

# TP #5: Mem Lab

Ecole Polytechnique de Montréal

Hiver 2019 — Durée: 2 heures et 10 minutes

## Présentation

Le *Mem Lab* a pour but de vous familiariser avec les mécanismes d'un système d'exploitation qui permettent la gestion de la mémoire virtuelle. Il est composé d'une unique question principale ainsi que d'une question bonus. **Il est conseillé de lire l'énoncé en entier avant de commencer le TP.**




**Attention:** Ce cours a une politique bien définie en termes de plagiat. L'archive que vous avez téléchargée sur Autolab a été générée pour votre groupe seulement. Il vous est interdit de partager votre code ou le code qui vous a été fourni, que ce soit en ligne (via un dépôt Git public par exemple) ou avec d'autres étudiants. Votre code sera systématiquement vérifié et le plagiat sera sanctionné. Afin d'éliminer tout doute quant à ce qui peut constituer du plagiat et pour connaître les sanctions, veuillez vous référer à la page Moodle du cours.

## Objectifs

A l'issue du TP, vous serez capables de:

- Effectuer la conversion entre adresses virtuelles et adresses physiques;
- Gérer le maintien des informations dans une table de pages et un *Translation Lookaside Buffer*;
- Appliquer une politique de remplacement donnée dans une table de pages ou un *Translation Lookaside Buffer* pleins.

## Enoncé

Question	Contenu		Barème
1	Traduction d'adresses virtuelles; gestion des défauts de page; politiques de remplacement pour TLB ( <i>Translation Lookaside Buffer</i> ) et tables de pages	2h10mn	16.0 pt

Vous allez implémenter un simulateur de MMU (*Memory Management Unit*). Pour rappel, il s'agit du composant matériel de votre ordinateur – géré par le système d'exploitation – qui est chargé de la conversion des adresses virtuelles, telles que vues par les processus en cours d'exécution, en adresses physiques, telles qu'utilisées par la mémoire centrale. Voici les informations que vous devrez considérer pour traiter la question:

- **La taille des pages gérées par votre MMU est de  $2^{15} = 32768$  octets;**
- Pendant la simulation, **un maximum de  $2^9 = 512$  pages**, numérotées  $0, \dots, 512 - 1$  seront utilisées. Vous pourrez limiter la taille de votre table de pages à ce nombre;
- Votre MMU possède **un TLB (*Translation Lookaside Buffer*) pouvant contenir  $2^4 = 16$  entrées**, de manière à accélérer la traduction des adresses;
- Le système a déterminé que pour la durée de la simulation, **seules 175 cadres sont libres en mémoire centrale**. Le premier numéro de cadre disponible en mémoire centrale est le numéro 270; tous les cadres disponibles en mémoire sont consécutifs et seront alloués par ordre croissant de numéro de cadre. Autrement dit, les pages lues par votre processus seront stockées en mémoire centrale **entre les adresses 0x870000 et 0xde7fff incluses** (faites le calcul pour vérifier!);
- Si le TLB est plein et que vous devez y insérer une traduction d'adresse, **vous utiliserez la politique de remplacement FIFO (First In First Out) pour sélectionner la traduction à supprimer;**
- Si la mémoire est pleine et que vous devez insérer une page en mémoire, **vous utiliserez la politique de remplacement FIFO (First In First Out) pour sélectionner la page à remplacer;**
- Si une page X **présente dans le TLB** est retirée de la mémoire où elle est remplacée par une page Y, alors **le TLB est mis à jour pour remplacer X par Y;**
- On considèrera qu'il n'y a pas de changement de contexte pendant l'exécution de la simulation; une seule table de page sera utilisée et le TLB ne sera jamais vidé;
- On ne considère pas non plus la présence de mémoire cache.

