

Les 5 pourquoi ?

- Méthode de résolution de problème
 - Technique développée par Sakichi Toyoda
 - Mise en œuvre par Toyota
-
- Un problème lorsqu'il est observé relève d'une série de causes
 - L'objectif est de remonter à l'origine du problème : la cause première

Les 5 pourquoi ?

- L'objectif est de poser 5 fois la question « pourquoi » face à une situation ou un problème
- En complément de cette technique, à chaque nouvelle réponse, on peut proposer une solution de correction.

Les 5 pourquoi ? Exemple 1

- Ma voiture ne démarre pas (le problème)

Pourquoi ? - La batterie n'est pas chargée.

Pourquoi (la batterie n'est-elle pas chargée) ? - L'alternateur ne fonctionne pas.

Pourquoi (...) ? - La courroie de l'alternateur est cassée.

Pourquoi ? - J'ai dépassé la durée préconisée par le constructeur et la courroie était usée.

Pourquoi ? - Je n'ai pas respecté les préconisations du constructeur (la cause première).

Les 5 pourquoi ? Exemple 2

- Un fichier critique sur mon PC n'est plus accessible. (Le problème)

Pourquoi ? – un virus a vérolé mon disque dur. (premier pourquoi)

Pourquoi ? – j'ai infecté mon PC avec la clé USB d'une connaissance. (second pourquoi)

Pourquoi ? – l'antivirus sur mon PC n'était pas à jour. (troisième pourquoi)

Pourquoi ? – je n'avais pas exécuté le dernier patch sécurité. (quatrième pourquoi)

Pourquoi ? – j'avais remis cela à plus tard. (cinquième pourquoi, une cause première)

Les 5 pourquoi ?

- Pourquoi 5 ? l'expérience montre que 5 suffisait généralement à arriver à la cause première
- A chaque étape, il peut y avoir plusieurs causes
- Deux personnes peuvent ne pas arriver à la même cause première ==> intérêt de faire cela en groupe avec l'ensemble des acteurs concernés

Allocation dynamique

- Tant que l'on a pas indiqué à malloc qu'une zone mémoire n'est pas réutilisable, il ne va pas sans servir pour de nouvelles allocations.
- Contrairement à l'allocation automatique, la « durée de vie » de l'allocation dynamique n'est pas liée à l'appel ou au retour de fonction.
- Si à un moment il n'existe plus de variable dans la pile (ou static) qui pointe directement ou indirectement vers une allocation dynamique alors il y a une « fuite mémoire » (memory leak).

Allocation dynamique

- Sous linux, le fonctionnement de malloc peut être paramétré via des variables d'environnement ou la fonction **mallopt** :
 - MALLOC_CHECK_ : vérification au moment de la libération (double free)
 - MALLOC_PERTURB_ : permet de remplir la mémoire avec une valeur à l'allocation et de la ré-initialiser à une autre valeur à la libération. Très utile !

Allocation : conclusion

- IMPORTANT : toute variable est soit dans la pile, soit dans le segment de donnée (static).
- En aucun cas une variable de votre programme peut avoir une adresse qui soit dans le tas.
- L'utilisation de la mémoire dynamique suppose donc l'utilisation de pointeurs permettant de gérer les adresses mémoires.
- La structuration des plages n'est pas connu par malloc : il est important d'interpréter ces zones correctement (taille).

malloc : question

#QDLE#Q#A*B#30#

- L'utilisation de malloc au profit de l'allocation automatique dans des fonctions récursives permet d'élargir la capacité de traitement de celle-ci :
 - A. vrai
 - B. faux



Question

#QDLE#Q#AB*C#40#

```
void rand(int *i) ; //génère un nombre aléatoire  
int main(){  
    int *j=NULL ;  
    rand(j) ;  
    return 0 ;  
}
```

- Ce code :
 - A – compile et fonctionne sans erreur
 - B – compile mais génère une erreur à l'exécution
 - C – ne compile pas

En bref :

- Allocation statique :
 - Variables globales
 - Chaînes constantes
 - ⇒ dans le binaire, adresse fixe dans la zone data
- Allocation automatique :
 - Variables locales
 - Paramètre de fonction
 - ⇒ réservé dans la pile en début de bloc, libéré en sortie de bloc.
Adresses variables
- Allocation dynamique :
 - malloc / free / sbrk
 - ⇒ Mémoire réservée à la demande dans le tas.

Les pointeurs

- Pour toute variable *foo i*, *&i* est de type *foo **
- Les pointeurs sont des variables qui contiennent des adresses mémoire.

*foo *p*

- La variable *p* contient une adresse
- L'adresse contenue dans *p* correspond au début de *sizeof(foo)* octets qui seront interprétés selon *foo*.

