Sistemas Operativos

Universidad Complutense de Madrid

2020-2021

Tutorial: Depuración de programas multihilo

GDB

Juan Carlos Sáez

Introducción

Objetivos

- Familiarizarse con los comandos básicos del depurador GDB para la depuración de programas multihilo
- Aprender a detectar puntos del programa donde se bloquean los hilos

Aspectos básicos de depuración con hilos

Depuración programa multihilo

- Cuando el depurador toma el control de un programa multihilo se paran todos los hilos del proceso
  - En este punto es posible inspeccionar el estado de cada hilo del programa y conocer la traza de funciones que le han llevado a la situación actual

Aspectos básicos de depuración con hilos (II)

Depuración programa multihilo

- El depurador GDB toma el control del proceso multihilo en 3 situaciones:
  1. Cuando el programa termina debido a un error crítico
     - Ejemplo: violación de segmento
  2. Cuando cualquiera de los hilos del programa alcance un *breakpoint*
  3. Cuando el usuario teclea CTRL+C desde el terminal donde se ejecuta el programa que está siendo depurado:
     - En este caso los hilos se paran en el punto de la ejecución donde estén en ese momento
- En este tutorial usaremos CTRL+C para tomar control del programa depurado y detectar los puntos del código donde se bloquea cada hilo

Comandos básicos de depuración con hilos

- `info threads`: Imprime un listado con información sobre cada uno de los hilos de la aplicación.
  - GDB asigna a cada hilo un identificador numérico para permitir la ejecución de comandos sobre hilos específicos.
  - El hilo que aparece marcado con "*" en el listado es el hilo actual sobre el cual se ejecutan los comandos de GDB por defecto

Ejemplo: Programa con 3 hilos - (Hilo 1 es el 'actual')

```
(gdb) info threads
  Id   Target Id         Frame
  1    Thread 0x7ffff7fd3700 (LWP 63321) "SumBug" 0x00007ffff7bc46dd in pthread_join (threadid=140737345865472
  2    Thread 0x7ffff781c700 (LWP 63325) "SumBug" __lll_lock_wait () at ../sysdeps/unix/sysv/linux/x86_64/lowl
* 3    Thread 0x7ffff701b700 (LWP 63326) "SumBug" __lll_lock_wait () at ../sysdeps/unix/sysv/linux/x86_64/lowl
```

- `thread` *`id_thread`*: Establece como hilo actual el hilo cuyo identificador es id_thread
  - Nos interesará cambiar el hilo actual para inspeccionar las propiedades de un hilo mediante comandos como `backtrace` or print'

Comandos básicos de depuración con hilos

- `backtrace`: Muestra la jerarquía de llamadas a funciones invocadas por el hilo actual en el punto actual de su ejecución.
  - Este comando realiza una inspección de la pila de llamadas del hilo.
    - El nivel #0 es la cima de la pila (punto de ejecución actual)
    - El #1 es el nivel inmediatamente anterior (desde donde se invocó la función actual)
  - Alternativamente, el comando puede invocarse tecleando `bt`

Ejemplo de backtrace - 2 niveles de llamadas

```
(gdb) backtrace
#0  0x00007ffff7bc46dd in pthread_join (threadid=140737345865472, thread_return=0x0) at pthread_join.c:90
#1  0x0000555555554906 in main () at partial_sum_bug.c:36
```

Comandos básicos de depuración con hilos

- `frame` *n*: Permite seleccionar el nivel *n* de la pila de llamadas que nos interesa inspeccionar
  - El número *n* se puede extraer a partir de la salida de `backtrace` (0 ó 1 en el ejemplo de arriba).
  - Al cambiar el nivel con `frame` se muestra la línea de código actual de ese nivel

Ejemplo (I)

- Ejecutaremos el programa SumBug
  - Variante del programa *partial_sum2.c*
  - Se crean 2 hilos para hacer la suma de los números del 1 al 10000 en paralelo
  - Se usa un mutex para proteger la actualización de la variable compartida `total_sum`

