Travail Pratique 3

Gestion de mémoire

À remettre, par le portail du cours le 10 décembre 2024 avant 23h59

Ce travail doit être fait en équipe de 2 ou 3.

1 Objectif

Ce travail consiste à simuler un système de gestion de la mémoire virtuelle. L'objectif est de reproduire un environnement où des programmes sont chargés dans la mémoire (RAM) ou déplacés dans la mémoire virtuelle lorsque la RAM est pleine, tout en respectant les règles de lecture et d'écriture. Il doit aussi définir pour chaque programme une section d'instructions (code) qui permet un accès en lecture uniquement, et une deuxième section pour les données dont accès se fait en lecture et écriture.

2 Travail à faire

Vous devez utiliser obligatoirement le fichier "**GabaritRapportTP3.docx**" pour le rapport, afin d'uniformiser la présentation pour le correcteur. Écrivez les noms et matricules au début du document. Lors de votre remise sur le portail du cours, incluez ce rapport ainsi que les codes sources, zippés dans un seul fichier (voir section 3). Notez que le code doit compiler sur la machine virtuelle fournie, puisque cette machine est celle du correcteur. Des erreurs de compilations entraîneront des pénalités pouvant aller jusqu'à 50 %. Notez également que 10 points sont donnés pour le respect et la qualité des biens livrables. Nous vous demandons d'implémenter le mini-système de gestion de mémoire en utilisant le langage C++. Nous vous conseillons d'utiliser les normes C++17. Assurez-vous que votre code est suffisamment commenté pour nous aider dans la compréhension.

2.1 Résumé des taches à effectuer

On pourrait résumer les taches comme suit :

1. Allouer dynamiquement de la mémoire dans le **heap** pour la RAM et la mémoire virtuelle.
2. Charger et exécuter plusieurs programmes en mémoire.
3. Gérer le swapping de programmes entre la RAM et la mémoire virtuelle.
4. Protéger les accès en lecture et écriture aux segments de mémoire des programmes.

Rappel : La **RAM** (Random Access Memory) est une mémoire rapide, mais limitée, qui sert de stockage temporaire pour les programmes et les données nécessaires à l'exécution active d'applications. Lorsque le processeur exécute un programme, les données et les instructions du programme sont chargées en RAM pour un accès rapide. La RAM est volatile, ce qui signifie que toutes les données qu'elle contient sont perdues lorsque l'ordinateur est éteint.

La **mémoire virtuelle** est une technique permettant de pallier la limite de capacité de la RAM. Elle utilise une partie du stockage de l'ordinateur (comme le disque dur ou le SSD) pour simuler une extension de la RAM. Cela permet aux systèmes d'exploitation de gérer des programmes qui nécessitent plus de mémoire que celle disponible physiquement en RAM.

En pratique, lorsque la RAM est pleine, le système déplace certaines données de la RAM vers la mémoire virtuelle. Ce processus est appelé **swapping**. Les données déplacées sont stockées dans un fichier spécial (souvent appelé fichier d'échange ou swap file) sur le disque dur. Cela libère de l'espace en RAM pour d'autres programmes ou données. Quand le programme a de nouveau besoin des données qui ont été swappées, elles sont ramenées en RAM.

2.2 Exigences Techniques

- **Mémoire :**
  - RAM **:** 4 Mo (simulée)
  - Mémoire virtuelle **:** 10 Mo (simulée aussi)

- **Concepts à utilisés :**
  - Classes et gestion dynamique de la mémoire
  - Structures de données (unordered_map, vector)
  - Accès et permissions mémoire

2.3 Fonctionnalités Requises

1. Initialisation de la RAM et de la Mémoire Virtuelle

- Simulez la RAM et la mémoire virtuelle en utilisant la structure de donnée approprié.
- Utilisez des tailles fixes : 4 Mo pour la RAM et 10 Mo pour la mémoire virtuelle.

