Exécution asynchrone

1) Définition du besoin

Un logiciel moderne, avec une inter-activité accru avec l'utilisateur, prends en compte de plus en plus d'actions asynchrones.

Une action est asynchrone si elle est en attente d'une ressource et que l'exécution principale n'a pas été bloquée. L'exécution de l'action reprend donc à un moment indéterminé donc asynchrone de l'exécution principale.

2) Première solution

Souvent ces actions sont le plus souvent reléguées au système par des appels de bas niveau qui présentent au mieux des mécanismes permettant d'exécuter des instructions pour reprendre le contrôle en fin d'action. La conséquence essentielle est le morcellement du programme en plusieurs parties difficilement maîtrisées car décousues et difficilement testable de par leur nature d'accès par des pointeurs bien souvent.

3) Solution moderne

À l'heure actuelle les systèmes modernes et les environnements de développement contenant des bibliothèques d'exécution (multimédia par exemple) présentent la notion de tâche élémentaire (traduction approximative du mot anglais "thread").

Cette notion est bien plus précise que la notion générale de tâches pour une application qui représentent des fonctionnalités identifiées d'ordinaires plus complexe. Une tâche au sein d'une application peut comporter un grand nombre de tâche élémentaires. La distinction est réelle même si le support de programmation est souvent le même. Pour toute sorte d'utilisation, les tâches élémentaires effectuent des actions asynchrones maîtrisées puisque s'intégrant dans le multi-tâche de l'application (notion de priorité, d'ordonnancement, d'attribution de l'exécution processeur) et non plus sur le support délicats de la hiérarchie des priorités d'interruption.

Une simple requête asynchrone sera attribué à une tâche élémentaire qui l'exécutera de façon séquentielle d'où une meilleure testabilité. Bien sûr, elle se bloquera en attente dune ressource comme un périphérique mais la tâche principale de l'application pourra s'exécuter en toute quiétude car les deux exécutions sont sous contrôle du multi-tâche applicatif.

4) Les limites

Face à cette expansion actuelle des logiciels interactifs auquel réponds de plus en plus de système, de langage et d'environnement de programmation, il manque cruellement de dévermineur réellement multi-tâche. Les dévermineurs actuels se contentent au mieux de geler l'application en cours et d'exécuter localement des pas à pas, alors qu'ils pourraient proposer d'atteindre individuellement chaque tâche du programme.

Le dévermineur doit visualiser l'état d'une tâche avec ses caractéristiques (mémoire consommée tas, pile, code; prochaine instruction à être exécutée; historique des appels) et proposer des commandes pour agir sur l'état en cours. Le dévermineur doit pouvoir lancer l'exécution d'une application ou prendre le contrôle d'une en cours d'exécution.

Il est clair que certains systèmes ont de l'avance comme Unix ainsi qu'un langage comme Ada où le multi-tâche est intégré au langage. Mais souvent les uns sont proposés en dehors des autres, limitant ainsi la puissance nécessaire pour déverminer des applications de plus en plus complexes en terme d'asynchronisme.

Le triplet gagnant pourrait être Mac OS X avec gnat et gdb.

Pascal Pignard, décembre 1996, juin 2002.