Terminal

```
osuser@debian:~/SumBug$ make
gcc -Wall -g -pthread -o SumBug partial_sum_bug.c
osuser@debian:~/SumBug$ ./SumBug
^C
osuser@debian:~/SumBug$
```

- El programa se queda bloqueado al ejecutarlo y tenemos que cancelar su ejecución con CTRL+C
- Usaremos el depurador GDB para:
  1. *Averiguar dónde está bloqueado cada hilo*
  2. *Detectar el error que lleva a ese bloqueo*

Ejemplo (II)

- Arrancamos el depurador (`$ gdb -tui <ejecutable>`)
- Cuando GDB muestre el *prompt*, "(gdb)", lanzamos el programa con el comando `run`
  - Uso: `run <argumentos_del_programa>`
  - En este caso el programa no acepta argumentos

Terminal

```
osuser@debian:~/SumBug$ gdb -tui ./SumBug
...
GNU gdb (Debian 7.12-6) 7.12.0.20161007-git
Copyright (C) 2016 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.
...
(gdb) run
Starting program: ./SumBug
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[New Thread 0x7ffff781c700 (LWP 63325)]
```

Ejemplo (III)

- Para que el depurador tome control del programa que está bloqueado, teclearemos CTRL+C
- Cuando GDB muestre el prompt obtendremos un listado de los *threads* del programa usando `info threads`

Terminal

```
^C
Thread 1 "SumBug" received signal SIGINT, Interrupt.
0x00007ffff7bc46dd in pthread_join (threadid=140737345865472, thread_return=0x0) at pthread_join
90  pthread_join.c: No existe el fichero o el directorio.
(gdb) info threads
  Id   Target Id         Frame
* 1    Thread 0x7ffff7fd3700 (LWP 63321) "SumBug" 0x00007ffff7bc46dd in pthread_join (threadid=1
  2    Thread 0x7ffff781c700 (LWP 63325) "SumBug" __lll_lock_wait () at ../sysdeps/unix/sysv/lin
  3    Thread 0x7ffff701b700 (LWP 63326) "SumBug" __lll_lock_wait () at ../sysdeps/unix/sysv/lin
(gdb)
```

- El programa consta de 3 hilos y el hilo actual tiene ID=1
- A continuación, averiguaremos dónde esta bloqueado cada hilo

Ejemplo (IV)

- Seleccionamos el hilo 1 y mostramos su pila de llamadas con `backtrace`

Terminal

```
(gdb) thread 1
[Switching to thread 1 (Thread 0x7ffff7fd3700 (LWP 63321))]
#0  0x00007ffff7bc46dd in pthread_join (threadid=140737345865472, thread_return=0x0) at pthread_
90  in pthread_join.c
(gdb) backtrace
#0  0x00007ffff7bc46dd in pthread_join (threadid=140737345865472, thread_return=0x0) at pthread_
#1  0x0000555555554906 in main () at partial_sum_bug.c:36
(gdb) frame 1
#1  0x0000555555554906 in main () at partial_sum_bug.c:36
36    pthread_join(th1, NULL);
```

- El hilo con ID=1 es el hilo main del programa, que ejecuta la función `main()`, como revela el listado de llamadas:
  1. `main()`
  2. `pthread_join()`
- El hilo está bloqueado en `pthread_join()` (invocada en la línea 36) esperando a que finalice otro hilo del programa

Ejemplo (V)

- Seleccionamos el hilo 2 y mostramos su pila de llamadas
  - Seleccionamos el frame número 2 para visualizar la última llamada que se produce dentro de nuestro código (el resto pertenece a *libpthread*)

