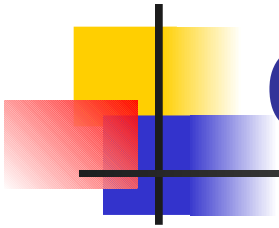
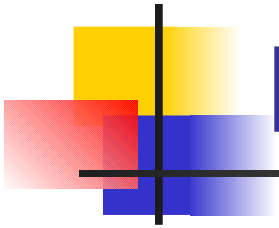


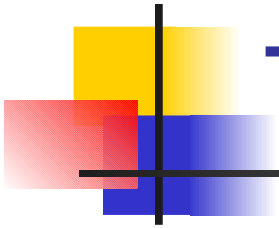
GESTION MEMOIRE

- Parmi les erreurs les plus typiques d'un programme C (ou C++) liées aux allocations dynamiques, on peut citer :
  - Les "fuites mémoires" (mémory leak)
  - l'utilisation de la mémoire après sa libération
  - la mémoire libérée plusieurs fois
  - des débordements de tableaux
  - l'utilisation de zones mémoire non initialisées.



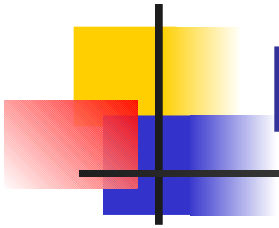
Des outils …

- Une des particularités des problème mémoire est la difficulté de trouver leur origine
- Des outils existent :
  - *Ils ne remplacent pas une bonne conception et une implémentation soignée, il les complète.*
  - *Quand on trouve une erreur, que fait-on ?*
    - *Le pire est possible …*
    - *précipitation vs qualité !*
    - *Trouver le bon niveau de correction : faute de frappe, erreur basique, erreur plus sophistiquée, problème de conception*



Types d'outils

- On peut citer des critères importants qui permettent de différencier les outils:
  - on doit modifier le code source pour pouvoir l'utiliser
  - on doit recompiler/relinker pour pouvoir l'utiliser
  - rien à faire
- besoins primaires :
  - intercepter tous les appels à malloc/calloc/free/realloc
  - utiliser un gestionnaire mémoire plus sophistiqué
    - en modifiant l'exécutable ou les librairies
    - en modifiant le code source
      - par exemple:
        - `#define malloc(n) mallocEx(n)`
        - `...`



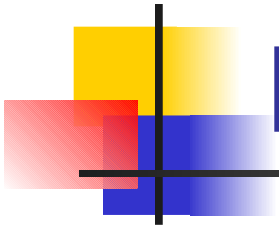
Exemples d'erreurs repérables

- Exemples d'utilisation de *valgrind* 3.0.1
  - http://valgrind.org
- Valgrind regroupe un ensemble d'outils de debug et profiling.
  - on s'intéresse ici à ***memcheck***, l'outil de debug des problèmes de gestion mémoire.

Valgrind ne nécessite aucune modification des sources ni recompilation !

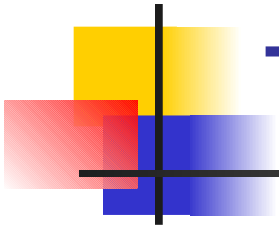
Valgrind utilise la table de symboles (option –g)

- Les exemples proviennent principalement du répertoire de tests de *mpatrol*



Débordements (1)

```
int main(void)
{
    char *p = NULL;

    if (p = (char *) malloc(8))
    {
        strcpy(p, "le monde");
        free(p); p = NULL;
    }
    return EXIT_SUCCESS;
}
```



Test

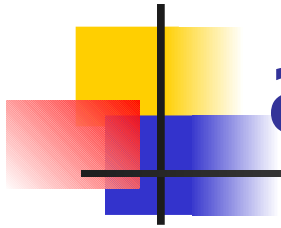
- Compilation & éditions de liens

```
gcc -g -ansi -Wall -c ex.c
gcc -g -o ex ex.o
```

- gdb

```
OK !
```

  - Bug actif mais invisible
  - Bug sans effet (à un instant t)



avec valgrind ...

Extraits

```
==1673== Invalid write of size 1
==1673==    at 0x80483E4: f (ex.c:17)
==1673==    by 0x804841D: main (ex.c:25)
==1673==  Address 0x1BA4D030 is 0 bytes after a block of size 8 alloc'd
==1673==    at 0x1B8FF896: malloc (vg_replace_malloc.c:149)
==1673==    by 0x80483C7: f (ex.c:15)
==1673==    by 0x804841D: main (ex.c:25)
```

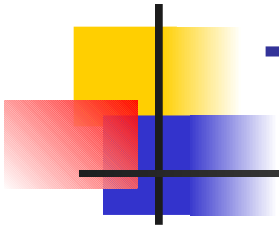


Table des symboles ?

