

Introduction au Développement en C sous UNIX

(4ème édition)

Bruno Quoitin
(bruno.quoitin@uclouvain.be)

Sébastien Barré
(sebastien.barre@uclouvain.be)

Virginie Van den Schrieck
(virginie.vandenschrieck@uclouvain.be)

Introduction

◆ **Constatation**

◆ **Programmation en C/C++**

- ◆ Toujours largement utilisée
- ◆ Systèmes d'exploitation
- ◆ Protocoles et applications réseau
- ◆ Systèmes embarqués
- ◆ Majorité des utilitaires/applications UNIX

◆ **Difficultés**

- ◆ Gestion de la mémoire: segmentation faults, bus errors, memory leaks, ...
- ◆ Erreurs de syntaxe / type
- ◆ Objets non trouvés: header (.h) / librairies / symboles
- ◆ Organisation / maintenance du code

Introduction

♦ But du cours:

- ♦ Donner quelques règles de bonne pratique
 - ♦ Organisation d'un projet
 - ♦ Utilisation d'un système de contrôle de version
 - ♦ Conventions de codage
- ♦ Introduire les outils disponibles sous UNIX / Linux
 - ♦ Compilation C: **gcc** (et **cpp**)
 - ♦ Débogueur: **gdb** (**ddd**), **valgrind**, **strace**
 - ♦ Automatisation de la compilation: **make**
 - ♦ Contrôle de version: **cvs** / **svn**
- ♦ Organisation
 - ♦ 1-2 heures de cours
 - ♦ Prérequis: notions de base de C

Table des matières

- ◆ **I. Recommandations générales**
- ◆ **II. Compilation**
- ◆ **III. Automatisation de la compilation**
- ◆ **IV. Débogage**
- ◆ **V. Contrôle de version**
- ◆ **VI. Conclusion**
- ◆ **VII. Références**

Processus de développement

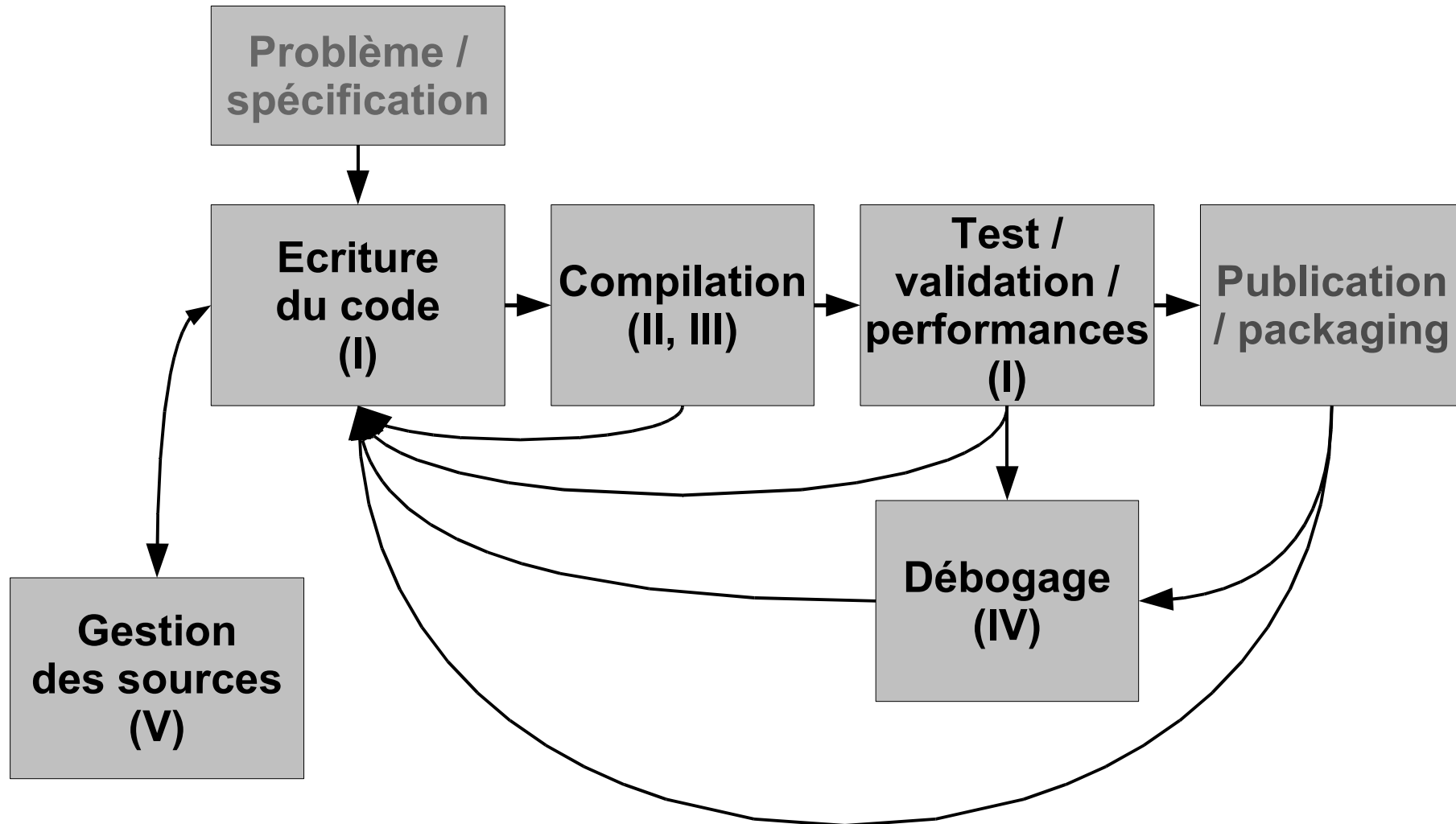
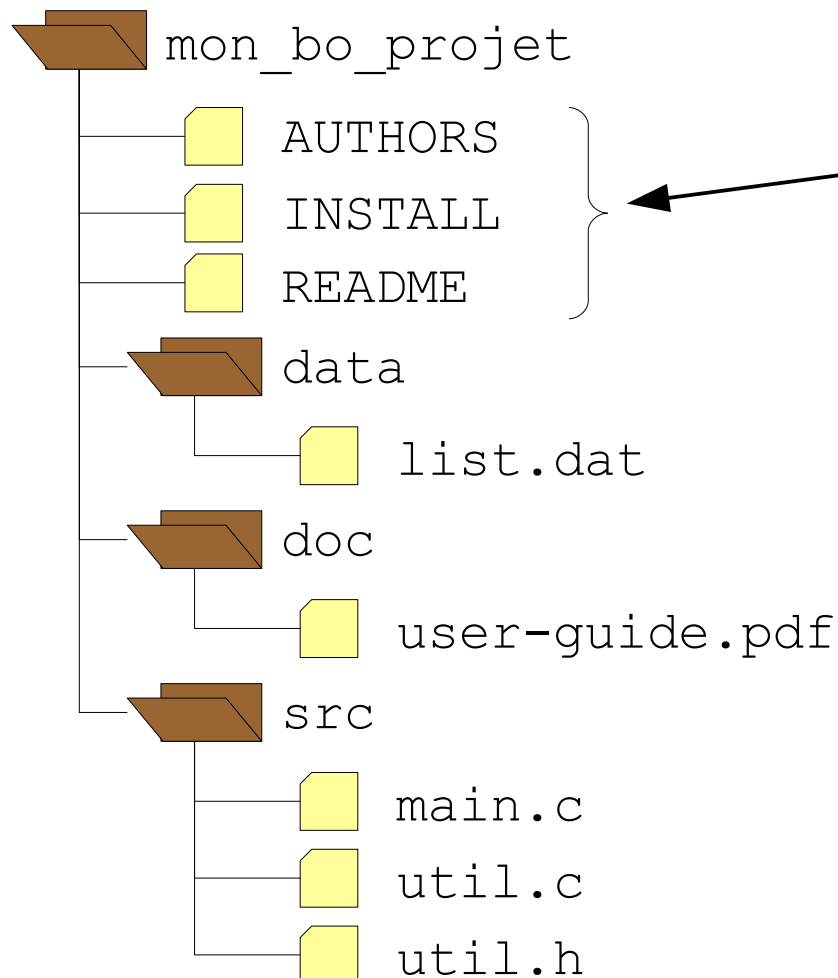


Table des matières

- ➔ **I. Recommendations générales**
- ◆ **II. Compilation**
- ◆ **III. Automatisation de la compilation**
- ◆ **IV. Débogage**
- ◆ **V. Contrôle de version**
- ◆ **VI. Conclusion**
- ◆ **VII. Références**

Recommendations

► Organisation des sources



Bonne pratique:

placez à la racine de votre projet les fichiers

- **README** décrivant ce que fait le projet
- **INSTALL** comment compiler et installer
- **AUTHORS** contenant la liste des auteurs

Bonne pratique:

séparez les fichiers selon leur fonction: sources (src), données (data), documentation (doc), ...

Recommendations

◆ Organisation des sources

```
/* =====  
 * My very first C program  
 * (C) 1980, B. Gates  
 * Intended license: public domain  
 * ===== */  
  
#include <stdio.h>  
  
// -----[ main ]-----  
/**  
 * Entry point  
 */  
main()  
{  
    printf("MS-DOS 0.1 starting...\n");  
    return 0;  
}
```

Bonne pratique:

au début du source
- décrivez le contenu
- copyright, licence,
auteur(s)

Bonne pratique:

- limitez la longueur
des lignes à 80 caractères
- limitez la longueur des
fonctions

Recommendations

◆ Gestion des erreurs

- ◆ Testez les arguments passés par l'utilisateur (nombre et valeurs)

```
if ((argc < 3) || (argc > 5)) {  
    fprintf(stderr, "Erreur: le nombre d'arguments doit être "  
                  "compris entre 2 et 4\n");  
    return(EXIT_FAILURE);  
}
```

- ◆ Codes de sortie standards (ou documentés)

EXIT_FAILURE / EXIT_SUCCESS (stdlib.h)

- ◆ Reportez les erreurs sur la sortie d'erreur standard !

```
fprintf(stderr, "Erreur: qqechose a foiré !\n");
```

Recommendations

◆ Gestion des erreurs

- ◆ Testez le code de retour des appels système !
 - ◆ Convention: 0 \Rightarrow succès, $< 0 \Rightarrow$ erreur
 - ◆ Vérifier dans les “*man pages*”

```
in_stream= fopen("list.txt", "r");  
if (in_stream == NULL) {  
    perror("Erreur: impossible d'ouvrir \"list.txt\"");  
    exit(EXIT_FAILURE);  
}
```

- ◆ Code d'erreur du dernier appel système disponible dans la variable globale **errno** (`errno.h`)

Bonne pratique:

`perror()` est une fonction standard qui affiche la raison pour laquelle un appel système a échoué (défini dans `stdio.h`).

Recommendations

◆ Gestion du tas (heap)

- ◆ Même `malloc()` peut échouer s'il n'y a plus assez de mémoire

```
char * str= (char *) malloc(100*sizeof(char));  
if (str == NULL) {  
    perror("Erreur: pas possible d'allouer de la mémoire");  
    exit(EXIT_FAILURE);  
}
```

- ◆ Ne pas oublier de libérer la mémoire

```
free(str);
```

Recommendations

- ◆ **Lisez la documentation (man pages) !!!**

- ◆ Tapez “*man* <cmd>” dans une console pour obtenir de l'info sur la commande ou l'appel système <cmd>.

man gdb -> donne de l'info sur le débogueur *gdb*

man fopen -> donne de l'info sur la fonction d'ouverture de fichier *fopen*

- ◆ Exemple: la fonction **gets()**

“BUGS

Never use gets(). Because it is impossible to tell without knowing the data in advance how many characters gets() will read, and because gets() will continue to store characters past the end of the buffer, it is extremely dangerous to use. It has been used to break computer security. Use fgets() instead.”

