

Principe des Systèmes d'Exploitation TP1

Chaîne de compilation C/UNIX et premiers appels système

Table des matières

Ex. 1 : Compilation d'un programme C mono-fichier :.....	3
Ex. 2 : Compilation de programme C multi-fichier.....	8
Ex. 3 Lancement d'un exécutable à partir d'un programme C et traitement des erreurs.....	9
Ex. 4 Exemple de création de processus fils dont le code est le même que celui de leur père.....	12

Ex. 1 : Compilation d'un programme C mono-fichier :

a)

```
[0]utilisateur@Ubuntu:~/Bureau$ gcc p1.c  
[0]utilisateur@Ubuntu:~/Bureau$ ls  
a.out  Disque D:\ (Windows).desktop  p1.c  
[0]utilisateur@Ubuntu:~/Bureau$ ./a.out  
Hello world
```

Le -v nous permet d'afficher toutes les étapes de la compilation du fichier (cf man gcc), il va successivement et automatiquement faire :

- 1

2/3

4

c)

```
[0]utilisateur@Ubuntu:~/Bureau$ ls
a.out  Disque D:\ (Windows).desktop  p1.c
[0]utilisateur@Ubuntu:~/Bureau$ gcc -S p1.c
[0]utilisateur@Ubuntu:~/Bureau$ ls
a.out  Disque D:\ (Windows).desktop  p1.c  p1.s
[0]utilisateur@Ubuntu:~/Bureau$ file p1.s
p1.s: assembler source text
[0]utilisateur@Ubuntu:~/Bureau$ gcc -c p1.s
[0]utilisateur@Ubuntu:~/Bureau$ file p1.o
p1.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), not stripped
```

Les commandes successives génèrent deux fichiers p1.o et p1.s (ainsi que a.out à la fin).

p1.s est de type assembler source text, voici son contenu :

```
[0]utilisateur@Ubuntu:~/Bureau$ cat p1.s
        .file   "p1.c"
        .section      .rodata
.LC0:
        .string "Hello world "
        .text
        .globl  main
        .type   main, @function
main:
.LFB2:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $.LC0, %edi
        call    puts
        movl    $0, %edi
        call    exit
        .cfi_endproc
.LFE2:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 4.8.4-2ubuntu1~14.04.3) 4.8.4"
        .section      .note.GNU-stack,"",@progbits
```

p1.o est de type binaire 64 bit, voici son contenu :

Avec la commande **od -h** :

```
[0]utilisateur@Ubuntu:~/Bureau$ od -h p1.o
00000000 457f 464c 0102 0001 0000 0000 0000 0000
00000020 0001 003e 0001 0000 0000 0000 0000 0000
00000040 0000 0000 0000 0000 0138 0000 0000 0000
00000060 0000 0000 0040 0000 0000 0040 000d 000a
00000100 4855 e589 00bf 0000 e800 0000 0000 00bf
00000120 0000 e800 0000 0000 6548 6c6c 206f 6f77
00000140 6c72 2064 0000 4347 3a43 2820 6255 6e75
00000160 7574 3420 382e 342e 322d 6275 6e75 7574
00000200 7e31 3431 302e 2e34 2933 3420 382e 342e
00000220 0000 0000 0000 0000 0014 0000 0000 0000
00000240 7a01 0052 7801 0110 0c1b 0807 0190 0000
00000260 001c 0000 001c 0000 0000 0000 0018 0000
00000300 4100 100e 0286 0d43 0006 0000 0000 0000
00000320 2e00 7973 746d 6261 2e00 7473 7472 6261
00000340 2e00 6873 7473 7472 6261 2e00 6572 616c
00000360 742e 7865 0074 642e 7461 0061 622e 7373
00000400 2e00 6f72 6164 6174 2e00 6f63 6d6d 6e65
00000420 0074 6e2e 746f 2e65 4e47 2d55 7473 6361
00000440 006b 722e 6c65 2e61 6865 665f 6172 656d
00000460 0000 0000 0000 0000 0000 0000 0000 0000
*
00000560 0000 0000 0000 0000 0020 0000 0001 0000
00000600 0006 0000 0000 0000 0000 0000 0000 0000
00000620 0040 0000 0000 0000 0018 0000 0000 0000
00000640 0000 0000 0000 0000 0001 0000 0000 0000
00000660 0000 0000 0000 0000 001b 0000 0004 0000
00000700 0000 0000 0000 0000 0000 0000 0000 0000
00000720 05b0 0000 0000 0000 0048 0000 0000 0000
00000740 000b 0000 0001 0000 0008 0000 0000 0000
00000760 0018 0000 0000 0000 0026 0000 0001 0000
00010000 0003 0000 0000 0000 0000 0000 0000 0000
00010020 0058 0000 0000 0000 0000 0000 0000 0000
00010040 0000 0000 0000 0000 0001 0000 0000 0000
00010060 0000 0000 0000 0000 002c 0000 0008 0000
00010100 0003 0000 0000 0000 0000 0000 0000 0000
00010120 0058 0000 0000 0000 0000 0000 0000 0000
00010140 0000 0000 0000 0000 0001 0000 0000 0000
```

