15
RAII, le mal nommé

RAII est l'acronyme de *Ressource Acquisition Is Initialization*. Il tend à promouvoir l'idée (parfaitement juste au demeurant) que chaque classe devrait être responsable des ressources qui la composent.

Mais s'il est bien sûr absolument nécessaire de s'assurer que chaque objet dispose des ressources dont il a besoin, ce que ne dit pas cet idiome, c'est qu'il faut surtout veiller à ce que la destruction d'un objet occasionne la libération de toutes les ressources dont il a la charge.

En effet, si le fait pour un objet de ne pas disposer des ressources dont il a besoin est assez embêtant en soi, il est pire encore qu'un objet ne libère pas toutes les ressources qu'il a utilisées en temps opportun, surtout si l'application est destinée à fonctionner comme un service.

Qu'il s'agisse du *lock* sur un fichier, d'un *mutex* qui occasionne un *dead lock* dans un système multithreadé, ou d'une ressource pour laquelle vous avez eu recours à l'allocation dynamique de la mémoire, il est primordial de veiller à libérer toutes les ressources dès que possible.

Gardez en effet toujours à l'esprit que, bien que les ordinateurs actuels disposent de ressources de plus en plus importantes, celles-ci ne sont **jamais** illimitées et que, tôt ou tard, elles viendront **fatalement** à manquer si vous ne les gérez pas correctement.

Bien sûr, vous arriverez beaucoup plus facilement à manquer de ressources sur un antique 386 qui ne dispose que de 1 à 4 Mb de mémoire, mais la seule différence avec votre quad core doté de 16 Gb de mémoire ou plus est que vous mettrez simplement plus longtemps à vous retrouver dans la même situation. Or, le manque de ressources aura strictement le même effet sur un antique 386 que sur votre quad core : la stabilité globale de tout le système sera compromise et risque de provoquer le plantage pur et simple de l'ordinateur.

Le terme RAII devrait donc régulièrement être mis en parallèle avec un autre terme, trop peu utilisé à mon goût : le *RFID* (*Resources Finalization is Destruction*) qui désigne selon moi beaucoup plus clairement la problématique à laquelle on est confronté.

L'ensemble des classes proposées par la bibliothèque standard de C++ présente cette caractéristique de savoir exactement ce qu'elles doivent faire des ressources qu'elles manipulent et quand elles doivent le faire. C++11 est d'ailleurs venu avec un ensemble de classes RAIIsantes qui permettent à leur utilisateur d'avoir la certitude que les ressources allouées à ses propres classes seront libérées en temps et en heure, l'aidant à développer du code résistant aux exceptions et par conséquent beaucoup plus robuste.

C'est dans cet ordre d'idée que je vous recommande l'utilisation des classes comme `std::string`, `std::vector` et autres collections fournies par la bibliothèque standard.