Programmation Multi-Tâches

Document de travail

Dimitry SOLET

19 avril 2024

Plan

Présentation du module

Objectifs :

- Comprendre les enjeux liés à la programmation multitâche :
  - Pourquoi en faire ? Quelles sont les difficultés ?
- Savoir identifier les situations de concurrence.
- Proposer des constructions permettant de pallier aux situations de concurrence.
- Mettre en œuvre les concepts du multitâche sous Linux.

Modalités pédagogiques :

- Entrelacement de cours magistraux et d'activités pratiques.
  - Cours magistraux : présentation des concepts théoriques.
  - Activités pratiques : mise en application des concepts selon le standard `POSIX`.
- Évaluation lors de la dernière séance.

Fonctionnement des séances

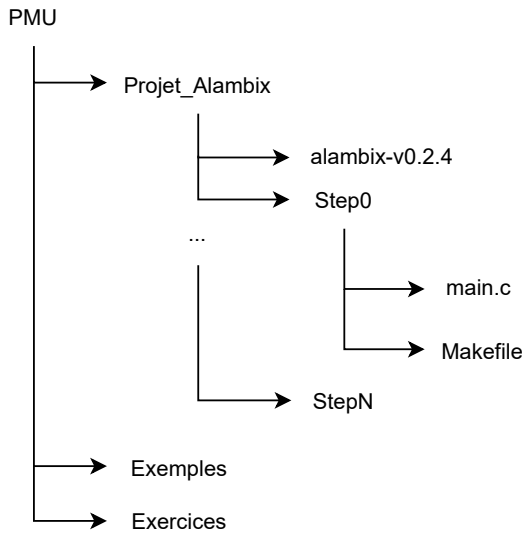
Les activités pratiques :

- *Lecture* : Cours sur la programmation multitâche en langage C selon le standard `POSIX`.
- *Exploration* : Ensemble d'exemples et d'exercices permettant de s'approprier les différentes notions.
- *Alambix* : Application ≪ fil rouge ≫ permettant de mettre en œuvre les différentes notions.

Utilisation d'un code couleur :

Lecture **Exploration** **Alambix**

Organisation du répertoire :



Planning prévisionnel

Séance 1	→ Introduction au module → Les processus
Séance 2	→ Les processus → Les threads
Séance 3	→ Synchronisation et gestion de la concurrence : Mutex, Barrière de synchronisation et Sémaphore
Séance 4	→ Communication : les files de messages → Développement d'une machine à état
Séance 5	→ Communication : les signaux
Séance 6	→ Temporisation → Développement d'un Watchdog
Séance 7	→ Gestion de la terminaison d'un thread en lecture : Les tubes et la fonction `select()`
Séance 8	Évaluation

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.

Plan

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

→ Analyser et exécuter `ex_02_fork_exec.c`

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

Exploration

Écrire un programme (`exo_2.c`) permettant de lancer le programme `/usr/bin/baobab` avec `execl`.

→ 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

→ Analyser et exécuter `ex_03_system.c`

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.

Analyser et exécuter `ex_05_fork_exec_wait.c`

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

Exploration

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`).

Plan

Les threads 1/4

Cours : Lire les diapositives 12 à 13

Exploration

Analyser et exécuter `ex_06_pthread_create.c`

- Observer les threads avec la commande `ps maux`.
- 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 2/4

Alambix

Relever le défi niveau 1 (version 1).

→ Appuyer sur `play again`

→ Quels problèmes peut-on constater ?

Remarque : À ce stade le carnet de commande doit se remplir mais la mission 1 n'est pas validée.

Cours : Lire la diapositive 15

Exploration

- Analyser et exécuter `ex_08_pthread_join.c`

Les threads 3/4

Exploration

- Écrire un programme (`exo_4.c`) qui créé un thread pour calculer la moyenne d'un tableau d'entiers. Ce thread doit calculer la moyenne et le `main` s'occupera d'afficher le résultat.
  - Aucun argument n'est passé au thread.
  - Le thread ne retourne aucune valeur.
  - Il est autorisée d'utiliser des variables globales.
- Modifier le code afin de ne pas avoir de variable globale. Il est maintenant possible de passer un argument au thread et de récupérer sa valeur de retour.

Les threads 4/4

Exploration

- Analyser et exécuter `ex_09_pthread_detach.c`

Alambix

Relever le défi niveau 1 (version 2).

→ Corriger le problème de mémoire.

Remarque : À ce stade la mission 1 n'est pas validée.

Cours : Lire les diapositives 16 à 17

Exploration

Analyser et exécuter `ex_10_pthread_attr.c`

Plan

Synchronisation : Mutex

Cours : Lire les diapositives 18 à 28

Exploration

(`exo_5.c`) Écrire un programme qui crée deux threads. Chaque thread doit incrémenter un compteur partagé 100000 fois. À la fin de l'exécution des threads, le programme doit afficher la valeur du compteur partagé.

- Mettre en évidence le problème observé.

Utiliser un mutex afin de corriger le problème.

Alambix

Terminer le défi niveau 1 (version 3).

Synchronisation : Barrière de synchronisation

Alambix

Relever le défi niveau 2 (version 1 : sans mécanisme de synchronisation).

→ Quel est le problème ?

Cours : Lire les diapositives 34 à 35

Exploration

Analyser et exécuter `ex_12_pthread_barrier.c`

Alambix

Relever le défi niveau 2 (version 2 : avec une barrière de synchronisation).

→ Corriger le problème avec l'utilisation d'une barrière

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 3 : avec des sémaphores **anonymes**.

Plan

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 : Files de messages

Exploration

- Compléter le code de la machine à états de la lampe avec la gestion de la file de messages et du thread (`lampe_single_mae_thread.c`).
  - Valider la bon fonctionnement en comparant avec la version sans thread (`lampe_single_mae_simple.c`).
- Modifier le code de la machine à état de votre projet de *langage C avancé* pour ajouter le thread et la file de messages.

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 (avec `sigaction`).

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.

Plan

Temporisation : Le minuteur

Fin des diapositives

Exploration : Analyser et exécuter `ex_19_timer.c`

Alambix : Relever le défi niveau 5

- Utiliser un minuteur (`timer_t`).

Temporisation : Watchdog

Watchdog

L'objectif est de développer un *watchdog* logiciel en se basant sur l'utilisation d'un minuteur (`timer_t`).

- Encapsulation de la ≪ complexité ≫ du `timer_t` dans un module.
- Pas de notion de période de répétition.
- Module multi-instance.

Ce module pourra être utilisé comme système d'alarme dans vos applications.

Exploration : Développement d'un *Watchdog*

- Compléter le code du module `watchdog` et vérifier son fonctionnement.
- `exo_11.c` Faire évoluer le programme d'utilisation du `watchdog` afin que :
  - Le calcul de la boucle main soit arrêté par le *watchdog*.
  - L'appuie sur `Ctrl+C` annule le *watchdog*.

Plan

Fin d'Alambix

Alambix : Relever les défis niveau 6 et 7