobre, date limite de remise**.

Le but de ce projet est de programmer un **jeu de petits chevaux dans lequel chaque joueur est un processus distinct et les joueurs communiquent via des tubes**.
On trouvera par exemple un dessin du plateau de jeu et la règle du jeu ici :
http://www.petits-chevaux.com/tout-sur-les-petits-chevaux.htm
Attention, le jeu en lui-même n'est pas l'objet principal de ce projet, le soin doit être mis sur la gestion des processus et des communications.

Dans l'idéal, on pourra faire jouer de 2 à 4 joueurs, avec 4 chevaux chacun. Si vous avez des difficultés, **commencez à réaliser de quoi faire jouer 2 joueurs, avec 1 cheval chacun**, voire avec des règles du jeu simplifiées (il vaut mieux rendre ceci bien réalisé et documenté, qu'une tentative de solution générale mal faite et/ou qui ne fonctionne pas).

On adoptera l'architecture suivante :

** les **4 processus** qui correspondent aux 4 joueurs vont s'informer de l'état du jeu par une **boucle de tubes** comme décrit par la figure ci-dessous.
Lorsqu'il a joué, chaque joueur envoie à son voisin (par le tube) ce qu'il vient de jouer, et les processus font tourner cette information dans la boucle (sur tout le plateau, donc) afin que tous les joueurs soient informés. Evidemment, choisir un sens de rotation.



** le **processus père synchronisera le jeu** de la façon suivante : il indiquera aux 4 joueurs, par un **tube spécifique** supplémentaire, le numéro du prochain joueur à jouer, puis attendra un retour avant de recommencer cette procédure. Typiquement, le retour sera "l'information a fait un tour complet du plateau (c'est à dire a atteint les 4 joueurs)". Attention, on pourra aussi penser au cas où un joueur est autorisé à jouer 2 fois de suite.
Ne pas oublier de repérer lorsque le jeu est terminé, pour arrêter proprement l'application.

** ce retour sera renvoyé au processus père par **un autre tube spécifique**.

Pour les "tirages de dé", on procédera par génération de nombres pseudo-aléatoires selon une loi de probabilité uniforme. On rappelle que ceci consiste à construire une suite u_k et nécessite donc l'initialisation de la "graine" (c'est à dire de u_0). Afin que la suite construite ne soit pas toujours la même, il convient de faire des initialisations différentes de la graine d'une exécution sur l'autre; pour cela on utilise habituellement le temps courant.
En C, l'initialisation de la graine se fera grâce à la fonction :

```
void srand (unsigned int seed);
```

et la génération de nombres pseudo-aléatoires, par la fonction :

```
int rand (void);
```

On pourra par exemple consulter `http://www.areaprog.com/c/cours-258-srand-et-rand-generation-de-nombres-pseudos-aleatoires` pour plus de détails et une illustration.

La position d'un cheval sur le plateau pourra être repérée par un numéro de case par rapport à son point de départ (départ case 0, cases suivantes 1, 2, 3,…).

On note que la règle du jeu prévoit que, dans certains cas, on puisse choisir de faire avancer un cheval sur le plateau ou de sortir un nouveau cheval. Si on souhaite coder cela, on prévoira une interaction conversationnelle avec l'utilisateur pour ces choix.

Vous êtes libres d'afficher l'information sur l'état du jeu comme vous l'entendez.

Nous vous suggérons de procéder progressivement comme suit.

1. Lire attentivement ci-dessus, puis dessiner l'organisation complète avec tous les processus et tubes, et bien spécifier quelles données seront transmises dans quel tube à quel moment. Autant que possible, faire au moins valider cela par votre encadrant de TP dans la séance du 5 octobre.

2. Écrire une application projet_0 qui crée les processus fils du processus initial. Assurez-vous que tous les processus ont correctement été créés (vous pourrez facilement vous assurer de la bonne création des processus en utilisant getpid() et getppid()).

3. Reprendre sous la forme d'une application projet_1 qui crée les processus fils du processus initial et met en place les tubes tels que décrit ci-dessus. Assurez-vous que les processus arrivent à communiquer (faites par exemple circuler un entier dans l'anneau), et que la

communication avec le processus père est également possible. Assurez-vous également que tous les descripteurs inutilisés ont bien été fermés, que tous les processus peuvent mourir, qu'il ne reste pas de zombie.

4. Lorsque vous serez sûrs que les processus et leur communication se mettent en place correctement, vous pourrez passer à la finalisation du projet.

Note. Les commandes suivantes pourront vous être utiles pour tester le comportement de votre application :
`pgrep -P pid` : affiche la liste des fils du processus de numéro pid
`lsof -p pid` : affiche la liste des fichiers ouverts (et leur type) par le processus de numéro pid
`pstree` : affiche les processus en cours d'exécution sous la forme d'un arbre

---

Réalisation et remise. A LIRE ATTENTIVEMENT

Vous devrez fournir un fichier .tar.gz contenant :

- tous vos **sources** *correctement organisés et commentés* (les noms des auteurs devront aussi figurer dans tous les sources),
- un **rapport** de présentation de votre travail (format pdf - classiquement environ 8 à 10 pages) qui présente clairement votre réflexion, vos choix de conception et de structures de données, vos réalisations (voire limitations), et qui permette d'utiliser votre projet (expliquer comment le compiler et l'exécuter),
- et un **Makefile** pour la compilation de vos sources.

NB. Le fichier envoyé doit permettre d'identifier clairement les noms des étudiants du binôme. Son identifiant devra être de la forme nom1_nom2.tar.gz où nom1 et nom2 sont les noms de famille des étudiants. Attention, il devra être produit à partir d'un répertoire lui-même d'identifiant nom1_nom2.
Placer votre hiérarchie de fichiers constituant le projet dans un répertoire nom1_nom2.
Faites tar cvf nom1_nom2.tar nom1_nom2
puis gzip -9 nom1_nom2.tar

Remise du projet (pour les grenoblois) **:** Un seul fichier .tar.gz envoyé par mail à Albin.Petit@univ-grenoble-alpes.fr et amadou.diarra@univ-grenoble-alpes.fr avant le 19 Octobre 2018 soir, extrême limite (tout projet arrivé en retard ne sera pas corrigé).