INF2610

TP #3: Comm Lab

Ecole Polytechnique de Montréal

Hiver 2019 — Durée: 2 heures et 10 minutes

Présentation

Le *Comm Lab* a pour but de vous familiariser avec les techniques classiques de communication inter-processus. Il se présente sous la forme de trois "puzzles" que vous devrez résoudre en programmant une solution à chacun d'entre eux. **Il est conseillé de lire l'énoncé en entier avant de commencer le TP.**

Attention: Ce cours a une politique bien définie en termes de plagiat. L'archive que vous avez téléchargée sur Autolab a été générée pour votre groupe seulement. Il vous est interdit de partager votre code ou le code qui vous a été fourni, que ce soit en ligne (via un dépôt Git public par exemple) ou avec d'autres étudiants. Votre code sera systématiquement vérifié et le plagiat sera sanctionné. Afin d'éliminer tout doute quant à ce qui peut constituer du plagiat et pour connaître les sanctions, veuillez vous référez à la page Moodle du cours.

Objectifs

A l'issue du TP, vous serez capables de:

- Créer des tubes de communication;
- Communiquer avec un processus au moyen d'un tube nommé;
- Communiquer avec un processus au moyen d'un tube anonyme;
- Dupliquer des descripteurs de fichier;
- Rediriger les flux d'entrée et de sortie d'un processus.

Énoncé

Vous trouverez par la suite les instructions précises pour chacun des puzzles à résoudre. Pour vous aider à gérer votre temps, nous indiquons pour chaque puzzle avec le symbole ◔ le temps qu'un élève finissant le TP dans le temps imparti devrait y consacrer. **Ces indications ne sont que des estimations pour que vous puissiez situer votre progression, alors pas de panique si vous dépassez la durée conseillée!**

Question	Contenu	◔	Barème
1	Tube de communication nommé; envoi de signaux	35mn	5.0 pt
2	Tube de communication anonyme	35mn	5.0 pt
3	Redirection de flux; duplication de descripteurs de fichiers; communication par tube anonyme	60mn	6.0 pt

Important: Tout au long de ce TP, vous devrez créer des processus fils pour y exécuter des programmes. **N'oubliez pas de systématiquement attendre après la fin des fils – en utilisant la fonction `wait` – afin de ne pas perturber la correction automatique.**

Puzzle 1 *(Le message secret)* *environ 35mn*

Nous vous fournissons pour cette question un jeton qui vous identifie de manière unique:

```
b4bf32d00a3f
```

Votre objectif est de transmettre ce jeton au programme `puzzle1/exchanger`, qui va se charger de valider le jeton et, en cas de succès, de vous transmettre un message en réponse. Vous utiliserez un premier tube de communication nommé pour envoyer le jeton au programme `exchanger`, et un second tube de communication nommé pour recevoir sa réponse.

☞ Lorsque vous lancez l'exécutable `puzzle1/exchanger`, un message d'erreur vous indique que deux tubes de communication nommés n'existent pas; c'est normal et ce sera à vous de les créer! Utilisez l'utilitaire `strace` sur le programme `exchanger` pour déterminer les chemins d'accès que vous devrez utiliser pour créer les tubes.

Indice: Le programme `exchanger` utilise l'appel système `openat` pour ouvrir le tube nommé. Dans le cas présent, cet appel système échoue avec l'erreur `ENOENT (No such file or directory)` que vous verrez apparaître dans la sortie du programme `strace`.

☞ Complétez la fonction `puzzle1` du fichier `puzzle1.c` pour envoyer le jeton ci-dessus au programme `exchanger`. Vous devez pour cela exécuter le programme `exchanger` depuis un processus fils, créer le tube de communication nommé et l'ouvrir en écriture pour transmettre le jeton. Notez que l'exécutable `exchanger` ne prend pas de paramètre.

☞ Le programme `exchanger` ne renverra un jeton que s'il reçoit le signal `SIGUSR2`. Complétez la fonction `puzzle1` pour envoyer le signal `SIGUSR2` au processus `exchanger` **après lui avoir transmis le message ci-dessus**.

☞ Complétez la fonction `puzzle1` pour récupérer le message envoyé par le programme `exchanger`. Vous devez pour cela compléter votre code précédent en créant le second tube de communication nommé et en l'ouvrant en lecture pour récupérer le message.

☞ Afin de vérifier que vous avez correctement récupéré le message, complétez la fonction `puzzle1` pour faire appel à la fonction `checkExchangerMessage` définie dans le fichier `libcommlab.h`. Cette fonction prend pour unique argument une chaîne de caractères, qui doit contenir le message renvoyé par le programme `exchanger`.

Attention:

- Lorsque vous envoyez une chaîne de caractères avec la fonction `write`, vous devez préciser le nombre de caractères à envoyer en argument de la fonction. Prenez soin d'inclure le caractère de terminaison '`\0`' de la chaîne lorsque vous comptez le nombre de caractères!
- Les tubes de communication nommés doivent être créés **depuis votre code** et non depuis le terminal Linux. Pensez également à supprimer les tubes créés après leur utilisation grâce à la fonction de librairie `remove`.