Les pointeurs : exemple

int i ;

- *i* représente 4 (variable) octets interprétés comme un entier, cette interprétation peut changer selon la plateforme.
- *&i* est une adresse sur 8 (variable) octets

*int *p=i ;*

- *p* représente une adresse, ici celle du début des 4 octets accessibles via la variable *i*
- *&p* représente une adresse : celle d'un pointeur sur un entier.

*int **q=&p ;*


- *q* représente une adresse, l'adresse contenue dans *q* est interprétée comme l'adresse de début d'une zone de 8 octets qui seront interprétés comme l'adresse de début d'une zone de 4 octets qui seront interprétés selon l'encodage des entiers de la plateforme.
-

Core

- Lors d'une faute mémoire (accès invalide à une page), le système génère une copie de la mémoire du processus sur le disque : un fichier **core**
- La commande ulimit permet de contrôler cette option : `ulimit -c` / `ulimit -c unlimited`

```
#include <stdlib.h>

int main(){
    char msg[]="ceci est un tres long message...";
    char *p=NULL;
    *p=0;
}
```



```
$ gcc bug.c
$ ulimit -c unlimited
$ ./a.out
Segmentation fault (core dumped)
$ ls -l core
-rw----- 1 allali 262144 core
$ strings core | grep ceci
ceci estH
ceci estH
ceci est un tres long message...
```

Core

- Le core contient l'ensemble des données mémoire : pile, tas, données, code...
- Il est possible de créer un core pour un processus actif avec la commande **gcore**
- Cela permet de contrôler la mémoire d'un processus à divers étapes de son exécution.
- On peut analyser un fichier core avec n'importe quel éditeur, par exemple :

```
$ hexdump -c core | less
```


mais l'interprétation en est très difficile !