Voici en quoi consiste une simulation:

1. Nous appellerons la **fonction `initMemoryState`, définie dans le fichier `memsim.c` et que vous devrez compléter**. Vous utiliserez cette fonction pour allouer toutes les structures de données que vous utiliserez pour sauver à chaque instant de la simulation l'état de la mémoire. Vous devrez renvoyer un pointeur vers une **structure de type `struct paging_system_state` qui est définie dans le fichier `memsim.h`**.
2. Autant de fois que nous avons des adresses virtuelles à traduire, nous appellerons la **fonction `processMemoryRequest`, définie dans le fichier `memsim.c` et que vous devrez compléter**. Cette fonction prend comme paramètres un pointeur vers votre structure sauvegardant l'état de la mémoire, ainsi qu'un pointeur vers une structure de type `struct memory_request` telle que définie dans le fichier `memsim/libmemsim.h`. Cette dernière contient notamment l'adresse virtuelle à traduire ainsi que des champs que votre fonction modifiera pour indiquer ce qui s'est passé pendant la traduction (y a-t-il eu un défaut de page, quelle est l'adresse physique renvoyée, etc.). Le détail des champs à modifier est spécifié dans le fichier `memsim/libmemsim.h`.
3. Une fois que la simulation est terminée, nous appellerons la **fonction `cleanMemoryState`, définie dans le fichier `memsim.c` et que vous devrez compléter**. Cette fonction devra désallouer toutes les structures de données que vous avez initialisé pour la simulation.

Pour chaque simulation, vous considèrerez qu'aucune page du processus en cours d'exécution ne se trouve initialement en mémoire centrale. La table de page ainsi que le TLB sont donc initialement vides et la toute première requête conduira forcément à un défaut de page, dont le traitement permettra de stocker la page voulue en mémoire centrale et de mettre à jour la table des pages et le TLB en conséquence.

Nous vous fournissons dans le fichier `memsim.h` des définitions possibles pour les structures de données que vous aurez à utiliser; **vous êtes libres de modifier ces structures de données ou de les utiliser**

telles quelles.

Votre code sera testé sur quatre simulations de difficulté croissantes et décrites dans les fichiers `memsim/testCase*.txt`:

- La simulation `memsim/testCase1.txt` est conçue pour ne jamais remplir complètement ni le TLB ni la mémoire centrale; vous ne devriez donc pas observer de remplacement dans le TLB, de remplacement en mémoire ou de défaut de page.
- La simulation `memsim/testCase2.txt` est conçue pour ne jamais remplir complètement la mémoire centrale, mais elle provoquera des remplacements dans le TLB. Vous ne devriez tout de même pas observer de remplacement en mémoire ou de défaut de page.
- La simulation `memsim/testCase3.txt` est conçue pour provoquer tous types de remplacement ainsi que des défauts de page.
- La simulation `memsim/testCase4.txt` a les mêmes caractéristiques que la précédente, mais elle est beaucoup plus fournie.

Si vous constatez des erreurs dans votre simulateur en lisant vos scores, l'information ci-dessus peut être très utile pour déboguer votre programme. **Nous vous conseillons de commencer à programmer une solution simple qui permet de réussir la première simulation; vous pourrez ensuite améliorer progressivement votre solution pour réussir les simulations suivantes.**

#### Question 1 (Simulateur de MMU) environ 2h10mn

Pour voir le score obtenu par votre simulateur au fur et à mesure que vous le développez, exécutez le programme `./memlab`, qui vous affichera le score pour chacune des quatre simulations.


Vous avez également la possibilité d'obtenir (beaucoup) plus de détail en sachant exactement ce qui était attendu pour chaque requête de la simulation. Pour avoir ce niveau de détail pour le fichier de simulation `memsim/testCase1.txt`, exécutez dans une fenêtre de terminal:


```
./memlab 1 memsim/testCase1.txt | less -r
```


Pour lister seulement les requêtes pour lesquelles vous obtenez des résultats différents de ce qui est attendu, utilisez plutôt:

```
SHOW_FAIL_ONLY=1 ./memlab 1 memsim/testCase1.txt | less -r
```

Vous pourrez ainsi faire défiler chacune des requêtes et comparer ce que votre simulation obtient à ce qui était attendu. Pour quitter la vue du programme `less`, appuyez simplement sur la touche `q` du clavier. Vous pouvez également remplacer `memsim/textCase1.txt` par un fichier de test que vous avez écrit.

