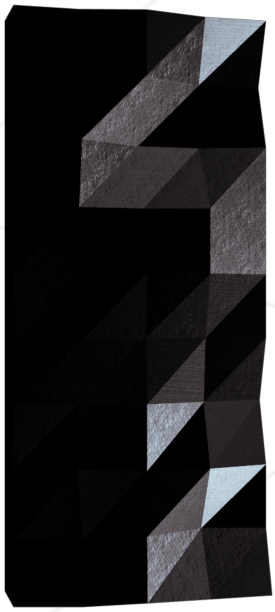

Philosophers

Qui aurait cru que philosopher serait si mortel ?

Résumé:
Ce projet est une introduction au threading et aux processus, et sur comment travailler sur le même espace mémoire.
Vous apprendrez à manipuler des threads.
Vous découvrirez les mutex, les sémaphores et la mémoire partagée.

Version: 9

Table des matières

Chapitre I

Introduction

La philosophie (du grec, *philosophia*, littéralement "amour de la sagesse") est l'étude de questions générales et fondamentales sur l'existence, la connaissance, les valeurs, la raison, l'esprit et le langage. De telles questions se posent souvent comme des problèmes à étudier ou à résoudre. Le terme a probablement été inventé par Pythagore (c. 570 - 495 AEC). Les méthodes philosophiques comprennent le questionnement, la discussion critique, l'argumentation rationnelle et la présentation systématique.

Les questions philosophiques classiques incluent : Est-il possible de savoir quelque chose et de le prouver ? Quel est le plus réel ? Les philosophes posent également des questions plus pratiques et concrètes telles que : Existe-t-il une meilleure façon de vivre ? Vaut-il mieux être juste ou injuste (si on peut s'en tirer comme ça) ? Les humains ont-ils le libre arbitre ?

Historiquement, la "philosophie" englobait tout corpus de connaissances. De l'époque du philosophe grec Aristote au 19ème siècle, la "philosophie naturelle" englobait l'astronomie, la médecine et la physique. Par exemple, les principes mathématiques de la philosophie naturelle de Newton, établis en 1687, ont par la suite été classés dans un livre de physique.

Au 19ème siècle, la croissance des universités de recherche modernes a amené la philosophie académique et d'autres disciplines à se professionnaliser et à se spécialiser. À l'ère moderne, certaines recherches qui faisaient traditionnellement partie de la philosophie sont devenues des disciplines universitaires distinctes, notamment la psychologie, la sociologie, la linguistique et l'économie.

D'autres enquêtes étroitement liées à l'art, à la science, à la politique ou à d'autres activités font toujours partie de la philosophie. Par exemple, la beauté est-elle objective ou subjective ? Existe-t-il plusieurs méthodes scientifiques ou une seule ? L'utopie politique est-elle un rêve porteur d'espoir ou une fantaisie sans espoir ? Les principaux sous-domaines de la philosophie académique comprennent la métaphysique ("soucieux de la nature fondamentale de la réalité et de l'être"), l'épistémologie (concernant "la nature et les fondements de la connaissance [et] ... ses limites et sa validité"), l'éthique, l'esthétique, philosophie politique, logique et philosophie des sciences.

Chapitre II

Règles communes

- Votre projet doit être écrit en C.
- Votre projet doit être codé à la Norme. Si vous avez des fichiers ou fonctions bonus, celles-ci seront inclues dans la vérification de la norme et vous aurez 0 au projet en cas de faute de norme.
- Vos fonctions de doivent pas s'arrêter de manière inattendue (segmentation fault, bus error, double free, etc) mis à part dans le cas d'un comportement indéfini. Si cela arrive, votre projet sera considéré non fonctionnel et vous aurez 0 au projet.
- Toute mémoire allouée sur la heap doit être libéré lorsque c'est nécessaire. Aucun leak ne sera toléré.
- Si le projet le demande, vous devez rendre un Makefile qui compilera vos sources pour créer la sortie demandée, en utilisant les flags `-Wall`, `-Wextra` et `-Werror`. Votre Makefile ne doit pas relink.
- Si le projet demande un Makefile, votre Makefile doit au minimum contenir les règles `$(NAME)`, `all`, `clean`, `fclean` et `re`.
- Pour rendre des bonus, vous devez inclure une règle `bonus` à votre Makefile qui ajoutera les divers headers, librairies ou fonctions qui ne sont pas autorisées dans la partie principale du projet. Les bonus doivent être dans un fichier différent : `_bonus.{c/h}`. L'évaluation de la partie obligatoire et de la partie bonus sont faites séparément.
- Si le projet autorise votre `libft`, vous devez copier ses sources et son Makefile associé dans un dossier libft contenu à la racine. Le Makefile de votre projet doit compiler la librairie à l'aide de son Makefile, puis compiler le projet.
- Nous vous recommandons de créer des programmes de test pour votre projet, bien que ce travail **ne sera pas rendu ni noté**. Cela vous donnera une chance de tester facilement votre travail ainsi que celui de vos pairs.
- Vous devez rendre votre travail sur le git qui vous est assigné. Seul le travail déposé sur git sera évalué. Si Deepthought doit corriger votre travail, cela sera fait à la fin des peer-evaluations. Si une erreur se produit pendant l'évaluation Deepthought, celle-ci s'arrête.

