Labo 6 – Polymorphisme et spécialisation

1. Présentation

1.1. Objectifs pédagogiques

- Appliquer le polymorphisme à un cas de spécialisation.
- Passer du monde procédural des API Windows à une encapsulation orientée objet C++.

1.2. Recommandations

- Dans un premier temps, les classes Thread et Mutex seront développées dans un programme de test séparé, permettant d'en faire la mise au point.
- Ensuite, elles seront intégrées dans l'application de commande de machine.

2. Tâches à effectuer

2.1. Objectif

Pour faciliter le contrôle de la machine par l'utilisateur, on souhaite que la commande du simulateur pendant le gravage soit faite par une tâche parallèle (thread séparé). Dans le même temps, l'utilisateur peut demander l'interruption prématurée de l'exécution.

2.2. Création de la classe Thread

Programmer une classe abstraite Thread encapsulant la notion de Thread Win32. Prévoir une méthode virtuelle pure protégée appelée « Execute ». Cette méthode sera appelée dans le contexte du Thread. Elle sera surchargée dans chaque sous classe et contiendra les instructions à exécuter dans le contexte du Thread. Ainsi, il sera possible de créer des sous classes de Thread spécialisées pour une tâche particulière par héritage et surcharge de la méthode Execute.
Les API Windows suivantes doivent être utilisées :

2.2.1. Création d'un thread

Il faut inclure <windows.h> pour disposer des fonctions, types et constantes de gestion de threads. Le type HANDLE est utilisé pour désigner un objet du système d'exploitation, comme un thread par exemple.

La fonction threadStart ci-dessous est une fonction à programmer, passée en paramètre à l'API CreateThread, et qui sera exécutée dans le contexte du thread à son démarrage. Elle peut être nommée différemment, mais doit comporter prototype exact mentionné.

```
DWORD WINAPI Thread::threadStart(LPVOID parameter);

HANDLE handle;
handle = CreateThread(NULL, 0, threadStart, parameter,
                      CREATE_SUSPENDED, &threadId);
```

2.2.2. Démarrage ou redémarrage d'un thread

```
ResumeThread(handle);
```

2.2.3. Suspension

```
SuspendThread(handle);
```

2.2.4. Attente de la fin d'un thread

Le temps d'attente « timeout » spécifié est en millisecondes, on peut utiliser la constante INFINITE pour attendre indéfiniment.

```
WaitForSingleObject(handle, timeOut);
```

2.2.5. Libération des ressources utilisées par un thread (destruction)

Le Thread ne doit plus être en exécution lors de l'appel de cette API.

```
CloseHandle(handle);
```

2.3. Création de la classe Mutex

Les applications multi-tâches demandent un soin particulier lors de l'accès concurrent à des ressources partagées.

En utilisant les API Windows ci-dessous, écrire une classe Mutex permettant de synchroniser l'accès aux ressources partagées :

2.3.1. Création du mutex

```
handle = CreateMutex(NULL, FALSE, NULL);
```

2.3.2. Destruction du mutex

```
CloseHandle(handle)
```

2.3.3. Attente de l'acquisition du mutex

La fonction WaitForSingleObject retourne la valeur WAIT_OBJECT_0 si le mutex a pu être acquis dans le temps à disposition, ou une valeur différente si le temps d'attente a été dépassé ou le mutex détruit.

```
WaitForSingleObject(handle, timeout);
```

2.3.4. Relâchement du mutex

```
ReleaseMutex(handle)
```

2.4. Programme de test

Créer une sous classe de Thread ayant les caractéristiques suivantes :

- Un nom de tâche (string)
- Une périodicité

Les instances de ce thread doivent afficher leur nom sur la console cout avec la périodicité paramétrée. Protéger l'accès à cout en utilisant un mutex global.

Créer 2 instances de cette classe de thread, démarrer les threads, et vérifier dans la console que chaque Thread affiche son nom avec la périodicité attendue.

2.5. Parallélisation de l'exécution du gravage

Intégrer les classes thread et mutex dans le programme de gravage.

Créer un thread spécialisé dans l'exécution de la séquence de gravage des formes géométriques.

Ce Thread attend de recevoir un ordre de démarrage du gravage. A ce moment, il exploite la liste des formes. L'interface utilisateur ne permet alors plus à l'utilisateur de modifier la liste pendant toute la durée du gravage. Par contre, l'utilisateur peut demander l'interruption de la séquence de gravage. Dans ce cas, le thread de gravage reçoit une demande d'interruption. Il termine le gravage de la forme géométrique en cours et arrête ensuite le gravage. Il se remet en attente d'un ordre de démarrage.

3. Travaux à rendre

3.1. Livrables

- Placez l'intégralité du dossier contenant votre projet Visual C++ 2008 dans le dossier dépôt étudiants.
- Une version imprimée du programme comportant votre nom et prénom.

3.2. Délai

Les travaux sont à rendre impérativement en l'état à la fin de la séance, et terminés après une semaine.