esiea
INGÉNIEUR·E·S D'UN NUMÉRIQUE UTILE

Programmation système

LAB1
Prise en main de l'environnement

Table des matières

By Rexy

1 Objectifs

Ce LAB permet de mettre prendre en main l'environnement minimal pour aborder les notions de programmation système.

- Documents :
  - Supports : « Linux – Les commandes » et « Linux – L'éditeur Vi » ;
  - Fiches : « Compilation » et « Débogage ».
- Notions abordées : l'environnement de développement, le débogage.
- Commandes et fichiers exploités : gcc, commandes de gestion de fichiers, ddd.
- Travail à rendre : Vous devrez répondre directement à plusieurs questions au sein de ce document. Vous le copierez sur Moodle sous le nom : « LAB0_*NOM*.pdf ».

Pour développer, vous devez utiliser les outils standards GNU (gcc + gdb/ddd) et un éditeur de texte type Vi/Emacs/Gedit, l'utilisation d'un IDE est interdite.

2 Travaux préparatoires

Pour vous familiariser à nouveau avec l'environnement :

- Lisez et effectuez les commandes décrites aux chapitres §1 à §3.7 du support « Linux - Les commandes.pdf » ;

3 Connaissance système

Objectif : Connaître le système sur lequel vous êtes en train de travailler : nom de la distribution, version du noyau, capacité de la machine (CPU, mémoire, carte mère et bios, cartes additionnelles), topologie du disque (marque, taille, partitionnement) ?

- Copiez ci-dessous les informations recherchées (les commandes utilisées et le résultat de celles-ci pour votre système).

CPU : 12th Gen Intel(R) Core(TM) i7-12700H (lscpu)

Taille mémoire vive : 15.9G (lsmem)

Carte mère et version du BIOS : Dell Inc. Model : 0MWGD4 / BIOS Version: 1.13.1 (dmidecode)

Cartes additionnelles : RTS5260 PCI Express Card Reader, Alder Lake PCH eSPI Controller (lspci)

Partitionnement du disque dur : (lsblk)

```
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
nvme0n1     259:0    0 476.9G  0 disk
├─nvme0n1p1 259:1    0  1000M  0 part /boot/efi
├─nvme0n1p2 259:2    0 459.1G  0 part /
└─nvme0n1p3 259:3    0  16.8G  0 part [SWAP]
```

- Comment lire le journal de démarrage du système (boot) ?
- Comment lire de manière continue le journal d'événement ?

journalctl -b

et

journalctl -f

- Ouvrez un terminal.

Dans quel répertoire vous trouvez-vous ?

Commande : **pwd**

/home/saml

- Dans votre répertoire de connexion, créez un répertoire `tmp` → **mkdir tmp**
- Positionnez les droits d'accès à `rwx r-x ---` pour `tmp` → **`chmod u=+rwx,g=+rx-w,o=-rwx tmp`**
- Copiez les fichiers `passwd`, `group` et `hosts` du répertoire `/etc` dans `tmp` ➔ `cp /etc/passwd tmp ; ➔ cp /etc/hosts tmp ; ➔ cp /etc/group tmp`
- Changez le nom de `hosts` en `hotes`. ➔ ~ mv tmp/hosts tmp/hotes
- Positionnez les droits d'accès à `rw- r--- ---` pour le fichier `hotes`. ➔ ~ chmod 640 tmp/hotes
- Lisez le contenu de `hotes`. ➔ ~ cat tmp/hotes

  Remarque : la lecture du fichier `~/tmp/hotes` est permise. Le fichier peut néanmoins être vide.
- Retirez au propriétaire le droit en lecture sur le fichier `hotes` et essayez de le lire.

Quel est la signification du message d'erreur obtenu ?

Commandes : ➔ ~ mv tmp/hosts tmp/hotes

→ cat tmp/hotes

Le message d'erreur indique que nous n'avons pas les droits de lecture sur le fichier (cat: tmp/hotes: Permission denied)

- Remettez pour le propriétaire le droit en lecture sur le fichier `hotes.` → `chmod u+r tmp/hotes`
- Retirez pour le propriétaire le droit en écriture sur le répertoire `tmp`. ➔ ~ chmod u-w tmp
- Essayez de détruire `hotes`.

Quel est la signification du message d'erreur obtenu ?

Commande : rm tmp/hotes

rm: cannot remove 'tmp/hotes': Permission denied

L'erreur signifie que nous n'avons pas les droits necessaires a la destruction du fichier car nous n'avons pas les droits d'ecriture sur le repertoire qui le contient

- Retirez pour le propriétaire le droit en lecture sur le répertoire `tmp.➔ ~ chmod u-r tmp`
- Essayez de lister le contenu de `tmp.`

Quel est la signification du message d'erreur obtenu ?

Commande : ls tmp

ls: cannot open directory 'tmp': Permission denied

Nous ne pouvons pas lister le contenu du repertoire car nous n’avons pas les droits de lecture dessus

- Lisez le contenu de `hotes.`

Pourquoi pouvez-vous le lire ?

