Programmation Multi-Tâches

Document de travail

Dimitry SOLET

25 avril 2022

ESEO Apprentissage

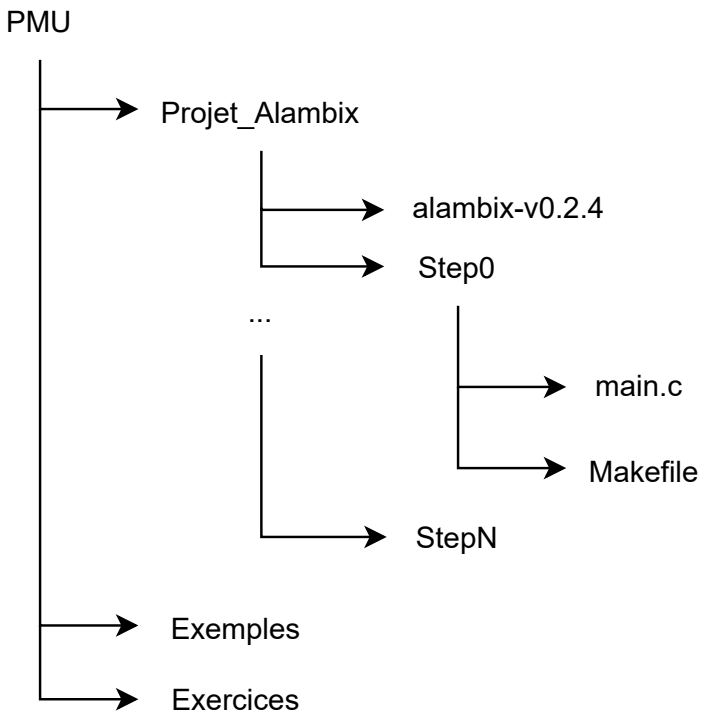
Présentation du module

Résumé :

- Nombre de séances : 8 , Linux, Autonomie

Déroulement en 3 phases :

Cours | **Exploration** | **Alambix**



Début avec Alambix

Objectif pédagogique

Apprendre par l'expérimentation ~~la gestion d'un bar et~~ la programmation multitâche sous un système Unix.

Les dépendances

- gcc, make
- pkg-config, libgtk-3-dev, libx11-dev

Alambix

→ Télécharger et placer la lib d'alambix dans votre dossier de travail.

→ Exécuter la commande `firefox alambix-v0.2.4/doc/html/index.html &`

- Appliquer Step 0.

Les processus 1/4

Cours : Lire les diapositives 1 à 6 (système)

Exploration

Vérifier que le `PID` d'Alambix est le même en console et dans l'IHM.

Cours : Lire la diapositive 6 (programmation)

Exploration

- → Analyser et exécuter `ex_01_fork.c`
- → Écrire un programme (`exo_1.c`) permettant d'exécuter 5 processus fils. Chacun d'entre eux devra afficher 5 fois d'affilé son numéro d'ordre entre 0 et 4.

Les processus 2/4

Cours : Lire la diapositive 7

Exploration

Écrire un programme (`exo_2.c`) permettant de lancer le programme `/usr/bin/baobab` avec `execl`, puis `execv` et `execvp`.

→ Que se passe-t-il au niveau de la console ?

→ Dans la console, faire `CTRL+C`. Qu'observe-t-on ?

Cours : Lire la diapositive 8

Exploration

→ Analyser et exécuter `ex_02_fork_exec.c`

→ Observer le lien de filiation (`ps axj`)

Les processus 3/4

Alambix

Appliquer Step 1.

Cours : Lire la diapositive 9

Exploration

→ Lire la documentation de la fonction `exit()` : `man 3 exit`

→ Analyser et exécuter `ex_04_fork_exec_exit.c`

- Observer le processus fils quand le père se termine : `ps j <id_fils>`

Cours : Lire la diapositive 10

Les processus 4/4

Exploration

Toujours avec `ex_04_fork_exec_exit.c`

- Tuer (commande `kill`) le processus fils puis observer son état avant 15 secondes.
- Observer le processus fils après 15 secondes.