core

```
0000000 177 E L F 002 001 001 \0 \0 \0 \0 \0 \0 \0 \0 \0
0000010 004 \0 > \0 001 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
0000020 @ \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
0000030 \0 \0 \0 \0 @ \0 8 \0 023 \0 \0 \0 \0 \0 \0 \0
0000040 004 \0 \0 \0 \0 \0 \0 \0 h 004 \0 \0 \0 \0 \0
0000050 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
0000060 d \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
0000070 \0 \0 \0 \0 \0 \0 \0 \0 001 \0 \0 \0 005 \0 \0
0000080 \0 020 \0 \0 \0 \0 \0 \0 \0 @ \0 \0 \0 \0 \0
0000090 \0 \0 \0 \0 \0 \0 \0 \0 \0 020 \0 \0 \0 \0 \0
00000a0 \0 020 \0 \0 \0 \0 \0 \0 \0 020 \0 \0 \0 \0 \0
00000b0 001 \0 \0 \0 004 \0 \0 \0 \0 \0 \0 \0 \0 \0
00000c0 \0 \0 ` \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
00000d0 \0 020 \0 \0 \0 \0 \0 \0 \0 020 \0 \0 \0 \0 \0
00000e0 \0 020 \0 \0 \0 \0 \0 001 \0 \0 \0 006 \0 \0
00000f0 \0 0 \0 \0 \0 \0 \0 \0 \0 020 ` \0 \0 \0 \0
0000100 \0 \0 \0 \0 \0 \0 \0 \0 \0 020 \0 \0 \0 \0 \0
0000110 \0 020 \0 \0 \0 \0 \0 \0 \0 020 \0 \0 \0 \0 \0
0000120 001 \0 \0 \0 005 \0 \0 \0 \0 @ \0 \0 \0 \0 \0
0000130 \0 0 <C0> S <BF> 177 \0 \0 \0 \0 \0 \0 \0 \0
0000140 \0 020 \0 \0 \0 \0 \0 \0 \0 <A0> 033 \0 \0 \0 \0
0000150 \0 020 \0 \0 \0 \0 \0 \0 001 \0 \0 \0 \0 \0 \0
0000160 \0 P \0 \0 \0 \0 \0 \0 \0 <D0> <DB> S <BF> 177 \0 \0
0000170 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
0000180 \0 \0 \0 \0 \0 \0 \0 \0 020 \0 \0 \0 \0 \0 \0
0000190 001 \0 \0 \0 004 \0 \0 \0 \0 P \0 \0 \0 \0 \0
00001a0 \0 <D0> <FB> S <BF> 177 \0 \0 \0 \0 \0 \0 \0 \0
00001b0 \0 @ \0 \0 \0 \0 \0 \0 @ \0 \0 \0 \0 \0 \0
00001c0 \0 020 \0 \0 \0 \0 \0 \0 001 \0 \0 \0 006 \0 \0
00001d0 \0 220 \0 \0 \0 \0 \0 \0 \0 020 <FC> S <BF> 177 \0 \0
00001e0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
00001f0 \0 \0 \0 \0 \0 \0 \0 \0 020 \0 \0 \0 \0 \0 \0
0000200 001 \0 \0 \0 006 \0 \0 \0 \0 <B0> \0 \0 \0 \0 \0
0000210 \0 0 <FC> S <BF> 177 \0 \0 \0 \0 \0 \0 \0 \0
0000220 \0 P \0 \0 \0 \0 \0 \0 P \0 \0 \0 \0 \0 \0
0000230 \0 020 \0 \0 \0 \0 \0 \0 001 \0 \0 \0 005 \0 \0
0000240 \0 \0 001 \0 \0 \0 \0 \0 \0 200 <FC> S <BF> 177 \0 \0
0000250 \0 \0 \0 \0 \0 \0 \0 \0 \0 020 \0 \0 \0 \0 \0
```

```
0000260 \0 0 002 \0 \0 \0 \0 \0 \0 020 \0 \0 \0 \0 \0
0000270 001 \0 \0 \0 006 \0 \0 \0 \0 020 001 \0 \0 \0 \0
0000280 \0 <A0> 034 T <BF> 177 \0 \0 \0 \0 \0 \0 \0 \0
0000290 \0 0 \0 \0 \0 \0 \0 \0 0 \0 \0 \0 \0 \0 \0
00002a0 \0 020 \0 \0 \0 \0 \0 \0 001 \0 \0 \0 006 \0 \0
00002b0 \0 @ 001 \0 \0 \0 \0 \0 \0 200 036 T <BF> 177 \0 \0
00002c0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
00002d0 \0 \0 \0 \0 \0 \0 \0 \0 020 \0 \0 \0 \0 \0 \0
00002e0 001 \0 \0 \0 004 \0 \0 \0 \0 ` 001 \0 \0 \0 \0
00002f0 \0 <A0> 036 T <BF> 177 \0 \0 \0 \0 \0 \0 \0 \0
0000300 \0 020 \0 \0 \0 \0 \0 \0 \0 020 \0 \0 \0 \0 \0
0000310 \0 020 \0 \0 \0 \0 \0 \0 001 \0 \0 \0 006 \0 \0
0000320 \0 p 001 \0 \0 \0 \0 \0 \0 <B0> 036 T <BF> 177 \0 \0
0000330 \0 \0 \0 \0 \0 \0 \0 \0 \0 020 \0 \0 \0 \0 \0
0000340 \0 020 \0 \0 \0 \0 \0 \0 \0 020 \0 \0 \0 \0 \0
0000350 001 \0 \0 \0 006 \0 \0 \0 \0 200 001 \0 \0 \0 \0
0000360 \0 <C0> 036 T <BF> 177 \0 \0 \0 \0 \0 \0 \0 \0
0000370 \0 020 \0 \0 \0 \0 \0 \0 \0 020 \0 \0 \0 \0 \0
0000380 \0 020 \0 \0 \0 \0 \0 \0 001 \0 \0 \0 006 \0 \0
0000390 \0 220 001 \0 \0 \0 \0 \0 \0 <F0> 200 \0 <FF> 177 \0 \0
00003a0 \0 \0 \0 \0 \0 \0 \0 \0 \0 002 \0 \0 \0 \0 \0
00003b0 \0 002 \0 \0 \0 \0 \0 \0 020 \0 \0 \0 \0 \0 \0
00003c0 001 \0 \0 \0 005 \0 \0 \0 \0 <B0> 003 \0 \0 \0 \0
00003d0 \0 ` 215 \0 <FF> 177 \0 \0 \0 \0 \0 \0 \0 \0
00003e0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
00003f0 \0 020 \0 \0 \0 \0 \0 \0 001 \0 \0 \0 004 \0 \0
0000400 \0 <D0> 003 \0 \0 \0 \0 \0 \0 200 215 \0 <FF> 177 \0 \0
0000410 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
0000420 \0 \0 \0 \0 \0 \0 \0 \0 020 \0 \0 \0 \0 \0 \0
0000430 001 \0 \0 \0 005 \0 \0 \0 \0 <F0> 003 \0 \0 \0 \0
0000440 \0 \0 ` <FF> <FF> <FF> <FF> <FF> \0 \0 \0 \0 \0 \0
0000450 \0 020 \0 \0 \0 \0 \0 \0 \0 020 \0 \0 \0 \0 \0
0000460 \0 020 \0 \0 \0 \0 \0 \0 \0 005 \0 \0 \0 P 001 \0
0000470 001 \0 \0 \0 C O R E \0 \0 \0 \0 \0 \0 \0
0000480 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
0000490 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 177 8 \0 \0
```

