Compilation: librairies, sécurité, optimisation.

Gaétan Richard
gaetan.richard@unicaen.fr

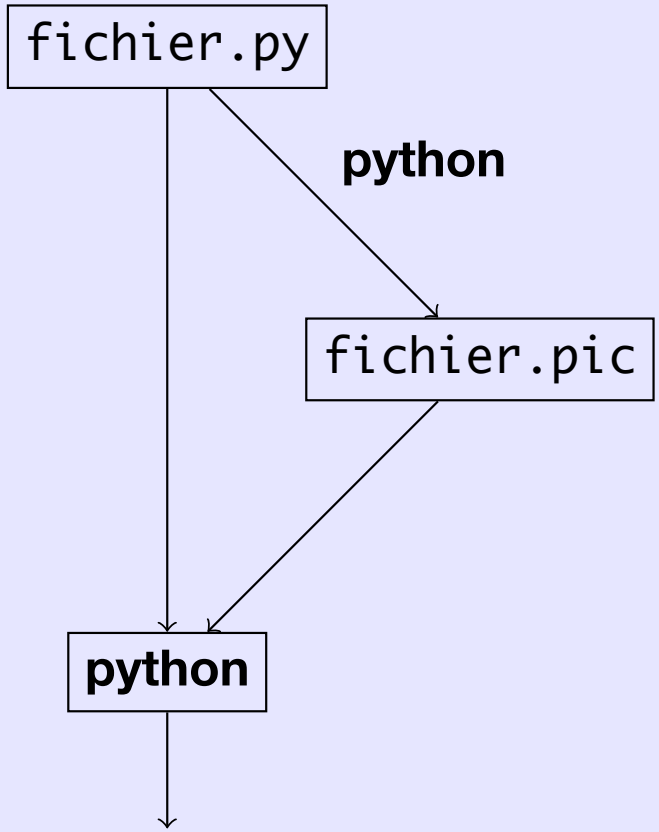
Langages et compilation — L3 informatique

1. Introduction

Du code source à l'exécution

Cas des langages interprétés:

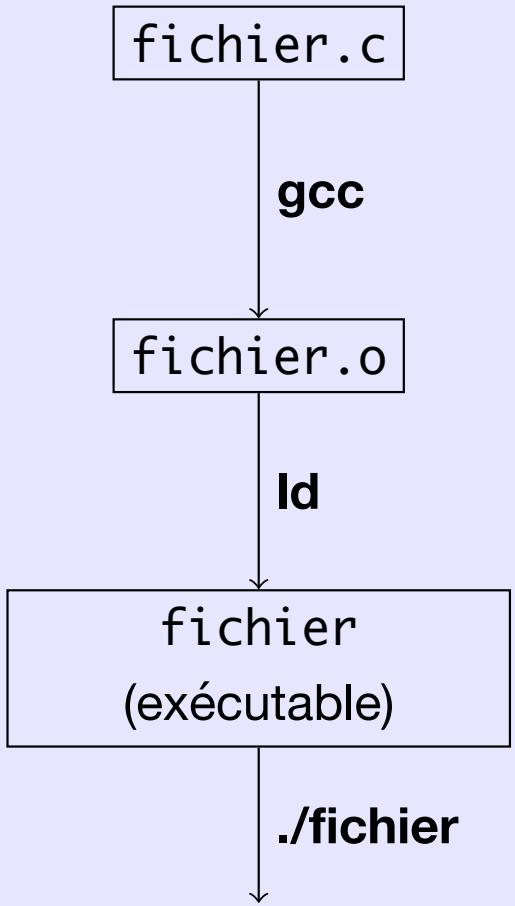
- le fichier source contient le code du programme;
- on convertit le langage vers un langage intermédiaire;
- un interpréteur se charge alors d'exécuter le source et de gérer les fonctions de bibliothèques.



Du code source à l'exécution

Cas des langages compilés.

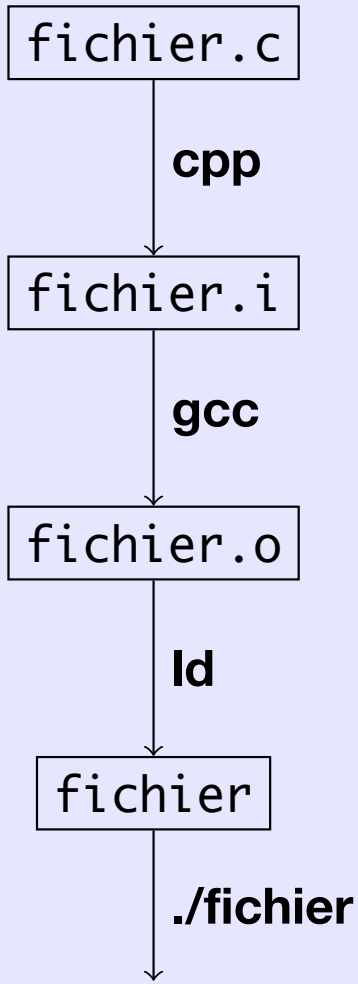
- le fichier source contient le code du programme;
- on compile le fichier pour produire du code exécutable;
- on fait le liens avec les fonctions de bibliothèques (linkage);
- on lance le programme.



Du code source à l'exécution

Cas des langages compilés avec précompilation.

- le fichier source contient le code du programme;
- on compile le fichier pour produire du code exécutable;
- on fait le liens avec les fonctions de bibliothèques (linkage);
- on lance le programme.



2. Binaire

C'est quoi exactement un binaire exécutable?

Tout d'abord il y a différent formats standardisés selon le système d'exploitation :

- PEF (*Preferred Executable Format*) / Mach-O (*Mach Object*) sous Mach OS / Mach OS X
- ELF (*Executable and Linkable Format*) pour Unix
- PE (*portable executable*) sous Windows

C'est quoi exactement un binaire exécutable?

Tout d'abord il y a différent formats standardisés selon le système d'exploitation :

- PEF (*Preferred Executable Format*) / Mach-O (*Mach Object*) sous Mach OS / Mach OS X
- ELF (*Executable and Linkable Format*) pour Unix
- PE (*portable executable*) sous Windows

ELF : format exécutable et liable

> *ELF [...] est un format de fichier binaire utilisé pour l'enregistrement de code compilé (objets, exécutables, bibliothèques de fonctions). Il a été développé par l'USL (Unix System Laboratories) pour remplacer les anciens formats a.out et COFF qui avaient atteint leurs limites. Aujourd'hui, ce format est utilisé dans la plupart des systèmes d'exploitation de type Unix (GNU/Linux, Solaris, IRIX, System V, BSD), à l'exception de Mac OS X.*

(Wikipedia)

ELF

man ELF

```
ELF(5)                          Linux Programmer's Manual                          ELF(5)

NAME
       elf - format of Executable and Linking Format (ELF) files

SYNOPSIS
       #include <elf.h>

DESCRIPTION
       The  header  file  <elf.h>  defines  the format of ELF executable binary files.  Amongst
       these files are normal executable files, relocatable object files, core files and shared
       libraries.

       An  executable  file  using the ELF file format consists of an ELF header, followed by a
       program header table or a section header table, or both.  The ELF header  is  always  at
       offset zero of the file.  The program header table and the section header table's offset
       in the file are defined in the ELF header.  The two tables describe the rest of the par-
       ticularities of the file.

       This header file describes the above mentioned headers as C structures and also includes
       structures for dynamic sections, relocation sections and symbol tables.

       The following types are used for N-bit architectures (N=32,64, ElfN stands for Elf32  or
       Elf64, uintN_t stands for uint32_t or uint64_t):

 Manual page elf(5) line 1 (press h for help or q to quit)
```

Exemple

```
    .file  "tiny.c"
  .text
  .globl  main
  .type  main, @function
main:
.LFB0:
  .cfi_startproc
  pushl  %ebp
  .cfi_def_cfa_offset 8
  .cfi_offset 5, -8
  movl  %esp, %ebp
  .cfi_def_cfa_register 5
  movl  $42, %eax
  popl  %ebp
  .cfi_def_cfa 4, 4
  .cfi_restore 5
  ret
  .cfi_endproc
.LFE0:
  .size  main, .-main
  .ident  "GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
  .section  .note.GNU-stack,"",@progbits
```

Mais ce n'est pas du binaire?