Recommendations

◆ Documentation

◆ Liste des fonctions C standard:

- ◆ <http://www.utas.edu.au/infosys/info/documentation/C/CStdLib.htm>
- ◆ http://www.acm.uiuc.edu/webmonkeys/book/c_guide/index.html

Tests

- ◆ **Pre-condition Tests**

- ◆ Vérifier des conditions dans le programme
- ◆ Tests internes

- ◆ **Validation Tests**

- ◆ Vérifier les fonctionnalités d'un programme

- ◆ **Regression Tests**

- ◆ Vérifier les fonctionnalités et l'interface d'un programme / librairie après modification
- ◆ Tests généralement externes (scripts, programme utilisateur, expect)

Pre-condition Tests

◆ Fonctions / macros utiles

◆ assert(CONDITION)

```
ptr= malloc(...);  
assert(ptr != NULL);
```

◆ abort()

```
if (CONDITION == 0)  
    abort();
```

◆ macros `__LINE__`, `__FILE__`(, `__FCT__`)

```
if (CONDITION == 0) {  
    fprintf(stderr, "Warning: blabla at line %d, in file %s\n",  
        __LINE__, __FILE__);  
}
```

Tests Unitaires

◆ Unit Testing

- ◆ CUnit (~JUnit): <http://cunit.sourceforge.net>
- ◆ Dans libgds: <http://libgds.info.ucl.ac.be>

◆ Guidelines

- ◆ Tests courts
- ◆ Une seule fonctionnalité par test
- ◆ Beaucoup de tests

Tests Unitaires

◆ Exemple

```
// -----[ test_bgp_filter_action_comm_add ]-----  
int test_bgp_filter_action_comm_add()  
{  
    filter_action_t * action= filter_action_comm_append(1234);  
    UTEST_ASSERT(action != NULL, "New action should not be NULL");  
    UTEST_ASSERT(action->code == FT_ACTION_COMM_APPEND,  
                  "Action code should be FT_ACTION_COMM_APPEND");  
    filter_action_destroy(&action);  
    UTEST_ASSERT(action == NULL, "Action should be NULL when destroyed");  
    return UTEST_SUCCESS;  
}
```

Recommendations

- ◆ **Définissez des convention de codage !**
 - ◆ Raisons:
 - ◆ Lisibilité et vérifiabilité du code accrues
 - ◆ Travail en groupe facilité
 - ◆ Que couvre une telle convention ?
 - ◆ Nomage des variables, fonctions, fichiers
 - ◆ Indentation, parenthésage
 - ◆ Types, structures, librairies autorisées
 - ◆ ... et tout ce que vous estimez important
 - ◆ Conventions de codage existantes:
 - ◆ Linux kernel: <http://www.linuxjournal.com/article/5780>
 - ◆ GNU standards: <http://www.gnu.org/prep/standards>
 - ◆ GNU indent: <http://www.gnu.org/software/indent>

Table des matières

- ◆ I. Recommandations générales
- ➔ **II. Compilation**
- ◆ III. Automatisation de la compilation
- ◆ IV. Débogage
- ◆ V. Contrôle de version
- ◆ VI. Conclusion
- ◆ VII. Références

GCC

◆ Exemple classique “Hello World”

```
/* =====  
 * Introduction au développement en C  
 * Programme d'exemple 1  
 *  
 * (c) 2006-2009, Université catholique de Louvain (UCLouvain)  
 *           Ecole Polytechnique de Louvain (EPL)  
 *           Département d'Ingénierie Informatique (INGI)  
 * ===== */  
  
#include <stdio.h>  
  
/* ----[ main ]----- */  
int main()  
{  
    printf("Hello World !\n");  
    return 0;  
}
```

GCC

◆ Compilation

- ◆ GNU C Compiler (**gcc**)
- ◆ Création d'un exécutable à partir d'un fichier source (vérification syntaxique, vérification de types, génération de code exécutable, ...)
- ◆ Exemple:

```
[toto@brol src] gcc -o prog1 prog1.c  
prog1.c:14:10: missing terminating " character  
prog1.c: In function `main':  
prog1.c:15: error: syntax error before "return"  
[toto@brol src]
```

En cas d'erreur, gcc indique

- 1). le fichier concerné
- 2). la ligne présumée de l'erreur
- 3). la cause de l'erreur

GCC

◆ Compilation

- ◆ GNU C Compiler (**gcc**)
- ◆ Création d'un exécutable à partir d'un fichier source (vérification syntaxique, vérification de types, génération de code exécutable, ...)
- ◆ Exemple:

```
[toto@brol src] gcc -o prog1 prog1.c  
[toto@brol src] ./prog1  
Hello World !  
[toto@brol src]
```

S'il n'y a ni erreur ni avertissement, gcc n'affiche pas de message.

GCC: Options

-Wall

Affiche tous les avertissements (*warnings*)

Bonne pratique:
toujours compiler
avec **-Wall** et
-Werror

-Werror

Considère les warnings comme des erreurs (empêche la compilation de se poursuivre en cas de *warning*).

-ansi

Vérifie que le code C est conforme à la norme ISO C90.

-pedantic

Affiche les avertissements recommandés par la norme ISO C.

-o <name>

Renomme le fichier de sortie en *<name>*.

Par défaut, le fichier généré est nommé ***"a.out"***.

GCC: Etapes de la compilation

1). Pré-processing:

```
gcc -o prog1.e -E prog1.c
```

2). Compilation:

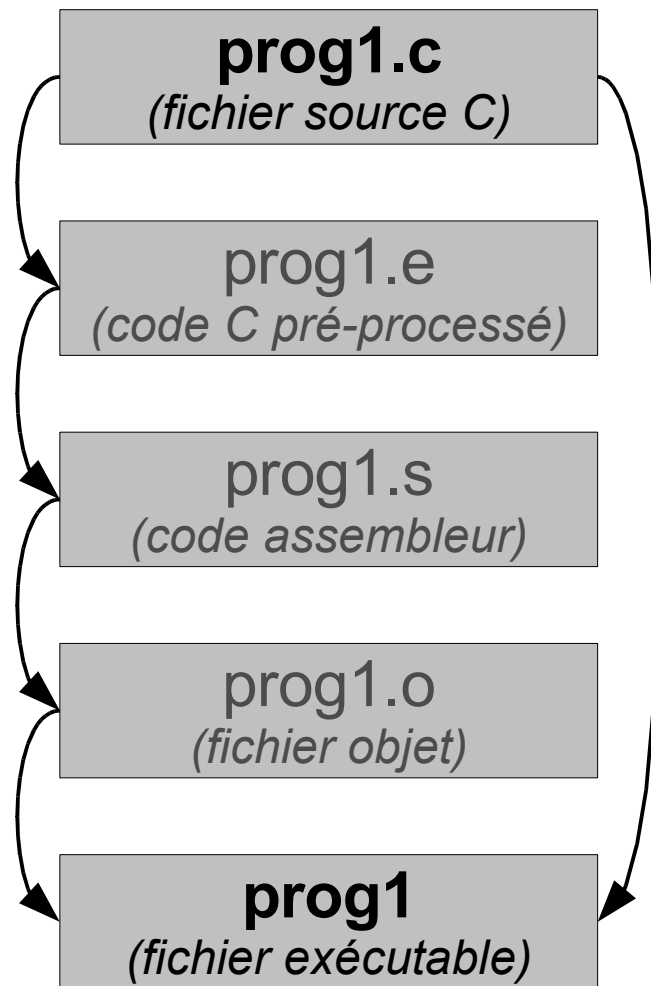
```
gcc -o prog1.s -S prog1.e
```

3). Assemblage:

```
gcc -o prog1.o -c prog1.s
```

4). Edition de liens (linkage):

```
gcc -o prog1 prog1.o
```



“Compilation”:

```
gcc -o prog1 prog1.c
```


GCC: Etapes de la compilation

1). Pré-processing:

```
gcc -o prog1.e -E prog1.c  
gcc -o util.e -E util.c
```

2). Compilation:

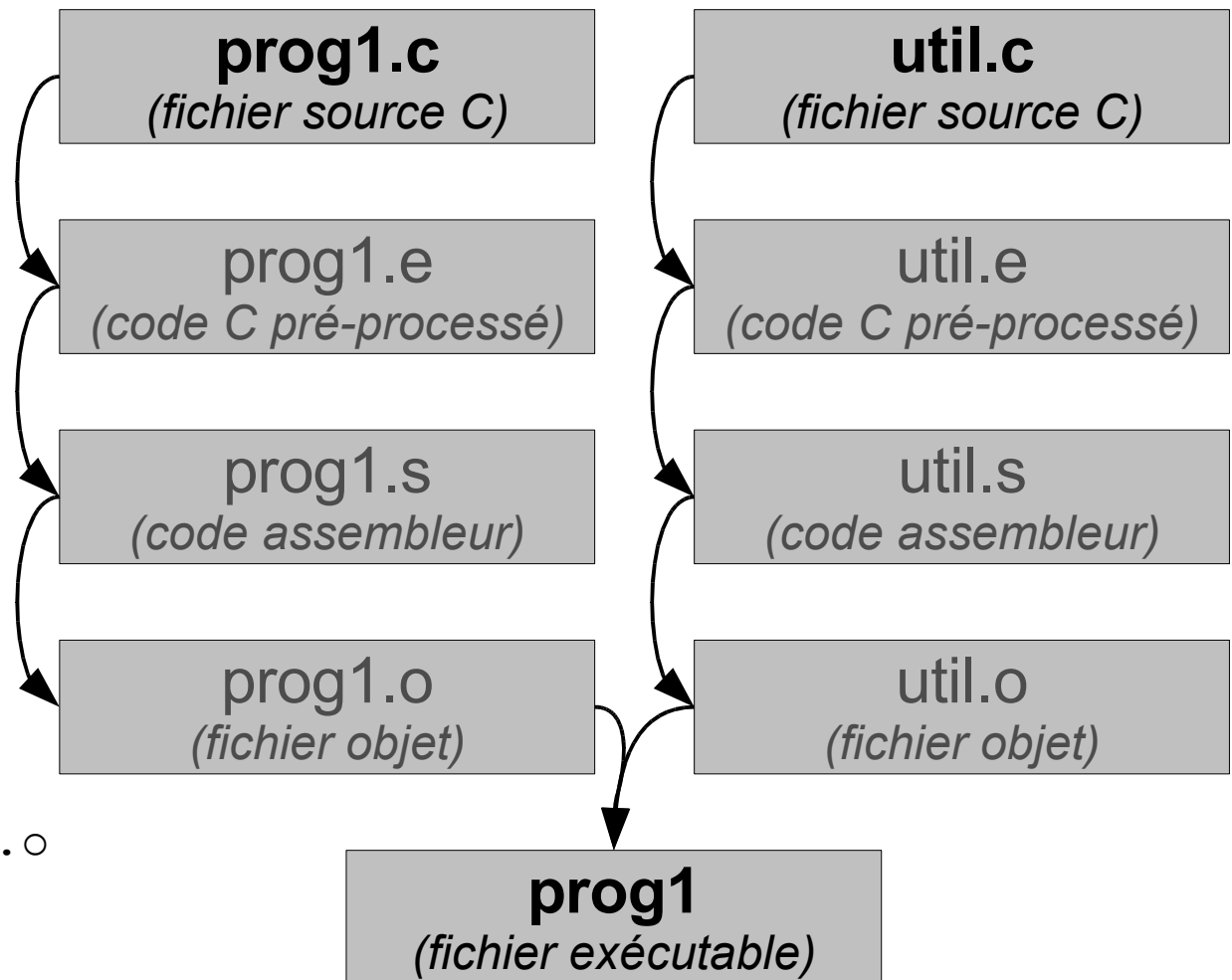
```
gcc -o prog1.s -S prog1.e  
gcc -o util.s -S util.e
```

3). Assemblage:

```
gcc -o prog1.o -c prog1.s  
gcc -o util.o -c util.s
```

4). Edition de liens (linkage):

```
gcc -o prog1 prog1.o util.o
```



GCC: Etapes de la compilation

1). Pré-processing:

```
gcc -o prog1.e -E prog1.c
```

2). Compilation:

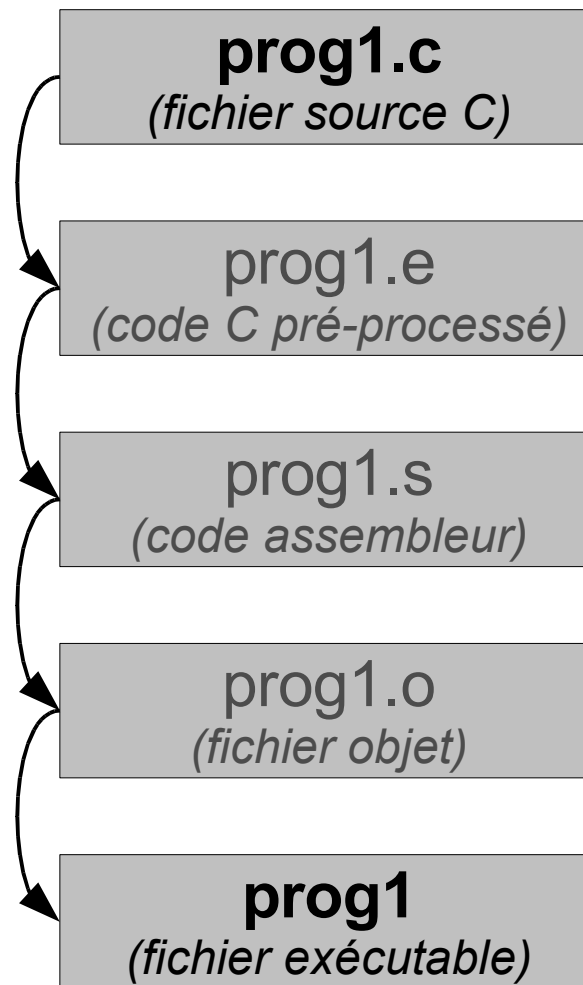
```
gcc -o prog1.s -S prog1.e
```

3). Assemblage:

```
gcc -o prog1.o -c prog1.s
```

4). Edition de liens (linkage):

```
gcc -o prog1 prog1.o
```



Erreurs de pré-processing

Header (.h) non trouvé,
macro incorrecte,
compilation conditionnelle
(#ifdef / #else / #endif),
...