Avec la commande **objdump -D** :

```
[0]utilisateur@Ubuntu:~/Bureau$ objdump -D p1.o

p1.o:          format de fichier elf64-x86-64


D  assemblage de la section .text :


0000000000000000 <main>:
   0:  55                      push    %rbp
   1:  48 89 e5                mov     %rsp,%rbp
   4:  bf 00 00 00 00          mov     $0x0,%edi
   9:  e8 00 00 00 00          callq   e <main+0xe>
  e:  bf 00 00 00 00          mov     $0x0,%edi
 13:  e8 00 00 00 00          callq  18 <main+0x18>


D  assemblage de la section .rodata :


0000000000000000 <.rodata>:
   0:  48                      rex.W
   1:  65                      gs
   2:  6c                      insb     (%dx),%es:(%rdi)
   3:  6c                      insb     (%dx),%es:(%rdi)
   4:  6f                      outsl    %ds:(%rsi),(%dx)
   5:  20 77 6f                and      %dh,0x6f(%rdi)
   8:  72 6c                   jb       76 <main+0x76>
  a:  64 20 00                and      %al,%fs:(%rax)


D  assemblage de la section .comment :


0000000000000000 <.comment>:
   0:  00 47 43                add      %al,0x43(%rdi)
   3:  43 3a 20                rex.XB  cmp (%r8),%spl
   6:  28 55 62                sub      %dl,0x62(%rbp)
   9:  75 6e                   jne      79 <main+0x79>
  b:  74 75                   je       82 <main+0x82>
  d:  20 34 2e                and      %dh,(%rsi,%rbp,1)
 10:  38 2e                   cmp      %ch,(%rsi)
 12:  34 2d                   xor      $0x2d,%al
 14:  32 75 62                xor      0x62(%rbp),%dh
 17:  75 6e                   jne      87 <main+0x87>
 19:  74 75                   je       90 <main+0x90>
 1b:  31 7e 31                xor      %edi,0x31(%rsi)
 1e:  34 2e                   xor      $0x2e,%al
 20:  30 34 2e                xor      %dh,(%rsi,%rbp,1)
 23:  33 29                   xor      (%rcx),%ebp
 25:  20 34 2e                and      %dh,(%rsi,%rbp,1)
```

Ex. 2 : Compilation de programme C multi-fichier

a)

```
[0]utilisateur@Ubuntu:~/Bureau$ ls
a.out  Disque D:\ (Windows).desktop  hello.c  hello.h  pl.c  pl.o  pl.s  prog.c
```

b)

```
[0]utilisateur@Ubuntu:~/Bureau$ gcc -c prog.c
[0]utilisateur@Ubuntu:~/Bureau$ gcc -c hello.c
[0]utilisateur@Ubuntu:~/Bureau$ ls
a.out  Disque D:\ (Windows).desktop  hello.c  hello.h  hello.o  pl.c  pl.o  pl.s  prog.c  prog.o
```