....

gdb

- gdb : GNU Debugger
- permet d'analyser la mémoire avec une couche d'interprétation.
- Il est possible de
 - prendre le contrôle d'un processus en cours d'exécution
 - de lancer un nouveau processus dans gdb
 - d'analyser la mémoire à posteriori hors exécution (core).

gdb : run

- On lance gdb en indiquant le binaire que l'on souhaite analyser, possiblement suivi d'un pid ou d'un fichier core
- gdb va charger le fichier binaire
- Dans le cas d'un pid, il va stopper le processus
- Il est possible de lancer un nouveau processus avec la commande :

run arg1 arg2 ...

- Ctrl-C permet de suspendre l'exécution du programme, « continue » de la reprendre.

gdb

- En l'absence d'informations additionnelles, gdb ne peut pas associer le contenu mémoire à des variables structurées et donc interpréter la mémoire.
- Il peut cependant indiquer la pile d'appel :

```
(gdb) backtrace
#0  0x00000000004004fa in f ()
#1  0x000000000040050f in g ()
#2  0x0000000000400522 in h ()
#3  0x0000000000400535 in main ()
```

- On voit ici, les adresses des fonctions (dans la zone de code) ainsi que la pile d'appel.

gdb

- Si le binaire contient des symboles de debug alors on peut :
 - lister le code (list)
 - placer un arrêt d'exécution sur une ligne spécifique (break)
 - afficher le contenu d'une variable (print ou display)
 - avancer d'une ligne de code (next ou step)

gdb : cas d'école !

- mon programme fait un SEGFAULT
- 1 ⇒ j'exécute (dans gdb ou bien core)
- 2 ⇒ j'identifie la cause directe du problème :

```
Reading symbols from a.out...done.
(gdb) r
Starting program: /tmp/a.out

Program received signal SIGSEGV, Segmentation fault.
__strcpy_sse2_unaligned ()
    at ../sysdeps/x86_64/multiarch/strcpy-sse2-unaligned.S:682
682    ../sysdeps/x86_64/multiarch/strcpy-sse2-unaligned.S: No such file or directory.
(gdb) bt
#0  __strcpy_sse2_unaligned ()
    at ../sysdeps/x86_64/multiarch/strcpy-sse2-unaligned.S:682
#1  0x00000000004006cc in main (argc=1, argv=0x7fffffffe058) at s.c:13
(gdb) up
#1  0x00000000004006cc in main (argc=1, argv=0x7fffffffe058) at s.c:13
13      strcpy(p,argv[i]);
(gdb) p p
$1 = 0x0
(gdb) p i
$2 = 0
(gdb) p argv[i]
$3 = 0x7fffffffe38d "/tmp/a.out"
(gdb)
```

gdb : cas d'école !

- je remonte la pile jusqu'à mon code
- je trouve la valeur qui pose problème (ici p)
- je liste le code :

```
(gdb) l
8      l+=1;
9      s=malloc(sizeof(char)*l);
10     if (s=NULL) return EXIT_FAILURE;
11     p=s;
12     for(i=0;i<argc;++i) {
13         strcpy(p,argv[i]);
14         p+=strlen(argv[i]);
15     }
16     puts(s);
17     return EXIT_SUCCESS;
```

- La valeur de p est « fixée » ligne 11. Je vérifie cela :
- « break 11 »
- « run »

gdb : cas d'école !

Starting program: /tmp/a.out

Breakpoint 1, main (argc=1, argv=0x7fffffffe058) at s.c:11

```
11  p=s;
```

```
(gdb) p s
```

```
$4 = 0x0
```

```
(gdb) l
```

```
6   int i,l=0;
```

```
7   for(i=0;i<argc;++i) l+=strlen(argv[i]);
```

```
8   l+=1;
```

```
9   s=malloc(sizeof(char)*l);
```

```
10  if (s=NULL) return EXIT_FAILURE;
```

```
11  p=s;
```

```
12  for(i=0;i<argc;++i) {
```

```
13      strcpy(p,argv[i]);
```

```
14      p+=strlen(argv[i]);
```

```
15  }
```

```
(gdb) b 9
```

Breakpoint 2 at 0x40066e: file s.c, line 9.

