tags: `homework`

SP HW1

410410060 林柔均

1. 設定中斷點，將中斷點設定在 main 函數

1. `gdb ./rdtse` 進入 gdb

```
 ~/De/h/Sy/hw1 ❯ gcc rdtsc.c -g -o rdtsc                                    00:02:24

 ~/De/h/Sy/hw1 ❯ gdb ./rdtsc                                                00:02:46
GNU gdb (GDB) 12.1
Copyright (C) 2022 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./rdtsc...
(gdb)
```

2. `b main` (breakpoint main) 設定中斷點在 main 函式
3. `r` (run) 執行

```
(gdb) b main
Breakpoint 1 at 0x11de: file rdtsc.c, line 28.
(gdb) r
Starting program: /home/rogewood/Desktop/homework/SystemProgramming/hw1/rdtsc

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.archlinux.org
Enable debuginfod for this session? (y or [n])
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0x7fffffffdc98) at rdtsc.c:28
28      {
(gdb)
```

2. 單步執行，遇到函數不會進入

1. `b 36` 設定中斷點在第36行(第一個函式)
2. `c` (continue) 持續執行直到遇到中斷點

3. `n` (next) 單步執行，遇到函式不進入

```
(gdb) b 36
Breakpoint 2 at 0x555555555212: file rdtsc.c, line 36.
(gdb) c
Continuing.
這個程式是量測一個指令執行的時間，但CPU可同時執行數十個指令
因此這些量測方法比較適合量測大範圍的程式碼


Breakpoint 2, main (argc=1, argv=0x7fffffffdc98) at rdtsc.c:36
36          cycles1 = rdtscp();
(gdb) n
37          tmp++;
(gdb)
```

3. 單步執行，遇到函數會進入

1. `b 38` 設定中斷點在第38行(第一個函式)
2. `c` (continue) 持續執行直到遇到中斷點
3. `s` (step) 單步執行，遇到函式進入函式

```
(gdb) b 38
Breakpoint 3 at 0x55555555521f: file rdtsc.c, line 38.
(gdb) c
Continuing.

Breakpoint 3, main (argc=1, argv=0x7fffffffdc98) at rdtsc.c:38
38          cycles2 = rdtscp();
(gdb) s
rdtscp () at rdtsc.c:19
19          __asm__ __volatile__("rdtscp":"=a"(lo), "=d"(hi));
```

4. 列印變數的值

- `p tmp` (print) 印出變數 tmp 的值

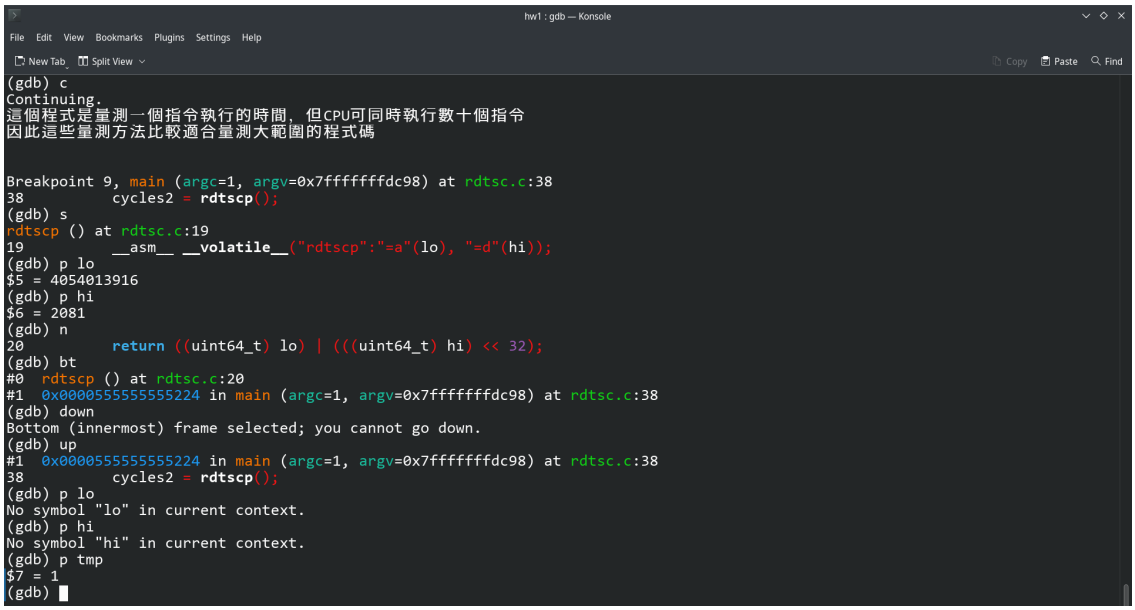
```
(gdb) c
Continuing.
這個程式是量測一個指令執行的時間，但CPU可同時執行數十個指令
因此這些量測方法比較適合量測大範圍的程式碼


Breakpoint 2, main (argc=1, argv=0x7fffffffdc98) at rdtsc.c:36
36          cycles1 = rdtscp();
(gdb) n
37          tmp++;
(gdb) p tmp
$1 = 0
(gdb)
```