Chapitre III

Vue d'ensemble

Voici les choses à savoir si vous souhaitez réussir cet exercice :

- Des philosophes (un philosophe minimum) sont assis autour d'une table ronde au centre de laquelle se trouve un grand plat de spaghetti.
- Les philosophes sont soit en train de **manger**, de **penser** ou de **dormir**.
  Lorsqu'ils mangent, ils ne peuvent pas penser ou dormir.
  Lorsqu'ils dorment, ils ne peuvent pas manger ou penser.
  Enfin, lorsqu'ils pensent, ils ne peuvent pas manger ou dormir.
- Il y a également des fourchettes sur la table. Il y a **autant de fourchettes que de philosophes**.
- Puisque servir et manger des spaghetti à l'aide d'une seule fourchette n'est pas chose facile, un philosophe prend la fourchette sur sa gauche et celle sur sa droite, soit une fourchette dans chaque main, afin de manger.
- Quand un philosophe a fini de manger, il repose les fourchettes sur la table et se met à dormir. Une fois réveillé, il se remet à penser. La simulation prend fin si un philosophe meurt de faim.
- Chaque philosophe a besoin de manger et ne doit pas mourir de faim.
- Les philosophes ne communiquent pas entre eux.
- Les philosophes ne peuvent pas savoir si l'un d'entre eux est sur le point de mourir.
- Inutile de préciser que les philosophes ne doivent pas mourir !

Chapitre IV

Règles communes

Vous devez faire un programme pour la partie obligatoire et, dans le cas où vous choisissez aussi de faire les bonus, un programme pour la partie bonus. Ces deux programmes doivent respecter les règles suivantes, communes aux deux parties :

- Les variables globales sont interdites !
- Chaque programme doit prendre les arguments suivants :
  `number_of_philosophers time_to_die time_to_eat time_to_sleep [number_of_times_each_philosopher_must_eat]`
  - `number_of_philosophers` : Le nombre de philosophes, mais aussi le nombre de fourchettes.
  - `time_to_die` (en millisecondes) : Si un philosophe n'a pas commencé à manger `time_to_die` millisecondes après le début de son précédent repas ou depuis le début de la simulation, il meurt.
  - `time_to_eat` (en millisecondes) : Le temps qu'un philosophe prend à manger. Pendant ce temps, un philosophe doit garder ses deux fourchettes.
  - `time_to_sleep` (en millisecondes) : Le temps qu'un philosophe passe à dormir.
  - `number_of_times_each_philosopher_must_eat` (argument optionnel) : Si tous les philosophes ont mangé au moins `number_of_times_each_philosopher_must_eat` fois, la simulation prend fin. Si cet argument n'est pas spécifié, alors la simulation prend fin à la mort d'un philosophe.
- Chaque philosophe se voit assigner un numéro entre `1` et `number_of_philosophers`.
- Le philosophe numéro `1` est assis à côté du philosophe numéro `number_of_philosophers`. Les autres suivent cette logique : philosophe numéro `N` est assis entre philosophe numéro `N` - 1 et philosophe numéro `N` + 1.