- apparemment la valeur de s est déjà à 0
- s est initialisée ligne 9
- « b 9 »
- « r »

gdb : cas d'école !

```
Breakpoint 2, main (argc=1,
argv=0x7ffffffe058) at s.c:9
9      s=malloc(sizeof(char)*l);
(gdb) next
10      if (s=NULL) return EXIT_FAILURE;
(gdb) p s
$5 = 0x602010 ""
(gdb) display s
1: s = 0x602010 ""
(gdb) n
```

```
Breakpoint 1, main (argc=1,
argv=0x7ffffffe058) at s.c:11
11     p=s;
1: s = 0x0
(gdb)
```

- je passe la ligne avec « next » puis je contrôle la valeur de s
- apparemment s change de valeur (de 0x602010 à 0x0)
- je fais un display puis j'exécute pas à pas.
- après la ligne 10, la valeur a changée...

gdb : autres

- Il existe d'autre interface pour gdb :
 - xxgdb
 - ddd
 - kgdb
 - etc...
- La plus part des environnements de développement propose un debugger intégré :
 - visual studio
 - qtcreator
 - kdevelop
 - etc...

question

#QDLE#Q#AB*C#20#

- malloc renvoie une adresse située :
 - A. dans le segment de donnée
 - B. dans le tas
 - C. dans la pile

tracer les accès mémoires

- valgrind [vælgrɪnd] est un outil qui intègre de nombreux tests pour l'exécution de programmes.
- Permet de détecter des erreurs d'exécution qui ne sont pas relevées par le système comme le dépassement d'un tableau par exemple.
- L'exécution du programme par valgrind est contrôlée pas à pas, ce qui ralentit beaucoup le programme.

valgrind

- Pour pouvoir fonctionner efficacement avec valgrind, le programme doit avoir été compilé avec -g et -O0
- ensuite, on lance le programme dans valgrind :
`valgrind ./a.out arg1 arg2 ...`
- valgrind va afficher les erreurs mémoires au fur et à mesure de leur apparition
- Il est possible de demander à valgrind de lancer gdb à chaque erreur :
`valgrind --vgdb-error=1 ./a.out`

exemple :

```
allali@hebus:/tmp$ valgrind --vgdb-error=1 ./a.out
==9539== Memcheck, a memory error detector
==9539== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==9539== Using Valgrind-3.10.0 and LibVEX; rerun with -h for copyright info
==9539== Command: ./a.out
==9539==
==9539== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==9539== /path/to/gdb ./a.out
==9539== and then give GDB the following command
==9539== target remote | /usr/lib/valgrind/../../bin/vgdb --pid=9539
==9539== --pid is optional if only one valgrind process is running
==9539==
==9539== Invalid write of size 4
==9539== at 0x400570: main (s.c:7)
==9539== Address 0x51fc050 is 0 bytes after a block of size 16 alloc'd
==9539== at 0x4C2ABA0: malloc (in /usr/lib/valgrind/vgpreload_memcheck.so)
==9539== by 0x40054E: main (s.c:4)
==9539==
==9539== (action on error) vgdb me ...
```

Toute erreur signalée par valgrind
mérite une analyse et, à de très rares
exceptions, doit être corrigée !

```
allali@hebus:/tmp$ gdb a.out
...
Reading symbols from a.out...done.
(gdb) target remote | /usr/lib/valgrind/../../bin/vgdb --pid=9539
...
Loaded symbols for /lib64/ld-linux-x86-64.so.2
0x0000000000400570 in main (argc=1, argv=0xfffff98) at s.c:7
7         t[i]=0;
(gdb) p t
$1 = (int *) 0x51fc040
(gdb) p i
$2 = 4
```

```
1 #include<stdlib.h>

3 int main(int argc, char **argv){
4     int *t=malloc(sizeof(int)*4);
5     int i;
6     for(i=0;i<6;++i)
7         t[i]=0;
8     return 0;
9 }
```

valgrind : fuite mémoire

#QDLE#Q#ABCD*#45#

- Une fuite mémoire correspond à de la mémoire encore réservée à la fin de votre programme
- Celle-ci peut être ?
 - A. dans la pile et le tas
 - B. dans la pile, le tas et le segment de données
 - C. uniquement dans la pile
 - D. uniquement dans le tas

valgrind : fuite mémoire

- A la fin de l'exécution, valgrind liste les blocs non libérés et les classe selon :
 - que plus rien ne pointe sur ce segment (definitively lost)
 - qu'un autre segment de mémoire pointe encore dessus (indirectly lost)
 - encore accessible par une variable de pile (still reachable)

The compiling trivia

#QDLE#Q#ABCD*E#25#

- Quelle commande permet de compiler le fichier toto.c en objet ?
 - A. gcc toto.c
 - B. gcc -E toto.c
 - C. gcc -S toto.c
 - D. gcc -c toto.c
 - E. gcc -fPIC toto.c