Erreurs de compilation

Erreur de syntaxe,
erreur de type,
symbole non défini
...

Erreurs de linkage

Librairie non trouvée,
symbole non défini,
définitions multiples,
...

GCC: Etapes de la compilation

♦ 1). Pre-Processing

- ♦ GNU C Pre-Processor (**cpp**)
- ♦ Transforme le programme avant la compilation
- ♦ Convertit les macros et directives:
 - ♦ `#define`, `#undef`
 - ♦ `#include`
 - ♦ `#ifdef`, `#else`, `#endif`
- ♦ Fichiers headers (.h) placés dans l'output
 - ♦ option “`-Idir`” permet d'indiquer où trouver les headers
 - ♦ Chemin standard: `/usr/include` (dépend du système)
- ♦ Supprime les commentaires

GCC: Etapes de la compilation

◆ 2). Compilation

- ◆ Génération de code, allocation des registres, optimisations ⇒ code d'assemblage

```
.file    "progl.c"
.section .rodata
.LC0:
.string  "Hello World !\n"
.text
.globl main
.type   main, @function
main:
    pushl    %ebp
    movl     %esp, %ebp
    subl     $8, %esp
    andl     $-16, %esp
    movl     $0, %eax
    subl     %eax, %esp
    movl     $.LC0, (%esp)
    call     printf
    movl     $0, %eax
```

GCC: Etapes de la compilation

◆ 3). Assemblage

- ◆ GNU Assembler (**as**)
- ◆ Traduction du code d'assemblage en langage machine ⇒ fichier “objet” (binaire)

```
[toto@brol src] hexdump prog1.o
00000000 457f 464c 0101 0001 0000 0000 0000 0000
00000010 0001 0003 0001 0000 0000 0000 0000 0000
00000020 00e0 0000 0000 0000 0034 0000 0000 0028
00000030 000b 0008 8955 83e5 08ec e483 b8f0 0000
00000040 0000 c429 04c7 0024 0000 e800 fffc ffff
00000050 00b8 0000 c900 00c3 6548 6c6c 206f 6f57
00000060 6c72 2064 0a21 0000 4347 3a43 2820 4e47
00000070 2955 3320 332e 352e 2820 6544 6962 6e61
00000080 3120 333a 332e 352e 312d 2933 0000 732e
00000090 6d79 6174 0062 732e 7274 6174 0062 732e
000000a0 7368 7274 6174 0062 722e 6c65 742e 7865
000000b0 0074 642e 7461 0061 622e 7373 2e00 6f72
...
```

GCC: fichier “objet”

- ◆ Fichier “objet”

- ◆ Plusieurs types d'objets

- ◆ Relocatable (créé par l'assembleur)
 - ◆ Executable (créé par le linker)
 - ◆ Shared (créé par le linker: librairies)

- ◆ Format:

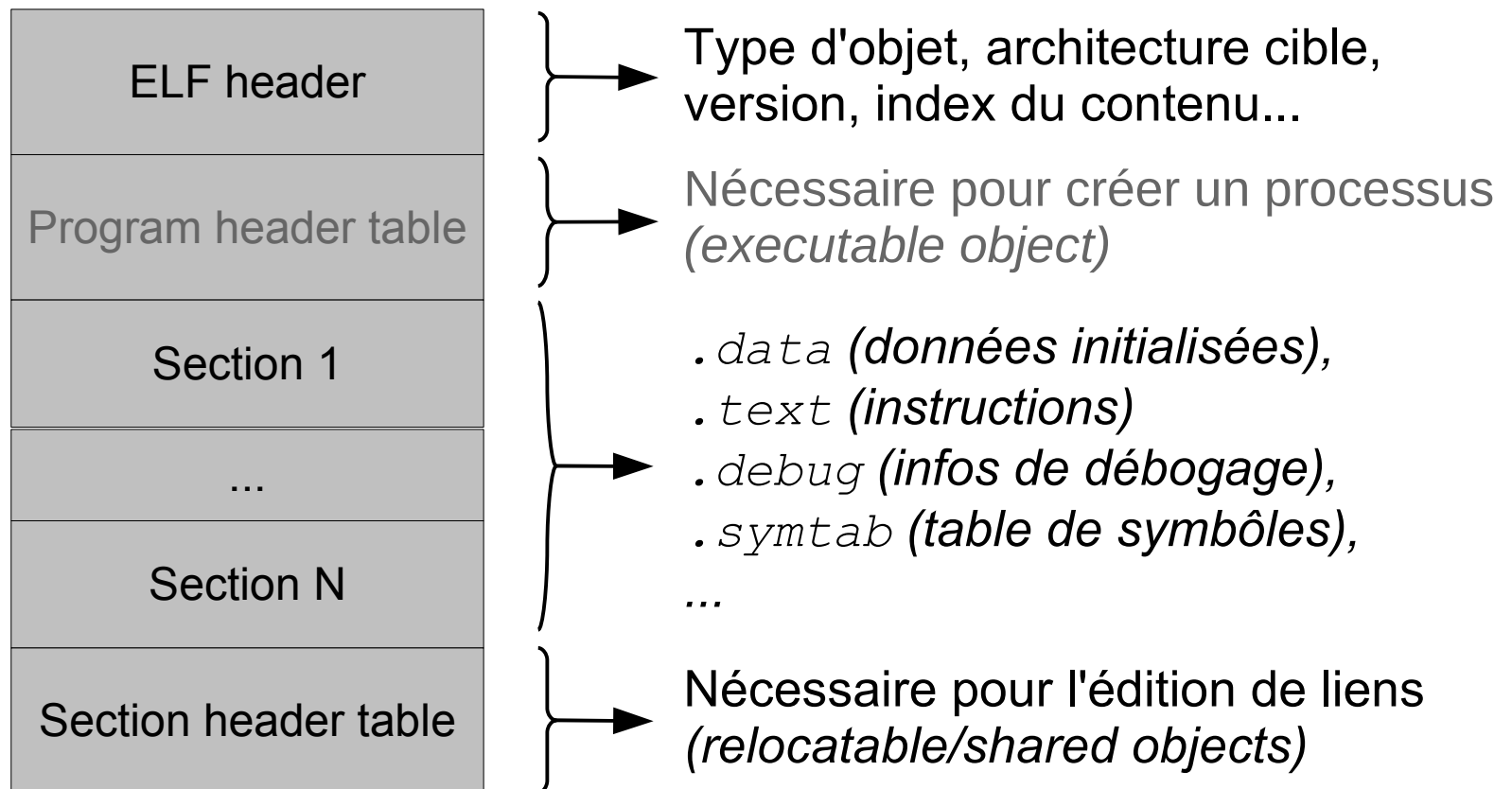
- ◆ *Executable and Linking Format* (ELF)
 - ◆ Autres formats: COM, a.out, COFF, ECOFF, EXE, Mach-O...

- ◆ Outils utiles:

- ◆ **objdump** : display content of object file
 - ◆ **nm** : display list of symbols in object file

GCC: fichier "objet"

◆ Structure générale d'un fichier ELF



GCC: fichier "objet"

◆ Contenu de l'objet

-x : show all sections

```
[toto@brol src] objdump -x prog1.o
```

```
prog1.o:      file format elf32-i386
prog1.o
architecture: i386, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x00000000
```

Position, taille et attributs
de chaque section

Sections:

Idx	Name	Size	VMA	LMA	File off	Algn
0	.text	00000023	00000000	00000000	00000034	2**2
		CONTENTS,	ALLOC,	LOAD,	RELOC,	READONLY,
1	.data	00000000	00000000	00000000	00000058	2**2
		CONTENTS,	ALLOC,	LOAD,	DATA	
2	.bss	00000000	00000000	00000000	00000058	2**2
		ALLOC				
3	.rodata	0000000f	00000000	00000000	00000058	2**0
		CONTENTS,	ALLOC,	LOAD,	READONLY,	DATA

...

GCC: fichier "objet"

◆ Contenu de l'objet

-s : show content of sections

```
[toto@brol src] objdump -s prog1.o
```

```
prog1.o:      file format elf32-i386
```

```
Contents of section .text:
```

```
0000 5589e583 ec0883e4 f0b80000 000029c4  U.....).
0010 c7042400 000000e8 fcffffff b8000000  ..$.
0020 00c9c3                                ...
```

code compilé
(instructions)

```
Contents of section .rodata:
```

```
0000 48656c6c 6f20576f 726c6420 210a00  Hello World !..
```

données
constantes

```
Contents of section .comment:
```

```
0000 00474343 3a202847 4e552920 332e332e  .GCC: (GNU) 3.3.
0010 35202844 65626961 6e20313a 332e332e  5 (Debian 1:3.3.
0020 352d3133 2900                                5-13).
```

GCC: fichier "objet"

◆ Contenu de l'objet

-d : disassemble

```
[toto@brol src] objdump -d prog1.o
prog1.o:      file format elf32-i386

Disassembly of section .text:

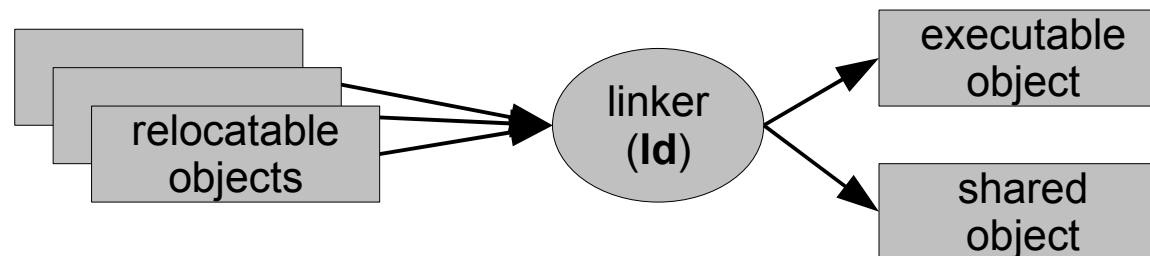
00000000 <main>:
  0:      55                push    %ebp
  1:      89 e5             mov     %esp,%ebp
  3:      83 ec 08         sub     $0x8,%esp
  6:      83 e4 f0         and     $0xfffffffff0,%esp
  9:      b8 00 00 00 00   mov     $0x0,%eax
  e:      29 c4             sub     %eax,%esp
10:      c7 04 24 00 00 00 00   movl    $0x0, (%esp)
17:      e8 fc ff ff ff   call    18 <main+0x18>
1c:      b8 00 00 00 00   mov     $0x0,%eax
21:      c9              leave
22:      c3              ret
```

*Le contenu doit être
similaire à prog1.s*

GCC: Etapes de la compilation

◆ 4). Edition de liens

- ◆ GNU linker (**ld**)
- ◆ Production d'un exécutable ou d'une librairie dynamique à partir d'un ensemble d'objets (relocatable)



- ◆ Lier avec les symboles non-définis (statiques):
 - ◆ utilise les sections `.reloc` et `.symtab` pour modifier le contenu de la section `.text`
 - ◆ “relocations”

Création d'une librairie

◆ Librairie statique (static library)

◆ Archive de fichiers objets (.o)

```
[toto@brol src] gcc -Wall -c hello.c
[toto@brol src] ar cru libhello.a hello.o
[toto@brol src] ranlib libhello.a
[toto@brol src]
```

Crée une archive avec les fichiers objets.

Indexe les symboles des différents objets de la librairie.

◆ Exemple

```
[toto@brol src] nm libbgpdump.a
libbgpdump.a(bgpdump_lib.o):
...
000000cb T _bgpdump_close_dump
00000000 T _bgpdump_open_dump
...
libbgpdump.a(bgpdump_mstream.o):
...
0000002a T _mstream_get
...
```

Création d'une librairie

◆ **Librairie dynamique (shared library)**

- ◆ Archive de fichier objets “position-independent” (PIC)
- ◆ Tous les systèmes ne supportent pas les librairies dynamiques.

```
[toto@brol src] gcc -fPIC -Wall hello.c  
[toto@brol src] gcc -shared -Wl,-soname,libhello.so.1 \  
-o libhello.so.1.0.1 hello.o -lc
```

Sous d'autres systèmes, la méthode diffère. Voir
<http://www.fortran-2000.com/ArnaudRecipes/sharedlib.html>

◆ Installation nécessaire

- ◆ Modifier `/etc/ld.so.conf` et lancer `/sbin/ldconfig`
- ◆ Ou utiliser `LD_LIBRARY_PATH`

En général lorsqu'on n'a pas les droits nécessaires pour modifier `ld.so.conf`

Utiliser une librairie

◆ API de la librairie

- ◆ API: types et fonctions fournis par la librairie généralement décrit dans des fichiers .h
- ◆ Type: statique ou dynamique ?