c)

```
[0]utilisateur@Ubuntu:~/Bureau$ gcc prog.o hello.o -o prog.x
[0]utilisateur@Ubuntu:~/Bureau$ ls
a.out  Disque D:\ (Windows).desktop  hello.c  hello.h  hello.o  pl.c  pl.o  pl.s  prog.c  prog.o  prog.x
```

d)

```
[0]utilisateur@Ubuntu:~/Bureau$ gcc prog.c hello.c -o prog.x
[0]utilisateur@Ubuntu:~/Bureau$ ls
a.out  Disque D:\ (Windows).desktop  hello.c  hello.h  hello.o  pl.c  pl.o  pl.s  prog.c  prog.o  prog.x
```

Exécution :

```
[0]utilisateur@Ubuntu:~/Bureau$ ./prog.x
Bonjour !
```



Ex. 3 Lancement d'un exécutable à partir d'un programme C et traitement des erreurs

a)

On réalise deux **ps au**, un avant l'exécution et un après l'exécution de ex3.x

```
[0]utilisateur@Ubuntu:~/Bureau$ ps au
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      1257  0.0  0.0  23008  2096 tty4      Ss+  09:34   0:00 /sbin/getty -8
root      1264  0.0  0.0  23008  2120 tty5      Ss+  09:34   0:00 /sbin/getty -8
root      1272  0.0  0.0  23008  2044 tty2      Ss+  09:34   0:00 /sbin/getty -8
root      1273  0.0  0.0  23008  2116 tty3      Ss+  09:34   0:00 /sbin/getty -8
root      1277  0.0  0.0  23008  2052 tty6      Ss+  09:34   0:00 /sbin/getty -8
root      1862  0.0  0.0  23008  2072 tty1      Ss+  09:34   0:00 /sbin/getty -8
root      1920  1.3  1.4 257240 59280 tty7      Ssl+ 09:34   0:54 /usr/bin/X -co
utilisat+ 3235  0.0  0.0  28272  3824 pts/0     Ss+  09:44   0:00 bash
utilisat+ 3969  0.2  0.0  28264  3824 pts/4     Ss   10:44   0:00 bash
utilisat+ 3981  0.0  0.0  25704  2592 pts/4     R+   10:44   0:00 ps au

[0]utilisateur@Ubuntu:~/Bureau$ ps au
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      1257  0.0  0.0  23008  2096 tty4      Ss+  09:34   0:00 /sbin/getty -8 38400 tty4
root      1264  0.0  0.0  23008  2120 tty5      Ss+  09:34   0:00 /sbin/getty -8 38400 tty5
root      1272  0.0  0.0  23008  2044 tty2      Ss+  09:34   0:00 /sbin/getty -8 38400 tty2
root      1273  0.0  0.0  23008  2116 tty3      Ss+  09:34   0:00 /sbin/getty -8 38400 tty3
root      1277  0.0  0.0  23008  2052 tty6      Ss+  09:34   0:00 /sbin/getty -8 38400 tty6
root      1862  0.0  0.0  23008  2072 tty1      Ss+  09:34   0:00 /sbin/getty -8 38400 tty1
root      1920  1.3  1.4 257240 59208 tty7      Ssl+ 09:34   0:55 /usr/bin/X -core :0 -seat seat
utilisat+ 3235  0.0  0.0  28272  3824 pts/0     Ss   09:44   0:00 bash
utilisat+ 3969  0.1  0.0  28264  3824 pts/4     Ss   10:44   0:00 bash
utilisat+ 3983  0.0  0.0   4196   792 pts/0     S+   10:45   0:00 ./ex3.x
utilisat+ 3986  0.0  0.0  25704  2620 pts/4     R+   10:45   0:00 ps au
```



On observe que ex3.x est bien dans la liste des processus une fois que son exécution a été effectuée.

ex3.x permet d'exécuter un émulateur de terminal de commande, via 'xterm'.

b)

On crée un fichier ex4.c avec le code ci-dessous, exécutable a.out affiche 'Hello World'.

