LEPL1503/LSINC1503 - Cours 5

O. Bonaventure, B. Legat, M. Baerts

☐ Full Width Mode ☐ Present Mode

Table of Contents

Clarifications projet

- Il est important de lire la partie multithread du syllabus pour savoir faire la deuxième partie
- Amenez le Raspberry aux séances de TP pour que votre tuteur puissent vérifier que vous savez l'utiliser et que votre code fonctionne dessus. Les mesures devront être faites sur le Raspeberry pour être valable. On ne considère pas les mesures sur une autre plateforme.
- "obligatoire" ne veut pas dire que vous aurez tous les points si vous n'implémentez pas le reste!

> Cette partie est obligatoire ! Vous trouverez dans l'annexe A des schémas expliquant ces neuf opérations ;

Chacun doit contribuer sur le Git

- Pour valider votre contribution au projet, il est nécessaire que vous ayez des merge requests en votre nom qui sont mergées avec des commits également en votre nom
- Ces merge requests doivent avoir été finies (donc passer les tests) puis mergées
- Elles doivent apporter une contribution significatives au code, par exemple pas juste modifier le README ou reformatter
- L'excuse du peer coding (vous programmez à deux sur un ordinateur) n'est pas valide
- Ne pas savoir exécuter le code chez soi n'est pas une excuse. Git est facile à installer sous n'importe quelle platforme et les tests s'exécute sur le runner GitLab.

Vérifiez qu'il n'y a pas de fuites de mémoires

Lors de la correction, nous vérifierons qu'il n'y a pas de fuites de mémoire avec Valgrind

```c
#include <stdlib.h>

void f(void)
{
    int *x = malloc(10 * sizeof(int));
    x[10] = 0; // problem 1: heap block overrun
} // problem 2: memory leak -- x not freed

int main(void)
{
    f();
    return 0;
} // From https://valgrind.org/docs/manual/quick-start.html
```

```
1 compile_and_run(valgrind, valgrind = true, verbose = true, cflags = valgrind_debug
  ? ["-g"] : String[])
```

```
Compiling : `/home/runner/.julia/artifacts/207eb5b8330e24674fe59b50d72f4b3d946219c8/tools/clang /tmp/jl_AB9WVe/main.c -o /tmp/jl_AB9WVe/bin`
```

```
Running : `valgrind /tmp/jl_AB9WVe/bin`
```

```
==5665== Memcheck, a memory error detector
==5665== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==5665== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==5665== Command: /tmp/jl_AB9WVe/bin
==5665==
==5665== Invalid write of size 4
==5665==    at 0x10915A: f (in /tmp/jl_AB9WVe/bin)
==5665==    by 0x109183: main (in /tmp/jl_AB9WVe/bin)
==5665==  Address 0x4a79068 is 0 bytes after a block of size 40 alloc'd
==5665==    at 0x4846828: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==5665==    by 0x109151: f (in /tmp/jl_AB9WVe/bin)
==5665==    by 0x109183: main (in /tmp/jl_AB9WVe/bin)
==5665==
==5665==
==5665== HEAP SUMMARY:
==5665==     in use at exit: 40 bytes in 1 blocks
==5665==   total heap usage: 1 allocs, 0 frees, 40 bytes allocated
==5665==
==5665== LEAK SUMMARY:
==5665==    definitely lost: 40 bytes in 1 blocks
==5665==    indirectly lost: 0 bytes in 0 blocks
==5665==      possibly lost: 0 bytes in 0 blocks
```

▶ **Pourquoi `valgrind` ne donne-t-il pas le numéro de ligne ?**

Interface par fichiers

On ne partage normalement pas de `.o` par `git` car ce n'est pas portable (e.g., ce `.o` ne marche pas sur Mac OS ni sur le Raspberry). Lorsque vous avez implémenté ces fonctions, supprimez ce fichier du git et ajoutez `objects/file.o` dans le `.gitignore`

> Comme vous n'aurez pas vu les fichiers au moment de commencer le projet, nous vous fournissons un fichier objet `file.o` dans le dossier objects. ... Cependant, vous devrez coder ces fonctions plus tard et ceci sera vérifié !

C'est une bonne chose d'écrire des unit tests mais comme les signatures des fonctions sont libres, nous utiliseront l'interface uniformisée par fichier pour testez votre code.

