SCIENCES SORBONNE UNIVERSITÉ

Examen Réparti 2 PR
Master 1 Informatique Jan 2020

UE 4I400

Année 2019-2020

2 heures – **Tout document papier autorisé**
Tout appareil de communication électronique interdit (téléphones. . .)

Introduction

- Le barème est sur 20 points et est donné à titre indicatif et susceptible d'être modifié.
- Il est demandé de répondre précisément et concisément aux questions.
- Dans le code C++ demandé vous vous efforcerez d'ecrire du code correct et compilable, mais les petites typos ne seront pas sanctionnées.
- De même les problèmes d'include et de namespace ne sont pas pertinents (pas la peine de qualifier les `std::`, ni de citer tous les `#include`)
- On suppose également que **tous les appels système se passent bien**, on ne demande pas de contrôler les valeurs de retour (donc pas de `perror` ou `errno.h`).

Le sujet est composé de quatre exercices indépendants qu'on pourra traiter dans l'ordre qu'on souhaite.

1 Shell et redirections (4,5 points)

On considère la réalisation d'une partie d'un interprète de commande type bash. On souhaite traiter l'exécution d'une commande arbitraire avec redirection de la sortie dans un fichier.

Un exemple d'invocation : `./redirect /bin/ls -l > log.txt`.

Question 1. (3,5 points) Ecrire un programme `main` qui :

- contrôle que l'avant dernier argument est bien une chaîne ">"
- considère que le dernier argument est un nom de fichier, qui sera créé s'il n'existe pas. Dans ce cas, les droits du fichier seront positionnés sur : `-rw-rw-r-` (read et write pour user et group, read pour other).
- invoque la commande correspondant aux arguments précédant le ">" dans un nouveau processus en *redirigeant* la sortie du processus dans ce fichier
- attend la fin de cette exécution et sort proprement

Question 2. (1 point) Que faut il modifier dans le programme pour que si le fichier existe, on ajoute à la fin du fichier plutôt que d'écraser son contenu ? (i.e. la redirection symbolisée par ">>" en shell).

2 Serveur TCP (5,5 points)

On souhaite construire un serveur de diffusion utilisant le protocole TCP. Le serveur attend les connections de clients sur le port "1664"; il peut traiter plusieur clients simultanément.

Chaque fois qu'un client écrit un message au serveur (les messages ont une taille fixe de 128 octets), celui-ci doit renvoyer le message reçu à tous les clients connectés.

L'objectif est de réaliser ce serveur de diffusion sans aucune concurrence (processus ou thread).

Question 1. (1,5 point) Sans fournir de code, expliquer informellement les étapes (appels système utiles) nécessaires avant de pouvoir *accepter* des connexions TCP sur le port donné. Expliquez en une ligne le rôle de chaque appel système utilisé, et la nature de ses arguments.

Question 2. (1 point) On souhaite stocker dans un `vector<int>` les filedescriptor correspondant aux sockets des clients conectés au serveur. Ecrivez une fonction : `void handleClientConnexion(int fdattente, vector<int> & clients)` qui traite l'arrivée d'un nouveau client qui vient de tenter une connexion sur le port 1664.

L'argument *fdattente* désigne le filedescriptor représentant la socket à laquelle les clients se connectent, et l'argument *clients* doit être mis à jour dans la fonction pour stocker le filedescriptor de la socket de discussion avec le client.

Question 3. (1,5 points) Donner le code de la fonction `void readAndBroadcast(int readfd, const vector<int> & clients)`, qui lit un message sur `readfd` et le rediffuse sur tous les clients représentés dans `clients`.

Question 4. (1,5 points) Sans fournir de code, expliquer informellement les étapes (appels système utiles) nécessaires pour pouvoir attendre simultanément une demande de connexion ou un message d'un des clients connectés avec un seul processsus, et le traiter. Expliquez en une ligne le rôle de chaque appel système utilisé, et la nature de ses arguments.

3 Famille de processus (6 points)

Question 1. (1,5 points) Ecrire un programme qui crée trois processus au total : le grand-père, le père et le petit-fils. Le grand-père affichera "GP" et son pid, le pere "P" et son pid et le petit-fils "PF" et son pid. On souhaite que l'ordre de ces affichages ne soit pas contraint ; cependant le programme doit se terminer proprement quand tous les affichages sont finis.

Question 2. (1,5 points) Sans fournir de code, expliquer informellement les étapes (appels système utiles) nécessaires pour pouvoir afficher dans le petit-fils le pid de son grand-père, et dans le grand-père le pid de son petit-fils. Expliquez en une ligne le rôle de chaque appel système utilisé, et la nature de ses arguments.

NB: on suppose dans la suite que *gpid* et $pfid$ sont des variables désignant les pid du grand-père et du petit-fils, définies et acessibles dans tous les processus.

Question 3. (1,5 points) On souhaite garantir que l'affichage du grand-pere soit réalisé *après* l'affichage du petit-fils. Expliquez comment réaliser ce comportement à l'aide de signaux: donner le code à exécuter par le grand-père pour que le signal reçu déclenche un affichage, et par le petit-fils pour envoyer le signal.

Question 4. (1,5 points) Sans fournir de code, expliquer informellement les étapes (appels système utiles) nécessaires, position dans le code déjà élaboré, pour que le grand-père ne puisse recevoir le signal que *après* avoir réussi à obtenir le pid de son petit-fils. Expliquez en une ligne le rôle de chaque appel système utilisé, et la nature de ses arguments.

4 Questions de synchronisation (4 points)

Question 1. (2 points) On vous demande le code de `void multilock (vector<mutex> & mutexes, vector<int> indexes)` où l'objectif est d'acquérir le sous ensemble des mutex dans le vector *mutexes* désignés par les indices dans *indexes*. Cette fonction est invoquée dans un contexte concurrent et ne doit jamais provoquer d'interblocage.

Par exemple on peut demander à acquérir les lock $indexes = \{8, 3, 5\}$ soit les verrous $mutexes[8]$, $mutexes[3]$, $mutexes[5]$.

Donnez le code de cette fonction *multilock* et de son symétrique `void multiunlock (vector<mutex> & mutexes, vector<int> indexes)` qui libère les mutex aux indices désignéss.

Question 2. (2 points) On suppose une interaction entre deux tâches qui échangent des données complexes (contenant des pointeurs, taille variable) que l'une produit et l'autre consomme. En max deux lignes par réponse, quels mécanismes faut-il mettre en place dans un contexte où :

1. Les deux tâches sont des threads au sein d'un seul processus
2. Les deux tâches sont des processus hébergés sur la même machine
3. Les deux tâches sont des processus distants, sur des machines différentes d'un même réseau