The compiling trivia

#QDLE#Q#AB*CD#25#

- Quelle commande permet de récupérer le source après l'étape de pré-compilation ?
 - A. gcc toto.c
 - B. gcc -E toto.c
 - C. gcc -S toto.c
 - D. gcc -c toto.c



The compiling trivia

#QDLE#Q#A*BCD#25#

- Quelle commande permet de lier toto.o et main.o en un exécutable ?
 - A. gcc toto.o main.o
 - B. gcc -shared toto.o main.o
 - C. gcc -static toto.o main.o
 - D. gcc -exec toto.o main.o



The compiling trivia

#QDLE#Q#ABCD*#25#

- Quelle commande permet de lier toto.o et main.o en une bibliothèque statique?
 - A. gcc -shared -o libfoo.a toto.o main.o
 - B. gcc -static -o libfoo.a toto.o main.o
 - C. gcc toto.o main.o -lfoo
 - D. ar rcs libfoo.a toto.o main.o



The compiling trivia

#QDLE#Q#ABCDEF*G#30#

- Pour pouvoir regrouper des objets (.o) dans une bibliothèque dynamique, il faut les compiler avec la ou les options :
 - A. -c
 - B. -E
 - C. -s
 - D. -fPIC
 - E. réponse A&B
 - F. réponse A&D
 - G. réponse B&E



The compiling trivia

#QDLE#Q#A*BCD#25#

- Quelle commande permet de lier toto.o et main.o en une bibliothèque dynamique?
 - A. gcc -shared -o libfoo.so toto.o main.o
 - B. gcc -dynamic -o libfoo.a toto.o main.o
 - C. gcc toto.o main.o -lfoo
 - D. ar rcsD libfoo.so toto.o main.o



Automatisation

- La compilation manuelle n'est pas possible sur un grand projet :
 - homogénéisation des options de compilation
 - gestion des dépendances (que doit-on recompiler?)
 - commandes et options spécifiques à une plateforme.
 - erreurs manuelles
 - ...
- On doit se munir d'outil pour automatiser cette étape.

Makefile

- make est un outil qui permet d'automatiser la création et la mise à jour de fichiers.
- make repose sur un fichier de description : **Makefile**
- Un Makefile est un ensemble de règles formatées :

```
cible : source1 source2 source3 ...
```

```
    commande1
```

```
    commande2
```

```
... .
```

- Si « cible » n'existe pas ou est moins récente que l'une des sources, alors les commandes sont exécutées.

Makefile : exemple

```
image_thumbail.jpg : image.jpg
    convert -size 80x80 image.jpg image_thumbnail.jpg
image.jpg : image.png
    convert image.png image.jpg
image.png : image.svg
    inkscape -z -e image.png image.svg
```

- make est récursif : si un fichier source n'existe pas, il cherche une règle pour le créer.
- initialement, make cherche à créer une seule cible : la première du Makefile ou celle(s) indiquée(s) sur la ligne d'appel : `make image.jpg`
- Si une source est manquante et qu'il n'y a pas de règle pour la créer : erreur
- Si une des commandes renvoie une valeur différente de 0, make s'arrête (tips : `commande || true`)

Makefile : variable

- make supporte la déclaration de variable :

`NOM=VALEUR`

- valeur peut comporter des espaces

`NOM = VALEUR`

- Si NOM est dans l'environnement, alors c'est la valeur de l'environnement qui est utilisée, sinon c'est VALEUR.
- `$(NOM)` est remplacé par VALEUR.

Makefile : variable, exemple

```
CONVERT=convert
THUMB_SIZE-=80x80
image_thumbail.jpg : image.jpg
    $(CONVERT) -size $(THUMB_SIZE) image.jpg image_thumbnail.jpg
image.jpg : image.png
    $(CONVERT) image.png image.jpg
image.png : image.svg
    inkscape -z -e image.png image.svg
```

\$ THUMB_SIZE=60x60 make

Makefile : spéciales

- Quelques variables spéciales pour l'écriture des commandes :

`cible : source1 source2`

- `$@` : cible
- `$<` : source1
- `$$` : source1 source2

- Ainsi :

```
image_thumbail.jpg : image.jpg
    $(CONVERT) -size $(THUMB_SIZE) image.jpg image_thumbnail.jpg
```

- Devient :

```
image_thumbail.jpg : image.jpg
    $(CONVERT) -size $(THUMB_SIZE) $< $@
```

Makefile : règles génériques

- Il est possible d'écrire des règles génériques :

```
% .jpg : % .png
```

```
convert $< $@
```

- Permet de convertir tout fichier png en jpg à l'aide de convert

```
%_thumb.jpg : % .jpg
```

```
convert -size 80x80 $< $@
```

- Il est enfin possible de séparer la liste des dépendances et la règle de création.