> Le Makefile génère un exécutable main. Vous pouvez l'utiliser de la façon suivante :
>
> `./main name_op input_file_A [-v] [-n n_threads] [-f output_stream] [-d degree] [input_file_B]`

Debugging

- Ne pas oublier d'utiliser `-g` lors de la compilation
- `lldb` : développé par LLVM comme `clang` mais marche aussi avec des binaires compilés avec `gcc` , plus facile à installer sur Mac OS
- `gdb` : développé par GNU comme `gcc` mais marche auss avec des binaires compilés avec `clang`
- Utilisez `-fsanitize=address` à la compilation pour que les bornes soient vérifiées.
    - Le code est peut-être un peu moins rapide donc c'est à désactiver pour les mesures
    - mais il n'y a pas de bonne raison de ne pas l'utiliser dans la phase de développement!

Exemple avec sanitize

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i = 0;
    int a[2];
    a[-1] = 1000000;
    a[2] = -1000000;
    printf("&i = %p\n", &i);
    printf("&a = %p\n", &a);
    printf("i = %d\n", i);
    return EXIT_SUCCESS;
}
```

```
Compiling : `/home/runner/.julia/artifacts/207eb5b8330e24674fe59b50d72f4b3d946219c8/tools/clang -Wno-array-bounds -g /tmp/jl_wzDZKr/main.c -o /tmp/jl_wzDZKr/bin`
```

```
Running : `/tmp/jl_wzDZKr/bin`
```

```
&i = 0x7ffe31dffa78
&a = 0x7ffe31dffa70
i = -1000000
```

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i = 0;
    int a[2];
    a[-1] = 1000000;
    a[2] = -1000000;
    printf("&i = %p\n", &i);
    printf("&a = %p\n", &a);
    printf("i = %d\n", i);
    return EXIT_SUCCESS;
}
```

```
Compiling : `/home/runner/.julia/artifacts/207eb5b8330e24674fe59b50d72f4b3d946219c8/tools/clang -Wno-array-bounds -fsanitize=address -g /tmp/jl_IqTuvY/main.c -o /tmp/jl_IqTuvY/bin`
```

```
Running : `/tmp/jl_IqTuvY/bin`
```

```
ProcessFailedException
```

```
=================================================================
==5673==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7f2a3b80002c at pc 0x55e78159cbca bp 0x7fff84add0f0 sp 0x7fff84add0e8
WRITE of size 4 at 0x7f2a3b80002c thread T0
    #0 0x55e78159cbc9  (/tmp/jl_IqTuvY/bin+0x162bc9)
    #1 0x7f2a3d82a1c9  (/lib/x86_64-linux-gnu/libc.so.6+0x2a1c9) (BuildId: 8e9fd827446c24067541ac5390e6f527fb5947bb)
    #2 0x7f2a3d82a28a  (/lib/x86_64-linux-gnu/libc.so.6+0x2a28a) (BuildId: 8e9fd827446c24067541ac5390e6f527fb5947bb)
    #3 0x55e7814653f4  (/tmp/jl_IqTuvY/bin+0x2b3f4)

Address 0x7f2a3b80002c is located in stack of thread T0 at offset 44 in frame
    #0 0x55e78159caaf  (/tmp/jl_IqTuvY/bin+0x162aaf)

  This frame has 2 object(s):
    [32, 36) 'i' (line 6)
    [48, 56) 'a' (line 7) <== Memory access at offset 44 underflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork
```

```julia
out_of_bounds = Example("out_of_bounds.c");
```

Matrices creuses

Deux format efficaces en taille mémoire, calcul et localité de la mémoire:

- Compressed Sparse Row (CSR)

- Compressed Sparse Column (CSC)

$$\begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 7 & 3 & 0 \\ 0 & 6 & 0 & 0 \end{pmatrix}$$

CSR (o-indexed)

$$\begin{aligned} \texttt{nzval} &= [5 \quad 8 \quad 7 \quad 3 \quad 6] \\ \texttt{colval} &= [0 \quad 1 \quad 1 \quad 2 \quad 1] \\ \texttt{rowptr} &= [0 \quad 1 \quad 2 \quad 4 \quad 5] \end{aligned}$$

CSC (o-indexed)

$$\begin{aligned} \texttt{nzval} &= [5 \quad 8 \quad 7 \quad 6 \quad 3] \\ \texttt{colptr} &= [0 \quad 1 \quad 4 \quad 5 \quad 5] \\ \texttt{rowval} &= [0 \quad 1 \quad 2 \quad 3 \quad 2] \end{aligned}$$

```
  Activating project at `~/work/LEPL1503/LEPL1503/Lectures`
```