Nous pouvons le lire car nous avons les droits en lecture dessus meme sans avoir les droits sur le dossier parent. Les droits de lecture sur le dossier parent n’influe que sur la capacite de lister ses elements mais pas sur les droits des fichiers a l’interieur

- Retirez pour le propriétaire le droit en exécution sur le répertoire `tmp`.➔ ~ chmod u-x tmp
- Essayez de vous positionner sur ce répertoire.
- Quel est la signification du message d'erreur obtenu ?

➔ ~ cd tmp

cd: permission denied: tmp

- Le message d’erreur indique qu’il n’est pas possible de se positionner dans le repertoire car nous n’avons pas les droits necessaires

- Essayez de lire le contenu de `hotes`.

Quel est la signification du message d'erreur obtenu ?

Le message obtenu signifie que nous ne pouvons plus lire le fichier car nous n’avons plus les droits en execution sur le dossier parent

4 Source, code assembleur, binaire et débogage.

- Lisez la fiche « rappel compilation » ;
- Réalisez le premier programme de la fiche `welcome.c` ;
- Lancez et analysez la compilation étape par étape comme décrite dans la fiche « rappel compilation ».

Copier ici, le **code assembleur** de ce programme.

```
        .file   "welcome.c"
        .text
        .section        .rodata
.LC0:
        .string "Rexy is welcome you"
        .text
```

```
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        leaq    .LC0(%rip), %rax
        movq    %rax, %rdi
        call    puts@PLT
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (GNU) 13.2.1 20230801"
        .section        .note.GNU-stack,"",@progbits
```

Pour connaître le type d'un fichier (binaire, son, image, etc.), on peut se fier à son extension (.mp3, .jpeg, etc.). Cela est limité surtout quand le nom du fichier ne possède pas d'extension. La commande `file` sous Linux permet de connaître le type d'un fichier en faisant abstraction de son nom.

Quel type de fichier votre compilateur a-t-il généré ? Récupérez un fichier binaire exécutable pour Windows (un .exe quelconque) et testez la commande `file` avec celui-ci ?

Type du fichier binaire Linux : ELF 64-bit LSB pie executable

Type du fichier binaire Windows : PE32 executable (GUI) Intel 80386, for MS Windows

- Analysez et écrivez le deuxième programme `sentence2words.c` ;
- Lancez-le pas à pas dans le débogueur (*ddd*, *gdb* ou autre) pour suivre l'évolution du chargement du tableau de mots.

Réalisez une copie d'écran du débogueur affichant la zone mémoire du tableau rempli. Dans cette copie d'écran, on doit aussi voir la fenêtre de résultat du programme affichant le tableau de mots.

5 Évolution de code

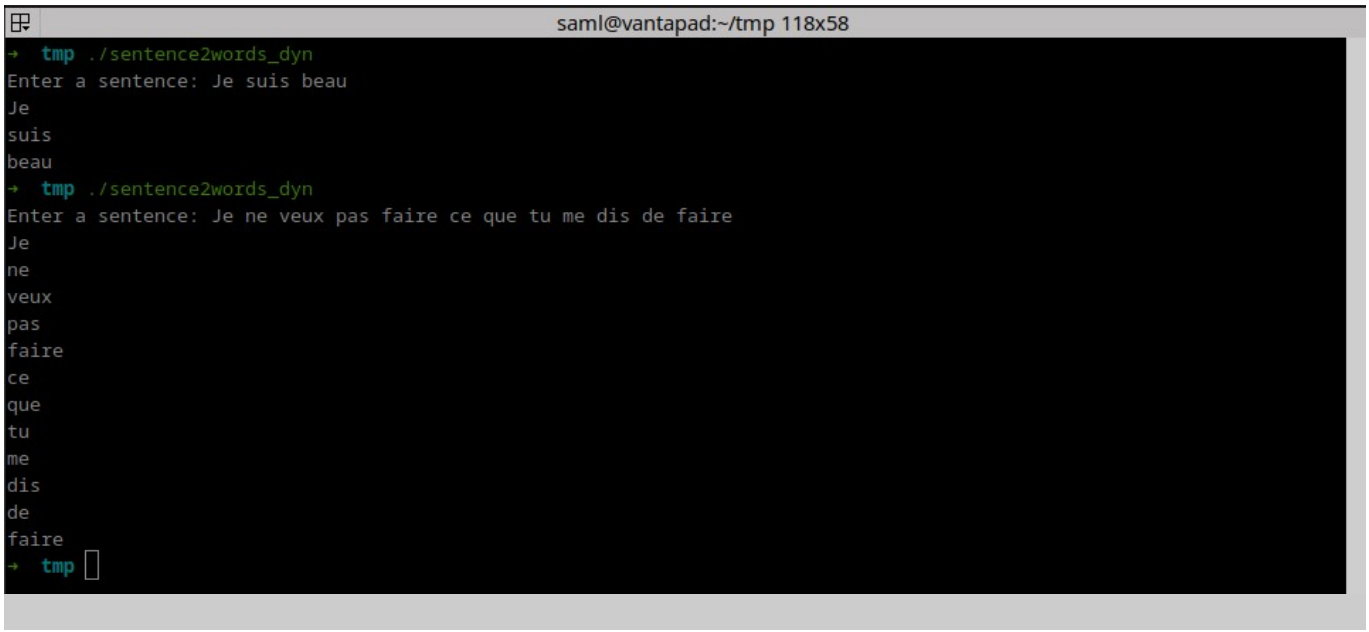
Proposez l’évolution suivante du code du deuxième programme :