En pratique, ce fichier est vraiment codé en binaire.

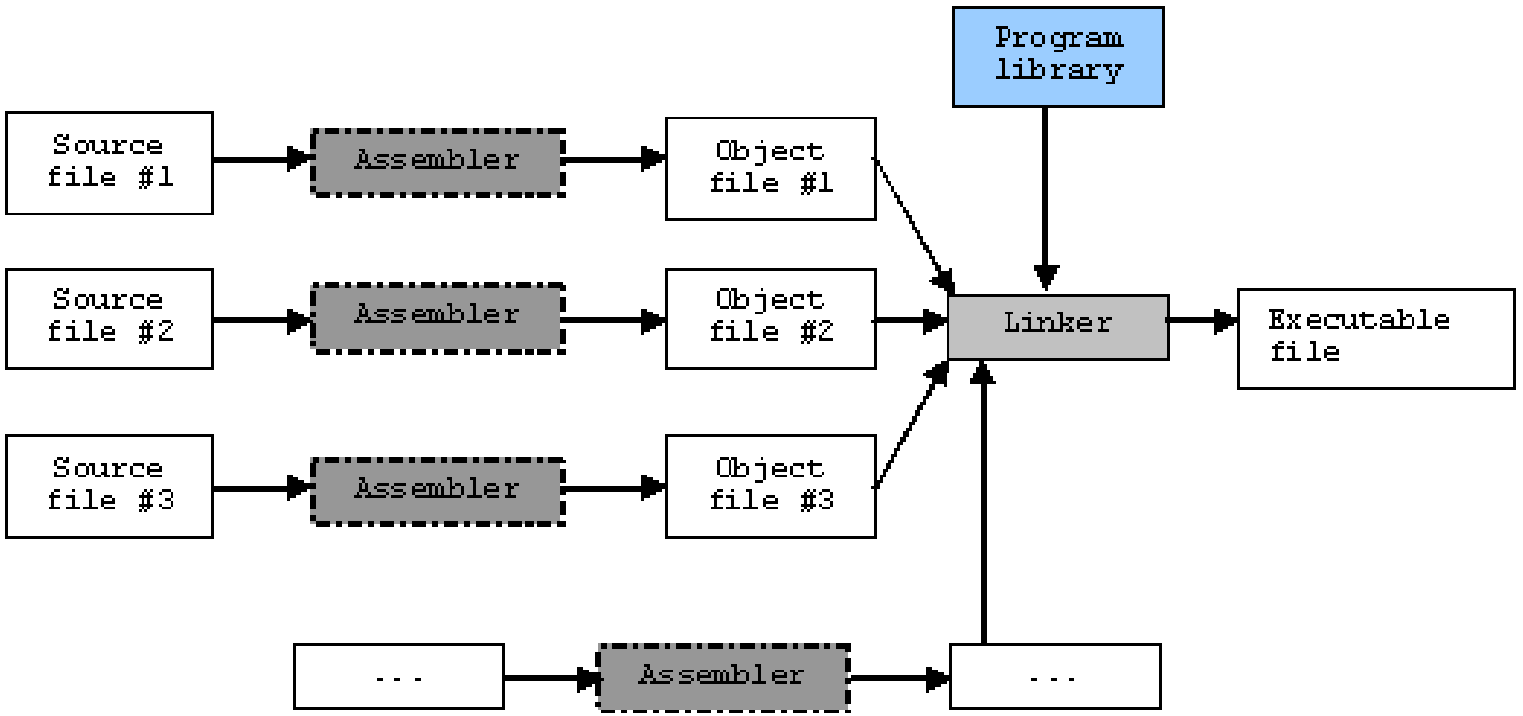
On peut toutefois à l'aide d'utilitaires passer du format textuel au format binaire et inversement.

Que veut dire format liable?

Grand principe : on ne va pas recompiler le même code plusieurs fois.

Idée : au lieu de recopier du code de librairie dans notre code puis de compiler le tout ensemble, on va pré-assembler et lier le tout au niveau du binaire (avec des fichiers ELF bas-niveau).

Illustration



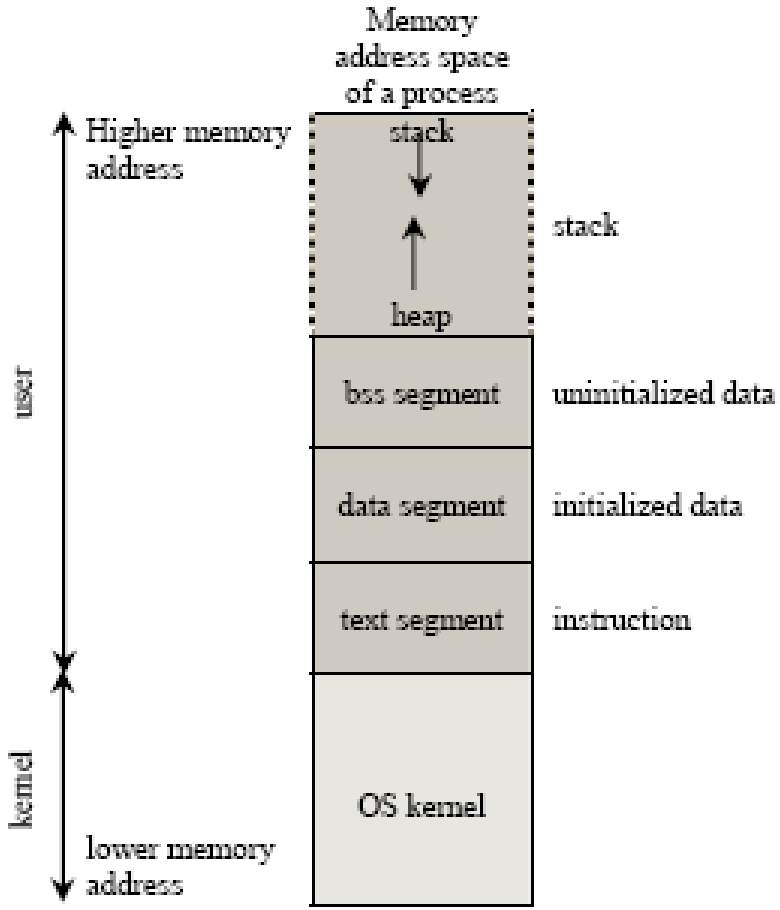
1

[1] source : http://www.tenouk.com

Comment le système charge un binaire sous forme de process?

1. Validation de la mémoire et de l'accès
   Le noyau lit les informations dans l'en-tête du programme et valide ou pas le droit d'accès, la demande de mémoire, etc puis *calcule les besoins en mémoire*.
2. Mise en place du process.
   allocation de mémoire, recopie les sections `.text` (le code) et `.data` (données initialisées) en mémoire, initialise les registres, saute à l'instruction de départ.

Illustration



[2]

[2] source : http://www.tenouk.com

3. Librairies

Les librairies

Grand principe : ne pas réinventer la roue.

Idée : mise en commun et normalisation / standardisation de code.

Exemple: librairie standard C.

Comment charger une librairie?

Il y a plusieurs façons.

- **statiquement** (mais sans recompiler) en assemblant les binaires (*static library*).
- **partagée** en assemblant leur binaires au moment de l'exécution (*shared library*).
- **dynamiquement** en assemblant leur binaires au besoin durant l'exécution (*dynamic library*).

