Bilan multitâches

Résumé

L'exécution en simultanée de tâche (processus ou thread) n'est pas une affirmation réel.
Les tâches on un temps partagé, à un instant qu'une seul tâche est exécutée.
L'ordonnanceur (scheduler), donne la main successivement à chaque tâche.
Le contexte d'exécution est sauvegarder
Une tâche peut être interrompue à tout moment.
Un appel système (read, fork, ...) n'est pas interruptible (c'est à dire atomique)

Processus OU Threads

1. Contexte d'exécution d'un thread est plus léger. => Chargement plus performant.
2. Processus = exécution séparées
   Threads = exécution non séparées => "plantage" d'un thread => arrêt de tout les threads
3. Les threads (à l'intérieur d'un processus) se partage les données (global et malloc). /!\ Accès concurrent /!
   Les processus ont des données séparées.

Serveur multithread

Serveur à modèle dynamique

- Pour chaque connexion client : création d'un thread pour gérer la communication avec le client.
  => Thread principal : se remet tout de suite en attente de connexion.

Ce type de serveur est appeler : Serveur à worker/modèle (=thread) dynamique.

- Juste le nombre de thread nécessaire.
- Pas de besoin de synchronisation entre le thread principal et les worker (thread).
- Pas sûr de pouvoir créer le thread. (si il y a saturation du nombre de thread créer sur le système), pas sûr de pouvoir assurer le service.

Serveur à modèle statique

Dans cette autre type de serveur multithread. Les thread gérant les clients sont créer au début. Donc nous devons fixer une limite.
Mécanisme :

- Les thread worker, attendent l'ouverture d'une connexion à gérer, et se remet en sommeil quand le client se déconnecte.
- Le thread principal, choisi un worker et le réveille sur une arrivée d'une connexion.

Ce type de serveur permet que l'on est sur de pouvoir assurer le service.
Cependant il faut donc mettre en place un mécanisme de synchronisation entre le trhead principal et les worker.

Mécanisme de synchronisation

Création d'un `#define` d'une nombre de worker.
Création d'un tableau de données de spécification d'un worker.

main :

```
canal = accept()
selectinner un worker dispo
le reveiller et lui passe le canal
Si pas disponible : tempo // implementer à la fin.s
```

thread : (worker)

```
Boucle infinie
        mise en attente :
                Tant que canal à -1: tempi

        Lorsque reveiller, gérer la session client

        indiquer disponible : remet son connal à -1
```

Pour savoir si un thread est disponible, il faut parcourir le tableau de données spécifique et regarder si la variable cannal est égale à -1.
Pour réveiller un worker (un thread) il faut lui indiquer un canal à gérer.