- Compilation & éditions de liens

```
gcc -ansi -Wall -c ex.c
gcc -o ex ex.o
```

- valgrind

```
=1724= Invalid write of size 1
=1724=    at 0x80483E4: f (in /home/MLV/TPs/ex)
=1724=    by 0x804841D: main (in /home/MLV/TPs/ex)
=1724=  Address 0x1BA4D030 is 0 bytes after a block of size
  8 alloc'd
=1724=    at 0x1B8FF896: malloc (vg_replace_malloc.c:149)
=1724=    by 0x80483C7: f (in /home/MLV/TPs/ex)
=1724=    by 0x804841D: main (in /home/MLV/TPs/ex)
```

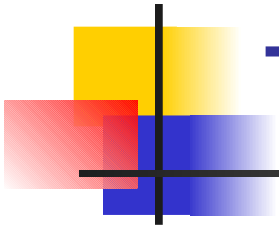


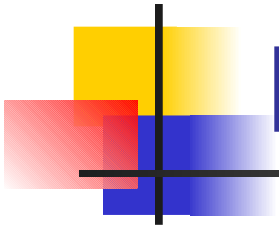
Table des symboles ?

- Suppression des symboles

```
strip ex
```

- valgrind

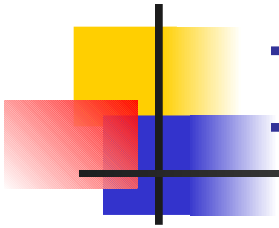
```
=813== Invalid write of size 1
=813==    at 0x80483E4: (within /home/MLV/TPs/ex)
=813==    by 0x804841D: (within /home/MLV/TPs/ex)
=813==    by 0x1B92EE3F: __libc_start_main (in /lib/tls/libc-2.3.5.so)
=813==    by 0x8048330: (within /home/MLV/TPs/ex)
=813==  Address 0x1BA4D030 is 0 bytes after a block of size 8 alloc'd
=813==    at 0x1B8FF896: malloc (vg_replace_malloc.c:149)
=813==    by 0x80483C7: (within /home/MLV/TPs/ex)
=813==    by 0x804841D: (within /home/MLV/TPs/ex)
=813==    by 0x1B92EE3F: __libc_start_main (in /lib/tls/libc-2.3.5.so)
=813==    by 0x8048330: (within /home/MLV/TPs/ex)
```



Lien avec debugger !

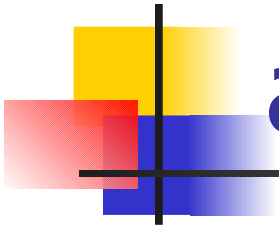
```
$ valgrind –db-attach=yes ex
```

À chaque erreur, valgrind propose de lancer le debugger pour attacher le processus !



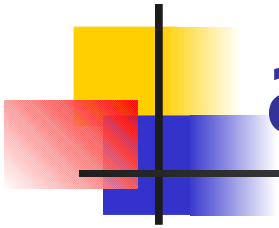
Incohérence malloc / free

```
/*
*Allocates a block of 16 bytes and then attempts to free the
* memory returned at an offset of 1 byte into the block.
*/


int main(void)
{
   char *p;
   if (p = (char *) malloc(16))
       free(p + 1);
   return EXIT_SUCCESS;
}
```



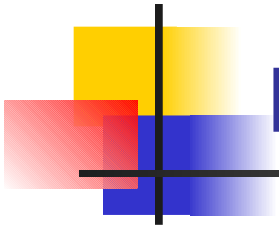
avec gdb ...

```
(gdb) run
Starting program: /home/MLV/TPs/GESTION-MEMOIRE/ex1

Program received signal SIGSEGV, Segmentation fault.
0x400a3e8a in free () from /lib/libc.so.6
(gdb) bt
#0 0x400a3e8a in free () from /lib/libc.so.6
#1 0x400a3bf4 in free () from /lib/libc.so.6
#2 0x0804846c in main () at ex1.c:15
#3 0x4003f280 in __libc_start_main () from
  /lib/libc.so.6
```



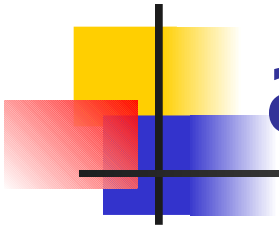
avec valgrind ...

```
==4511== Memcheck, a.k.a. Valgrind, a memory error detector
==4511== Invalid free() / delete / delete[]
==4511== at 0x4002A885: free (vg_replace_malloc.c:231)
==4511== by 0x804846B: main (ex1.c:15)
==4511== by 0x4025A27F: __libc_start_main (in
   /lib/libc-2.2.4.so)
==4511== by 0x8048360: (within /home/MLV/TPs/GESTION-
   MEMOIRE/ex1)
==4511== Address 0x411AC025 is 1 bytes inside a block of size
   16 alloc'd
==4511== at 0x4002A563: malloc (vg_replace_malloc.c:153)
==4511== by 0x804844F: main (ex1.c:14)
```