Librairie statique

- Collection de fichiers objets
- historiquement premier type de librairies
- création avec `ar` (*archiver*)

```
ar rcs my_library.a file1.o file2.o
```

Librairie statique

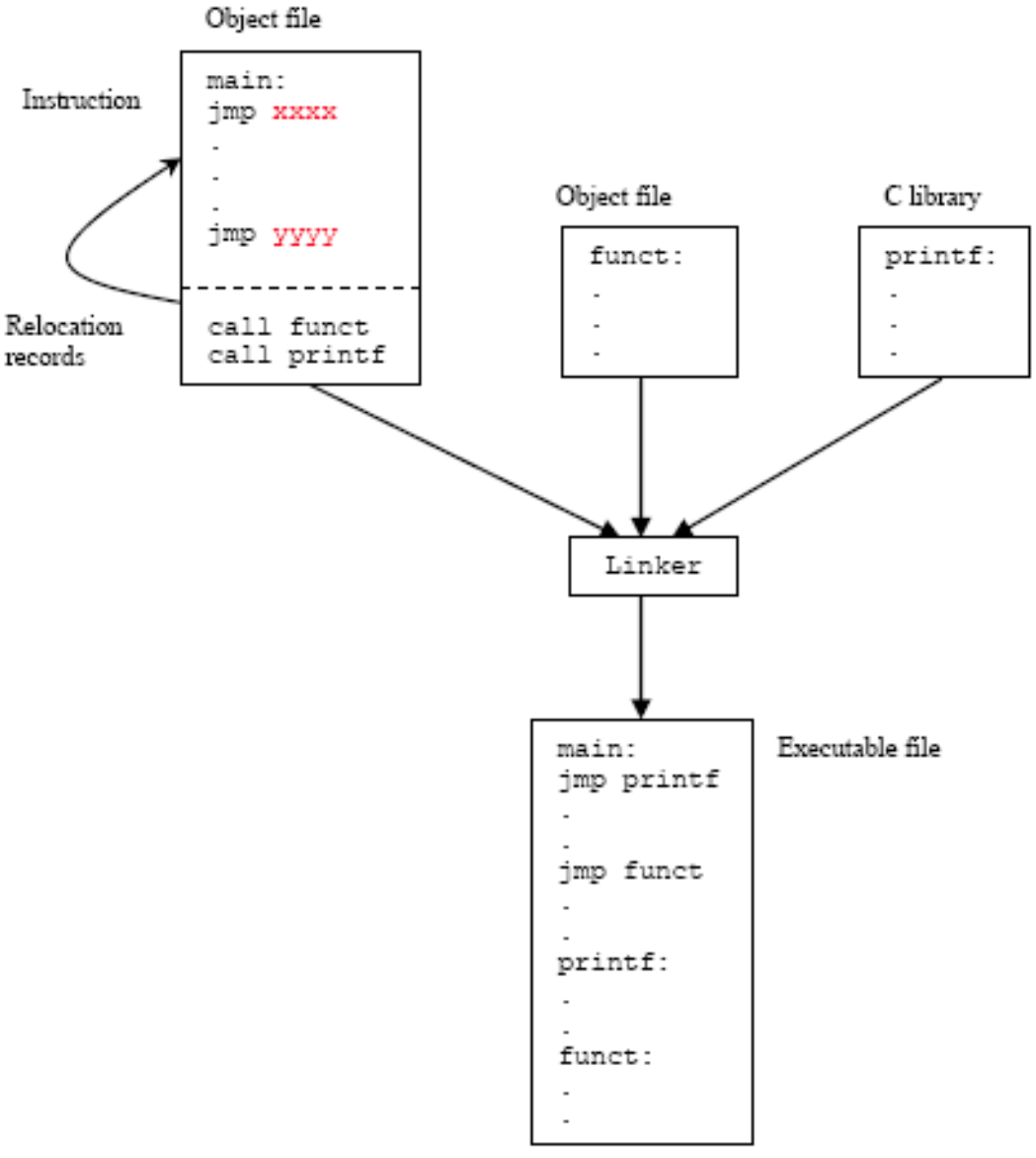
- Collection de fichiers objets
- historiquement premier type de librairies
- création avec `ar` (*archiver*)

```
ar rcs my_library.a file1.o file2.o
```

Avantages/Inconvénients

+ liens vers (binaire de) progs existants sans recompiler
  (à la préhistoire : compiler prenait beaucoup de temps)
+ Théorie : 1 à 5% plus rapide qu'autres librairies.
  Pratique pas forcément.
+ utilisation pour dev qui ne veulent pas donner leurs sources
− pour programmeur utilisant la librairie
− prend de la place.

Illustration



3

[3] source : http://www.tenouk.com

Librairie partagées

- Collection de fichiers objets
- chargée par le programme au démarrage

Librairie partagées

- Collection de fichiers objets
- chargée par le programme au démarrage

Avantages/Inconvénients

+ Permet de faire une mise à jour des libraries tout en conservant le support pour des librairies plus anciennes et incompatibles
+ Cacher/remplacer une librairie lors de l'exécution d'un programme spécifique.
+ Exécutable moins gros
− Il faut installer la librairie partagée!

Principe sous linux

Plusieurs noms.

- soname ln -s vers real name
- real name file with actual library code
- linker name (soname without version) link to the latest soname

Connaître les dependances d'un programme?

En général on en a au moins 2 :

- `/lib/ld-linux.so.N` (chargement de librairies)
- `/libc.so.N` (libc)

Principe sous linux

Plusieurs noms.

- soname ln -s vers real name
- real name file with actual library code
- linker name (soname without version) link to the latest soname

Connaître les dependances d'un programme?

En général on en a au moins 2 :

- `/lib/ld-linux.so.N` (chargement de librairies)
- `/libc.so.N` (libc)

```
$ ldd /bin/chmod
  linux-gate.so.1 =>  (0xb76eb000)
  libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7525000)
  /lib/ld-linux.so.2 (0xb76ec000)
```

Remarques

ATTENTION

Ne pas utiliser ldd sur un programme en lequel vous n'avez pas confiance!
(voir man ldd)

Changement de soname / version.

Si incompatible car l'ABI (*Application Binary Interface*) change.

Librairies chargée dynamiquement

- Collection de fichiers objets
- Chargement pas nécessairement au démarrage du programme. e.g. chargement *plugin* lorsque nécessaire.

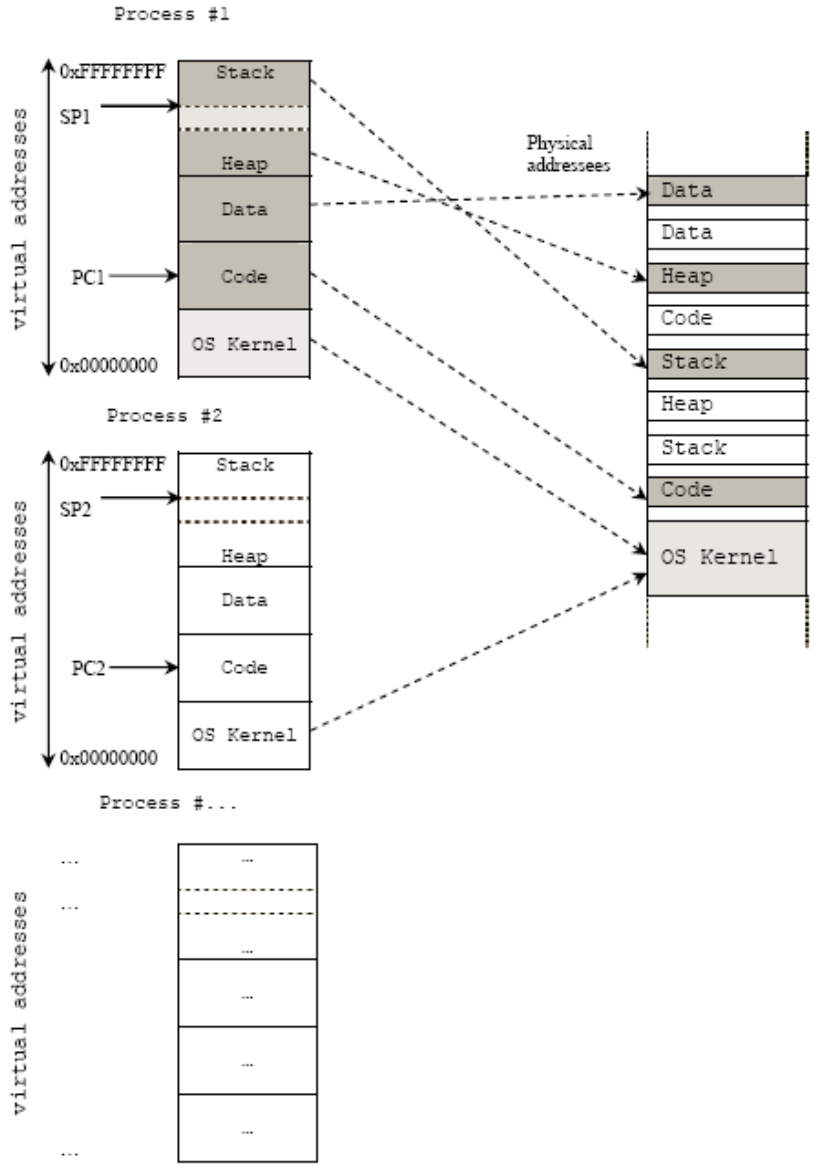
Sous linux :

- format identique : `.o` ou `.so`
- différence : utilisation d'une API spéciale via lib `<dlfcn.h>`
  fonctions : `dlopen`/ `dlclose`, `dlsym` (qui va chercher symboles dans librairies) et `dlerror`.