Exploration

Analyser et exécuter `ex_05_fork_exec_wait.c`

- Que peut-on constater sur le moment de terminaison du processus père ?

Écrire un programme qui lance dix fils qui effectuent une ≪ course ≫ et qui affiche à la fin l'ordre des fils (pid et ordre d'activation). Chaque fils effectuera *n* tours d'une boucle vide. *n* est choisit au hasard entre 5000 et 10000 (`exo_3.c`).

Les threads 1/3

Cours : Lire les diapositives 12 à 13

Exploration

Analyser et exécuter `ex_06_pthread_create.c`

- Observer les threads avec la commande `ps mauj`.
- Modifier l'exemple de façon à ce que le thread principal se termine avant les autres threads.

Cours : Lire la diapositive 14

Exploration

Analyser et exécuter `ex_07_pthread_exit.c` pour les deux versions (voir commentaires dans le code)

- Quelle différence peut-on observer entre les deux versions ?

Les threads 2/3

Alambix

Relever le défi niveau 1 (version 1).

→ Appuyer sur `play again`

→ Quels problèmes peut-on constater ?

Cours : Lire la diapositive 15

Exploration

- Analyser et exécuter `ex_08_pthread_join.c`
- Écrire un programme (`exo_4.c`) composé de deux threads « secondaires » :
  - `thread_lecture` : lit des caractères au clavier.
  - `thread_affichage` : affiche les caractères.
  - Les deux threads se terminent si l'utilisateur tape une certaine lettre (préalablement définie ou choisie au hasard) au clavier.

Les threads 3/3

Exploration

- Analyser et exécuter `ex_09_pthread_detach.c`

Alambix

Relever le défi niveau 1 (version 2).

→ Corriger le problème de mémoire.

Cours : Lire les diapositives 16 à 17

Exploration

Analyser et exécuter `ex_10_pthread_attr.c`

Synchronisation : Mutex

Cours : Lire les diapositives 18 à 28

Alambix

Terminer le défi niveau 1 (version 3).

Exploration

(`exo_5.c`) Implémentation d'une variante du jeu du nombre mystère.

- Toutes les *x* secondes le nombre mystère est modifié par l'augmentation ou la diminution d'une valeur *n*.
  - Cette modification est indiquée à l'utilisateur par un message à l'écran.
  - *x* et *n* sont des valeurs aléatoires. *x* est compris entre 1 et 5, et *n* est compris entre 0 et 50.
- Le temps du jeu est limité à 60 secondes.
- Le nombre mystère doit rester entre 0 et 250.

Synchronisation : Condition

Alambix

Relever le défi niveau 2 (version 1).

→ Quel est le problème ?

Cours : Lire les diapositives 31 à 33

Exploration

- Analyser et exécuter `ex_11_pthread_cond.c`
- Proposer une amélioration de l'exercice `exo_4.c`.

Alambix

Relever le défi niveau 2 (version 2).

→ Corriger le problème avec l'utilisation des conditions.

Synchronisation : Barrière de synchronisation

Cours : Lire les diapositives 34 à 35

Exploration

Analyser et exécuter `ex_12_pthread_barrier.c`

Alambix

Relever le défi niveau 2 (version 3) avec une barrière.

→ Repartir du niveau 1 version 3.

Synchronisation : Sémaphore

Cours : Lire les diapositives 36 à 41

Exploration

Analyser et exécuter `ex_13_sem.c`

- Lancer l'exécutable dans deux shells
- Observer le contenu du dossier `/dev/shm`

Alambix

Relever le défi niveau 2 (version 4) avec des sémaphores anonymes.

