

Programmation Multi-Tâches

Document de travail

Dimitry SOLET

21 mars 2024



- 1 Introduction au module
- 2 Les processus
- 3 Les threads

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'appropriier les différentes notions.
- *Alambix* : Application « fil rouge » permettant de mettre en œuvre les différentes notions.

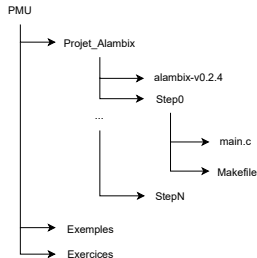
Utilisation d'un code couleur :

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 <code>select()</code>
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

- 1 Introduction au module
- 2 Les processus
- 3 Les threads

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`

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

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

- 1 Introduction au module
- 2 Les processus
- 3 Les threads

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 ?

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`

Exploration

- Écrire un programme (`exo_4.c`) qui crée 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.

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`