◆ Lier à une librairie statique

◆ Exemple

```
[toto@brol src] gcc -o main.o -c main.c  
[toto@brol src] gcc -o hello main.o libhello.a
```

- ◆ Si la librairie est situé ailleurs, par exemple dans /usr/local/lib

```
[toto@brol src] gcc -L/usr/local/lib -o hello main.o libhello.a
```

Utiliser une librairie

- ◆ **Lier à une librairie dynamique**
 - ◆ Exemple

```
[toto@brol src] gcc -o main.o -c main.c  
[toto@brol src] gcc -o hello -lhello main.o
```

Table des matières

- ◆ I. Recommandations générales
- ◆ II. Compilation
- ➔ **III. Automatisation de la compilation**
- ◆ IV. Débogage
- ◆ V. Contrôle de version
- ◆ VI. Conclusion
- ◆ VII. Références

Compilation automatisée

- ◆ **Compiler un gros projet manuellement**
 - ◆ peut causer des **erreurs** et être **fastidieux**

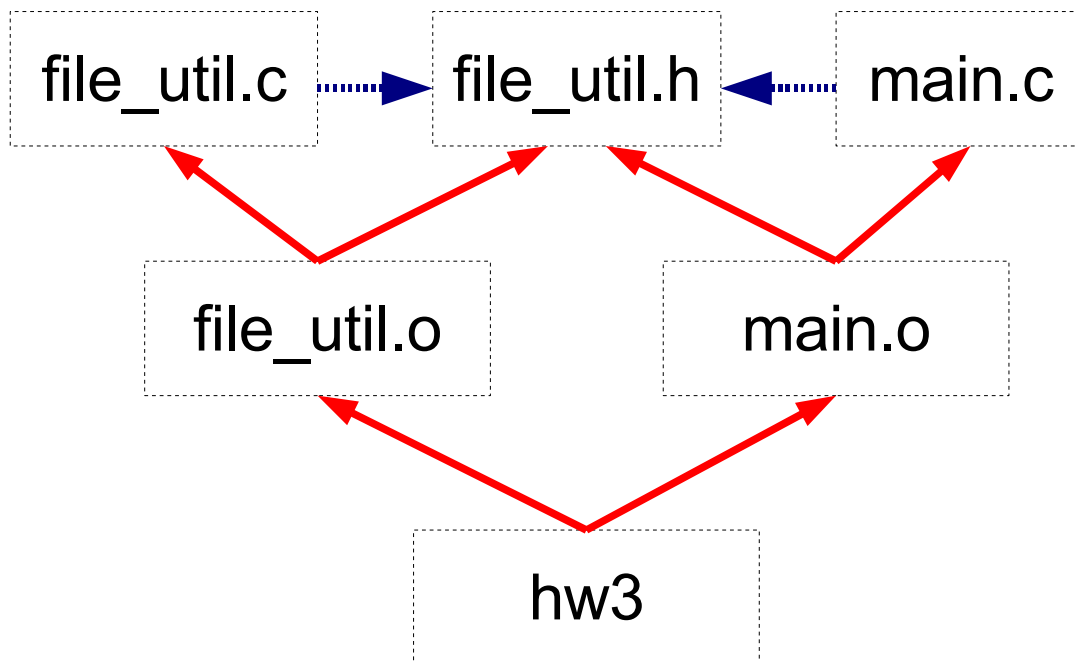
- ◆ Solution 1: créer un script shell

```
#!/bin/bash
gcc -c list_util.c -o list_util.o
gcc -c main.c -o main.o
gcc -o hw3 list_util.o main.o
```

- ◆ Solution 2: utiliser l'utilitaire **make**
 - ◆ Make détermine **automatiquement** quelles parties d'un programme doivent être recompilées
 - ◆ Se base sur un fichier de configuration: le **Makefile**
 - ◆ Définit des règles de compilation et des dépendances

Make

◆ Graphe de dépendances



Exemple:

Si `file_util.c` est modifié, alors `file_util.o` et `hw3` doivent être reconstruits.

Exemple:

Si `file_util.h` est modifié, alors tous les fichiers objets ainsi que `hw3` doivent être reconstruits.



Makefile

- ◆ **Le fichier Makefile sert à définir**
 - ◆ les dépendances entre fichiers:
 - ◆ 1 *cible* dépend de 0-*n* *dépendances*
 - ◆ par exemple: `main.o` dépend de `main.c` et `file_util.h`
 - ◆ Les règles de transformation
 - ◆ Comment générer une *cible* à partir de ses *dépendances*
 - ◆ par exemple: comment passer de `main.c` à `main.o`
 - ◆ **générique**: permet de fonctionner avec `gcc`, `javac`, `latex`, ...

Makefile

◆ Syntaxe

```
cible : [dépendance1 [, dépendance2...]]  
  <tab>commande1  
  <tab>commande2  
  ...
```

Exemple:

```
hw3: main.o list_util.o  
    gcc -o hw3 main.o list_util.o  
  
main.o: main.c list_util.h  
    gcc -c main.c -o main.o  
  
list_util.o: list_util.c list_util.h  
    gcc -c list_util.c -o list_util.o
```

Ce **Makefile** représente le graphe de dépendances de **hw3**.

Make et Makefile

◆ Invocation de make

- ◆ La *cible* par défaut est la première qui apparaît dans le **Makefile**.

```
[toto@brol src] make  
gcc -c main.c -o main.o  
gcc -c list_util.c -o list_util.o  
gcc -o hw3 main.o list_util.o
```

- ◆ Possibilité d'appeler une autre règle

```
[toto@brol src] make main.o
```

- ◆ Utiliser un autre fichier **Makefile**

- ◆ Par défaut: fichier **Makefile** du répertoire courant

```
[toto@brol src] make -f Makefile.solaris
```

Make

- ◆ **Recompilation partielle par make**
 - ◆ ne recompile que le nécessaire
 - ◆ se base sur la date de modification des fichiers

```
[toto@brol src] make
gcc -c main.c -o main.o
gcc -c list_util.c -o list_util.o
gcc -o hw3 main.o list_util.o
[toto@brol src] rm hw3
[toto@brol src] make
gcc -o hw3 main.o list_util.o
[toto@brol src] vi list_util.c
[toto@brol src] make
gcc -c list_util.c -o list_util.o
gcc -o hw3 main.o list_util.o
```

Première compilation:
tout est compilé.

Suppression de hw3:
il faut seulement relinker.

Edition de
list_util.c:
il faut recompiler
list_util.o
et relinker.

Makefile

◆ Clean (*phony target*)

- ◆ Règle conventionnelle pour “nettoyer” le projet
 - ◆ Supprimer tous les fichiers objets
 - ◆ Supprimer d'autres fichiers générés durant la compilation

◆ Comment ?

```
.PHONY: clean  
  
clean:  
    rm -f main.o list_util.o hw3
```

← .PHONY spécifie une liste de cibles qui ne sont pas des fichiers. C'est nécessaire en cas de présence d'un fichier nommé “clean”.

◆ Utilisation

```
[toto@brol src] make clean
```

ATTENTION !

Il faut faire attention à ne pas mettre de fichier “source” (.c, .h) dans la règle `clean`

Makefile

◆ Règles génériques (*suffix rules*)

- ◆ Type de règle spécial s'appliquant aux fichiers qui correspondent aux extensions spécifiées.

Par exemple:

```
.SUFFIXES: .c .o
```

```
.c.o:
```

```
gcc -c $< -o $@
```

← **.SUFFIXES** spécifie une liste d'extensions utilisées dans les "*suffix rules*".

← **.c.o** définit une règle de transformation de fichiers terminés par **.c** (comme **main.c**) en fichiers terminés par **.o** (comme **main.o**).

Lorsque la règle s'applique, **\$<** sera remplacé par le nom du fichier source et **\$@** par le nom du fichier cible.

Makefile

```
SRCS= \
    list_util.c \
    main.c
OBJS= $(SRCS:.c=.o)

hw3: $(OBJS)
    $(CC) -o hw3 $(OBJS)

.SUFFIXES: .c .o

.c.o:
    $(CC) -c $< -o $@

.PHONY: clean

clean:
    rm -f hw3 $(OBJS)
```

SRCS et **OBJS** sont des variables définies par l'utilisateur. **CC** est une variable standard de make qui contient le nom du compilateur C par défaut. En général: **cc** ou **gcc**.

Le contenu de **OBJS** est obtenu en transformant le contenu de **SRCS**.

Conseil:
lisez les **Makefiles**
écrits par des développeurs
expérimentés. C'est instructif !!

Utilisation récursive

◆ Grands projets

- ◆ Organisés en une hiérarchie de répertoires
- ◆ Pratique courante:
 - ◆ 1 Makefile par sous-répertoire
 - ◆ Chaque Makefile appelle les Makefiles des ss-répertoires

```
SUBDIRS=src1 src2
```

```
all-recursive:  
  for subdir in $(SUBDIRS); do \  
    (cd $$subdir && $(MAKE)) \  
  done;
```

Bon à savoir...

◆ automake / autoconf

- ◆ permettent de générer des fichiers Makefile
- ◆ processus de compilation “*standard*”

```
[toto@brol src] ./configure  
[toto@brol src] make  
[toto@brol src] make install
```

- ◆ processus de compilation + portable
- ◆ relativement **complexe** à apprendre
- ◆ bonne documentation disponible chez RedHat

http://sourceware.org/autobook/autobook/autobook_toc.html

◆ makedepend

- ◆ Génère automatiquement des règles de dépendance (et les ajoute au fichier Makefile)

Table des matières

- ◆ I. Recommandations générales
- ◆ II. Compilation
- ◆ III. Automatisation de la compilation
- ➔ **IV. Débogage**
 - ◆ Principes
 - ◆ GDB
 - ◆ VALGRIND
- ◆ V. Contrôle de version
- ◆ VI. Conclusion
- ◆ VII. Références

Débogage

♦ Erreurs fréquentes en C

- ♦ Accès en dehors des bornes d'un tableau
- ♦ Utilisation d'une variable non initialisée
- ♦ Utilisation d'un pointeur non initialisé
- ♦ Pas de libération des ressources

♦ Symptômes

- ♦ Message “*Segmentation fault*”
- ♦ Message “*Bus error*”
- ♦ Fonctionnement non déterministe
- ♦ Plus de mémoire disponible

Débogage

◆ Exemple:

- ◆ programme qui prend un nom de fichier en ligne de commande et imprime son contenu (ligne par ligne)

```
01  #include <stdio.h>
02  #include <stdlib.h>
03
04  #define BUFFER_SIZE 100
05
06  main(int argc, char * argv[])
07  {
08      FILE * in_stream;
09      char * buffer= NULL;
10
11      /* Vérifie la ligne de commande */
12      if (argc != 2) {
13          fprintf(stderr, "Error: missing argument\n");
14          exit(EXIT_FAILURE);
15      }
16
```


Débogage

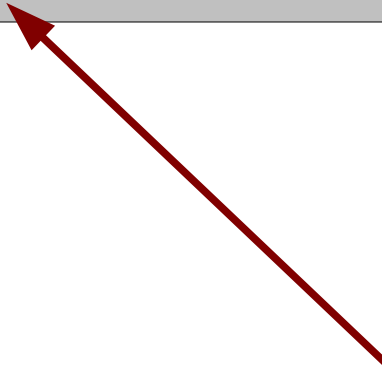
◆ Exemple: suite...

```
17  /* Ouvre le fichier en lecture... */
18  if ((in_stream= fopen(argv[1], "r")) == NULL) {
19      fprintf(stderr, "Error: could not open \"%s\" ",
20              argv[1]);
21      exit(EXIT_FAILURE);
22  } else
23      printf("Fichier \"%s\" ouvert :-)\n", argv[1]);
24
25  /* ... et lit les utilisateurs */
26  while (!feof(in_stream)) {
27      if (fgets(buffer, BUFFER_SIZE, in_stream) == NULL)
28          break;
29      printf("-> %s", buffer);
30  }
31
32  fclose(in_stream);
33  exit(EXIT_SUCCESS);
34 }
```

Débogage

◆ Compilation et exécution

```
[toto@brol src] gcc -o prog3 prog3.c
[toto@brol src] cat list.txt
Eddard Stark
Jon Snow
Robert Baratheon
[toto@brol src] ./prog3 list.txt
Fichier "list.txt" ouvert :-)
Segmentation fault
```



Une erreur de segmentation
s'est produite. Il y a probablement
une erreur ...