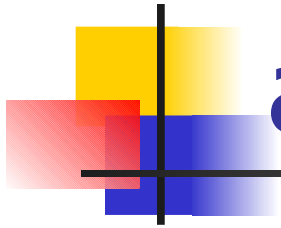
memory leak

```
int main(void)
{
  char *p;
  if (p = (char *) malloc(16))
  {
     strcpy(p, "hello");
     printf(p);
  }
  return EXIT_SUCCESS;
}
```



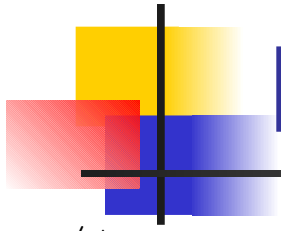
avec gdb ...

- rien n'est jamais signalé !



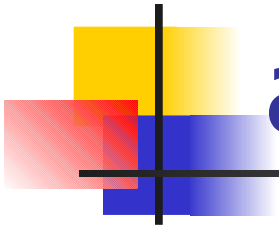
avec Valgrind –leak-check=yes

```
==4658== malloc/free: in use at exit: 16 bytes in 1 blocks.
==4658== malloc/free: 1 allocs, 0 frees, 16 bytes allocated.
==4658== For counts of detected errors, rerun with: -v
==4658== searching for pointers to 1 not-freed blocks.
==4658== checked 3606064 bytes.
==4658== 16 bytes in 1 blocks are definitely lost in loss record 1 of 1
==4658== at 0x4002A563: malloc (vg_replace_malloc.c:153)
==4658== by 0x804847F: main (ex1leak.c:11)
==4658== by 0x4025A27F: __libc_start_main (in /lib/libc-2.2.4.so)
==4658== by 0x8048390: (within /home/MLV/TPs/GESTION-MEMOIRE/ex1leak)
==4658==
==4658== LEAK SUMMARY:
==4658== definitely lost: 16 bytes in 1 blocks.
==4658== possibly lost: 0 bytes in 0 blocks.
==4658== still reachable: 0 bytes in 0 blocks.
==4658== suppressed: 0 bytes in 0 blocks.
==4658== Reachable blocks (those to which a pointer was found) are not shown.
==4658== To see them, rerun with: --show-reachable=yes
```



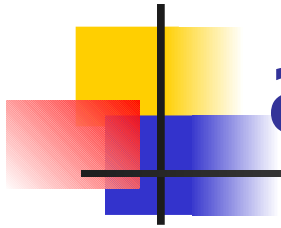
Réutilisation de mémoire libérée

```
/*
*Allocates a block of 16 bytes and then attempts to free the
* memory returned at an offset of 1 byte into the block.
*/


int main(void)
{
   char *p;
   if (p = (char *) malloc(16))
   {
       free(p);
       p = (char *) realloc(p, 32);
   }
   return EXIT_SUCCESS;
}
```



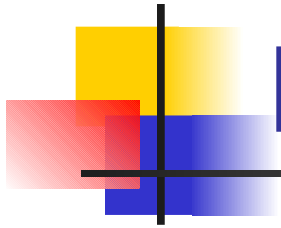
avec gdb ...

- Plantage simple ...



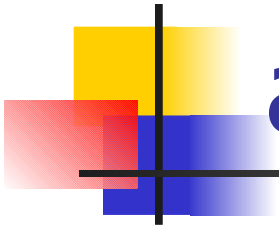
avec Valgrind

```
==4666== Invalid free() / delete / delete[]
==4666== at 0x4002ABF4: realloc (vg_replace_malloc.c:310)
==4666== by 0x80484A9: main (ex2.c:18)
==4666== by 0x4025A27F: __libc_start_main (in /lib/libc-2.2.4.so)
==4666== by 0x8048390: (within /home/MLV/TPs/GESTION-MEMOIRE/ex2)
==4666== Address 0x411AC024 is 0 bytes inside a block of size 16 free'd
==4666== at 0x4002A885: free (vg_replace_malloc.c:231)
==4666== by 0x8048499: main (ex2.c:17)
==4666== by 0x4025A27F: __libc_start_main (in /lib/libc-2.2.4.so)
==4666== by 0x8048390: (within /home/MLV/TPs/GESTION-MEMOIRE/ex2)
```



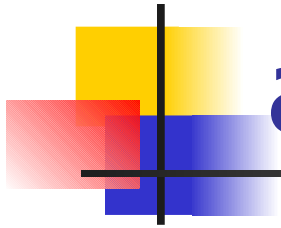
Réutilisation de mémoire libérée

```
/*
* Detecting use of free memory
*
* Allocates a block of 16 bytes and then immediately frees it. A
* NULL character is written into the middle of the freed memory.
*/

int main(void)
{
   char *p;
   if (p = (char *) malloc(16))
   {
        free(p);
        p[8] = '\0';
   }
   return EXIT_SUCCESS;
}
```



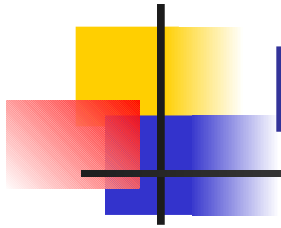
avec gdb …

- Exécution normale !