Makefile et compilation

```
CC=gcc
CFLAGS=-Wall

prog : prog.o hash.o
    $(CC) -o $@ $^
prog.o : prog.c hash.h
hash.o : hash.c hash.h
%.o :
    $(CC) $(CFLAGS) $< -o $@
```

- La dernière règle explique comment générer un fichier .o
- les deux règles au dessus donnent les dépendances

Makefile : PHONY

- On peut vouloir écrire des règles qui ne génèrent pas de fichier :

`all`

`install`

`clean`

`distclean`

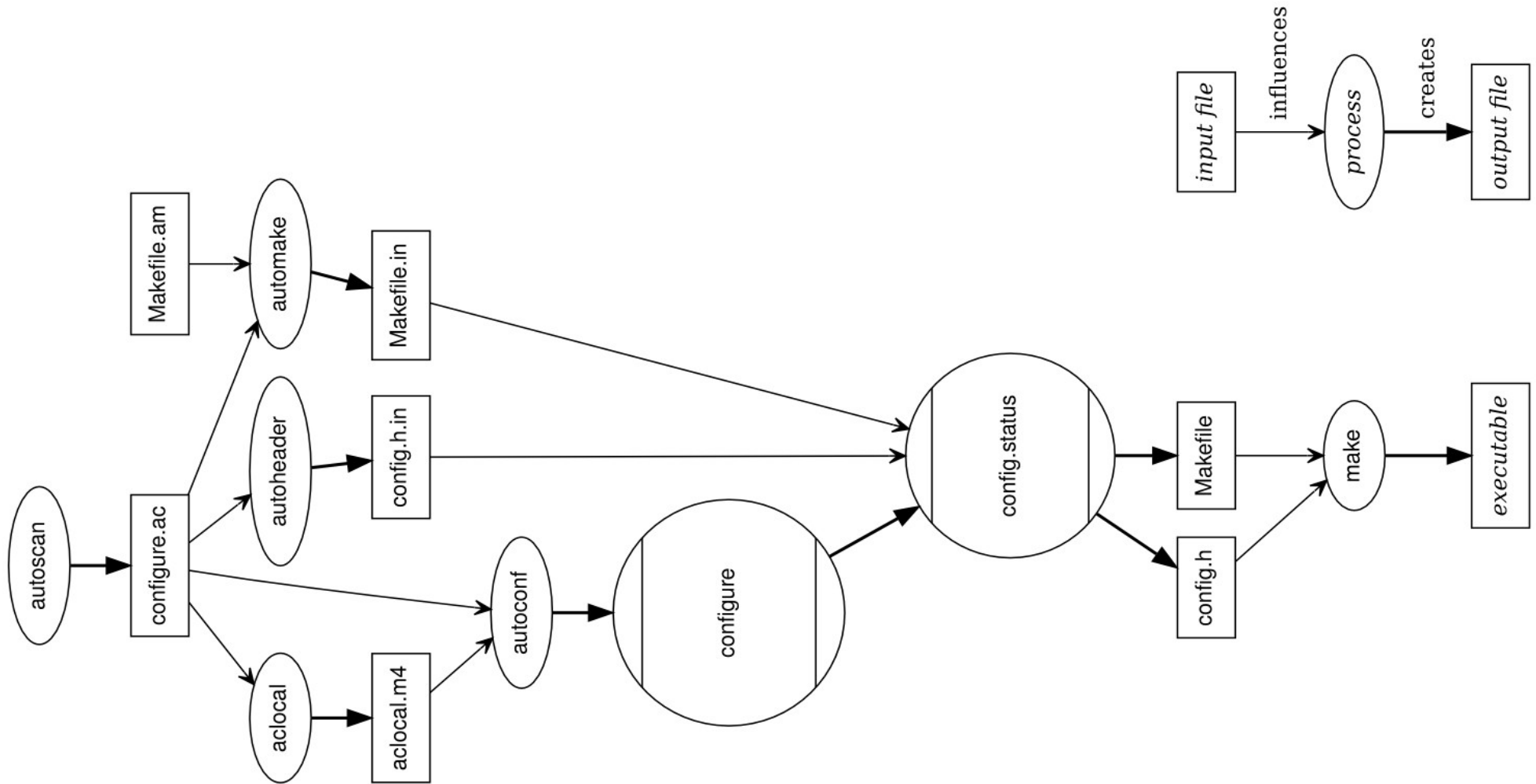
- On indique cela à make :

```
.PHONY=all install clean distclean
```

Makefile, automake...