... Mais où ?

Débogage

◆ **Débogage**

- ◆ Re-lecture des sources
 - ◆ ne suffit pas toujours à localiser l'erreur
 - ◆ programmes longs / complexes
- ◆ L'utilisation d'un débogueur (debugger) permet de
 - ◆ suivre l'exécution pas-à-pas
 - ◆ définir des points d'arrêt (breakpoints)
 - ◆ définir des watchpoints
 - ◆ tracer la pile d'exécution (stack trace)
- ◆ Sous UNIX (Linux): **gdb**

Débogage: principes

◆ Breakpoints

- ◆ Arrêt du programme à une adresse de programme donnée
 - ◆ Lors de l'*instruction fetching*

◆ Hardware

- ◆ registres spéciaux du CPU (*debug registers* sur i386)
- ◆ limite sur le nombre de breakpoints (4 sur i386)
- ◆ génération d'une exception (*trap*)

◆ Software

- ◆ changements dynamiques du code (*int3* sur i386)
- ◆ fonctionne même si pas de support du CPU
- ◆ utile si beaucoup de breakpoints
- ◆ plus lent/complexé qu'en hardware

Débogage: principes

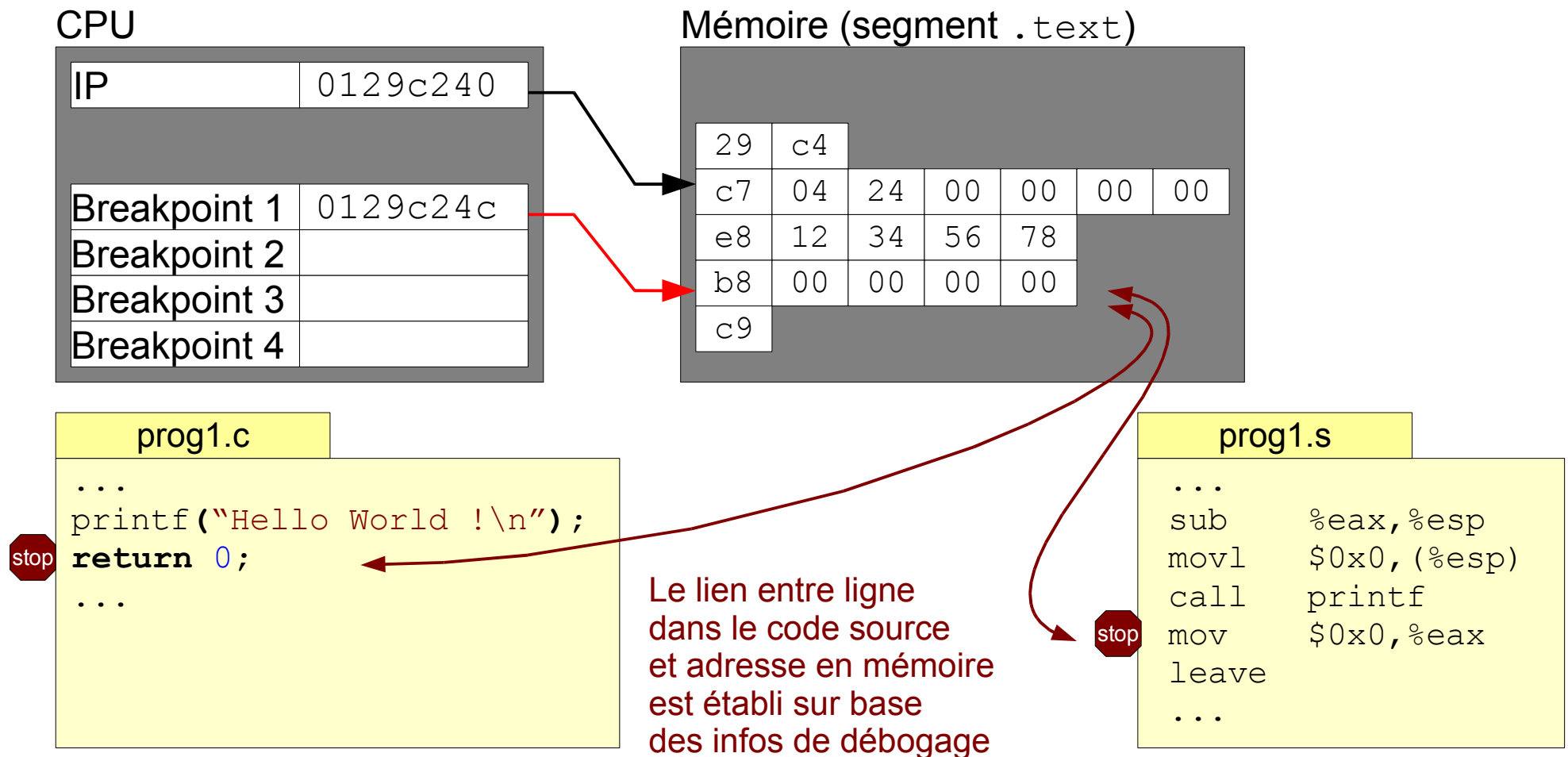
◆ Interruption de débogage (i386)

```
.text  
.global main  
  
main:  
    int3  
    .end
```

```
[toto@bro1 src] gcc -o int3 int3.s  
[toto@bro1 src] ./int3  
Trace/breakpoint trap
```

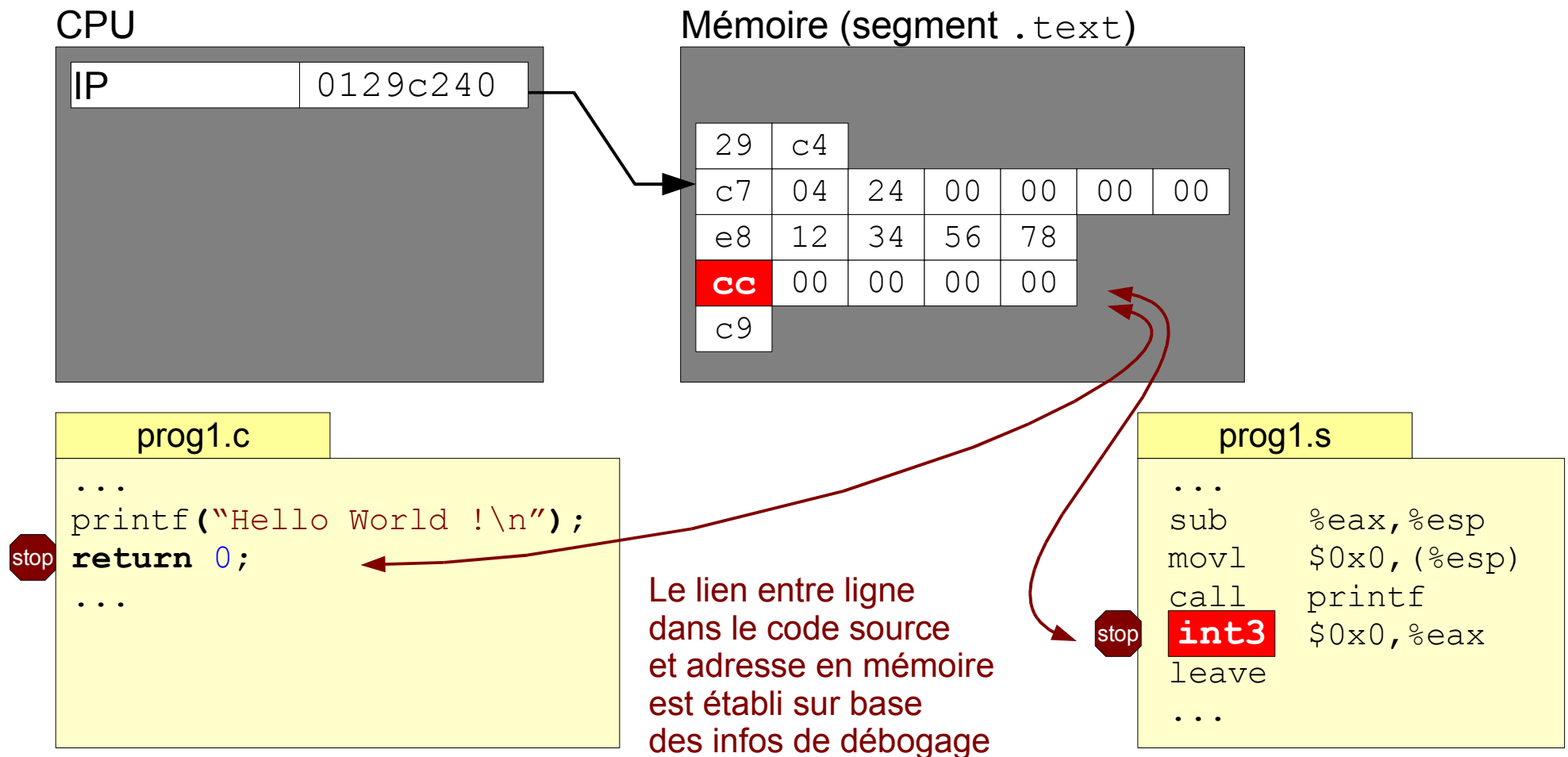
Débogage: principes

◆ Points d'arrêt matériels



Débogage: principes

◆ Points d'arrêt logiciels



Débogage: principes

◆ Watchpoints

- ◆ Arrêt du programme lors de l'accès à une zone de mémoire

◆ Hardware

- ◆ registres spéciaux (adresse + longueur)
- ◆ limite sur nombre de watchpoints (i386: 4)
- ◆ limite sur taille des watchpoints (i386: 1/2/4 octets)
- ◆ en lecture/écriture

◆ Software

- ◆ single-stepping + comparaison avant/après de la zone surveillée
- ◆ plus lent/complexé qu'en hardware
- ◆ en écriture uniquement !

Débogage: principes

◆ Watchpoints matériels

CPU

IP	0129c240
Watchpoint 1	0129c24c
Length 1	2
Watchpoint 2	
Length 2	

Mémoire (segment .data)

29	c4					
c7	04	24	00	00	00	00
e8	12	34	56	78		
b8	00	00	00	00		
c9						

prog1.c

```
...  
short int value= 5;  
printf("Val: %d\n", value);  
return 0;  
...
```

prog1.s

```
...  
sub    %eax,%esp  
movl   $0x0, (%esp)  
call   printf  
mov    $0x0,%eax  
leave  
...
```

Le lien entre ligne dans le code source et adresse en mémoire est établi sur base des infos de débogage

Table des matières

- ◆ I. Recommendations générales
- ◆ II. Compilation
- ◆ III. Automatisation de la compilation
- ◆ IV. Débogage
 - ◆ Principes
 - GDB
 - ◆ VALGRIND
- ◆ V. Contrôle de version
- ◆ VI. Conclusion
- ◆ VII. Références

Débogage: GDB

◆ Préparation du programme

- ◆ Compilation du programme avec les symboles de débogage: option “-g”

```
[toto@brol src] gcc -g -o prog3 prog3.c
```

◆ Résultat:

- ◆ L'exécutable contient des sections `.debug_XXX`
- ◆ Noms des symboles (fonctions, variables)
- ◆ Lien entre le code source (instruction C) et le code binaire (adresse dans le segment `.text`)

GDB

- ◆ **Lancement du débogueur**
 - ◆ Interface utilisateur (ligne de commande)

```
[toto@bro1 src] gdb prog3
GNU gdb 6.3-debian
Copyright 2004 Free Software Foundation, Inc.
...
(gdb)
```

Prompt de gdb:
les commandes
de débogage peuvent
être entrées ici...

Exécutable à charger:
à spécifier en ligne de
commande...


GDB

◆ Utilisation de base

- ◆ Quitter le débogueur: <ctrl-D> ou “quit”
- ◆ Lancer l'exécution du programme

```
[toto@brol src] gdb prog3
GNU gdb 6.3-debian
Copyright 2004 Free Software Foundation, Inc.
...
(gdb) run list.txt
Starting program:../prog3 list.txt
Fichier "list.txt" ouvert :-)
-> Eddard Stark

Program received signal SIGSEGV, Segmentation fault.
0x4007a295 in _IO_getline_info () from /lib/tls/libc.so.6
(gdb)
```



Une erreur s'est produite. Le programme a causé une erreur de segmentation (SEGV).

