INGI1113 - Semaine 2

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

9 février 2012

Rappel sur les threads

- Différence entre threads/processus ?
- Producteur/consommateur ?

Rappel sur les threads

- Différence entre threads/processus ?
  - Changement de contexte plus rapide dans les threads
  - Espace d'adressage partagé entre les threads.
  - Les processus offrent une isolation par rapport à des erreurs.
- Producteur/consommateur ?

Rappel sur les threads

- Différence entre threads/processus ?
  - Changement de contexte plus rapide dans les threads
  - Espace d'adressage partagé entre les threads.
  - Les processus offrent une isolation par rapport à des erreurs.
- Producteur/consommateur ?
  - Utiliser un buffer partagé
  - Utiliser deux sémaphores full/empty

Exemple : Producteur/Consommateur

Deux sémaphores - *full* et *empty* (initialisée à N - taille du buffer)

```
1   Producteur:
2     /* Est-ce qu'il y a de l'espace pour un element en plus? */
3     semaphore_down(empty)
4     mutex_lock(mutex)
5       /* Ajoute l'element */
6     mutex_unlock(mutex)
7     semaphore_up(full)
8     /* Signaler aux consommateurs qu'il y a un element en plus */
9
10  Consommateur
11    /* Est-ce qu'il y a un element a consommer? */
12    semaphore_down(full)
13    mutex_lock(mutex)
14      /* Consomme l'element */
15    mutex_unlock(mutex)
16    semaphore_up(empty)
```

phtread.h

- Une librairie pour les threads et mutex.

```
/** Création d'un thread */
int pthread_create(...)

/** Terminer un thread */
void pthread_exit(void *retval);

/** Attendre un thread */
int pthread_join(pthread_t thread, void **retval);

/** Les mutex */
pthread_mutex_*
...
```

- man pthread.h
- man pthread_create/join/exit...
- man pthread_mutex_*

semaphore.h

- Une librairie pour les sémaphores

```
/** Création d'un sémaphore */
sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value);

/** sem_up */
int    sem_post(sem_t *);

/** sem_down */
int    sem_wait(sem_t *);

/** Fermer le sémaphore*/
int    sem_close(sem_t *);
...
```

- `man semaphore.h`
- `man sem_open/close/...`

Débuggage

- Erreurs fréquentes en C :
  - Accès en dehors des bornes d'un tableau
  - Utilisation d'une variable non-initialisée
  - Utilisation d'un pointeur non-initialisé
- Techniques de débugging :
  - Utilisation des printf(...) pour voir le contenu des variables.
  - Utilisation d'un débogueur (gdb).

GDB sous Linux

- Déterminer plus précisement l'endroit de l'erreur
- Afficher la stack-trace
- Afficher le contenu des variables
- Mettre des "breakpoints"
- Exécution pas-à-pas.

Préparation du programme

```
gcc -g -o main main.c
```

-g ajoute à l'exécutable les symboles de déboguage nécessaires pour pouvoir l'utiliser avec gdb.

Utilisation de gdb

```
1   #include <stdio.h>
2
3   void calc (int *tab, int i, int num)
4   {
5           tab[i] = num / i;
6   }
7
8   void iter (int *tab, int num)
9   {
10          int i;
11          printf("Iterating\n");
12          for (i = 0; i <= 10; i++) {
13                  calc(tab, i, num);
14          }
15  }
16
17  void main (void)
18  {
19          int *tab = NULL;
20          int num = 20;
21          int i;
22
23          iter(tab, num);
24
25          for (i = 0; i < 10; i++) {
26                  printf("%d ", tab[i]);
27          }
28          printf("\n");
29  }
```

Utilisation de gdb

- Exécuter gdb : `gdb main`

```
cpaasch@cpaasch:~$ gdb main
GNU gdb (GDB) 7.2-ubuntu
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from ~/main...done.
(gdb)
```

Utilisation de gdb

- Terminer gdb : `quit`
- Exécution du programme : `run [args]`

```
(gdb) run
Starting program: ~/main

Program received signal SIGFPE, Arithmetic exception.
0x000000000040056a in calc (tab=0x0, i=0, num=20) at main.c:5
5               tab[i] = num / i;
(gdb)
```

- Affichage de la stacktrace : `backtrace`

```
(gdb) backtrace
#0  calc (tab=0x0, i=0, num=20) at main.c:5
#1  0x000000000040059d in iter (tab=0x0, num=20) at main.c:12
#2  0x00000000004005d1 in main () at main.c:22
(gdb)
```

- Arrèter l'exécution du programme : `kill`

```
(gdb) kill
Kill the program being debugged? (y or n) y
(gdb)
```

Utilisation de gdb

- Mise d'un breakpoint :
  - `break [function]`
  - `break [filename:linenum]`
  - `break [*address]`
  - `...`

```
(gdb) break iter
Breakpoint 2 at 0x400580: file main.c, line 11.
(gdb)
```

- Redémarrer le programme :

```
(gdb) run
Starting program: ~/main

Breakpoint 1, iter (tab=0x0, num=20) at main.c:11
11              printf("Iterating\n");
(gdb)
```

Utilisation de gdb

- Se promener dans l'exécution :
  - `step` - Continue l'exécution jusqu'à la prochaine ligne du code source
  - `next` - Continue l'exécution, mais ne s'arrête pas dans les appels de fonctions
  - `continue` - Continue l'exécution jusqu'à la fin du programme

```
(gdb) run
Starting program: ~/main

Breakpoint 1, iter (tab=0x0, num=20) at main.c:11
11              printf("Iterating\n");
(gdb) next ======= On n'entre pas dans le code de printf() =======
Iterating
12              for (i = 0; i <= 10; i++) {
(gdb) step
13                      calc(tab, i, num);
(gdb) step ======= On entre dans le code de calc() =======
calc (tab=0x0, i=0, num=20) at main.c:5
5               tab[i] = num / i;
(gdb)
```

Utilisation de gdb

- Afficher le contenu d'une variable :
  - `p/x` - hex
  - `p/d` - signed integer
  - `p/f` - floating point
  - `p/c` - character
  - `...`

```
(gdb) p/d num
$3 = 20
(gdb) p/d i
$4 = 0
```

- Changer la valeur d'une variable : `set var [var] = [value]`

```
(gdb) set var i = 1
(gdb) p/d i
$5 = 1
(gdb)
```

Utilisation de gdb

- Continuer l'exécution : `continue`

```
(gdb) continue
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x00000000004005ad in calc (tab=0x0, i=1, num=20) at main.c:5
5               tab[i] = num / i;
(gdb)
```

- `tab=0x0` — tab n'a pas été initialisée...

```
void main (void)
{
        int *tab = NULL; <========== Pointeur non-initialisé
        int num = 20;
        int i;
```

GDB et les threads

- Notifie lors de la création de nouveaux threads
- `info threads` afficher de l'info sur les threads
- `thread [thread_no]` switcher entre les threads
- `thread apply [thread_no] [all] args` envoyer la commande *all* vers le thread *thread_no*

Plus d'infos : http ://sourceware.org/gdb/onlinedocs/gdb/Threads.html

GDB-Documentation

- http ://www.gnu.org/software/gdb/
- http ://www.yolinux.com/TUTORIALS/GDB-Commands.html
- **http ://www.google.com** :)