PC 只是一個 Program Counter, 無法告訴你問題發生時, 執行到哪一個 function. 如果你能跑 gdb 的話, 只要問題發生時, 使用 bt (backtrace) 這個命令, 即可知道問題發生的地方在哪. 但是, 你所描述的這個問題, 只有 ARM 系列的 CPU 及 2.6.x 的 kernel 才會看得到. 通常在 embedded device 上, 要用 gdb 來 remote debug 也不是一件簡單的事.

在下提供一個方法, 可以不需要 gdb, 使程式發生問題時印出 backtrace 的內容, 配合 addr2line 找出問題發生的地方(檔案,行數,function). 但在這之前, 可以先試著使用 gcc 的檢查功能,

1. 編譯時, 增加 -Wcast-align -Wpadded -Wpacked
2. 修正所有編譯時出現的 Warning

如果程式碼不小, 那這可能會需要相當多的時間修正, 及重新編譯, 有些地方也可能修了之後, 打壞了原本的架構, 而且, 這功能只能告訴你, "這樣的程式碼, 有可能會發生這個問題", 不保證能指出真正發生問題的地方.

如果上述這個方法仍不能解決你的問題, 那麼, 後述的這個方法或許可以試試看. (後面的內容十分冗長)

[原理概述]

我們要在程式有問題的地方, 讓它中斷, 然後印出 backtrace 來讓 addr2line 反查. 因此, 我們分成以下三個步驟說明之.

1. 用 signal 讓程式在有問題的地方中斷
2. 印出 backtrace
3. 使用 addr2line 來解析其內容

[實作方法]

1. 用 signal 讓程式在有問題的地方中斷:
   在 kernel 的文件 -- Documentation/arm/mem_align 裡有提到, 當有 memory alignment 的問題發生時, kernel 會做出一些處置, 像你所看到的訊息, 即是 kernel 印出 warning, 這個行為是可以透過 /proc 變更的. 以下的指令可以檢視目前的設定,

   ```
   # cat /proc/cpu/alignment
   User:           0
   System:         577
   ```

```
Skipped:        0
Half:           9076
Word:           4479
Multi:          0
User faults:    4 (signal)  <----- 此為目前設定
```

關於 User faults, 有以下 5 種設定,

```
  0 - ignore (預設值)
  1 - warn
  2 - fixup
  3 - fixup+warn
  4 - signal
  5 - signal+warn (我們需要這個)
```

我們需要將行為模式設成 4 或 5, 理由是 4 和 5 都會在發生問題之時送出 SIGBUS, 如果程式裡沒有 signal handler, process 就會被 kill 掉. 我們要利用這個 signal 來中斷有問題的地方, 並且印出 backtrace. 因此我們先更改模式為 5,

```
# echo 5 > /proc/cpu/alignment
```

2. 印出 backtrace
   這個部份, 需要利用 #include <execinfo.h>; 裡的 backtrace(). 如果 source code 分成很多 .c 或 .cpp 的話, 請找出 main() 所在的那個檔案來增加以下的 code,

```
-----------------------------------------------------------------
#include <execinfo.h>;
#include <signal.h>;
#include <stdlib.h>;

/* 此為 signal handler */
static void catch_sig(int sig) {
  void *trace[128];
  int n = backtrace(trace, sizeof(trace) / sizeof(trace[0]));
  backtrace_symbols_fd(trace, n, 1);
  exit(0);
}
```

```
/* 此函式指定 SIGBUS 的 handler 為 catch_sig() */
static void set_signals(void) {
  struct sigaction act;
  sigemptyset(&amp;act.sa_mask);
  act.sa_flags = 0;
  act.sa_handler = catch_sig;
  sigaction(SIGBUS, &amp;act, NULL);
}

/* 在你的 main() 的前頭, 呼叫 set_signals() */
int main() {
  set_signals();
      :
      :
}
```

編譯時, 務必增加 -g -rdynamic 選項.

例: gcc -o prog -g -rdynamic prog.c

重新編譯及執行後, 若發生 Alignment trap, kernel 會送給這個 process 一個 SIGBUS 的 signal, 而我們在 set_signal() 裡, 設定了收到 SIGBUS 時要執行 catch_sig(), 而在 catch_sig() 裡, 我們利用 backtrace() 來取得 stack frame 的位址之後, 用 backtrace_symbols_fd() 來印出比較看得懂的資訊 (但還是得配合 addr2line 來解讀), 印出的內容看起來像是以下這樣:

```
# ./prog
./prog[0x8048743] --+-→; 前面這兩行必指向 catch_sig(), 可以忽略
[0xffffe420] -------+
./prog(func+0x1d)[0x804878c] →; 問題發生在這裡
./prog(main+0xa3)[0x804883d] →; main() 呼叫了 func()
/lib/i686/cmov/libc.so.6(__libc_start_main+0xe5)[0xb7e21455]
./prog[0x8048691]
```

從這兒, 我們可以看到問題出現的地方在 func() 的 0x1d, 接下來, 下面將介紹 addr2line 來更進一步解析問題出現的位置.

3. addr2line

addr2line 的用法很簡單, 如下所示:

用法: **addr2line -e 執行檔 -f 位址**

承上例:

 # addr2line -e prog -f 0x804878c

func ----------→; 函式名稱

/home/haha/prog.c:312 --→; 檔案及行數

最後, 只要再注意一件事即可, 就是 312 這個行數, 還不是問題發生的正確位置, 要如何取得確切的位置呢? 假設以下內容為 prog.c 的一部份,

```
309 void func() {
310     unsigned char *data = (unsigned char *)malloc(16);
->; 311     struct st_bug *ptr = (struct st_bug *)data;
 * 312     printf("Hi, I am here\n");
                :
387 }
```

我們發現到, 剛才 addr2line 解析出來的行數是 312, 但是問題其實不在 312, 而是 311, 那是因為呼叫 function 時, 要把返回的位址記錄在 Stack 裡的緣故, 因此當程式執行到有問題的 311 行時, process 會接到 kernel 送來的 SIGBUS, 然而, 在呼叫 signal handler -- catch_sig() 前, 因為 catch_sig() 執行完後, 要回到的位址就是 312, 所以會先把 312 的位址 先存在 Stack 裡, 再呼叫 catch_sig(), 而我們在 catch_sig() 裡呼叫 backtrace() 時, 它只是忠實的告訴我們每一個 function 被呼叫時, Stack 裡所預存的返回位址是什麼而已. (亦即每一個 stack frame 當時所紀錄的 位址). 因此, addr2line 看到的行數, 需要上移一行, 才是真正問題發生的地方. (在此例也就是指 311 行)

[此方法的其他應用]

利用"實作方法"的 2 和 3, 也可以用來捕捉 SIGSEGV (Segmentation Fault,記憶體區段錯誤) 所發生的地方. 甚至是其他的 signal 發生時, 程式執行的位置.