Exemple

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

int main(int argc, char **argv) {
  void *handle;
  double (*cosine)(double);
  char *error;

  handle = dlopen (”/lib/libm.so.6”, RTLD_LAZY);
  if (!handle) {
    fputs (dlerror(), stderr);
    exit(1);
  }

  cosine = dlsym(handle, ”cos”);
  if ((error = dlerror()) != NULL)  {
    fputs(error, stderr);
    exit(1);
  }

  printf (”%f\n”, (*cosine)(2.0));
  dlclose(handle);
}
```

Illustration



SP - Stack Pointer (the address hold by ESP register.
PC - Program counter.

4

[4] source : http://www.tenouk.com

4. Compilation séparée en C (par l'exemple)

Outils

- le préprocesseur (**cpp**);
- le compilateur (**gcc**);
- l'assembleur (**as**);
- le linker (**ld**).

Un exemple

Exf.c:

```
1 int f (int x) {
    return 2*x;
3 }
```

Exf.h:

```
1 int f (int i);
```

Un exemple (suite)

Exg.c:

```
1 #include "Exf.h"

3 int g (int i,int j) {
    while (i < j+1) i=f(i);
5   return j;
  }
```

Exg.h:

```
  #include "Exf.h"
2

  int g (int i,int j);
```

Un exemple (fin)

Exmain.c:

```c
#include <stdio.h>
#include "Exg.h"

int main ( void ) {
  int i;
  i = g(3,4);
  printf("%d",i);
}
```

Pré-compilation

Commandes: `cpp Exg.c -o Exg.i; cpp Exmain.c -o Exmain.i`

Résultat: interprétation des directives.

Génération du code assembleur

Commandes: `gcc -S Exf.c ; gcc -S Exg.i ; gcc -S Exmain.i`

Résultat:

```
        .file   "Exf.c"
        .text
        .globl  f
        .type   f, @function
f:
.LFB0:
        .cfi_startproc
        pushq   %rbp
<...>
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   f, .-f
        .ident  "GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3"
        .section        .note.GNU-stack,"",@progbits
```

Assemblage du code

Commande: `gcc -c Exf.s`

Résultat:

Un fichier binaire `Exf.o` que l'on peut analyser avec **objdump**:

- **objdump -f Exf.o** donne la table des symbole;
- **objdump -d Exf.o** visualise le code binaire.

Création d'une librairie statique

Commande: `ar rcs libEx.a exf.o exg.o`

Résultat: Une archive *ar* contenant les deux fichiers. Cette archive peut être analysée par **objdump**

Edition de lien

Programme: `ld`

commande:
```
ld -m elf_x86_64 -static -z relro -o Ex
/usr/lib/x86_64-linux-gnu/crt1.o
/usr/lib/x86_64-linux-gnu/crti.o
/usr/lib/gcc/x86_64-linux-gnu/4.6/crtbeginT.o
-L/usr/lib/gcc/x86_64-linux-gnu/4.6 Exmain.o libEx.a
–start-group -lgcc -lgcc_eh -lc –end-group
/usr/lib/gcc/x86_64-linux-gnu/4.6/crtend.o
/usr/lib/x86_64-linux-gnu/crtn.o
```

Résultat: l'exécutable.

Taille

Problème:

```
$ ls -lh Ex
-rwxr-xr-x 1 richardg personnels 863K févr. 21 13:21 Ex
```

5. Buffer overflows

Débordement de tampon

Principe général: profiter que le programme ne vérifie pas que la longueur des données saisies dans le tampon pour écraser des données dans l'espace d'adressage du processus et lui faire exécuter par exemple du code malicieux qu'on injectera.

Plusieurs approches

- (*stack overflow*) débordement dans la pile
- (*heap overflow*) débordement dans le tas
- (*integer overflow*) index en dehors d'un tableau, pas seulement trop grand, peut être aussi un entier négatif là où on attend normalement un entier positif.

Les pages suivantes (en particulier les figures) sont tirées du site `http://www.tenouk.com/Bufferoverflowc/`

Débordement de tampon

Principe général: profiter que le programme ne vérifie pas que la longueur des données saisies dans le tampon pour écraser des données dans l'espace d'adressage du processus et lui faire exécuter par exemple du code malicieux qu'on injectera.

Plusieurs approches

- (*stack overflow*) débordement dans la pile
- (*heap overflow*) débordement dans le tas
- (*integer overflow*) index en dehors d'un tableau, pas seulement trop grand, peut être aussi un entier négatif là où on attend normalement un entier positif.

Les pages suivantes (en particulier les figures) sont tirées du site `http://www.tenouk.com/Bufferoverflowc/`

Dans quelles conditions peut-on avoir ce problème?

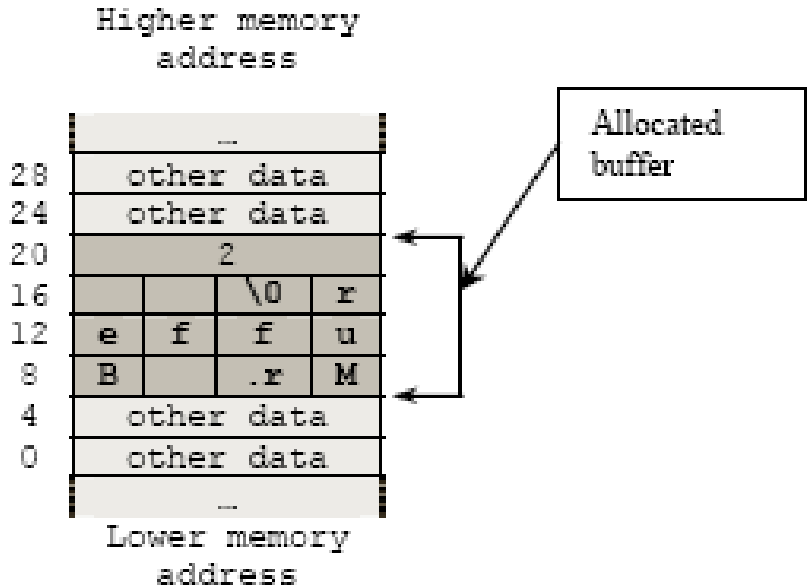
1. **Pas de sûreté du typage**.
   Par exemple en C et C++ il y a des fonctions de la librairie standard (gestion mémoire, manipulation de chaînes de caractères) qui ne vérifient ni la taille des tableaux, ni les types (c'est au programmeur d'utiliser les versions sûres de ces fonctions ou bien de faire les vérifications).
2. **Accès ou copie d'un tampon sur la pile de manière non sûre**.
   Exemple: déclaration de 100 bytes, mais recopie de 150 bytes.
3. **Emplacement du tampon proche de section critique sur la pile**.
   Exemple: l'emplacement réservé pour le buffer est proche du bloc d'activation d'un appel de fonction.

Les tampons et la pile

Code C

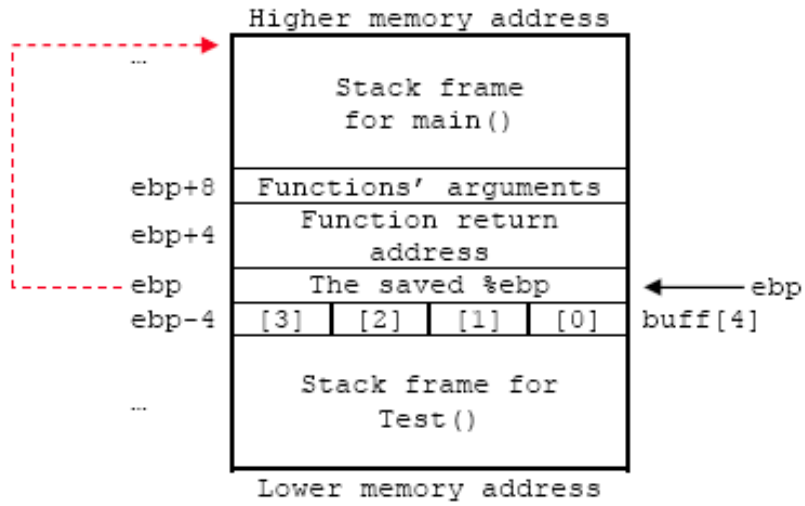
```
// in C, for string array, it is NULL (\\0) terminated
char Name[12] = "Mr. Buffer";
int num = 2;
```

Sur la pile:



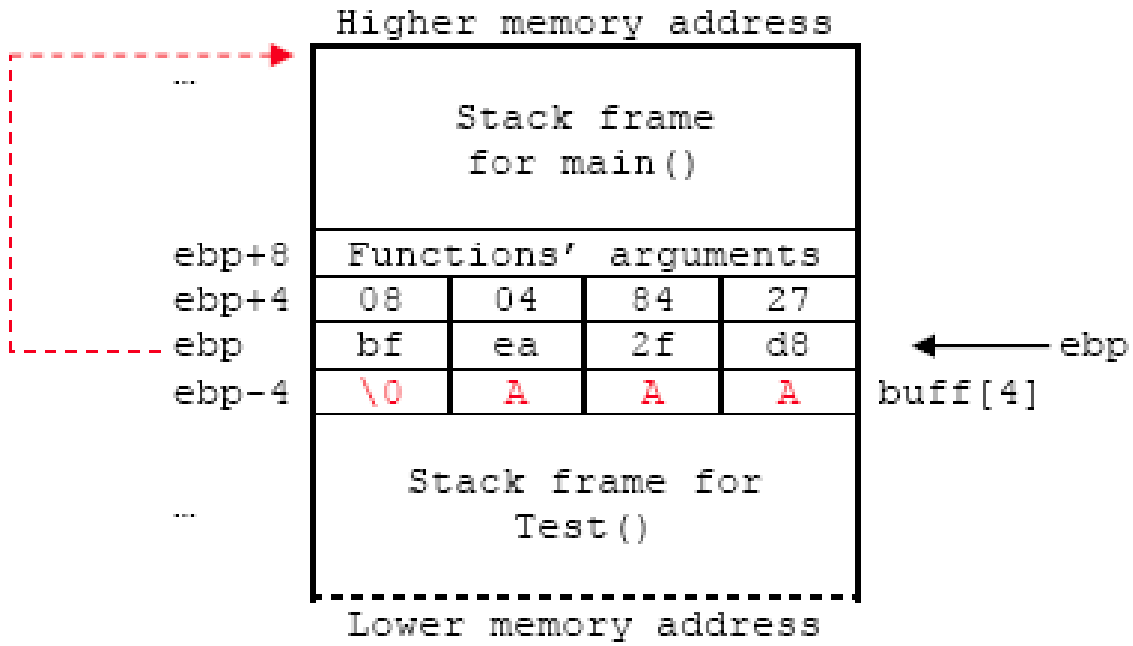
Exemple

