ESIEE AMIENS	TRAVAUX PRATIQUES N°3 Matière : Programmation multitâches Remis par : S.Pomportes / Y. Guerroudj J. Olive / C. Sarroche	Année : I4 GRIT Temps : 4h

OBJECTIFS

Comprendre le fonctionnement des tubes ainsi que leur utilisation.

REMARQUES

Vous placerez l'ensemble de vos fichiers sources dans le répertoire PMTP3 de votre répertoire personnel (commande mkdir $HOME/PMTP3).
Vous rédigerez un compte rendu au format openoffice ou txt.
Chaque fichier (compte rendu ou code) doit comporter vos noms au début.

Les tubes

1. Présentation

Les pipes, en français tubes, sont l'implantation Unix des tampons gérés suivant le schéma producteur/consommateur. On représente leur principe de fonctionnement avec des sémaphores :

```
Producteur              Consommateur
...                     ...
P(S1)                   P(S2)
écrire (case(i))        lire (case(j))
i = suiv_prod(i)        j = suiv_cons (j)
V(S2)                   V(S1)
...                     ...
```

S1 et *S2* sont initialisés ainsi : *Init(S1,N)* et *Init(S2,0)*. Le tampon est visualisé ci-dessous :

2. Utilisation en C

Pour gérer et utiliser un tube :

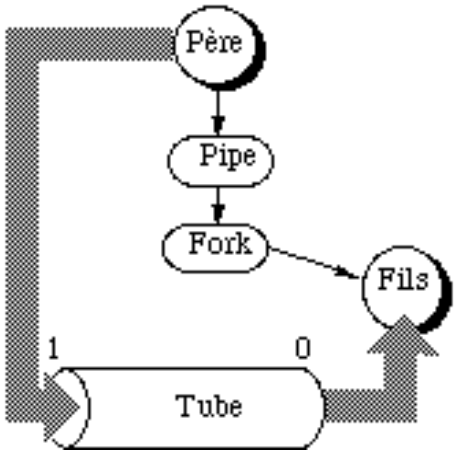
- on déclare un tableau d'entiers de 2 éléments :
  ```
  int Tube[2];
  ```
- on crée le tube en appelant la fonction `pipe ()` :
  ```
  pipe (Tube);
  ```
- on lit dans le tube en utilisant le premier élément du tableau, par exemple pour y lire un caractère, on fera :
  ```
  read (Tube[0], &c,1);
  ```
- on écrit dans le tube en utilisant le second élément du tableau, par exemple pour y écrire un caractère, on fera :
  ```
  write (Tube[1], &c,1);
  ```

3. Exercice préliminaire

Ecrivez un programme dans lequel le père et le fils communiquent via un tube (pipe). Le père attend des caractères depuis le clavier et les écrit dans le tube. Le fils (le consommateur) se contente d'afficher ce qu'il a lu dans le tube. Quand ***<CTRL D>*** est tapé, le programme se termine : le fils meurt puis le père affiche « mon fils est mort » et se termine.

Contraintes :

- Le fils ne connaît pas le nombre d'écrivains accédant au tube.
- Vous ne devez pas utiliser de signaux.



Si vous tournez cette page, attendez-vous au pire …

4. Le Trône de Fer (S05E08 : La bataille de Durlieu)

Le roi des marcheurs blanc (appelons le comme ça pour éviter le spoil) à levée une armée de mort-vivants qu'il lance contre le village portuaire de Durlieu. La taille de l'armée n'est pas connu à l'avance mais la GRIT n'ayant pas les moyens du studio « El Ranchito » nous nous limiterons à une centaine d'unités (au-delà ce sont les PCs qui risquent de souffrir). Heureusement les morts-vivants ne sont plus ce qu'ils étaient et se feront tuer sans difficulté.

Réécrivons un peu l'histoire (mais pas trop) :

Les morts-vivants se présentent un à un devant Jon Snow (via un tube) qui utilisera alors son arme favorite pour les détruire : grand-griffe alias SIGUSR2 (attention, il est interdit de le laisser utiliser SIGUSR1 qui, pour rappel, est le sabre laser de SKYWALKER dans la guerre des étoiles. Évitons donc le mélange des genres qui pourrait finir à la sawDust : https://www.youtube.com/watch?v=HcNoB4O9Mg8)

Une fois le dernier mort-vivant éliminé, Jon Snow devra affronter le marcheur blanc. Pour le trouver il pourra utiliser les informations qu'il aura soutiré aux mort-vivants avant qu'ils ne meurent.

De son coté, le marcheur blanc est fou de rage à la mort de son dernier soldat, et décide de s'attaquer à Snow. Contrairement à ce dernier, le marcheur sait où trouver son ennemi (grâce à une indiscrétion de dieu lors de sa naissance). Les coups les plus mortels sont autorisés et même conseillés.

Qui sera le vainqueur ? Sans doute le premier qui attaquera mais, qui est-ce ? Prévoyez un résultat aléatoire afin de pouvoir tester les différents cas.

Que dire de plus sinon que Dieu a créé Jon Snow et le marcheur blanc.

Ps : vous rendrez un programme C commenté à la fin duquel vous expliquerez brièvement comment vous avez traduit ce scénario en programme Unix.

Avant de passer à l'exercice suivant, vérifiez que votre programme est judicieusement commenté, qu'il est bien construit, qu'il n'a pas de bug et ne risque pas d'avoir de comportement non souhaité (fermez-vous tous les descripteurs de fichiers ouverts avant de quitter ? N'y a-t-il pas un risque qu'une partie du code ne s'exécute pas dans certains cas ? Traquez les cas particuliers de changements de contexte !).

5. Pour aller plus loin

Les tubes sont implémentés à l'aide des sémaphores (cf Section 1 : Présentation). Il est donc tout à fait possible d'utiliser les tubes pour émuler des sémaphores. C'est ce que vous allez faire ici.

1. En premier lieu, programmez les fonctions P(S), V(S) et e(S, N) en utilisant uniquement des tubes. Mettez ces trois fonctions dans un fichier « sematube.c ». Créez le fichier sematube.h correspondant. Notez qu'aucun de ces fichiers n'a de fonction « main ».
2. Testez vos sémaphores sur des exemples simples (interblocage, exclusion mutuelle, etc.) en expliquant à chaque fois ce que vous faites. Chacun de vos exemples devra se trouver dans un fichier *.c différent, ce fichier *.c faisant appel aux fonctions de sémaphores que vous avez implémentées en incluant (#include) le fichier sematube.h (qui, rappelons-le, ne contient PAS les sources des fonctions que vous avez implémentées, uniquement leurs prototypes !).
3. Créez le fichier Makefile permettant de compiler vos exemples. Les appels à *make* pourraient être du type « make interblocage » ou « make mutex » par exemple. Une recherche sur google (mots clefs *makefile simple howto*) vous permettra sûrement d'obtenir les renseignements nécessaires pour créer ce Makefile si vous ne savez pas comment cela fonctionne.