␣ → 18	h → 117	p → 125	x → 133	5 → 214
a → 110	i → 118	q → 126	y → 134	6 → 215
b → 111	j → 119	r → 127	z → 135	7 → 216
c → 112	k → 120	s → 128	0 → 209	8 → 217
d → 113	l → 121	t → 129	1 → 210	9 → 218
e → 114	m → 122	u → 130	2 → 211	
f → 115	n → 123	v → 131	3 → 212	
g → 116	o → 124	w → 132	4 → 213	

Figure 1: L'alphabet à utiliser pour le puzzle 2.

Puzzle 2 *(Le télégraphe)* *environ 35mn*

On vous fournit un télégraphe poussiéreux que vous allez utiliser pour envoyer un message. Votre chargé de lab écoute sur le câble[1], et vous attribuera des points pour le puzzle 3 s'il ou elle reçoit le bon message. Le télégraphe a un fonctionnement un peu spécial cependant, alors lisez bien les instructions qui suivent...

Nous avons programmé pour vous une interface logicielle pour utiliser le télégraphe: le programme `puzzle2/telegraph`. Vous devez l'exécuter depuis un processus fils, créé dans la fonction `puzzle2` du fichier `puzzle2.c`, en utilisant une des fonctions de la famille `exec`; il attend comme unique argument le numéro d'un descripteur de fichier qui est l'un des deux côtés (le côté lecture) d'un tube anonyme que vous devez créer. Vous utiliserez l'autre côté de ce tube pour envoyer des messages au processus `telegraph`. Notez que le programme `telegraph` prend soin de fermer le descripteur de fichier qui lui est passé après utilisation. Cependant, vous devez vous assurer que le processus `telegraph` n'a pas plus de descripteurs de fichiers ouverts que nécessaire!

Votre chargé de lab s'attend à recevoir le message suivant, qui contient d'ailleurs un jeton permettant de vous identifier:

```
token is 857528a05d0f end
```

La subtilité est que vous devez envoyer le message octet par octet en utilisant les codes des caractères donnés en figure 1, dans lequel chaque caractère est représentée par un code sur un octet dont la valeur se situe entre 0 et 255. Au lieu d'envoyer le message initial caractère par caractère, vous enverrez donc une suite d'octets (variables de type `unsigned char`) dans laquelle chaque octet codera pour un caractère du message initial. Le processus `telegraph` va lire les données octet par octet (en utilisant également des variables de type `unsigned char`) depuis le tube, et enverra le message à votre chargé de lab quand il lira l'octet 0. Assurez-vous donc de bien terminer votre message par l'octet 0!

☞ Complétez la fonction `puzzle2` du fichier `puzzle2.c` pour envoyer le jeton grâce au programme `telegraph`. Commencez par utiliser la fonction `pipe` pour créer un tube anonyme, puis créez un processus fils dans lequel vous exécuterez le programme `telegraph`. Vous enverrez le message ci-dessus depuis le processus père.

[1] En réalité il n'écoute pas vraiment sur le câble, mais nous allons prétendre que c'est le cas.

Puzzle 3 *(La citation brouillée)* *environ 1h*

Nous vous donnons quatre exécutables `exc1`, `exc2`, `exc3` et `exc4` dans le répertoire `puzzle3`. Chacun d'eux imprime sur le sortie standard et / ou la sortie d'erreurs une partie d'une citation célèbre (ou pas); certains de ces exécutables le font d'ailleurs seulement si on leur fournit la bonne chaîne de caractères en entrée. Votre objectif est d'imprimer la citation en entier et dans le bon ordre, sur la sortie standard seulement. Vous devez réussir à faire cela en complétant la fonction `puzzle3` du fichier `puzzle3.c`; ce code sera exécuté par le mécanisme d'évaluation qui comparera la sortie de votre programme à la citation originale. Vous devez respecter certaines contraintes pour pouvoir obtenir des points pour cette question:

- Chacun des quatre exécutables doit être exécuté une fois (et une fois seulement);
- Vous ne pouvez pas rediriger les flux vers des fichiers ordinaires autres que le fichier `tmpfile`, qui est initialement créé dans le répertoire `puzzle3` et vidé avant chaque exécution de la question (nous nous en chargeons pour vous!). Vous ne pouvez pas non plus rediriger des flux vers des périphériques tels que `\dev\null`;
- Vous ne pouvez pas utiliser plusieurs tubes anonymes pour établir une connexion entre la sortie standard d'un processus et l'entrée standard d'un autre. Vous devez vous limiter à un seul tube anonyme;
- Vous ne pouvez pas utiliser de fonctions pour endormir le processus, ni de boucles for ou while;
- Vous ne pouvez pas utiliser la fonction de librairie `system`;
- Vous ne pouvez pas écrire directement dans des fichiers ou des flux (que ce soit avec `write` ou des fonctions comme `printf`), ni créer ou détruire des fichiers; toutes les opérations d'écriture doivent être réalisées par les processus que nous vous fournissons.