GDB

◆ Examiner les sources

◆ `list <file>:<line>; list ; ...`

```
[toto@brol src] gdb prog3
GNU gdb 6.3-debian
Copyright 2004 Free Software Foundation, Inc.
```

```
...
```

```
(gdb) list prog3.c:1
```

```
01  #include <stdio.h>
02  #include <stdlib.h>
03
04  #define BUFFER_SIZE 100
05
06  main(int argc, char * argv[])
07  {
08      FILE * pIn;
09      char * pcBuffer= NULL;
10
```

} 10 lignes (max)

```
(gdb) list
```

```
11  /* Vérifie la ligne de commande */
12  if (argc != 2) {
```

} 10 lignes de plus
(max)

```
...
```


GDB

◆ Définir un breakpoint

◆ **break** *<line>; break <fct-name>; break *<address>*

```
[toto@bro1 src] gdb prog3
GNU gdb 6.3-debian
Copyright 2004 Free Software Foundation, Inc.
...
(gdb) break 26
Breakpoint 1 at 0x804855f: file prog3.c, line 26.
(gdb) run list.txt
Starting program:../prog3 list.txt
Fichier "list.txt" ouvert :-)
```



```
Breakpoint 1, main (argc=2, argv=0xbffff5f4) at prog3.c:26
26          while (!feof(pIn)) {
(gdb)
```

GDB

◆ Breakpoints

◆ Définir un breakpoint

◆ `break <line>`

◆ Définir une condition

◆ `condition <bp-#> <expression>` (sur bp existant)

◆ `break <line> if <expression>` (nouveau bp)

◆ Désactiver

◆ `disable <bp-#>`

◆ Supprimer

◆ `clear <line>`

◆ `delete <bp-#>`

(tous les bps de la ligne)
(par numéro de bp)

GDB

◆ Continuer l'exécution...

```
[toto@brol src] gdb prog3
GNU gdb 6.3-debian
Copyright 2004 Free Software Foundation, Inc.
...
Breakpoint 1, main (argc=2, argv=0xbffff5f4) at prog3.c:26
26      while (!feof(pIn)) {
(gdb) continue
Continuing.
-> Eddard Stark

Breakpoint 1, main (argc=2, argv=0xbffff5f4) at prog3.c:26
26      while (!feof(pIn)) {
(gdb) continue
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x4007a295 in _IO_getline_info () from /lib/tls/libc.so.6
(gdb)
```

GDB

◆ Exécution pas-à-pas

```
[toto@brol src] gdb prog3
GNU gdb 6.3-debian
Copyright 2004 Free Software Foundation, Inc.
...
Breakpoint 1, main (argc=2, argv=0xbffff5f4) at prog3.c:26
26      while (!feof(pIn)) {
(gdb) step
27      if (fgets(pcBuffer, BUFFER_SIZE, pIn) == NULL)
(gdb) print pcBuffer
$1 = 0x0
(gdb) step

Program received signal SIGSEGV, Segmentation fault.
0x4007a295 in _IO_getline_info () from /lib/tls/libc.so.6
(gdb)
```

L'erreur s'est produite lors de l'exécution de la ligne 27.

Raison: pcBuffer non alloué.

Débogage: GDB

◆ Exécution pas à pas

◆ Avance à la ligne suivante (ou N lignes)

◆ `step [N]`

◆ `next [N]` (comme `step`, n'entre pas dans les fonctions)

◆ Avance à l'instruction suivante (ou N instructions)

◆ `stepi [N]`

◆ `nexti [N]` (comme `stepi`, n'entre pas dans les fonctions, i.e. `call`)

◆ Affiche la valeur de variables / expressions

◆ `print <expression>`

◆ Termine l'exécution d'une fonction (*même stack frame*)

◆ `finish`

GDB

◆ Autres commandes


◆ Watchpoints

◆ `watch <expression>`

◆ Back trace

◆ `backtrace [full]`

```
(gdb) break 5
...
(gdb) run
...
(gdb) backtrace
#0  fct_b () at prog7.c:5
#1  0x080483a3 in fct_a () at prog7.c:10
#2  0x080483ba in main () at prog7.c:15
```



```
#include <stdio.h>

void fct_b(char * msg)
{
    printf(msg);
}

void fct_a(char * msg)
{
    fct_b(msg);
}

int main()
{
    fct_a("Hello World !\n");
    return 0;
}
```

DDD: un GUI pour GDB

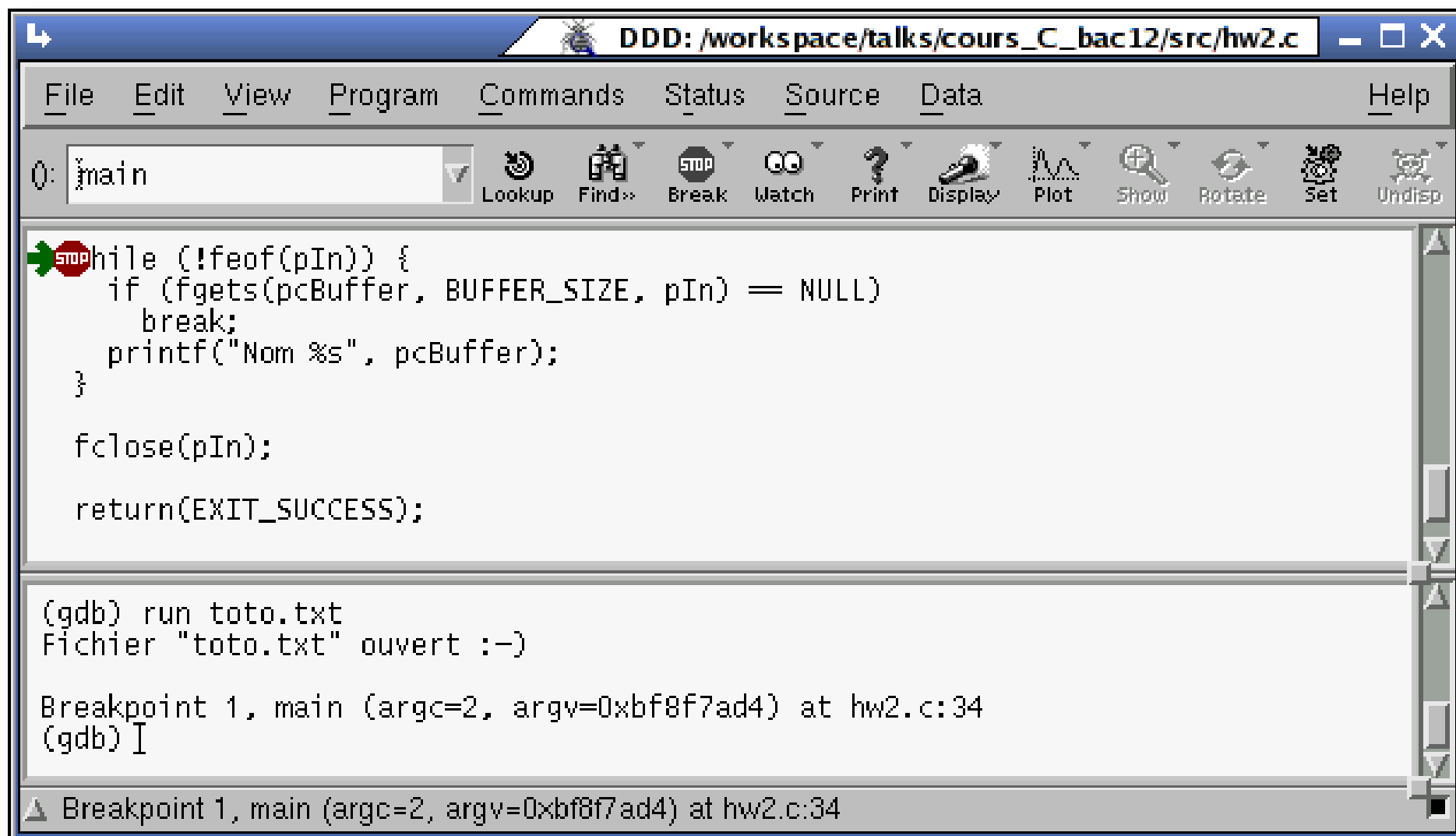


Table des matières

- ◆ I. Recommandations générales
- ◆ II. Compilation
- ◆ III. Automatisation de la compilation
- ◆ IV. Débogage
 - ◆ Principes
 - ◆ GDB
 - ➔ VALGRIND
- ◆ V. Contrôle de version
- ◆ VI. Conclusion
- ◆ VII. Références

Débogage: memory leaks

- ◆ **Traçage des accès en mémoire**
 - ◆ Permet de détecter
 - ◆ l'accès à des zones non initialisées
 - ◆ l'accès hors des bornes d'un tableau,
 - ◆ les zones de mémoire non libérées
 - ◆ Les tentatives de libération de zones non allouées
 - ◆ Traçage non exhaustif
 - ◆ uniquement le long du chemin d'exécution

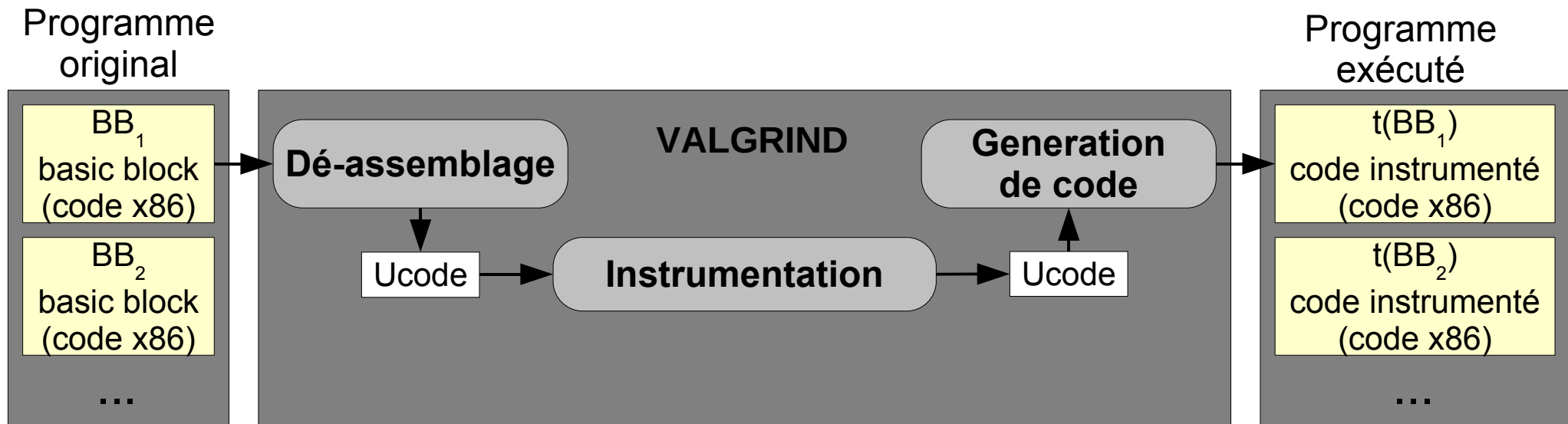
Valgrind

- ◆ **Plate-forme d'analyse de programmes**
 - ◆ Analyse par simulation DBI (Dynamic Binary Instrumentation)
 - ◆ **Memcheck**: détection d'erreurs de mémoire
 - ◆ **Cachegrind**: “cache misses” analyser
 - ◆ **Massif**: space profiler
 - ◆ ...
- ◆ Utilisé par de grands projets
 - ◆ OpenOffice, MySQL, Gimp, Python, ...