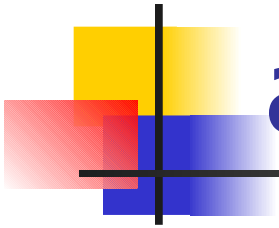
avec Valgrind

```
==4679== Invalid write of size 1
==4679== at 0x8048473: main (ex3.c:18)
==4679== by 0x4025A27F: __libc_start_main (in /lib/libc-2.2.4.so)
==4679== by 0x8048360: (within /home/MLV/TPs/GESTION-MEMOIRE/ex3)
==4679== Address 0x411AC02C is 8 bytes inside a block of size 16 free'd
==4679== at 0x4002A885: free (vg_replace_malloc.c:231)
==4679== by 0x8048469: main (ex3.c:17)
==4679== by 0x4025A27F: __libc_start_main (in /lib/libc-2.2.4.so)
==4679== by 0x8048360: (within /home/MLV/TPs/GESTION-MEMOIRE/ex3)
```



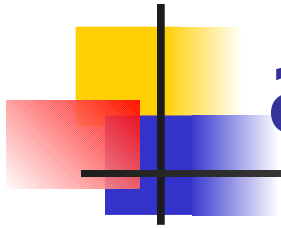
Débordements (2)

```
/*
* Using overflow buffers
*
* Allocates a block of 16 bytes and then copies a string of 16
* bytes into the block. However, the string is copied to 1 byte
* before the allocated block which writes before the start of the block
*/


int main(void) {
   char *p;
   if (p = (char *) malloc(16)) {
        strcpy(p - 1, "this test fails!");
        free(p);
   }
   return EXIT_SUCCESS;
}
```



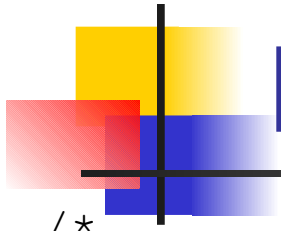
avec gdb …

- Plantage simple.



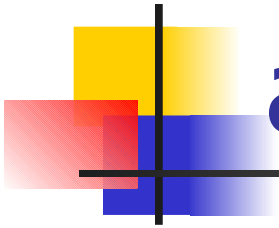
avec Valgrind

```
==4692== Invalid write of size 1
==4692== at 0x400224D2: strcpy (mac_replace_strmem.c:173)
==4692== by 0x80484A0: main (ex4.c:20)
==4692== by 0x4025A27F: __libc_start_main (in /lib/libc-2.2.4.so)
==4692== by 0x8048390: (within /home/MLV/TPs/GESTION-MEMOIRE/ex4)
==4692== Address 0x411AC023 is 1 bytes before a block of size 16 alloc'd
==4692== at 0x4002A563: malloc (vg_replace_malloc.c:153)
==4692== by 0x804847F: main (ex4.c:18)
==4692== by 0x4025A27F: __libc_start_main (in /lib/libc-2.2.4.so)
==4692== by 0x8048390: (within /home/MLV/TPs/GESTION-MEMOIRE/ex4)
```



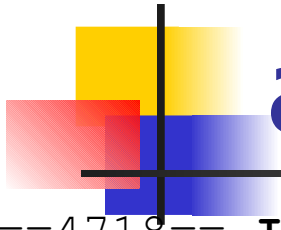
Débordements (3)

```
/*
* Bad memory operations
* Allocates a block of 16 bytes and then attempts to zero the contents
   of
* the block. However, a zero byte is also written 1 byte before and 1
* byte after the allocated block.
*/


int main(void) {
   char *p;
   if (p = (char *) malloc(16)) {
       memset(p - 1, 0, 18);
       free(p);
   }
   return EXIT_SUCCESS;
}
```