```
/* test buffer program */
#include <unistd.h>
void Test(){
  char buff[4];
  printf("Some input: ");
  gets(buff);
  puts(buff);
}

int main(int argc, char *argv[]){
  Test();
  return 0;
}
```



Example : lorsque le tampon déborde

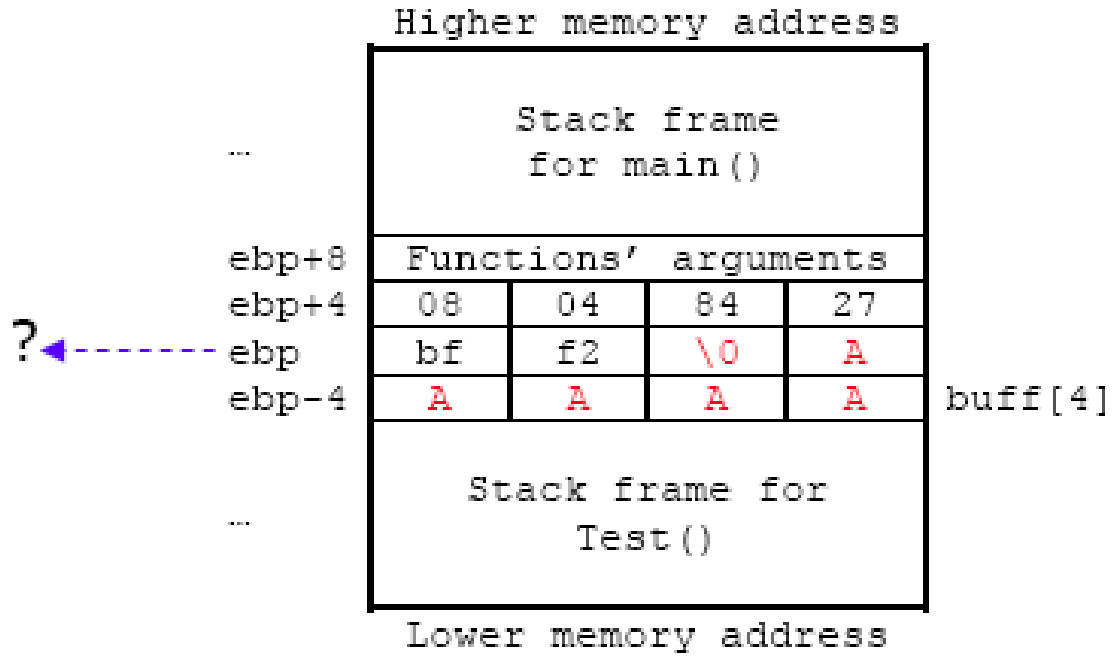
Saisie: AAA



Example : lorsque le tampon déborde

Saisie: AAAAA

L'ancienne valeur du registre "pile de fonction" est perdue (notre notation en MVàP : fp, ici ebp).

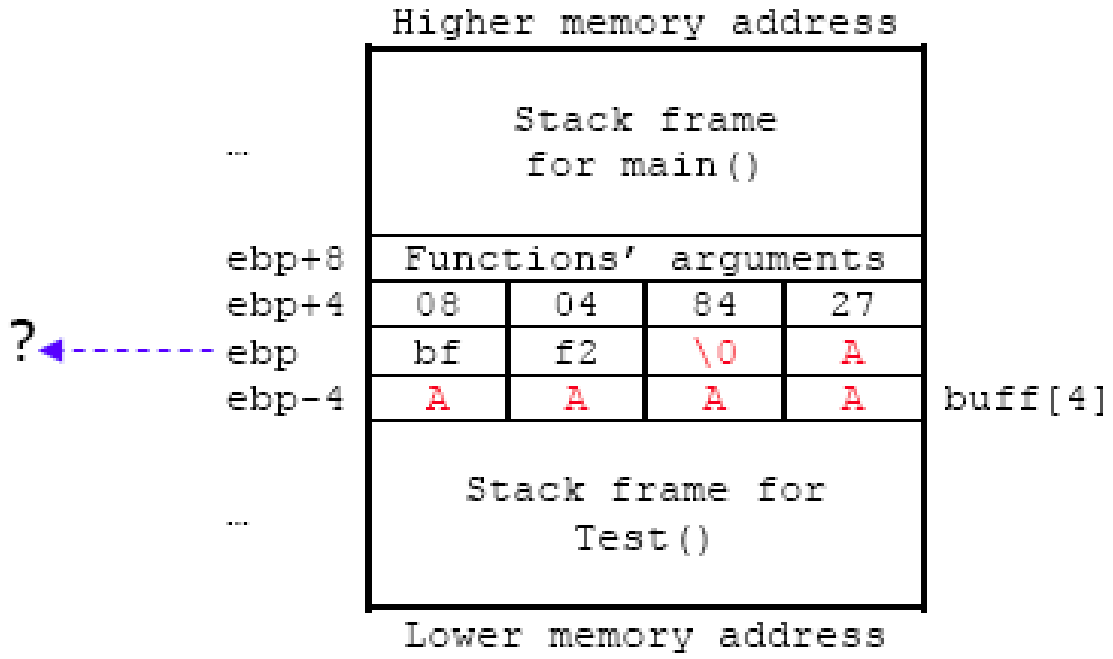


Le tampon a débordé dans la zone critique (bloc d'activation de la fonction appelante)

Example : lorsque le tampon déborde

Saisie: AAAAA

L'ancienne valeur du registre "pile de fonction" est perdue (notre notation en MVàP : fp, ici ebp).

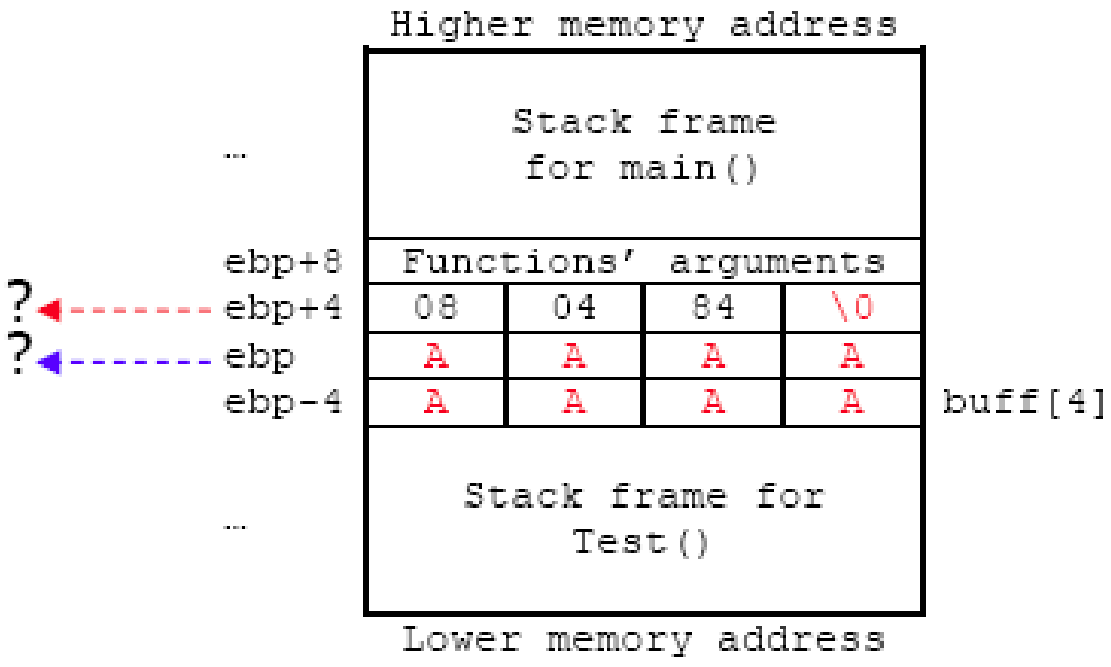


Le tampon a débordé dans la zone critique (bloc d'activation de la fonction appelante)

Example : lorsque le tampon déborde

Saisie: AAAAAAAA

L'adresse de retour dans le code (paramètre du RETURN pour la MVàP) est compromise.

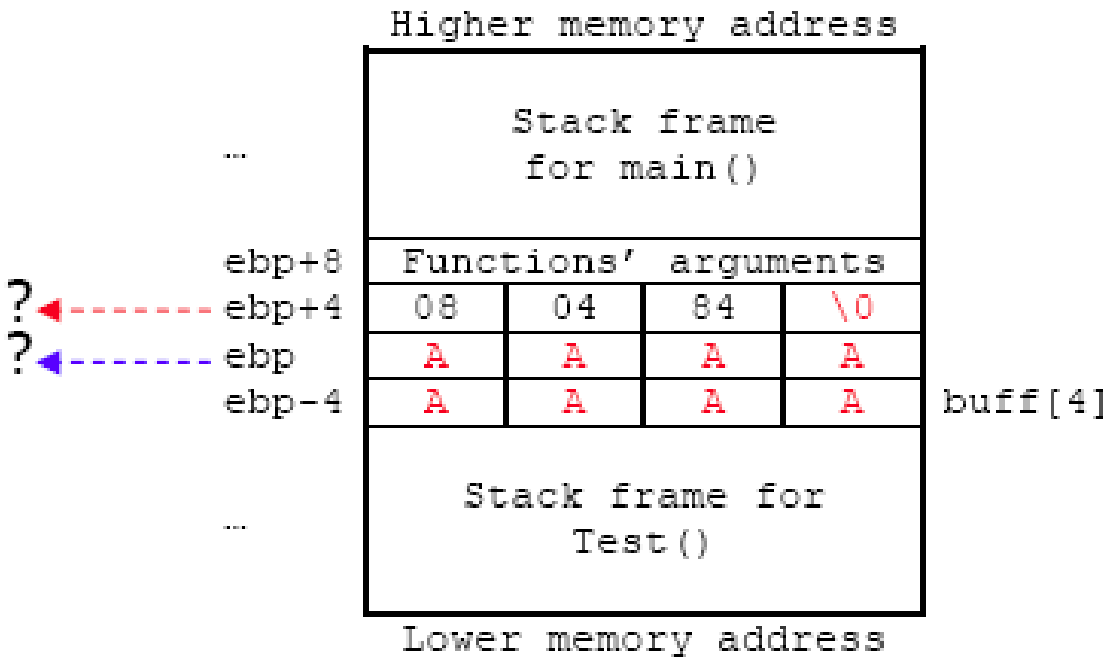


Le tampon a débordé encore plus loin dans la zone critique (bloc d'activation de la fonction appelante).

Example : lorsque le tampon déborde

Saisie: AAAAAAAA

L'adresse de retour dans le code (paramètre du RETURN pour la MVàP) est compromise.