 Complétez les fonctions `initMemoryState` et `cleanMemoryState` dans le fichier `memsim.c`.

 Complétez les fonctions `getPageOffset`, `getPageNumber` et `getStartPageAddress` dans le fichier `memsim.c`. Le rôle de ces fonctions est précisé en commentaire. Il n'est pas nécessaire d'utiliser ces fonctions pour la suite de l'exercice, mais la correction de ces fonctions sera évaluée dans la note.

 Afin de vous guider dans l'exercice, nous vous fournissons un modèle pour la fonction `processMemoryRequest`. Complétez les fonctions requises par `processMemoryRequest` dans le fichier `memsim.c`.

# Instructions

## Travailler sur le TP

Toutes vos solutions pour ce TP doivent être écrites dans les fichiers `memsim.c` et `memsim.h`. **Seuls ces deux fichiers seront pris en compte pour évaluer votre travail; il est donc inutile, voire contre-productif, de modifier les autres fichiers que nous vous fournissons!**

## Compiler et exécuter le TP

Nous vous fournissons tous les scripts et librairies pour vous permettre de compiler les sources du TP. Pour compiler le TP initialement et après chacune de vos modifications sur le code source:

```
Console
```

```
$ make
```

lorsque vous vous situez dans le répertoire de base du laboratoire. Si la compilation se déroule sans problème, vous pouvez ensuite exécuter le programme:

```
Console
```

```
$ ./memlab
```

qui va tester votre solution pour le TP.

## Evaluer votre travail

Nous vous fournissons les scripts qui vous permettront d'évaluer votre travail autant de fois que vous le souhaitez. Il vous suffit d'exécuter:

```
Console
```

```
$ ./grade.sh
```

pour avoir un rapport détaillé sur votre travail.



**Information:** Les scripts que nous vous fournissons vous donnent une indication mais pas une garantie sur la note finale que vous obtiendrez. Seule l'évaluation par les serveurs d'Autolab sera prise en compte (mais si vous respectez bien les consignes ci-dessus, les deux notes devraient être identiques!).

## Rendre votre travail

Votre travail doit être rendu sur Autolab **avant la fin de cette séance de TP**. Aucune autre forme de remise de votre travail ne sera acceptée. Les retards ou oublis seront sanctionnés comme indiqué sur la page Moodle du cours.

Lorsque vous souhaitez soumettre votre travail – vous pouvez le faire autant de fois que vous le souhaitez pendant la séance –, créez l'archive de remise en effectuant:

```
Console
```

```
$ make handin
```

Cela a pour effet de créer le fichier `handin.tar.gz` que vous devrez remettre sur Autolab. Seul le dernier fichier remis sera pris en compte pour l'évaluation finale.

## Evaluation

Ce TP est noté sur 20 points (comprenant un point bonus), répartis comme suit:

- /16.0 pts: Simulateur de MMU
- /4.0 pts: Clarté du code

Une première note sur 16 points vous est donnée par le script d'auto-évaluation (voir ci-dessus) ainsi que par Autolab. N'hésitez pas à exécuter le script d'auto-évaluation pour connaître le barème détaillé. Les 4 points restants seront évalués par la suite par vos chargés de laboratoire, qui vous feront des commentaires via la plateforme Autolab.

## Ressources

Ce TP vous laisse beaucoup d'autonomie pour programmer votre solution. Le cours constitue une première ressource pour résoudre ce TP. Si vous avez besoin d'informations sur la syntaxe d'une fonction ou d'un programme en particulier, les *manpages* sont une ressource précieuse.



**Attention:** Copier puis coller du code tout prêt à partir d'Internet (par exemple depuis Stack Overflow) n'est pas considéré comme du travail original et peut être considéré comme du plagiat. Les *manpages* et les documentations officielles, en revanche, vous aident à apprendre à construire un code par vous-même. Privilégiez cette solution pour améliorer votre apprentissage, et n'hésitez pas à solliciter votre chargé de laboratoire si vous avez une question.