- Makefile n'est pas spécifique à la compilation de programme
- Il ne gère pas, entre autre, la dépendance entre les fichiers sources (on peut utiliser `gcc -MM source1.c source2.c ...` pour cela)
- La prise en compte de l'environnement (compilateur, options, bibliothèque...) peut ce faire :
 - par l'édition des variables du Makefile (options de compilations, répertoire d'installation...)
 - par l'écriture de plusieurs Makefile (un par système)
 - par l'utilisation d'un générateur :
 - automake / autoconf
 - cmake

auto-tools : automake / autoconf



cmake

- Cmake est un outil simplifié permettant la compilation de sources C et C++.
- C'est un outil multiplateforme sous licence BSD
- Nécessite la présence d'un fichier CMakeLists.txt
- gestion automatique des dépendances
- peut générer des makefile
- Simple d'utilisation / facile à prendre en main
- Exemple: un projet <<HELLO>> avec une bibliothèque dans le répertoire Hello et un programme d'exemple dans le répertoire Demo

cmake

- `./CMakeLists.txt`:

```
cmake_minimum_required (VERSION 2.6)
project (HELLO)
```

```
add_subdirectory (Hello)
add_subdirectory (Demo)
```

- `./Hello/CMakeLists.txt`

```
add_library (Hello hello.c)
```

- `./Demo/CMakeLists.txt`

```
include_directories (${HELLO_SOURCE_DIR}/Hello)
link_directories (${HELLO_BINARY_DIR}/Hello)
add_executable (helloDemo demo.c demo_b.c)
target_link_libraries (helloDemo Hello)
```

cmake:cross platform make

- cmake est un système de compilation cross-platform. Il ne compile pas directement mais génère des fichiers dans différents formats :
 - Makefile
 - projet Visual Studio
 - Borland Makefile
 - projet Xcode
 - Kate
 - ...
- cmake utilise les fichiers CMakeLists.txt et génère des fichiers en fonction de la plate-forme de compilation (Makefile, visual, xcode....).

cmake : les bases

- La déclaration de variables :
 - `set(NAME VALUE)`
 - `${NAME}`
 - lors de l'appel à cmake : `cmake -DNAME=VALUE`
- Variables standards :
 - `CMAKE_INCLUDE_PATH` (pour les .h)
 - `CMAKE_LIBRARY_PATH` (pour la recherche de .so)
 - `DESTDIR` (pour l'installation)
 - `CMAKE_BUILD_TYPE` (Debug, Release)
- Dans le `CMakeLists.txt` :

– <code>CMAKE_C_FLAGS</code>	– <code>CMAKE_CURRENT_SOURCE_DIR</code>
– <code>CMAKE_C_FLAGS_DEBUG</code>	– <code>CMAKE_CURRENT_BINARY_DIR</code>
– <code>CMAKE_C_FLAGS_RELEASE</code>	– <code>CMAKE_SOURCE_DIR</code>

cmake : les bases (2)

- `add_executable(name sources)`
- `add_library(name STATIC sources)`
- `add_library(name SHARED sources)`
- `target_link_libraries(name libs)`
- `include_directories(dir1 dir2...)`
- `add_custom_command`

cmake : utilisation

- cmake supporte l'out-source building : c'est à dire la compilation en dehors du répertoire des sources :
- On suppose : projet/CMakeLists.txt
- alors on peut faire :

```
mkdir projet-build
```

```
cmake ../projet
```

```
make
```

- et

```
mkdir projet-debug
```

```
cmake -DCMAKE_BUILD_TYPE=Debug ../projet
```

```
make
```

Question

#QDLE#Q#AB*#20#

- cmake analyse les dépendances :
 - A. en énumérant les fonctions appelées
 - B. en traçant les inclusions

Question

#QDLE#Q#A*BCD#35#

- J'ai une bibliothèque dynamique libtoto.so compilée à partir de toto.c et toto.h. J'ai un programme de démo demo.c qui utilise cette bibliothèque. L'ensemble est compilé. Si je modifie toto.c je dois :
 - A. re-compiler la bibliothèque
 - B. re-compiler l'exécutable
 - C. réponse A & B
 - D. ne rien faire.

Question

#QDLE#Q#ABC*D#25#

- J'ai une bibliothèque dynamique libtoto.so compilée à partir de toto.c et toto.h. J'ai un programme de démo demo.c qui utilise cette bibliothèque. L'ensemble est compilé. Si je modifie toto.h je dois :
 - A. re-compiler la bibliothèque
 - B. re-compiler l'exécutable
 - C. réponse A & B
 - D. ne rien faire.