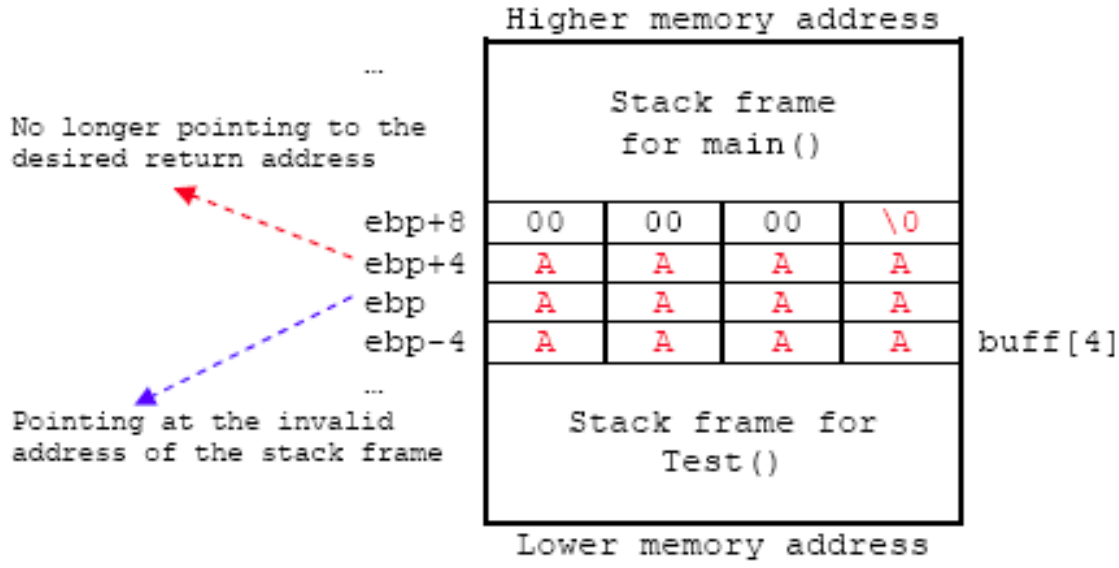
Le tampon a débordé encore plus loin dans la zone critique (bloc d'activation de la fonction appelante).

Example : lorsque le tampon déborde

Saisie:

AAAAAAAAAAAA

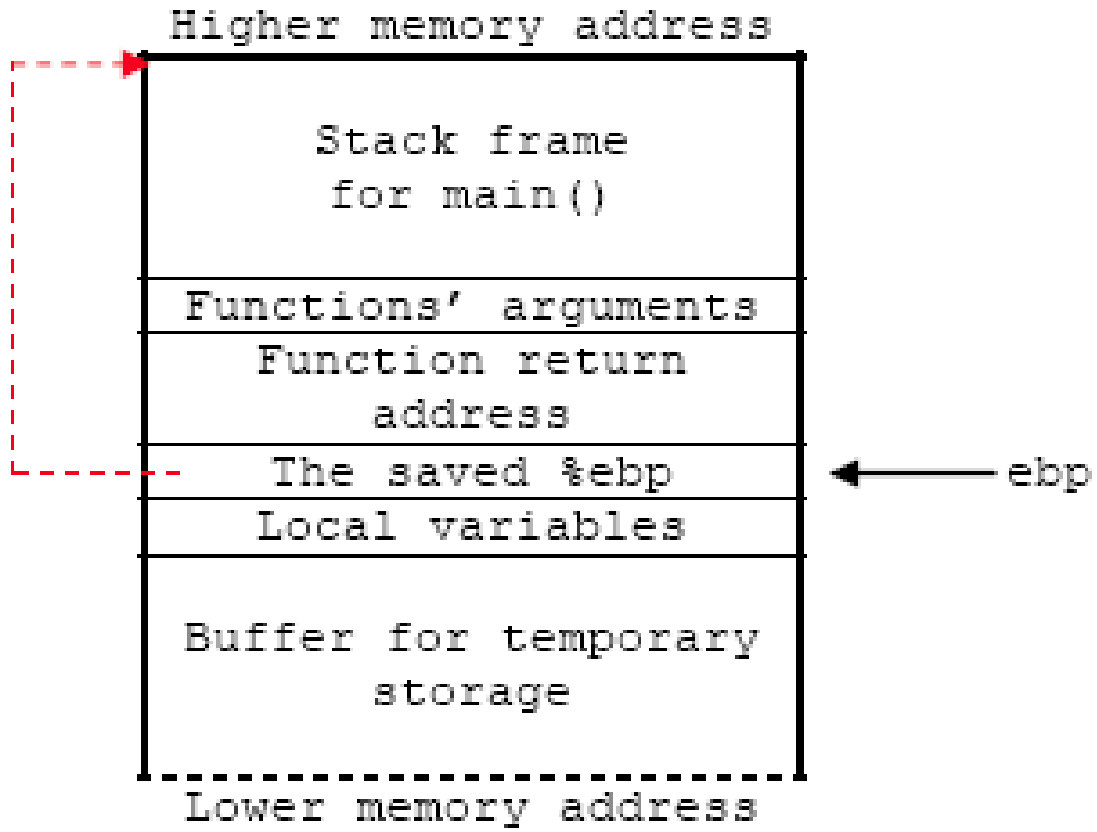
L'adresse de retour est complètement corrompue.



Example : lorsque le tampon déborde

État normal:

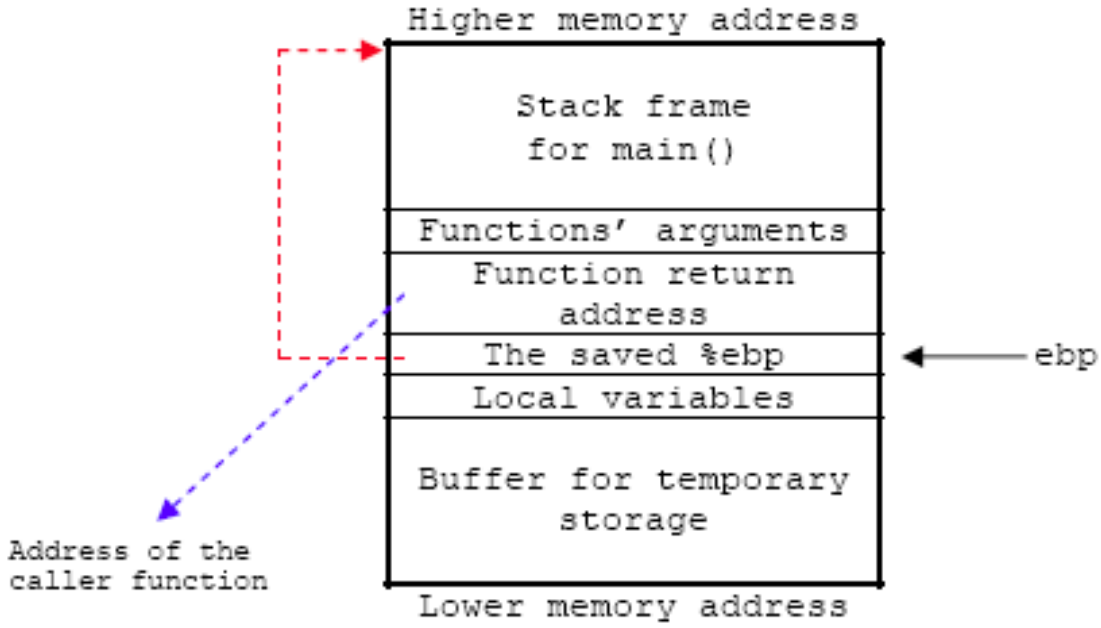
adresse de La pile de la fonction appelante.



Example : lorsque le tampon déborde

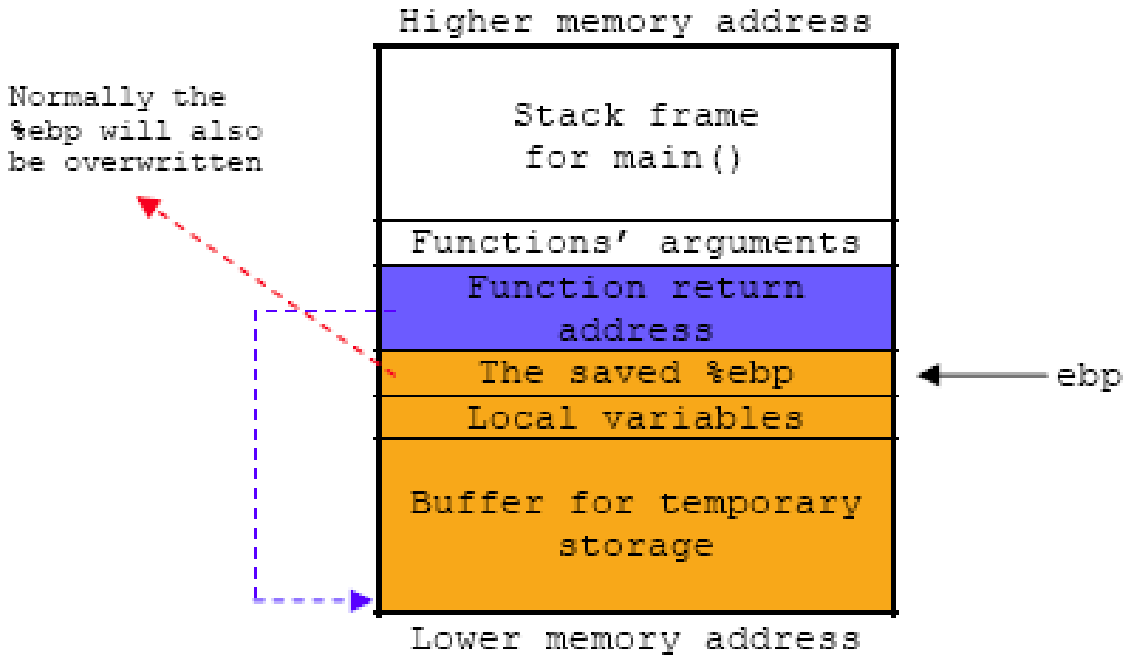
État normal:

adresse du code de retour.



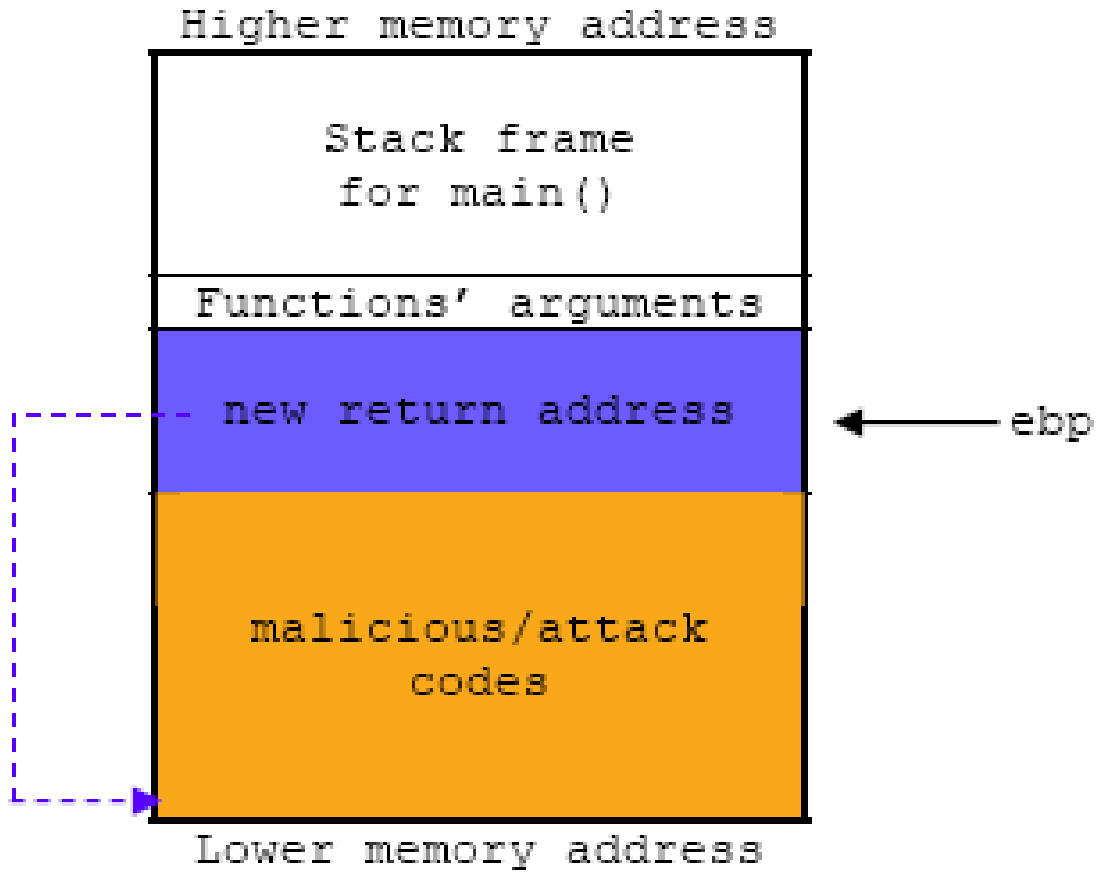
Example : lorsque le tampon déborde

Attaque par débordement de pile: corruption et changement de l'adresse du code de retour!.



Example : lorsque le tampon déborde

Attaque par débordement de pile: corruption adresse du code de retour redirigé vers code injecté dans buffer.



Contre-mesure

En pratique, il existe des protections et on ne peut pas forcément exécuter n'importe quelle code en mémoire.

NX bit: pour *Never eXecute*.

Dissociation entre zones de mémoire contenant des instructions, donc exécutables, et zones contenant des données.

Le nom précis diffère selon le fabriquant de processeur Intel XD bit (eXecute Disable), AMD Enhanced Virus Protection, ARM XN (eXecute Never).

Contre-attaque

En présence du NX bit, on peut exécuter du code déjà présent (et donc forcément exécutable) par exemple celui de la fonction `System()` de la libc, qui permet d'exécuter un programme arbitraire.

Nom de l'attaque : **return-to-libc**

Contre-mesure: *address space layout randomisation* (ASLR) qui charge les fonctions en mémoires à des positions aléatoires. Cette méthode n'est pas forcément suffisante sur un système 32-bits (seuls 16 bits sont aléatoires, une attaque par force brute est possible).

Comment se protéger?

Le plus simple reste toutefois de n'utiliser que des librairies sûres de C et de faire des contrôles dans les programmes.

6. Optimisation

Optimisation

Objectif: produire du code plus efficace (pas forcément optimal).

Transformations indépendantes de la machine:

- simplifier le code;
- enlever les calculs ou instructions inutiles;
- n'exécuter les calculs qu'une fois;
- spécialiser du code (dépliage de code).

Transformations dépendantes de la machine:

- remplacer des instructions complexes par des plus simples;
- exploiter les instructions spéciales du processeur;
- exploiter la hiérarchie mémoire (registre, cache, …) ;
- exploiter le parallélisme permis par le matériel

Manipulation du code

Suppression du code inaccessible

