INGI1113 - Semaine 3

Christoph Paasch — christoph.paasch@uclouvain.be
Fabien Duchêne — fabien.duchene@uclouvain.be

16 février 2012

Les IPCs

- SysV et POSIX

Les IPCs

- SysV et POSIX
  - Interfaces différentes qui fournissent sémaphores et mémoires partagées (entre autres)
  - Les IPCs SysV sont antérieurs aux IPCs POSIX

Les IPCs

- SysV et POSIX
  - Interfaces différentes qui fournissent sémaphores et mémoires partagées (entre autres)
  - Les IPCs SysV sont antérieurs aux IPCs POSIX
- Problème : les IPCs ne sont pas implémentés partout

Les IPCs

- SysV et POSIX
  - Interfaces différentes qui fournissent sémaphores et mémoires partagées (entre autres)
  - Les IPCs SysV sont antérieurs aux IPCs POSIX
- Problème : les IPCs ne sont pas implémentés partout
  - OK : Linux, FreeBSD, Solaris
  - KO : Mac OS/Darwin (sémaphores et mémoire partagées seulement)

Exemple : Lister et supprimer les IPCs

- SysV

```
fab@Crashtest:~$ ipcs -s
------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0xcbc384f8 6127616    fab        600        1
0x0056a4d5 6684673    fab        660        1
fab@Crashtest:~$ ipcrm -S 0xcbc384f8
```

Exemple : Lister et supprimer les IPCs

- SysV

```
fab@Crashtest:~$ ipcs -s
------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0xcbc384f8 6127616    fab        600        1
0x0056a4d5 6684673    fab        660        1
fab@Crashtest:~$ ipcrm -S 0xcbc384f8
```

- POSIX

```
fab@Crashtest:~$ ls -l /dev/shm
total 3
-rw-r--r-- 1 xxxx etinfo 16 feb 24 21:09 sem.accesBB
-rw-r--r-- 1 xxxx etinfo 16 feb 24 21:09 sem.empty
-rw-r--r-- 1 xxxx etinfo 16 feb 24 21:09 sem.full
fab@Crashtest:~$ rm /dev/shm/xxxx
```

Exemple : Lister et supprimer les IPCs

- SysV

```
fab@Crashtest:~$ ipcs -s
------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0xcbc384f8 6127616    fab        600        1
0x0056a4d5 6684673    fab        660        1
fab@Crashtest:~$ ipcrm -S 0xcbc384f8
```

- POSIX

```
fab@Crashtest:~$ ls -l /dev/shm
total 3
-rw-r--r-- 1 xxxx etinfo 16 feb 24 21:09 sem.accesBB
-rw-r--r-- 1 xxxx etinfo 16 feb 24 21:09 sem.empty
-rw-r--r-- 1 xxxx etinfo 16 feb 24 21:09 sem.full
fab@Crashtest:~$ rm /dev/shm/xxxx
```

- Lorsque vous travaillez sur une machine partagée (Sirius ?) **n'oubliez pas de supprimer vos sémaphores** après avoir travaillé. Regardez `man ipcrm`.

Valgrind : description

Valgrind est un outil de déboggage et de profilage de code permettant notamment de détecter :

- les fuites de mémoire (memory leaks)

Valgrind : description

Valgrind est un outil de déboggage et de profilage de code permettant notamment de détecter :

- les fuites de mémoire (memory leaks)
- les accès incorrects à la mémoire (menants parfois à une erreur de segmentation)

Valgrind : description

Valgrind est un outil de déboggage et de profilage de code permettant notamment de détecter :

- les fuites de mémoire (memory leaks)
- les accès incorrects à la mémoire (menants parfois à une erreur de segmentation)
- l'utilisation d'une zone mémoire non initialisée.

Exemple : détection d'une fuite de mémoire

Soit le code suivant :

```
1  #include <stdlib.h>
2
3  int
4  main(int argc, char * argv[])
5  {
6      char * ptrChars = malloc(6 * sizeof(char));
7      ptrChars[0]= 'H';
8      //free(ptrChars);
9      return 0;
10 }
```

dans lequel l'espace mémoire alloué pour ptrChars n'est jamais libéré.