5. 使用 bt、up、down，印出 caller 和 callee 各自的變數

- `p lo`、`p hi` 印出變數的值

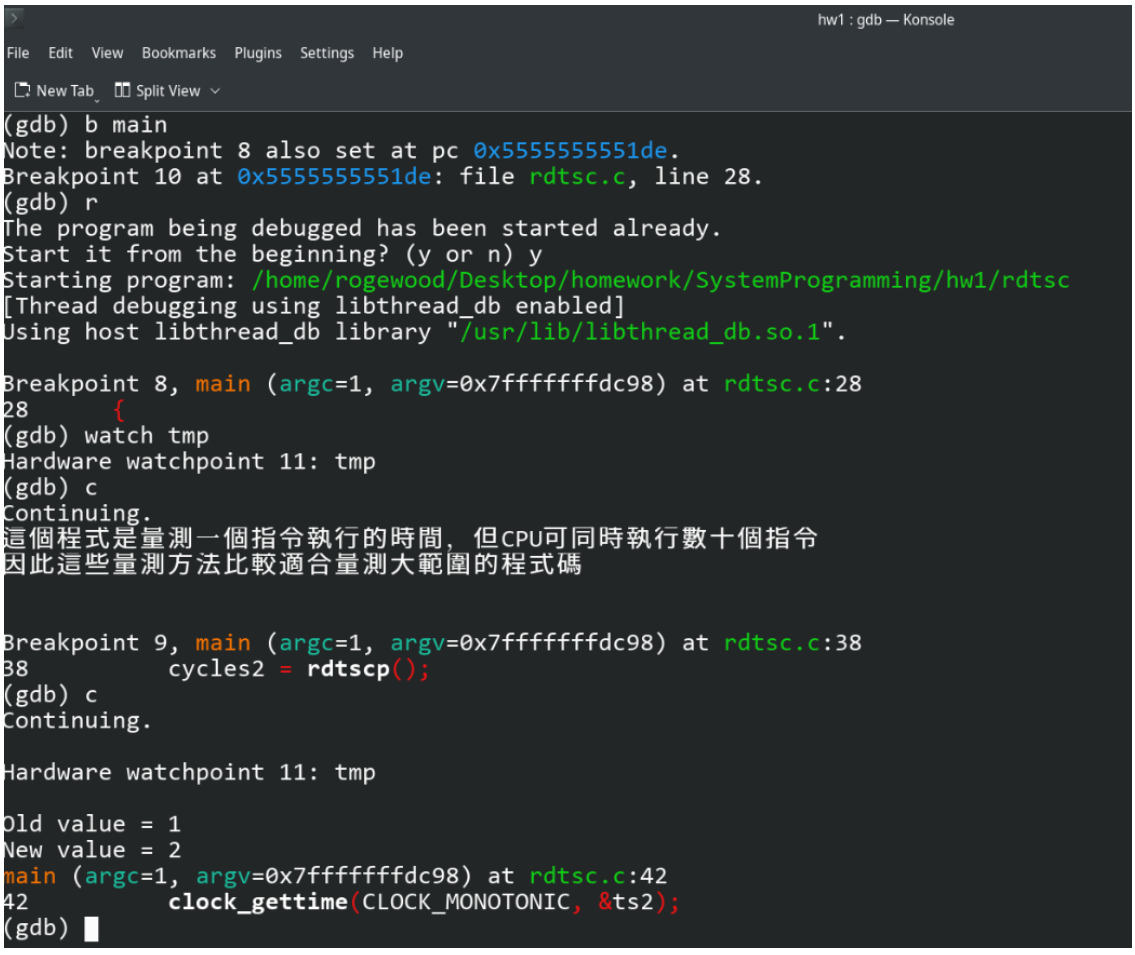
1. `bt` (backtrace) 進入函式的堆疊追蹤
2. `down` 往下層(函式裡面)
3. `up` 往上層(函式外面)

4. 印出變數的值



6. 使用 watch 查看變數被修改的情況

- `watch tmp` 監控變數 tmp 的更動



7. 修改程式碼，故意存取錯誤的記憶體

- `vim rdtsc.c` 製造錯誤的存取記憶體

```
27 int main(int argc, char **argv)
28 {
29     int tmp=0;
30     uint64_t cycles1, cycles2;
31     struct timespec ts1, ts2;
32     int *ptr;
33     printf("%d", *ptr);
34     printf("這個程式是量測一個指令執行的時
35     printf("因此這些量測方法比較適合量測
```

重新編譯執行 gdb 過後，會發現 ptr 為初始化，並出現 `Segmentation fault`