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char* argv){

    usleep(7000000); /* sleep 7 seconds */
    execl("./a.out", "a.out", NULL);
    /* si l'appel à exec fonctionne, le reste
    du programme ne s'exécute jamais */
    printf("ne s'affiche jamais");
    return 0 ;|
}
```

```
[0]utilisateur@Ubuntu:~/Bureau$ gcc -o ex ex4.c
[0]utilisateur@Ubuntu:~/Bureau$ ls
a.out      ex      ex3.c~  ex4.c  ex4.x  hello.h  pl.c  pl.s  prog.o
Disque D:\ (Windows).desktop  ex3.c  ex3.x  ex4.c~  hello.c  hello.o  pl.o  prog.c  prog.x
[0]utilisateur@Ubuntu:~/Bureau$ ./ex
Hello world
```

c) On passe ex.x en paramètre dans la fonction execl(), sachant que cet exécutable n'existe pas.

execl("./a.out", "a.out", NULL);  execl("./ex.x", "ex.x", NULL);

```
[0]utilisateur@Ubuntu:~/Bureau$ gcc -o ex ex4.c
[0]utilisateur@Ubuntu:~/Bureau$ ./ex
ne s'affiche jamais[0]utilisateur@Ubuntu:~/Bureau$
```

Le programme ne va pas trouver d'exécutable a ce chemin d'accès et va donc passer au printf() après le execl().

d)

```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>

int main(int argc, char* argv){

    //usleep(7000000); /* sleep 7 seconds */
    execl("./ex.x", "ex.x", NULL);
    if(errno==ENOENT){
        perror("Erreur");
        _exit(1);
    }
    /* si l'appel à exec fonctionne, le reste
    du programme ne s'exécute jamais */
    printf("ne s'affiche jamais");
    return 0;
}
```

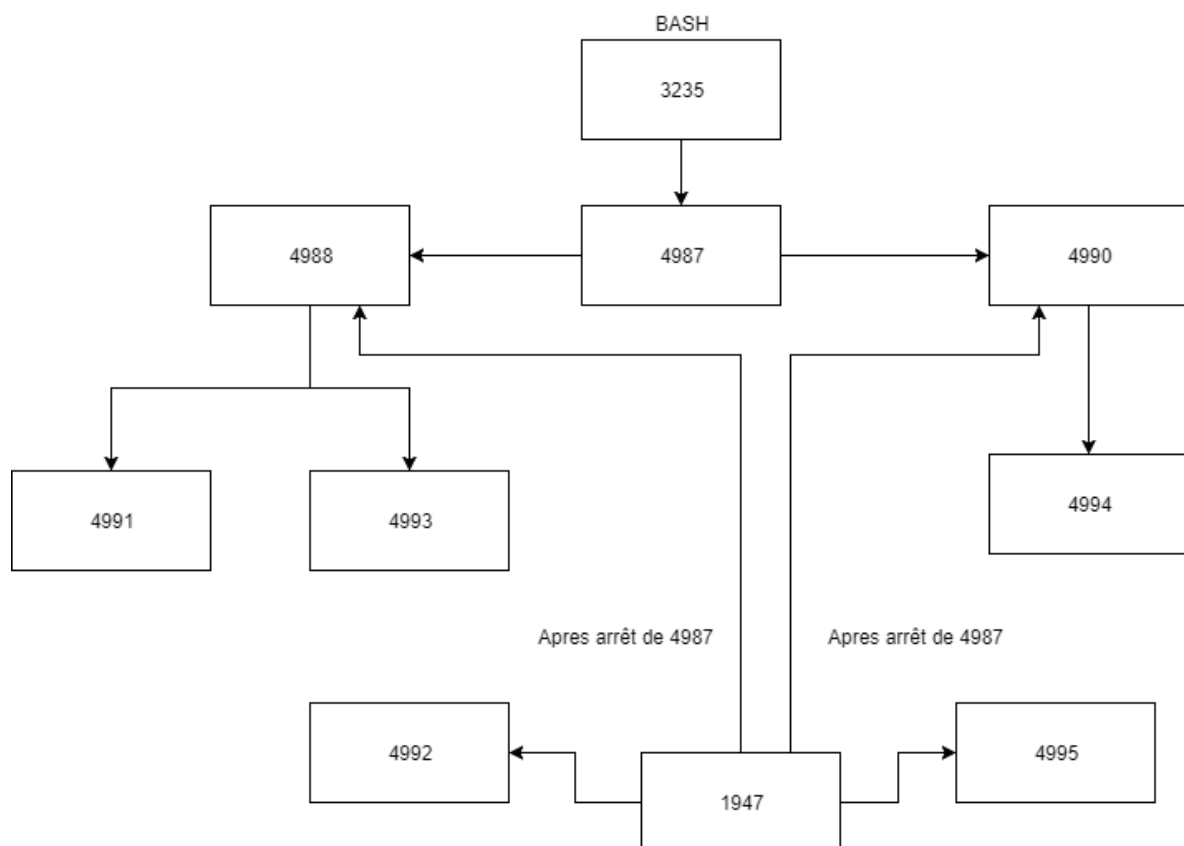
```
[0]utilisateur@Ubuntu:~/Bureau$ gcc -o ex ex4.c
[0]utilisateur@Ubuntu:~/Bureau$ ./ex
Erreur: No such file or directory
```

Ex. 4 Exemple de création de processus fils dont le code est le même que celui de leur père.

a) 8 processus sont créés à l'exécution de ce programme (2^3 car il y a 3 parcours de boucle).

b)

