Université Paul Sabatier **Licence Informatique – L3/S6**

31062 Toulouse Cedex 9

Le système UNIX

Travaux pratiques

TP n°3 – Signaux et tubes

1 – signal() / sigaction()

Écrire un programme qui boucle et qui ne peut être détruit qu'après avoir reçu 10 caractères CTRL_C (Signal `SIGINT`).

Donner une version du programme utilisant la primitive `signal` puis une seconde version utilisant `sigaction`.

2 – fork(), signal(), pause(), kill() / sigaction(), sigsuspend(), kill()

Écrire un programme dans lequel un processus et son fils se synchronisent en utilisant les signaux pour écrire, à tour de rôle, un message personnalisé sur la sortie standard.

Donner une version du programme utilisant la primitive `pause` puis une seconde version utilisant `sigsuspend`.

3 – fork(), pipe(), read(), write(), close(), signal(), pause(), kill()

On se propose de simuler une chaîne de fabrication d'un certain type de produit. Cette chaîne est composée de 4 processus `ouvrier` qui travaillent en parallèle.

Le premier processus `ouvrier` est chargé d'accueillir la clientèle : il reçoit une requête de la part d'un processus `client` et lance le processus de fabrication du produit désiré en transmettant les caractéristiques de la commande au deuxième ouvrier de la chaîne.

Chaque ouvrier de production (les trois ouvriers suivants) transmet le produit au processus `ouvrier` suivant après avoir réalisé sa tâche.

Quand le produit arrive au dernier `ouvrier`, celui-ci prévient le `client` en lui envoyant le signal `SIGUSR1` afin qu'il vienne chercher son produit.

Chaque `ouvrier` tient compte du travail du précédent qui lui est indiqué par un code sur un entier.

Dans cette application le nombre maximum de clients est fixé à `NBCLIENTS`.

Quand il n'y a plus de clients susceptibles de faire des commandes, les ouvriers se terminent (`exit(0);`) et le programme principal rend la main au processus shell.

☞ Faire un schéma des communications entre les différents processus et les structures de données nécessaires et le faire **valider** par l'enseignant de TP.

- ☞ Écrire la fonction `ouvrier` qui définit le comportement de l'ouvrier de numéro `num`. L'en-tête de cette fonction est : `void ouvrier(int num);`

- ☞ Écrire la fonction `void client1(int num);` qui définit le comportement d'un client de type 1 et de numéro `num` avec comme contrainte : le client attend la fin de la fabrication du produit pour continuer son activité.

- ☞ Écrire la fonction `void client2(int num);` qui définit le comportement d'un client de type 2 et de numéro `num` avec comme contrainte : le client n'attend pas la fin de la fabrication du produit pour continuer son activité.

- ☞ Écrire le programme principal d'une application comportant 4 ouvriers et `NB_CLIENTS1` (paramètre du programme) clients de type 1.

- ☞ Dans une seconde version, remplacer les clients de type 1 par des clients de type 2.

- ☞ Enfin, une troisième version du programme devra supporter `NB_CLIENTS1` de type 1 et `NB_CLIENTS2` de type 2 (le nombre de clients étant toujours des paramètres du programme).

Question optionnelle ouverte

Écrire une commande `Mailer` paramétrée par l'identité d'un utilisateur à qui on veut envoyer un message électronique.

Le processus père sera chargé de collecter le message écrit au clavier (terminaison de la saisie par Ctrl-D) et le communiquera à son fils en utilisant un tube de communication.

Le processus fils enverra alors ce message par courrier électronique à l'utilisateur concerné.

Le père attendra l'envoi du message pour signifier que la commande s'est bien déroulée.