Le respect de ces contraintes sera vérifié par nos scripts d'évaluation ainsi que par votre chargé de lab. Attention, vous n'obtiendrez aucun point pour cette question si l'une de ces contraintes n'est pas respectée! Voici la citation originale que vous devrez imprimer sur la sortie standard:

> *One man's crappy software is another man's full time job. (Jessica Gaston)*

Et voici la ligne de commande qui permet – si exécutée depuis le répertoire `puzzle3` – d'imprimer l'entièreté de la citation à l'aide des quatre exécutables:

```
./exc1 2>&1 > tmpfile; ./exc2 2>&1 | ./exc3 2>&1 ; ./exc4 < tmpfile
```

Le but est de reproduire exactement le comportement de cette ligne de commande au moyen de tubes anonymes et de redirections, en utilisant notamment les fonctions `pipe` et `dup2`.

☞ Complétez la fonction `puzzle3` du fichier `puzzle3.c` pour imprimer la citation originale sur la sortie standard. Chaque parent doit attendre la fin de ses fils avant de terminer son exécution.

Instructions

Travailler sur le lab

Toutes vos solutions pour ce lab doivent être écrites dans les fichiers `puzzle1.c`, `puzzle2.c` et `puzzle3.c`. **Seul ce fichier sera pris en compte pour évaluer votre travail; il est donc inutile, voire contre-productif, de modifier les autres fichiers que nous vous fournissons!**

Compiler et exécuter le lab

Nous vous fournissons tous les scripts et librairies pour vous permettre de compiler les sources du lab. Pour compiler le lab initialement et après chacune de vos modifications sur le code source:

```
Console
$ make
```

lorsque vous vous situez dans le répertoire de base du laboratoire. Si la compilation se déroule sans problème, vous pouvez ensuite exécuter le programme:

```
Console
$ ./commlab
```

qui va lancer successivement vos solutions pour les trois puzzles.

Evaluer votre travail

Nous vous fournissons les scripts qui vous permettront d'évaluer votre travail autant de fois que vous le souhaitez. Il vous suffit d'exécuter:

```
Console
$ ./grade.sh
```

pour avoir un rapport détaillé sur votre travail.

Information: Les scripts que nous vous fournissons vous donnent une indication mais pas une garantie sur la note finale que vous obtiendrez. Seule l'évaluation par les serveurs d'Autolab sera prise en compte (mais si vous respectez bien les consignes ci-dessus, les deux notes devraient être identiques!).

Rendre votre travail

Votre travail doit être rendu sur Autolab **avant la fin de cette séance de TP**. Aucune autre forme de remise de votre travail ne sera acceptée. Les retards ou oublis seront sanctionnés comme indiqué sur la page Moodle du cours.

Lorsque vous souhaitez soumettre votre travail – vous pouvez le faire autant de fois que vous le souhaitez pendant la séance –, créez l'archive de remise en effectuant:

```
Console
$ make handin
```

Cela a pour effet de créer le fichier `handin.tar.gz` que vous devrez remettre sur Autolab. Seul le dernier fichier remis sera pris en compte pour l'évaluation finale.

Evaluation

Ce lab est noté sur 20 points, répartis comme suit:

- /5.0 pts: Code du puzzle 1

- /5.0 pts: Code du puzzle 2
- /6.0 pts: Code du puzzle 3
- /4.0 pts: Clarté du code

Une première note sur 16 points vous est donnée par le script d'auto-évaluation (voir ci-dessus) ainsi que par Autolab. N'hésitez pas à exécuter le script d'auto-évaluation pour connaître le barème détaillé. Les 4 points restants seront évalués par la suite par vos chargés de laboratoire, qui vous feront des commentaires via la plateforme Autolab.

Ressources

Ce lab vous laisse beaucoup d'autonomie pour programmer votre solution. Le cours constitue une première ressource pour résoudre ce lab. Si vous avez besoin d'informations sur la syntaxe d'une fonction ou d'un programme en particulier, les *manpages* sont une ressource précieuse.

Pour ce lab, vous pourriez avoir envie de vous documenter sur les fonctions `fork`, `open`, `read`, `write` et `dup2` (entre autres!). Le programme `strace` vous sera certainement très utile également.

Attention: Copier puis coller du code tout prêt à partir d'Internet (par exemple depuis Stack Overflow) n'est pas considéré comme du travail original et peut être considéré comme du plagiat. Les *manpages* et les documentations officielles, en revanche, vous aident à apprendre à construire un code par vous-même. Privilégiez cette solution pour améliorer votre apprentissage, et n'hésitez pas à solliciter votre chargé de laboratoire si vous avez une question.