Exemple : détection d'une fuite de mémoire

Exécution de valgrind avec la directive **–leak-check=yes** afin de détecter les fuites de mémoire.

```
fab@Crashtest:~$ valgrind --leak-check=yes ./memory_leak
==8994== Memcheck, a memory error detector
==8994== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==8994== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==8994== Command: ./memory_leak
==8994==
==8994==
==8994== HEAP SUMMARY:
==8994==     in use at exit: 6 bytes in 1 blocks
==8994==   total heap usage: 1 allocs, 0 frees, 6 bytes allocated
==8994==
==8994== 6 bytes in 1 blocks are definitely lost in loss record 1 of 1
==8994==    at 0x4C244E8: malloc (vg_replace_malloc.c:236)
==8994==    by 0x4004FC: main (test.c:6)
==8994==
==8994== LEAK SUMMARY:
==8994==    definitely lost: 6 bytes in 1 blocks
==8994==    indirectly lost: 0 bytes in 0 blocks
==8994==      possibly lost: 0 bytes in 0 blocks
==8994==    still reachable: 0 bytes in 0 blocks
==8994==         suppressed: 0 bytes in 0 blocks
==8994==
==8994== For counts of detected and suppressed errors, rerun with: -v
==8994== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 4 from 4)
fab@Crashtest:~$
```

Exemple : accès non autorisé à une zone de mémoire

Soit le code suivant :

```
1   #include <stdlib.h>
2   int
3   main(int argc, char * argv[])
4   {
5       char * ptrChars = malloc(6 * sizeof(char));
6       ptrChars[0]= 'H';
7       ptrChars[12]= 'W';
8       free(ptrChars);
9       ptrChars[1]= 'e';
10      return 0;
11  }
```

dans lequel on écrit (notamment) dans une zone de mémoire n'étant plus allouée au programme.

Exemple : accès non autorisé à une zone de mémoire

Exécution de valgrind :

```
fab@Crashtest:~$ valgrind --leak-check=yes ./seg_fault
==9065== Memcheck, a memory error detector
==9065== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==9065== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==9065== Command: ./seg_fault
==9065==
==9065== Invalid write of size 1
==9065==    at 0x400560: main (seg_fault.c:7)
==9065==  Address 0x518b04c is 6 bytes after a block of size 6 alloc'd
==9065==    at 0x4C244E8: malloc (vg_replace_malloc.c:236)
==9065==    by 0x40054C: main (seg_fault.c:5)
==9065==
==9065== Invalid write of size 1
==9065==    at 0x400577: main (seg_fault.c:9)
==9065==  Address 0x518b041 is 1 bytes inside a block of size 6 free'd
==9065==    at 0x4C240FD: free (vg_replace_malloc.c:366)
==9065==    by 0x40056E: main (seg_fault.c:8)
==9065==
==9065==
==9065== HEAP SUMMARY:
==9065==     in use at exit: 0 bytes in 0 blocks
==9065==   total heap usage: 1 allocs, 1 frees, 6 bytes allocated
==9065==
==9065== All heap blocks were freed -- no leaks are possible
==9065==
==9065== For counts of detected and suppressed errors, rerun with: -v
==9065== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 4 from 4)
fab@Crashtest:~$
```

Exemple : appel système avec des paramètres invalides

Soit le code suivant :

```
1  #include <stdlib.h>
2  #include <unistd.h>
3  int main()
4  {
5          int *p;
6          p = malloc(10);
7          read(0, p, 100);
8          free(p);
9          return 0;
10 }
```

dans lequel on tente d'écrire 100 bytes dans un espace mémoire de 10 bytes.

Exemple : appel système avec des paramètres invalides

Exécution de valgrind :

```
fab@Crashtest:~$ valgrind --leak-check=yes ./syscall
==9206== Memcheck, a memory error detector
==9206== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==9206== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==9206== Command: ./syscall
==9206==
==9206== Syscall param read(buf) points to unaddressable byte(s)
==9206==    at 0x4EEC630: __read_nocancel (syscall-template.S:82)
==9206==    by 0x4005AF: main (syscall.c:7)
==9206==  Address 0x518b04a is 0 bytes after a block of size 10 alloc'd
==9206==    at 0x4C244E8: malloc (vg_replace_malloc.c:236)
==9206==    by 0x400595: main (syscall.c:6)
==9206==
==9206== HEAP SUMMARY:
==9206==     in use at exit: 0 bytes in 0 blocks
==9206==   total heap usage: 1 allocs, 1 frees, 10 bytes allocated
==9206==
==9206== All heap blocks were freed -- no leaks are possible
==9206==
==9206== For counts of detected and suppressed errors, rerun with: -v
==9206== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 4 from 4)
fab@Crashtest:~$
```

Exemple : free invalide

Soit le code suivant :

```
1   #include <stdlib.h>
2   int main()
3   {
4           int *p, i;
5           p = malloc(10*sizeof(int));
6           for(i = 0;i < 10;i++)
7                   p[i] = i;
8           free(p);
9           free(p);
10          return 0;
11  }
```

dans lequel on tente de libérer une zone mémoire ayant déjà été libérée.