- Au lancement du programme, la phrase initiale est demandée à l'utilisateur ;
- Le tableau est dimensionné dynamiquement en fonction de cette phrase ;
- L'affichage du tableau est réalisé par l'appel d'une fonction.

Copier votre code sur Moodle sous le nom de fichier « LAB1_*NOM*.c ».

Rappel : un fichier source qui ne compile pas n'est pas corrigé !

Copiez ci-dessous une copie d'écran du résultat de son exécution



6 Amélioration de la fiche « débogage »

L'objectif est d'améliorer cette fiche, exploitez `ddd` sur un programme utilisant des fonctions et expliquez le rôle des fichiers `core`.

Comment utiliser le debugger pour comprendre les appels de fonction ?

/!\ Attention : Il est necessaire d'avoir des fonctions (autres que la fonction main) afin d'utiliser cette methode.

1. Compiler en mode débogage

- `gcc -g -o mon_programme mon_programme.c`

2. Lancer GDB

- `gdb ./mon_programme`

3. Ajouter le breakpoint a main

- `breakpoint main`

4. Démarrer l'exécution

- `run`

5. Exécuter pas à pas

- `step (autant de fois que necessaire pour rentrer dans la fonction d'interet`

6. Afficher la pile des appels de fonction

- `backtrace`

Exemple de resultat :

```
(gdb) step
109       __printf_buffer_init (&buf->base, buf->stage, array_length (buf->stage),
(gdb) step
__printf_buffer_init (mode=__printf_buffer_mode_to_file, len=128, base=0x7fffffffd9c0 "", buf=0x7fffffffd990)
    at ../include/printf_buffer.h:125
125       buf->write_ptr = base;
(gdb) backstrace
Undefined command: "backstrace".  Try "help".
(gdb) backtrace
#0  __printf_buffer_init (mode=__printf_buffer_mode_to_file, len=128, base=0x7fffffffd9c0 "", buf=0x7fffffffd990)
    at ../include/printf_buffer.h:125
#1  __printf_buffer_to_file_init (buf=buf@entry=0x7fffffffd990, fp=fp@entry=0x7ffff7e3f5c0 <_IO_2_1_stdout_>)
    at printf_buffer_to_file.c:109
#2  0x00007ffff7c60e40 in __vfprintf_internal (s=0x7ffff7e3f5c0 <_IO_2_1_stdout_>,
    format=0x555555556006 "Enter a sentence: ", ap=ap@entry=0x7fffffffda90, mode_flags=mode_flags@entry=0)
    at vfprintf-internal.c:1522
#3  0x00007ffff7c562ff in __printf (format=<optimized out>) at printf.c:33
#4  0x000055555555532f in main () at sentence2words_dyn.c:38
(gdb)
```

Comment utiliser un fichier core et pourquoi ?

Un fichier core sert a investiguer un crash (typiquement, un segmentation fault). Il s'ouvre en utilisant la commande suivante :

- `gdb /chemin/vers/votre_programme core`

Apres avoir ouvert le fichier core, on peut commencer a investiguer la cause du crash, notamment avec les commandes « backtrace » (bt) ou « list » :

```
Program received signal SIGFPE, Arithmetic exception.
0x0000555555555232 in countWords (sentence=0x7fffffffdb70 "asd a sda sd a sd asd as dasdasdasd ")
    at intentional_segfault.c:17
17                i=4/0;
(gdb) bt
#0  0x0000555555555232 in countWords (sentence=0x7fffffffdb70 "asd a sda sd a sd asd as dasdasdasd ")
    at intentional_segfault.c:17
#1  0x000055555555536f in main () at intentional_segfault.c:44
(gdb) list
12      int countWords(const char* sentence) {
13          int wordCount = 1;
14          for (int i = 0; sentence[i] != '\0'; i++) {
15              if (sentence[i] == ' ') {
16                  wordCount++;
17                  i=4/0;
18              }
19          }
20          return wordCount;
21      }
(gdb)
```

Copiez votre fiche « débogage » sur Moodle sous le nom « debug_*NOM*.pdf »

Vérifiez que vous avez bien copié 3 fichiers sur Moodle : ce LAB, le code de l'évolution du programme `sentence2words.c` et votre fiche débogage.