2. Création et Simulation de Programmes

- Chaque programme est représenté par un vecteur contenant des instructions.
- Créez des programmes avec des tailles et instructions variables pour tester le comportement de la mémoire virtuelle.

3. Gestion de la Mémoire et Swapping

- **Chargement des Programmes :**

Lorsqu'un programme est chargé, vérifiez si suffisamment d'espace contigu est disponible dans la RAM. Si la RAM n'a pas assez d'espace pour charger un nouveau programme, effectuez un swapping : transférez un programme de la RAM vers la mémoire virtuelle.

- **Swapping :**

Choisissez un processus en RAM (par exemple, le moins récemment utilisé ou LIFO ou FIFO) et déplacez-le vers la mémoire virtuelle. Libérez ensuite l'espace RAM occupé par le programme swappé pour charger le nouveau programme.

4. Vérification des Permissions de Lecture et Écriture :

Chaque programme doit être chargé avec des permissions d'accès :

- Instructions : en lecture seule
- Données : en lecture/écriture

Toute tentative de modification des instructions devrait être bloquée.

5. Affichage de l'État de la Mémoire

Affichez en temps réel l'état de la RAM et de la mémoire virtuelle. En incluant :

- La liste des programmes en RAM et en mémoire virtuelle.
- La taille de chaque programme ainsi que l'adresse de début dans la mémoire
- Le statut(optionnel).

2.4 Classes et Structure de Code

Vous devriez implémenter les classes associées afin d'avoir un code clean dont chaque classe effectue un traitement particulier.

1. MemorySegment

Représente un segment de la mémoire alloué à un programme, avec un accès contrôlé en lecture et écriture.

2. Program

Contient le nom, la taille et les instructions d'un programme. Il est a noté que 20% de la taille du

programme sera consacré à son segment d'instructions et le reste pour le segment des données.
Gère l'état du programme (chargé en RAM ou swappé en mémoire virtuelle).

3. MemoryManager

Gère les opérations de chargement, de swap et de libération de mémoire.
Suivi de la RAM et de la mémoire virtuelle.
Vérifie la disponibilité de la RAM et applique les permissions d'accès.
Teste et valide pour chaque segment d'un programme (instructions et données) si l'accès en écriture ou lecture est autorisé.

Important : lors de la manipulation des segments des programmes ainsi que leurs emplacements dans la RAM, il faut s'assurer de bien calculer des index de début et de fin pour ne pas avoir un écrasement ou chevauchement entre les programmes

3 Ce que vous devez rendre

Vous devez rendre un fichier **.zip** comportant **uniquement** les fichiers suivants :

- Le fichier **RapportTP3.pdf** contenant votre rapport
- Les fichiers **main.cpp, MemoryManager.h MemorySegment.h, Program.h ainsi que les .cpp respectives** contenant votre code. Assurez-vous de commenter votre code raisonnablement. Un code mal documenté pourra être pénalisé.
- Le fichier **NoteTp3.xls** contenant le barème (ajoutez simplement les noms et matricules afin de faciliter le travail du correcteur).

Vous ne devez en aucun cas ajouter un autre fichier ou répertoire. N'incluez aucun exécutable, ni aucun fichier généré par votre environnement de développement. De plus, il est de votre responsabilité de vérifier après la remise que vous nous avez envoyé les bons fichiers (**non vides et non corrompus**), sinon vous pouvez avoir un zéro.

Tolérance zéro vis-à-vis des erreurs de compilation :
Vos programmes doivent contenir zéro erreur de syntaxe. Un programme qui ne compile pas, peu importe la raison, sera fortement pénalisé.

Plagiat : Tel que décrit dans le plan de cours, le plagiat est interdit. Une politique stricte de tolérance zéro est appliquée en tout temps et sous toutes circonstances.

Attention! Tout travail remis en retard se verra pénalisé. Le retard débute dès la limite de remise dépassée (dès la première minute). Un retard excédant une journée provoquera le rejet du travail pour la correction et la note de 0 pour ce travail.

Bon travail !