avec gdb …

- Exécution normale

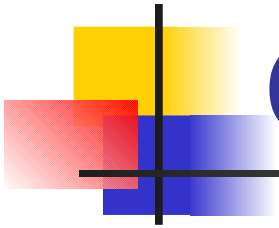


avec Valgrind

```
==4718== Invalid write of size 4
==4718== at 0x402C6789: memset (in /lib/libc-2.2.4.so)
==4718== by 0x4025A27F: __libc_start_main (in /lib/libc-2.2.4.so)
==4718== by 0x8048390: (within /home/MLV/TPs/GESTION-MEMOIRE/ex6)
==4718== Address 0x411AC023 is 1 bytes before a block of size 16 alloc'd
==4718== at 0x4002A563: malloc (vg_replace_malloc.c:153)
==4718== by 0x804847F: main (ex6.c:16)
==4718== by 0x4025A27F: __libc_start_main (in /lib/libc-2.2.4.so)
==4718== by 0x8048390: (within /home/MLV/TPs/GESTION-MEMOIRE/ex6)
==4718== Invalid write of size 1
==4718== at 0x402C6790: memset (in /lib/libc-2.2.4.so)
==4718== by 0x4025A27F: __libc_start_main (in /lib/libc-2.2.4.so)
==4718== by 0x8048390: (within /home/MLV/TPs/GESTION-MEMOIRE/ex6)
==4718== Address 0x411AC034 is 0 bytes after a block of size 16 alloc'd
==4718== at 0x4002A563: malloc (vg_replace_malloc.c:153)
==4718== by 0x804847F: main (ex6.c:16)
```

Débordements (3)
La magie a des limites ...

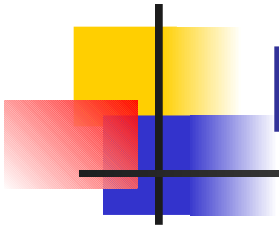
```
/* Checking memory accesses
   Allocates a single byte of memory and then attempts to read
   the byte as a word, resulting in some uninitialised bytes
   being read.
*/

int main(void)
{
   int *p;
   int r;
   if (p = (int *) calloc(1, 1)) {
       r = p[0];
       free(p);
   }
   return EXIT_SUCCESS;
}
```



Quelques outils

- Purify (Rational / IBM) : le plus connu !
- DevPartner (compuWare) (ex-BoundChecker)
- Insure++ (ParaSoft)
- et en GPL :
  - memprof,
  - dmalloc,
  - mpatrol,
  - Boehm Collector,
  - ...

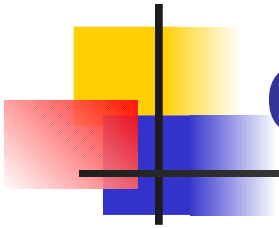
les différents outils sont souvent complémentaires !



PROFILER

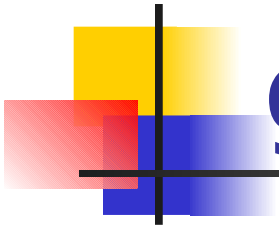
- *Objectifs:*
  - Analyser les performances et localiser les fonctions consommant le plus de CPU
  - déterminer la fréquence des appels et permettre une classification en fonction du contexte (à partir de quelle fonction la fonction étudiée a-t-elle été appelée ?).

Un outil de profiling ne remplace pas une bonne conception et une implémentation soignée, il les complète.



outils ...

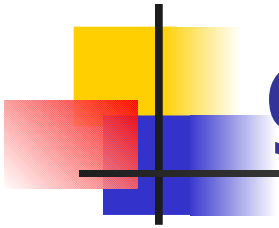
- Nous étudierons ici un outil assez simple et généraliste:
  - gprof de GNU
    http://sourceware.org/binutils/docs/gprof/index.html
  - puis deux variantes à usage spécifique :
    - time
    - strace



gprof

- *gprof* est compatible avec *gcc*.

- ***Utilisation :*** *gprof* se déroule en trois phases:
  - compilation spéciale pour ajouter automatiquement au code des instructions de profiling.
    - Il faut ajouter les options *-pg* à la compilation ET à l'édition de liens.
  - exécution du programme : cela crée un fichier binaire de données, par défaut *gmon.out*
  - exécuter *gprof* pour exploiter les données collectées à l'exécution et obtenir un rapport

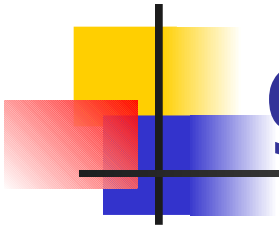


gprof

```
$ gcc -ansi -Wall -c -pg test.c
$ gcc -pg -o test test.o
$ ./test ...
```

- Génère gmon.out dans le répertoire courant

```
$ gprof ./test
....
```



gprof : données obtenues

- `$ gprof mon_executable`
  - (avec gmon.out présent dans le répertoire courant)

- *Flat profile*
  - liste des fonctions avec le nombre d'appels et le temps total passé dedans, en incluant ou pas les fonctions appelées.
  - Permet de repérer très rapidement les quelques fonctions prenant beaucoup de temps.

gprof : ...file

sample coun... as...