```
[0]utilisateur@Ubuntu:~/Bureau$ ./ex
fils (4987) de (3235). i=0
fils (4988) de (4987). i=0
fils (4987) de (3235). i=1
fils (4988) de (4987). i=1
fils (4990) de (4987). i=1
fils (4991) de (4988). i=1
fils (4987) de (3235). i=2
fils (4994) de (4990). i=2
fils (4991) de (4988). i=2
[0]utilisateur@Ubuntu:~/Bureau$ fils (4992) de (1947). i=2
fils (4988) de (1947). i=2
fils (4990) de (1947). i=2
fils (4993) de (4988). i=2
fils (4995) de (1947). i=2
```



c) Les processus changent de père pendant l'exécution car le processus père peut s'arrêter car l'exécution du programme est trop rapide, le processus fils va donc se faire adopter par un autre processus.

```
[0]utilisateur@Ubuntu:~/Bureau$ ./ex
fils (4987) de (3235). i=0
fils (4988) de (4987). i=0
fils (4987) de (3235). i=1
fils (4988) de (4987). i=1
fils (4990) de (4987). i=1
fils (4991) de (4988). i=1
fils (4987) de (3235). i=2
fils (4994) de (4990). i=2
fils (4991) de (4988). i=2
[0]utilisateur@Ubuntu:~/Bureau$ fils (4992) de (1947). i=2
fils (4988) de (1947). i=2
fils (4990) de (1947). i=2
fils (4993) de (4988). i=2
fils (4995) de (1947). i=2
```

Ici, le processus 4988 était au départ le fils du processus 4987 et est devenu le fils du processus 1947 lors de la dernière itération de la boucle.

d) Ici, aucun processus ne change de père lors de l'exécution.

```
[0]utilisateur@Ubuntu:~/Bureau$ ./ex
fils (5049) de (3235). i=0
fils (5049) de (3235). i=1
fils (5049) de (3235). i=2
fils (5050) de (5049). i=0
fils (5050) de (5049). i=1
fils (5050) de (5049). i=2
fils (5054) de (5050). i=2
fils (5053) de (5050). i=1
fils (5053) de (5050). i=2
fils (5051) de (5049). i=1
fils (5052) de (5049). i=2
fils (5055) de (5053). i=2
fils (5051) de (5049). i=2
fils (5056) de (5051). i=2
```

Représentation de l'arbre a partir du processus père (3235) et ses fils créés par le programme.