```
        goto L
        { code inaccessible }
    L:
```

Simplification des branchements

```
        goto L1
        { code }
    L1:
        goto L2
        { code }
    L2:
        { code }
```

Méthode: se fait par analyse du graphe de flot de contrôle (enchaînements des blocs de code)

Suppression calculs inutiles

Méthode: Ne pas recalculer la valeur d'une expression dont les éléments n'ont pas été changés.

```
main() {
        int a, b, c, d, e;
        init(&a, &b, &c, &d, &e);
              b = a + 42;
        c = b + 3;
        if (c > e) c = d;
        e = a + 42;
        ...
    }
}
```

Sortir un calcul d'expression d'une boucle (invariants de boucle)

Méthode: Sortir un calcul d'expression d'une boucle (invariants de boucle)

```
for (i=0; i < n; i++) {
  for (j = 0; j < m; j++) {
    A[i][j] = B[i][j] * c + 3 * E[i];
  }
}
```

Sortir un calcul d'expression d'une boucle (invariants de boucle)

Méthode: Sortir un calcul d'expression d'une boucle (invariants de boucle)

```
for (i=0; i < n; i++) {
  for (j = 0; j < m; j++) {
    A[i][j] = B[i][j] * c + 3 * E[i];
  }
}
```

```
for (i=0; i < n; i++) {
  t = 3 * E[i];
  for (j = 0; j < m; j++) {
    A[i][j] = B[i][j] * c + t;
  }
}
```

Dépliage de boucles

Transformation: le code

```
Pour i de 1 à 3 faire
2    A[i] = 0
```

devient

```
A[1] = 0
2 A[2] = 0
A[3] = 0
```

et d'autres ...

Optimisations possibles:

- réduction des variables d'instructions;
- propagation des copies;
- propagation de constantes;
- simplification algébrique;
- suppression des vérifications inutiles;
- ...

Problèmes:

- Introduction possible de bugs;
- Changement d'un comportement non-défini.

7. Grammaire antlr

Antlr, ça ne sert à rien?

Kudos. I'm actually really liking ANTLR! I have a pretty darn good velocity with a rapid prototyping project I am doing in my Google 20% time. For example, I just discovered a feature in rewrite rules that does exactly what I need (referencing previous rule ASTs, p. 174 in your book). It took me about 5 minutes to get this to work and remove an ugly wart from my grammar. Hats off!
Guido van Rossum, Inventor of Python.

Antlr, ça ne sert à rien?

ANTLR is an exceptionally powerful and flexible tool for parsing formal languages. At Twitter, we use it exclusively for query parsing in Twitter search. Our grammars are clean and concise, and the generated code is efficient and stable. The book is our go-to reference for ANTLR v4 – engaging writing, clear descriptions and practical examples all in one place.
Samuel Luckenbill, Senior Manager of Search Infrastructure, Twitter, inc.

Exemples

À défaut d'acheter le livre, vous pouvez vous reporter vous au site d'antlr sur lequel il y a beaucoup d'exemples, en particulier ceux du livre.