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
61.38	5.61	5.61	21	0.27	0.40	compress
31.35	8.47	2.87	9336894	0.00	0.00	...xposant2
4.4...	8.88	0.41	...1	0.41	9.14	comp...all
...	...11	0.22	...	...	...00	tri...
...	...14	0.04	...	...	...4	cre...
...	...14	0.00	...	...	...0	aj...
...	...14	0.00	...	...	...0	longueur_int
...	9.14	0.00	...	...	...0	nb_caracteres
0.00	9.14	0.00	48	0.00	0.00	cat2string
0.00	9.14	0.00	42	0.00	0.00	extract_rep
0.00	9.14	0.00	21	0.00	0.00	add_file
0.00	9.14	0.00	3	0.00	0.00	add_dir

Temps passé dans la fonction

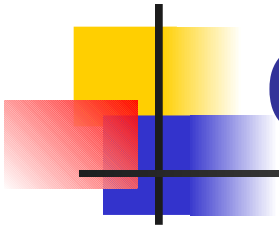
Nombre d'appels

Temps moyen in incluant les descendants

Temps cumulé dans cette fonction et celles au-dessus

Temps passé dans la fonction (sans les descendants)

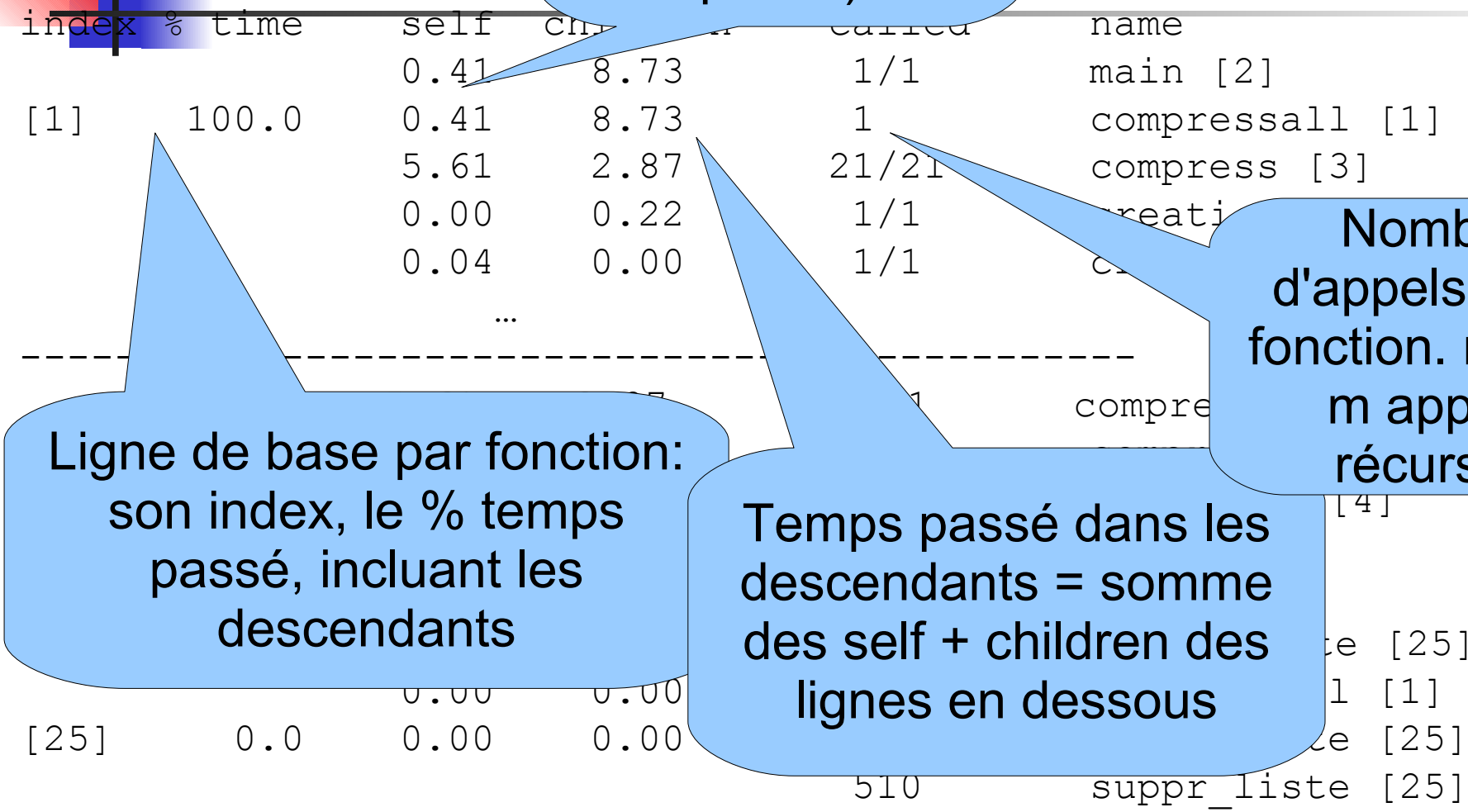
Temps moyen passé dans la fonction



Call graph

- Beaucoup plus de précisions sur le contexte :
  - Contributions par appelant / appelée
- Pour chaque fonction, on a :
  - les fonctions appelantes (les lignes au-dessus de la ligne de référence commençant par [i])
  - proportion du temps pris par la fonction et par ses "enfants" (l'appel des sous-fonctions)
  - nombre d'appels de la fonction, en indiquant aussi les appels récursifs
  - pour chaque appel d' "enfant", contribution de la fonction au nombre total d'appel de cet "enfant".