Compilateur JIT / Instrumentation

◆ Principes

- ◆ Code x86: centaines d'instructions différentes
- ◆ Traduction en UCode, de type RISC
 - ⇒ Moins d'instructions
- ◆ Instrumentation de l'UCode



Compilateur JIT / Instrumentation

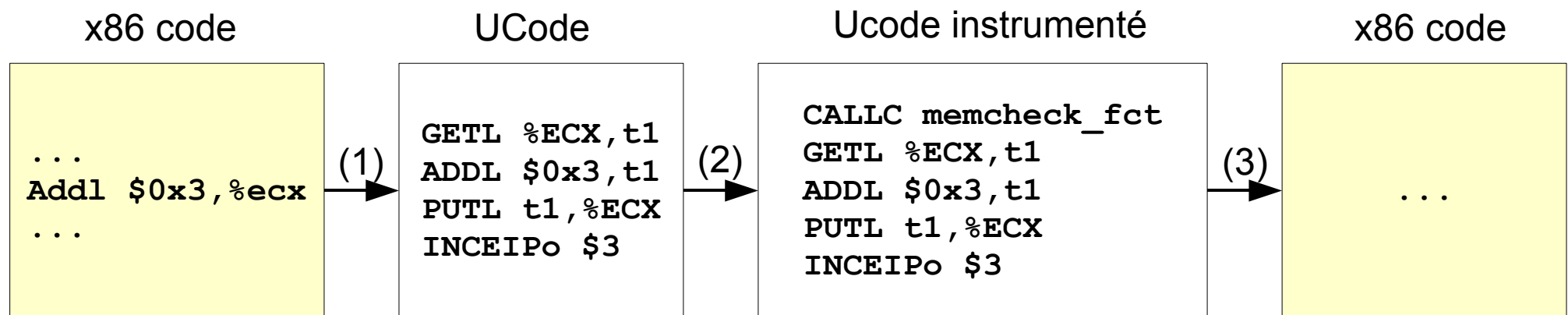
◆ Exemple

1) Instruction x86 traduite en instructions UCode

2) Instructions UCode ajoutées

Par exemple, appel de la fonction `memcheck_fct`

3) Code x86 généré



Valgrind

◆ MemCheck

◆ Détecte

- ◆ Les pertes de mémoire (memory leaks)
- ◆ Accès incorrects à la mémoire (heap)
- ◆ Zones de mémoire utilisées sans être initialisées

◆ Grâce à l'instrumentation du code

- ◆ Espionne les opérations UCode de lecture et d'écriture
- ◆ Maintient des metadatas (“shadow values”) pour chaque registre et zone de mémoire utilisée
- ◆ Chaque zone de mémoire allouée / libérée est pistée

Valgrind

◆ MemCheck

- ◆ Remplace les fonctions **malloc**, **realloc**, **free**, ...
 - ◆ garde une trace des allocations (emplacement + taille)
 - ◆ place des “*red-zones*” autour des zones de mémoire

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

```
int main()
```

```
{
```

```
    char * ptr= malloc(10);
```

```
    *ptr= 'A';
```

```
    *(ptr+1)= 'B';
```

```
    free(ptr);
```

```
    return 0;
```

```
}
```

Appel de la fonction
malloc() de Valgrind

...	...
0x0010	10
...	...

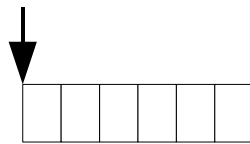
Appel de la fonction
free() de Valgrind

...	...
0x0010	10
...	...

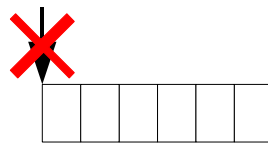
Valgrind

◆ MemCheck: Memory leaks

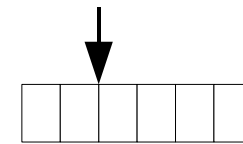
◆ Classification



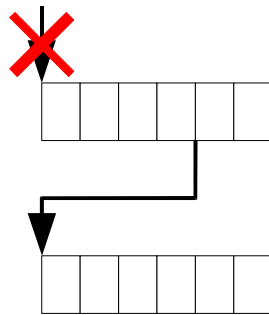
Not-leaked



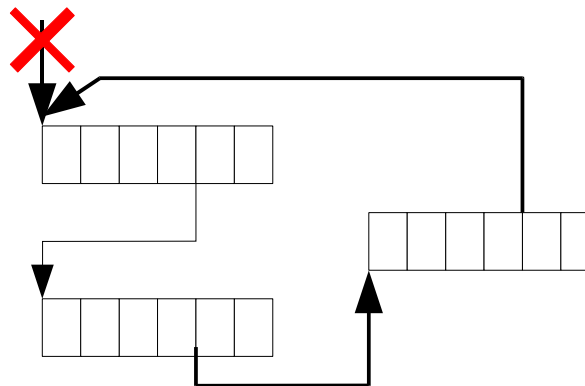
Directly leaked



Possibly leaked



Indirectly leaked



Indirectly leaked

(voir présentation de Julian Seward, FOSDEM'06)

Memory Leaks

◆ Exemple

```
#include <stdlib.h>

int main()
{
    char * ptrChars= malloc(6 * sizeof(char));
    ptrChars[0]= 'H';
    //free(ptrChars);
    return 0;
}
```

Memory Leaks

◆ Exemple

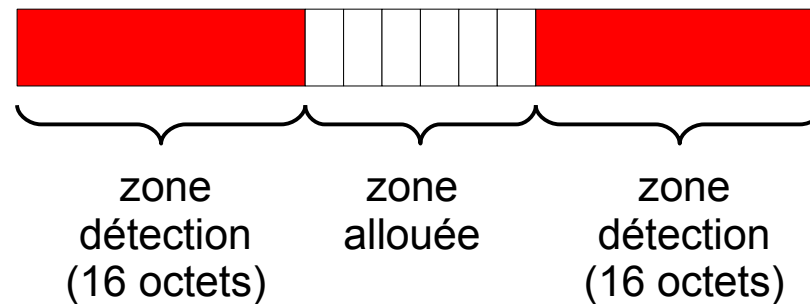
```
[toto@brol src] valgrind -leak-check=yes prog6
...
==13082== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 14 from 1)
==13082== malloc/free: in use at exit: 6 bytes in 1 blocks.
==13082== malloc/free: 1 allocs, 0 frees, 6 bytes allocated.
==13082== For counts of detected errors, rerun with: -v
==13082== searching for pointers to 1 not-freed blocks.
==13082== checked 76,196 bytes.
==13082==
==13082== 6 bytes in 1 blocks are definitely lost in loss record 1 of 1
==13082==    at 0x401C867: malloc (vg_replace_malloc.c:149)
==13082==    by 0x804839F: main (prog6.c:5)
==13082==
==13082== LEAK SUMMARY:
==13082==    definitely lost: 6 bytes in 1 blocks.
==13082==    possibly lost: 0 bytes in 0 blocks.
==13082==    still reachable: 0 bytes in 0 blocks.
==13082==    suppressed: 0 bytes in 0 blocks.
```

Invalid Heap Access

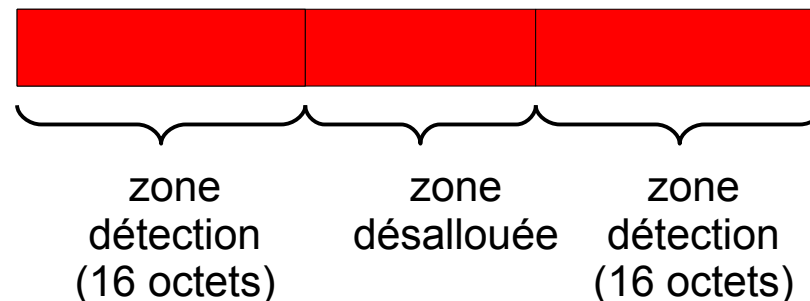
◆ MemCheck: Accès invalides au tas (heap)

- ◆ En écriture et lecture
- ◆ Autres outils (ElectricFence, ...) ne détectent que les écritures incorrectes

Accès incorrects:



Accès à des zones libérées:



Attention: une fois désallouée, une zone peut être ré-allouée plus tard. Conséquence, des accès incorrects peuvent ne pas être détectés !

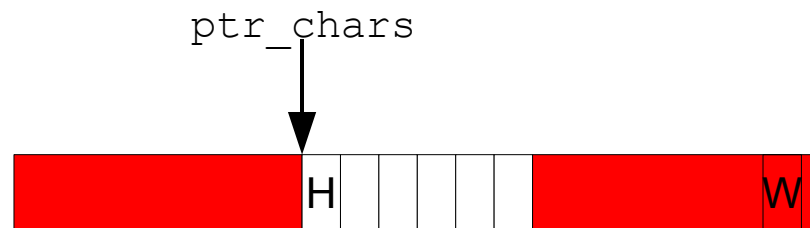
(voir présentation de Julian Seward, FOSDEM'06)

Invalid Heap Access

◆ Exemple

```
#include <stdlib.h>

int main()
{
    char * ptr_chars= malloc(6 * sizeof(char));
    ptr_chars[0]= 'H';
    ptr_chars[12]= 'W';
    free(ptr_chars);
    ptr_chars[1]= 'e';
    return 0;
}
```



Invalid Heap Access

◆ Exemple

```
[toto@brol src] gcc -g -o prog4 prog4.c
[toto@brol src] ./prog4
[toto@brol src] valgrind -leak-check=yes prog4
==7643== Memcheck, a memory error detector.
==7643== Copyright (C) 2002-2007, and GNU GPL'd, by Julian Seward et al.
...
==7643== Invalid write of size 1
==7643==    at 0x80483EF: main (prog4.c:7)
==7643==    Address 0x415602E is 0 bytes after a block of size 6 alloc'd
==7643==    at 0x401C867: malloc (vg_replace_malloc.c:149)
==7643==    by 0x80483DF: main (prog4.c:5)
==7643==
==7643== Invalid write of size 1
==7643==    at 0x8048401: main (prog4.c:9)
==7643==    Address 0x4156029 is 1 bytes inside a block of size 6 free'd
==7643==    at 0x401D4FE: free (vg_replace_malloc.c:233)
==7643==    by 0x80483FC: main (prog4.c:8)
...
```

Invalid Stack Access

◆ Attention!

- ◆ Les erreurs d'accès à la pile (stack) ne sont pas détectées!

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

```
int main()
```

```
{
```

```
    char array_chars[6];
```

```
    array_chars[0]= 'H';
```

```
    array_chars[12]= 'W';
```

```
    return 0;
```

```
}
```

Ici, le tableau `array_chars[]` est alloué sur la pile. Valgrind ne détecte pas l'écriture invalide de la valeur 'W'.

Valgrind

◆ MemCheck: Zones non initialisées

- ◆ Chaque bit dans les zones de mémoire et registres est associé à un “*shadow bit*” qui indique si la zone a été initialisée (affectée)
- ◆ Propage les shadow bits avec les opérations en Ucode + avertit lors de leur utilisation

```
if (...GARBAGE...)
    branchement dépend de valeur non-initialisée
```

```
* (...GARBAGE...)
    adresse mémoire dépend de valeur non-initialisée (dé-référencement)
```

```
syscall (...GARBAGE...)
    paramètre d'appel système dépend de valeur non-initialisée
```

Résumé

- ◆ **Quel outil utiliser ?**

- ◆ Segmentation fault: **gdb**, **valgrind**
- ◆ Memory leaks: **valgrind**

- ◆ **Attention !!!**

- ◆ Les outils tels que **gdb** et **valgrind** ne tracent que le chemin d'exécution. Ils ne peuvent détecter des erreurs dans des parties de programme non exécutées !

Table des matières

- ◆ I. Recommandations générales
- ◆ II. Compilation
- ◆ III. Automatisation de la compilation
- ◆ IV. Débogage
- ➔ **V. Contrôle de version**
- ◆ VI. Conclusion
- ◆ VII. Références

Contrôle de version

◆ Un contrôle de version, mais...

◆ À quoi ça sert ?

◆ Conserver un **historique des modifications**

- ◆ Ca “*marchait*”, vous faites une modification et “**plouf!**”
 - ◆ possibilité de retourner chercher l'ancienne version
- ◆ Gestion des versions successives du projet
 - ◆ possibilité de récupérer une ancienne version, d'y apporter des corrections et de la fournir aux utilisateurs.

◆ **Travail collaboratif**

- ◆ Permet de travailler à plusieurs sur un projet

Contrôle de version

- ◆ **Un contrôle de version, mais...**

- ◆ Comment ça marche ?

- ◆ ***“Repository central”***

- ◆ contient le projet et son historique
 - ◆ typiquement sur un serveur dédié

- ◆ ***“Working copy”***

- ◆ **Copie locale** du repository
 - ◆ Un développeur travaille uniquement sur **sa copie**. Il ne gêne donc pas les autres développeurs.
 - ◆ lorsqu'il a fini ses modifications, il les transfère (**commit**) dans le repository.
 - ◆ plusieurs *working copies* sont possibles sur plusieurs workstations en même temps

- ◆ Exemples: **CVS** ou **SVN** (subversion)

SVN en INGI

- ◆ Inscription via <https://scm.info.ucl.ac.be/cgi-bin/inscription.sh>

**Gestion des repositories
pour TP de groupes**

Server info: scm

- Tue Feb 10 2009 09h33:16
- HTTPD: Apache/2.2.3 (CentOS)
- OS: Linux 2.6.18-92.1.22.el5xen kernel
- Arch: i686

Veuillez compléter les paramètres du nouveau projet

Informations générales

Nom du groupe ou du projet :

☐ Intégration dans un repository monitoré de travaux de groupes.

Cours concerné :

Adresse(s) email du(des) membre(s) du projet

@student.uclouvain.be

Envoyer le formulaire

(Caractères autorisés : alphanumériques, "-", ".", "_"; les champs doivent commencer par une lettre.)