Concernant les *logs* de votre programme :

- Tout changement d'état d'un philosophe doit être formatté comme suit :
  - `timestamp_in_ms X has taken a fork`
  - `timestamp_in_ms X is eating`
  - `timestamp_in_ms X is sleeping`
  - `timestamp_in_ms X is thinking`
  - `timestamp_in_ms X died`

  Remplacez `timestamp_in_ms` *par le timestamp actuel en millisecondes et* `X` *par le numéro du philosophe.*
- Tout message affiché ne doit pas être mélangé avec un autre message.
- Il ne doit pas y avoir plus de 10 ms entre la mort d'un philosophe et l'affichage du message annonçant sa mort.
- Encore une fois, les philosophes doivent éviter de mourir.

Chapitre V

Partie obligatoire

Nom du programme	`philo`
Fichiers de rendu	`Makefile, *.h, *.c, dans un dossier philo/`
Makefile	`NAME, all, clean, fclean, re`
Arguments	`number_of_philosophers time_to_die time_to_eat time_to_sleep [number_of_times_each_philosopher_must_eat]`
Fonctions externes autorisées	`memset, printf, malloc, free, write, usleep, gettimeofday, pthread_create, pthread_detach, pthread_join, pthread_mutex_init, pthread_mutex_destroy, pthread_mutex_lock, pthread_mutex_unlock`
Libft autorisée	`Non`
Description	`Philosophers avec des threads et des mutex`

Les règles spécifiques à la partie obligatoire sont :

- Chaque philosophe doit être représenté par un *thread*.
- Une fourchette est placée entre chaque paire de philosophes. Cela signifie que, s'il y a plusieurs philosophes, chaque philosophe a une fourchette à sa gauche et une à sa droite. S'il n'y a qu'un seul philosophe, il n'y aura donc qu'une seule fourchette sur la table.
- Afin d'empêcher les philosophes de dupliquer les fourchettes, vous devez protéger leur état en mémoire avec un mutex pour chacune d'entre elle.

memset
printf
malloc
free
write,
usleep
gettimeofday
pthread_create. = Function to init the thread but you create it first with pthread_t
pthread_detach = Je ne suis pas certain de pk je peux prendre ça sans pthread_exit
pthread_join = Pthread_join is like the wait for the forks you pass the pthread_t struct and you can grasp the result of the thread there
pthread_mutex_init = To init a mutex you first have to create it like this pthread_mutex_t mutex; Then you can use mutex_init to init it and use it. I don't know how now to use it as a global variable tho
pthread_mutex_destroy = Used in conjonciton with pthread_mutex_create you just drestroy it at the end in the main.
pthread_mutex_lock = This is actually how you lock something in a thread with this function calling the adress of the mutex
pthread_mutex_unlock = Same thing but it unlocks the thread or action within a thread once its called

Chapitre VI

Partie bonus

Nom du programme	`philo_bonus`
Fichiers de rendu	`Makefile, *.h, *.c, dans un dossier philo_bonus/`
Makefile	`NAME, all, clean, fclean, re`
Arguments	`number_of_philosophers time_to_die time_to_eat time_to_sleep [number_of_times_each_philosopher_must_eat]`
Fonctions externes autorisées	`memset, printf, malloc, free, write, fork, kill, exit, pthread_create, pthread_detach, pthread_join, usleep, gettimeofday, waitpid, sem_open, sem_close, sem_post, sem_wait, sem_unlink`
Libft autorisée	`Non`
Description	`Philosophers avec des processus et des sémaphores`

Le programme de la partie bonus prend les mêmes arguments que celui de la partie obligatoire. Il doit respecter les règles énoncées dans le chapitre *Règles communes*.

Les règles spécifiques à la partie bonus sont :

- Toutes les fourchettes sont au centre de la table.
- Elles n'ont pas d'état spécifique en mémoire, mais le nombre de fourchettes disponibles est représenté par un sémaphore.
- Chaque philosophe est représenté par un processus différent. Cependant, le processus principal ne doit pas être un philosophe.

Les bonus ne seront évalués que si la partie obligatoire est PARFAITE. Par parfaite, nous entendons complète et sans aucun dysfonctionnement. Si vous n'avez pas réussi TOUS les points de la partie obligatoire, votre partie bonus ne sera pas prise en compte.

Chapitre VII

Rendu et peer-evaluation

Rendez votre travail sur votre dépot `Git` comme d'habitude. Seul le travail présent sur votre dépot sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

Dossier de la partie obligatoire : `philo/`

Dossier de la partie bonus : `philo_bonus/`