call grap



Temps passé dans la fonction seule (idem flat profile)

Nombre d'appels de la fonction. n+m si m appels récursifs

Ligne de base par fonction: son index, le % temps passé, incluant les descendants

Temps passé dans les descendants = somme des self + children des lignes en dessous

call graph

```
index % time    self  children    called     name
                0.41    8.73       1/1       main [2]
[1]     100.0   0.41    8.73       1         compressall [1]
                5.61    2.87      21/21      compress [3]
                0       0.22       1/1         ation_arbre [6]
                        0.00       1/1       cr       n_codage [7]
-----                                                    
[3]                     2.87
                        .87
                        .00 9
-----                                                  --
                 ...
                                              suppr_liste [25]
                0.00    0.00                 compressall [1]
[25]      0.0   0.00    0.00       1+510     suppr_liste [25]
                                   510       suppr_liste [25]
```

Temps passé dans la fonction dans les appels de cet appelant

Temps passé dans les descendants quand compressall est appelé par main

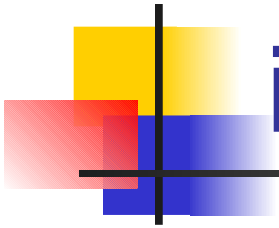
nombre d'appels de cet appelant / nombre d'appels total

call graph

```
index  % time    self  children    called     name
                0.41    8.73       1/1       main [2]
[1]     100.0   0.41    8.73       1         compressall [1]
                5.61    2.87      21/21      compress [3]
                0.00    0.22       1/1       creation_arbre [6]
                0.0     0.00       1/1       creation_codage [7]

-----------------------------------------------
                                  21/21      c
[3]      92.                                 co
                                  /9336894   exp
-----------                      ------------

                                  510        suppr_liste [25]
                                  1/1        compressall [1]
[25]      0.0   0.00    0.00      1+510      suppr_liste [25]
                                  510        suppr_liste [25]
```

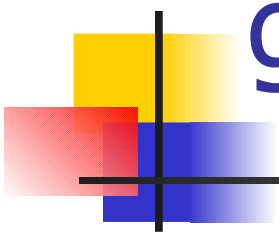
Temps passé dans compress et dans ses enfants quand il est appelé par compressall

Contribution de compressall aux appels de compress



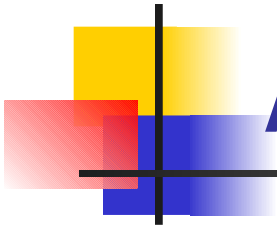
index by function name

- Simple récapitulatif de la liste des fonctions avec leur numéro associé dans le rapport.



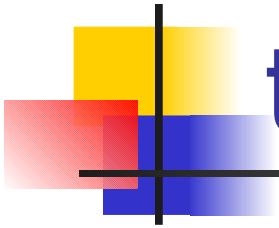
gprof : Statistiques ...

- les nombres d'appels sont comptabilisés => donc précis et répétables
- Les temps sont échantillonnés (défaut 10 ms), donc sujet à des erreur statistiques
  - Importance d'avoir une exécution assez longue
  - Possibilité de sommer des « run » ...
    - Option -s ou --sum



Autres outils …

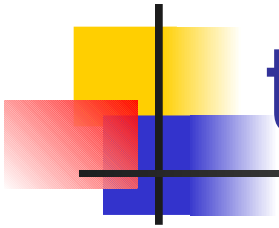
- Commerciaux
  - Quantify (rational => IBM) : Unix/Windows
  - DevPartner (compuWare)



time

- L'outil le plus simple !
  - donne le temps d'exécution d'une commande:
    - le temps écoulé
    - le temps CPU en mode utilisateur
    - le temps CPU en mode noyau (exécution d'appels systèmes)

☞*la version GNU rajoute, si possible, des informations sur l'utilisation de la mémoire (swap) et des I/O.*

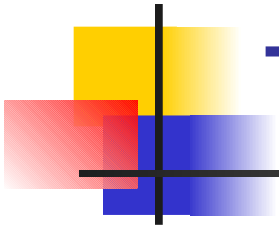


time

- Lancement :

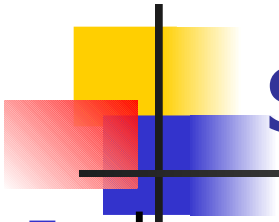
```
$ time [options] cmd [args cmd]
$ time irtar -f test.iar -z -c .
13.54user 0.34system 0:27.75elapsed

$ time "--format=%S;%U" --output=allTimes
   --append cmd …

$ time -v irtar
```

infos aussi complètes que possible : CPU-Mémoire-I/O



Tracer les appels systèmes

- *strace* (*truss* sur d'autres Unix) trace uniquement les appels systèmes et les signaux.
- Lancement:
  - strace [options] commande [arguments de la commande]
- On peut obtenir la liste exhaustive des appels systèmes avec les arguments d'appels et le code retour