**Vous recevrez une confirmation
par e-mail accompagnée de
votre mot de passe.**

- ◆ Utilisation décrite sur le Wiki étudiants
<http://wiki.student.info.ucl.ac.be/index.php/Subversion>

SVN

◆ Création d'une Working Copy

- ◆ Commande `svn checkout <url>`

```
[toto@brol ~] svn --username toto checkout  
https://scm.info.ucl.ac.be/studentsvn/hw  
Password for 'toto': *****  
Checked out revision 0.  
[toto@brol ~] cd hw  
[toto@brol hw] ls -a  
.  ..  .svn  
[toto@brol hw]
```

Nom du projet.

URL complète du projet à récupérer.

Un répertoire `.svn` est créé dans chaque répertoire de la *working copy*.
Ce répertoire maintient des informations sur l'état de la *working copy*.

Il ne faut pas le supprimer ni en modifier le contenu !!!

SVN

◆ Ajout d'un fichier dans le projet

- ◆ Commande `svn add <path_to_file>`

```
[toto@brol hw] svn add -N src
A          src
[toto@brol hw] svn add src/hw.c
A          src/hw.c
[toto@brol hw]
```

Ajout d'un répertoire
(non récursivement).

Ajout d'un fichier.

ATTENTION !

Après la commande `svn add`, le fichier n'est pas encore copié dans le repository central !

SVN

◆ Envoyer un fichier vers le repository central

- ◆ Commande `svn commit <path_to_file>`

```
[toto@brol hw] svn commit -N src
Adding          src
Committed revision 1.
[toto@brol hw] svn commit src/hw.c
Adding          src/hw.c
Committed revision 1.
```

Numéro de révision
(incrémenté à chaque commit)

Note: il est possible d'ajouter un commentaire lors du commit du fichier avec l'option **-m** *<msg>*

```
[toto@brol hw] svn commit -m "Correction bug" src/hw.c
```

Bonne pratique: utiliser **-m** permet de se souvenir plus tard de la modification qui avait été apportée au fichier.

SVN

- ◆ **Mettre à jour une copie locale**
 - ◆ Récupère les modifications apportées par les autres
 - ◆ Commande **svn update** *<path_to_file>*

```
[toto@brol hw] svn update src/hw.c
A      src/hw.c
Updated to revision 2.
[toto@brol hw]
```

SVN

◆ Notes:

- ◆ Les commandes **add**, **commit** et **update** peuvent être utilisées avec plusieurs fichiers en même temps

```
[toto@bro1 hw] svn add src/main.c src/list_util.c src/list_util.h
```

ou bien

```
[toto@bro1 hw] svn add src/*.c
```

- ◆ Il est recommandé de **NE PAS** placer dans le projet les fichiers compilés (objets et exécutables)
 - ◆ Les autres développeurs ne travaillent peut-être pas sur la même plateforme que vous.

SVN

◆ Résolution de conflits

- ◆ modifications simultanées \Rightarrow conflit ?

```
[bob@truc src] vi hw.c
[bob@truc src] svn commit hw.c
Sending      hw.c
Committed to revision 3.
```

Bob a modifié **hw.c** et l'a envoyé dans le repository

```
[toto@brol src] vi hw.c
[toto@brol src] svn commit hw.c
svn: Commit failed (details follow)
svn: Out of date: 'hw.c'
[toto@brol src] svn update hw.c
C      src/hw.c
Updated to revision 3.
[toto@brol src] svn commit hw.c
svn: Commit failed (details follow)
svn: Aborting commit: 'hw.c' remains in conflict
```

Dans le même temps, Toto a modifié **hw.c**. Lorsqu'il veut l'envoyer dans le repository, **svn** râle...

CONFLIT!

SVN

◆ Résolution de conflits

- ◆ Il faut décider **manuellement** quelle version garder !

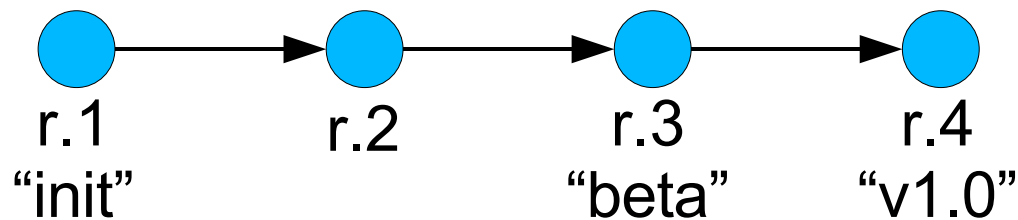


- ◆ La plupart du temps, **svn** parvient à résoudre les conflits automatiquement.

SVN Avancé

◆ Tagging

- ◆ Donner un nom à une révision particulière de l'ensemble d'un projet



SVN Avancé

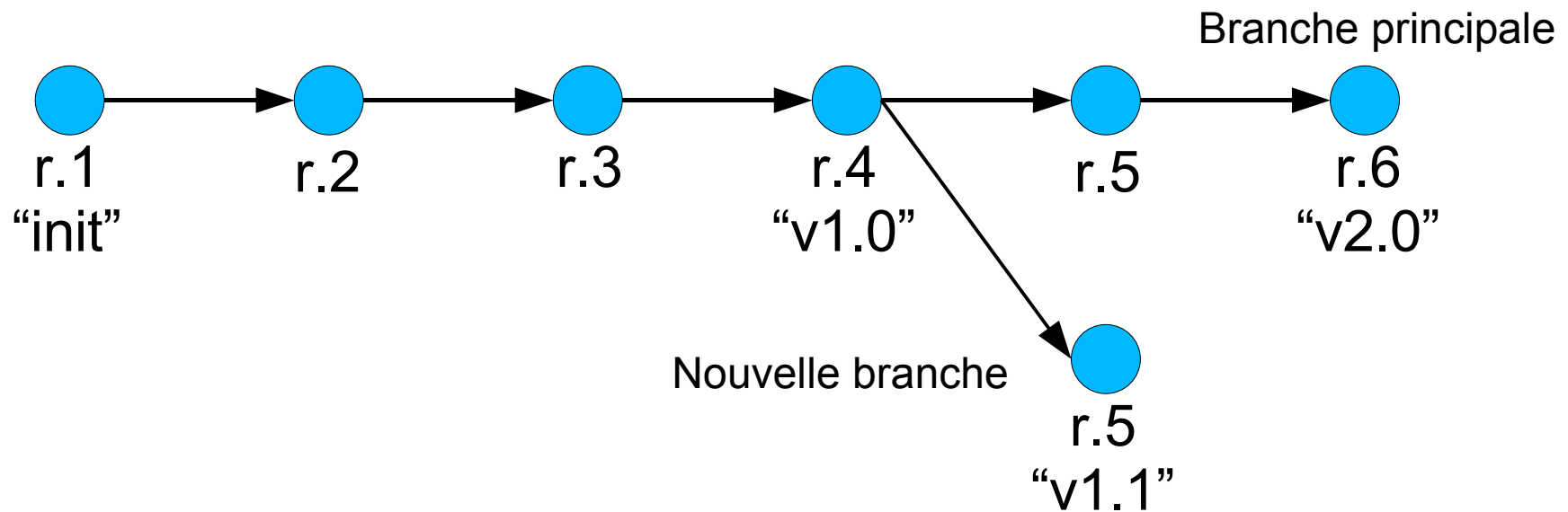
◆ Tagging

```
[toto@brol src] svn copy  
  https://scm.info.ucl.ac.be/studentsvn/hw/trunk  
  https://scm.info.ucl.ac.be/studentsvn/hw/tags/release-1.0
```


SVN Avancé

◆ Branching

- ◆ Modifications isolées pour une version particulière
- ◆ Maintenance d'une version dans une branche
- ◆ Développement expérimental



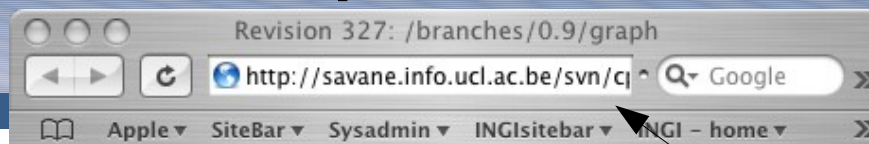
SVN Avancé

◆ Branching

```
[toto@brol src] svn copy  
https://scm.info.ucl.ac.be/studentsvn/hw/tags/release-1.0  
https://scm.info.ucl.ac.be/studentsvn/hw/branches/release-1  
.0
```

```
[toto@brol src] cd /tmp  
[toto@brol tmp] svn checkout  
https://scm.info.ucl.ac.be/studentsvn/hw/branches/release-1  
.0 hw  
[toto@brol tmp] cd hw/src  
[toto@brol src] vi hw.c  
[toto@brol src] svn commit hw.c
```

SVN par le web



Revision 327: /branches/0.9/graph

- [..](#)
- [INSTALL](#)
- [LICENSE](#)
- [Makefile](#)
- [Makefile.in.in](#)
- [TODO](#)
- [binarysimple.hh](#)
- [binarysimple.icc](#)
- [branch/](#)
- [configure](#)
- [configure.ac](#)
- [doxygen.conf](#)
- [doxygen.hh](#)
- [examples/](#)
- [graph.hh](#)
- [graphutils.h](#)
- [graphutils.icc](#)
- [misc/](#)
- [path/](#)
- [path.hh](#)
- [path.icc](#)
- [shortdesc.ac](#)
- [stlutility.icc](#)
- [var.icc](#)
- [view/](#)
- [view.icc](#)

URL complète du projet à visualiser.
C'est la même URL que celle utilisée
avec la commande **svn**.

Powered by [Subversion](#) version 1.1.4 (r13838).

Table des matières

- ◆ I. Recommandations générales
- ◆ II. Compilation
- ◆ III. Automatisation de la compilation
- ◆ IV. Débogage
- ◆ V. Contrôle de version
- ➔ **VI. Conclusion**
- ◆ VII. Références

Recommendations



1). Lisez le code de programmeurs expérimentés afin de comprendre comment il est organisé !

Il existe un grand nombre de projets Open-Source. Préférez la lecture de projets de grande taille et qui sont réellement utilisés (par exemple le kernel Linux)

2). Programmez, essayez, faites des erreurs!

C'est ainsi que vous ferez évoluer votre technique de programmation.

Table des matières

- ◆ I. Recommandations générales
- ◆ II. Compilation
- ◆ III. Automatisation de la compilation
- ◆ IV. Débogage
- ◆ V. Contrôle de version
- ◆ VI. Conclusion
- ➔ **VII. Références**

Références (1)

- ◆ **“Advanced Programming in the UNIX(R) Environment (2nd edition)”**, de W. Richard Stevens et Stephen A. Rago. Addison-Wesley, 2005.
- ◆ **“The Practice of Programming”**, de Brian W. Kernighan et Rob Pike. Addison-Wesley, 1999.
- ◆ **“Write Portable Code: An Introduction to Developing Software for Multiple Platforms”**, de Brian Hook. NO STARCH PRESS, 2005.
- ◆ **“Managing Projects with GNU Make, Third Edition”**, de Robert Mecklenburg. O'Reilly, 2004.
- ◆ **“Pragmatic Version Control Using Subversion”**, de Mike Mason. O'Reilly, 2005.
- ◆ **“Version Control with Subversion”**, de Ben Collins-Sussman, Brian W. Fitzpatrick et C. Michael Pilato. O'Reilly. Accessible gratuitement <http://svnbook.red-bean.com/>
- ◆ **“The C Programming Language (2nd edition)”**, par Brian W. Kernighan et Dennis M. Ritchie. Prentice Hall, Inc., 1988.
- ◆ **“Intel 80386 Reference Programmer's Manual”**, Intel Corporation (unofficial), 1986. Accessible à partir de <http://pdos.csail.mit.edu/6.828/2005/readings/i386/toc.htm>

Références (2)

- ◆ **“Executable and Linkable Format (ELF) Specification (version 1.2)”**, Tool Interface Standard (TIS) Committee, 1995.
- ◆ **“Debugging with GDB: The gnu Source-Level Debugger (9th Edition)”**, Richard Stallman, Roland Pesch, Stan Shebs, et al., 2004.
- ◆ **“GDB Internals: A guide to the internals of the GNU debugger (2nd Edition)”**, John Gilmore and Stan Shebs, 2004.
- ◆ **“Dynamic Binary Analysis and Instrumentation”**, Nicholas Nethercote, PhD Dissertation, University of Cambridge, November 2004.
- ◆ **“Using Valgrind to detect undefined value errors with bit-precision”**. Julian Seward and Nicholas Nethercote, Proceedings of the USENIX'05 Annual Technical Conference, Anaheim, California, USA, April 2005.
- ◆ **“Good Practices in Library Design, Implementation, and Maintenance (Version 0.1)”**. Ulrich Drepper, Red Hat Inc. March 7, 2002
- ◆ Ce cours est partiellement inspiré d'une formation RedHat http://www.redhat.com/docs/wp/intro_dev/index.html