Exploration

Développer une barrière de synchronisation avec un sémaphore et un mutex. (`exo_6.c`)

Communication : Mémoires partagées

Cours : Lire les diapositives 42 à 47

Exploration

Analyser le code d'exemple se trouvant à l'adresse :

`https://man7.org/linux/man-pages/man3/shm_open.3.html`

Exploration

- Écrire 3 programmes (dossier `exo_shm`) qui se partagent une structure de temps (minute, seconde) en mémoire partagée :
  - `gestion_temps.c` : Initialise la mémoire partagée et toutes les secondes met à jours la structure partagée.
  - `affiche_temps.c` : Affiche la structure à chaque fois que l'utilisateur appui sur la touche 'c'.
  - `fin_temps.c` : Termine les deux autres programmes.

La structure peut (doit) se composer d'autres champs (sémaphore, booléen).

Communication : Files de messages

Cours : Lire les diapositives 48 à 54

Exploration : Analyser et exécuter `ex_15_mq.c`

Alambix : Relever le défi niveau 3

- version 1 : Initialiser une file de message et gérer l'envoie des message.
  - Observer ce qui se passe en redémarrant l'application.
  - Corriger le problème manuellement.
- version 2 : Corriger le problème observé.
  - Interrompre le programme (`Ctrl+C`) après avoir déposé le 1er message, tester le redémarrage de l'application.
- version 3 : Corriger le problème observé.

Alambix : Relever le défi niveau 4

.

Communication : Les tubes

Cours : Lire les diapositives 55 à 57

Exploration : `exo_7.c`, `exo_8.c`, `exo_9.c`

- Écrire un programme qui créé un tube, puis écrit une donnée de type `char*` dedans et enfin lit cette donnée une par une.
- Écrire un programme composé d'un père et d'un fils. Le père récupère une chaîne de 20 caractères au clavier puis l'envoie dans le tube. Le fils récupère la chaîne de caractère puis l'affiche à l'écran.
  - Il faut fermer la partie du tube qui n'est pas utilisée avec `close()`.
- Écrire un programme créant trois processus : deux écrivains et un lecteur. Les deux écrivains écrivent respectivement les séquences *ABC...Z* et *abc...z* par bloc de trois caractères (attendre 3 secondes entre les blocs). Le lecteur lit dans le tube par bloc de 4 caractères et affiche ce qu'il lit.

Communication : Les signaux 1/2

Cours : Lire les diapositives 58 à 59

Exploration : Analyser et exécuter `ex_16_signal_list.c`

Cours : Lire les diapositives 60 à 61

Exploration : Analyser et exécuter `ex_17_fork_exec_wait_signal.c`

- Tester `Ctrl+C`.

Cours : Lire les diapositives 62 à 66

Exploration : Analyser et exécuter `ex_18_fork_exec_wait_sigaction.c`

- Tester `Ctrl+C`.

Communication : Les signaux 2/2

Alambix

Ajouter un gestionnaire de signal pour éviter l'apparition de zombis suite à l'ouverture de l'aide.

Exploration

- `exo_10.c` Écrire un programme qui crée un fils qui fait un calcul sans fin. Le processus père propose alors un menu :
  - L'appuie sur la touche $'s'$ endort le fils.
  - L'appuie sur la touche $'r'$ redémarre le fils.
  - L'appuie sur la touche $'q'$ tue le fils puis termine le père.

Temporisation

Fin des diapositives

Exploration : Analyser et exécuter `ex_19_timer.c`

Alambix : Relever le défi niveau 4

- Utiliser un minuteur (`timer_t`).

Exploration : Développement d'un *Watchdog*

- Développer le code des fonctions pour le *watchdog* (voir `watchdog.h`.
- `exo_10.c` Utiliser le watchdog à la place du *timer* dans l'exemple 19.
- `exo_11.c` Faire évoluer le programme afin que :
  - Le calcul de la boucle main soit arrêté par le *watchdog*.
  - L'appuie sur `Ctrl+C` annule le *watchdog*.

Fin d'Alambix

Alambix : Relever les défis niveau 6 et 7