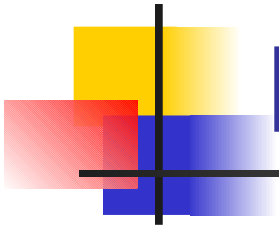
strace : extraits

- strace envoie le résultat sur la sortie erreur

```
execve("../xxx/bin/irtar", ["../xxx/bin/irtar", "-f", "../../../../tmp/kk.igz", "-z", "-c",
   "CORPUS/text-only/"], [/* 66 vars */]) = 0
uname({sys="Linux", node="PF-LINUX-5", ...}) = 0
brk(0)                                  = 0x8050000
open("/etc/ld.so.preload", O_RDONLY)    = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=70548, ...}) = 0
close(3)                                = 0
open("/lib/tls/libc.so.6", O_RDONLY)    = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\220O\1"..., 512) = 512
lstat64("CORPUS/text-only/", {st_mode=S_IFDIR|0777, st_size=4096, ...}) = 0
getcwd("/home/MLV/IRTAR/_PF_", 1024)    = 21
chdir("../../../../tmp")                = 0
lstat64("CORPUS/text-only/geo", {st_mode=S_IFREG|0777, st_size=102400, ...}) = 0
read(4, "Welcome to the Calgary/Canterbur"..., 4096) = 2479
read(4, "", 4096)                       = 0
```

strace : résumé avec -c

% time	seconds	usecs/call	calls	errors	syscall
59.41	0.022979	14	1657		read
31.30	0.012107	23	537		write
5.66	0.002191	46	48	1	open
1.24	0.000481	11	44		munmap
0.66	0.000257	5	48		lstat64
0.57	0.000221	5	43		mmap2
0.45	0.000174	4	47		close
0.35	0.000134	3	47		fstat64
0.09	0.000035	7	5		brk
0.08	0.000032	6	5		old_mmap
0.06	0.000022	6	4		chdir
0.05	0.000020	5	4		getdents64
0.03	0.000011	4	3		getcwd
0.02	0.000007	7	1		set_thread_area
0.02	0.000006	3	2		fcntl64
0.01	0.000004	4	1		uname
100.00	0.038681		2496	1	total



Les performances

- les problématiques sont liées
  - performances
  - sécurisation (redondance, cryptage, …)
  - fiabilité
  - simplicité
- Amélioration des performances / de la sécurité
  - penser aux autres problèmes que peut générer la solution
  - penser aux autres problèmes que peut résoudre la solution

s'adapter au contexte: besoins / moyens

Quelques exemples de C

- Lecture d'1 gros fichier (50Mo)
  - Appel système read 1 byte / call
  - Appel système read 50Mo / call
  - Lib C standard : fread 1 byte / call
  - Lib C standard : fread 50Mo / call

Lecture gros fichier (x 200)

```
static void readByByte()
{
  int i, nb;
  int fd = open(strFileName, O_RDONLY);
  assert(fd >= 0);
  for (i=0; i<sizeof(buf); ++i)
  {
    nb = read(fd, BUF(i), 1);
    assert(nb == 1);
  }
  close(fd);
}
```

Lecture gros fichier (x 1)

```
static void readAll()
{
  int nb;
  int fd = open(strFileName, O_RDONLY);
  assert(fd >= 0);
  nb = read(fd, buf, sizeof(buf));
  close(fd);
}
```

Lecture gros fichier (x 5)

```
static void freadByByte()
{
  int i, nb;
  FILE* f = fopen(strFileName, "r");
  assert(f);
  for (i=0; i<sizeof(buf); ++i)
  {
    nb = fread(BUF(i), 1, 1, f);
    assert(nb == 1);
  }
  fclose(f);
}
```

entrées/sorties bufférisées !

Lecture gros fichier (x 1)

```
static void freadAll()
{
  int nb;
  FILE* f = fopen(strFileName, "r");
  assert(f);
  nb = fread(buf, sizeof(buf), 1, f);
  assert(nb == 1);
  fclose(f);
}
```

Surcoût de la lib standard négligeable !

Parcours de la mémoire (x 1)

- Parcours naturel

```
for (y=0; y<HEIGHT; ++y)
{
  for (x=0; x<WIDTH; ++x)
  {
    n += tab[y][x];
  }
}
```

Parcours de la mémoire (x 40)

- Parcours non naturel

```
for (x=0; x<WIDTH; ++x)
{
  for (y=0; y<HEIGHT; ++y)
  {
    n += tab[y][x];
  }
}
```