Terminal

```
(gdb) thread 2
[Switching to thread 2 (Thread 0x7ffff781c700 (LWP 63325))]
#0  __lll_lock_wait () at ../sysdeps/unix/sysv/linux/x86_64/lowlevellock.S:135
135 ../sysdeps/unix/sysv/linux/x86_64/lowlevellock.S: No existe el fichero o el directorio.
(gdb) backtrace
#0  __lll_lock_wait () at ../sysdeps/unix/sysv/linux/x86_64/lowlevellock.S:135
#1  0x00007ffff7bc5bb5 in __GI___pthread_mutex_lock (mutex=0x555555755080 <mutex>) at ../nptl/pthread_mutex_lo
#2  0x0000555555554889 in partial_sum (arg=0x7fffffffe458) at partial_sum_bug.c:21
#3  0x00007ffff7bc34a4 in start_thread (arg=0x7ffff781c700) at pthread_create.c:456
#4  0x00007ffff7905d0f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:97
(gdb) frame 2
#2  0x0000555555554889 in partial_sum (arg=0x7fffffffe458) at partial_sum_bug.c:21
21      pthread_mutex_lock(&mutex);
```

- El hilo con ID=2 ha sido creado vía `pthread_create()`
  - Esto lo podemos saber porque el hilo invoca `start_thread()`
- El hilo 2 está bloqueado en la línea 21 del programa, invocando `pthread_mutex_lock`

Ejemplo (VI)

- Seleccionamos el hilo 3 y mostramos su pila de llamadas
  - Seleccionamos el frame número 2 para visualizar la última llamada que se produce dentro de nuestro código (el resto pertenece a *libpthread*)

Terminal

```
(gdb) thread 3
[Switching to thread 3 (Thread 0x7ffff701b700 (LWP 63326))]
#0  __lll_lock_wait () at ../sysdeps/unix/sysv/linux/x86_64/lowlevellock.S:135
135 ../sysdeps/unix/sysv/linux/x86_64/lowlevellock.S: No existe el fichero o el directorio.
(gdb) backtrace
#0  __lll_lock_wait () at ../sysdeps/unix/sysv/linux/x86_64/lowlevellock.S:135
#1  0x00007ffff7bc5bb5 in __GI___pthread_mutex_lock (mutex=0x555555755080 <mutex>) at ../nptl/pthread_mutex_lo
#2  0x000055555555486c in partial_sum (arg=0x7fffffffe450) at partial_sum_bug.c:19
#3  0x00007ffff7bc34a4 in start_thread (arg=0x7ffff701b700) at pthread_create.c:456
#4  0x00007ffff7905d0f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:97
(gdb) frame 2
#2  0x000055555555486c in partial_sum (arg=0x7fffffffe450) at partial_sum_bug.c:19
19      pthread_mutex_lock(&mutex);
```

- El hilo con ID=3 también es un hilo creado vía `pthread_create()`
- El hilo 3 está bloqueado en la línea 19 del programa, invocando `mutex_lock()`

Ejemplo (VII)

Información extraída con el depurador

- Hilo1 está esperando en `pthread_join()` a que acabe uno de los otros dos hilos
- Hilo2 está intentando adquirir un mutex en la línea 21 del programa (`partial_sum()`)
- Hilo3 está intentando adquirir el mismo mutex que Hilo2 en la línea 19 del programa (`partial_sum()`)

Conclusiones

- Alguno de los hilos del programa es el propietario del mutex, pero no sabemos cuál
- El problema está en la implementación de la función `partial_sum()`, que ejecutan los hilos 2 y 3

Ejemplo (VIII)

```
1  #include <pthread.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int total_sum = 0;
6
7  pthread_mutex_t mutex;
8
9
10 void * partial_sum(void * arg) {
11   int j = 0;
12   int tmp=0;
13   int ni=((int*)arg)[0];
14   int nf=((int*)arg)[1];
15
16   for (j = ni; j <= nf; j++)
17     tmp+=j;
18
19    pthread_mutex_lock(&mutex);
20    total_sum += tmp;
21    pthread_mutex_lock(&mutex);
22
23    pthread_exit(0);
24 }
25 ...
```

- El hilo 2 ha adquirido el mutex (en línea 19) pero intenta re-adquirirlo después (línea 21) y por eso se bloquea
  - En realidad el mutex se debería liberar al salir de la sección crítica (línea 21)
- El hilo 3 se bloquea en la línea 19 porque el hilo 2 es el propietario del mutex
- **Solución**: reemplazar línea 21 por `pthread_mutex_unlock(&mutex);`