Exemple : free invalide

Exécution de valgrind :

```
fab@Crashtest:~$ valgrind --leak-check=yes ./double_free
==9254== Memcheck, a memory error detector
==9254== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==9254== Using Valgrind-3.6.0.SVN-Debian and LibVEX; rerun with -h for copyright info
==9254== Command: ./double_free
==9254==
==9254== Invalid free() / delete / delete[]
==9254==    at 0x4C240FD: free (vg_replace_malloc.c:366)
==9254==    by 0x400586: main (double_free.c:9)
==9254==  Address 0x518b040 is 0 bytes inside a block of size 40 free'd
==9254==    at 0x4C240FD: free (vg_replace_malloc.c:366)
==9254==    by 0x40057A: main (double_free.c:8)
==9254==
==9254==
==9254== HEAP SUMMARY:
==9254==     in use at exit: 0 bytes in 0 blocks
==9254==   total heap usage: 1 allocs, 2 frees, 40 bytes allocated
==9254==
==9254== All heap blocks were freed -- no leaks are possible
==9254==
==9254== For counts of detected and suppressed errors, rerun with: -v
==9254== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 4 from 4)
fab@Crashtest:~$
```

OProfile : description

- **OProfile** est un outil Linux permettant de faire du profilage de code.
- Surtout utilisé pour le profilage du kernel Linux
- Capable d'afficher des informations des "events" (propre au type de CPU) `ophelp`

```
1  christoph@linux:~$ ophelp
2  oprofile: available events for CPU type "Intel Architectural Perfmon"
3
4  See Intel 64 and IA-32 Architectures Software Developer's Manual
5  Volume 3B (Document 253669) Chapter 18 for architectural perfmon events
6  This is a limited set of fallback events because oprofile doesn't know your CPU
7  CPU_CLK_UNHALTED: (counter: all)
8          Clock cycles when not halted (min count: 6000)
9  INST_RETIRED: (counter: all)
10         number of instructions retired (min count: 6000)
11 LLC_MISSES: (counter: all)
12         Last level cache demand requests from this core that missed the LLC (min count: 6000)
13         Unit masks (default 0x41)
14         ----------
15         0x41: No unit mask
16 LLC_REFS: (counter: all)
17         Last level cache demand requests from this core (min count: 6000)
18         Unit masks (default 0x4f)
19         ----------
20         0x4f: No unit mask
21 BR_INST_RETIRED: (counter: all)
22         number of branch instructions retired (min count: 500)
23 BR_MISS_PRED_RETIRED: (counter: all)
24         number of mispredicted branches retired (precise) (min count: 500)
```

OProfile : exemple d'utilisation

- Charger le module :

```
1 christoph@linux:~$ modprobe oprofile
```

- Préparer oprofile :

```
1 christoph@linux:~$ opcontrol --no-vmlinux --image=[path to my binary]
```

Vous pouvez spécifier des events (provénant de `ophelp`) avec `--event=...`

- Démarrez votre programme, puis lancez le profiling :

```
1 christoph@linux:~$ opcontrol --start
```

OProfile : exemple d'utilisation

- Arrètez : `opcontrol --stop`

```
1  christoph@linux:~$ opcontrol --stop
```

- Récuperer un rapport :

```
1  christoph@linux:~$ opreport -l [path to my binary]
2  Overflow stats not available
3  CPU: Intel Architectural Perfmon, speed 2667 MHz (estimated)
4  Counted CPU_CLK_UNHALTED events (Clock cycles when not halted) with a unit mask of 0x00 (
       No unit mask) count 90000
5  samples  %        symbol name
6  3118     99.6803  process_produce
7  7         0.2238  matrix_queue_get
8  2         0.0639  process_consume
9  1         0.0320  matrix_queue_add
```

- Avant de faire un nouveau profiling :

```
1  christoph@linux:~$ opcontrol --reset
2  christoph@linux:~$ opcontrol --shutdown
```

Ressources :

- `man oprofile/opcontrol/opreport`
- `http://oprofile.sourceforge.net/`
- `http://www.centos.org/docs/5/html/Deployment_Guide-en-US/ch-oprofile.html`
- `http://www.ibm.com/developerworks/linux/library/l-oprof/index.html`
- `http://www.google.com`