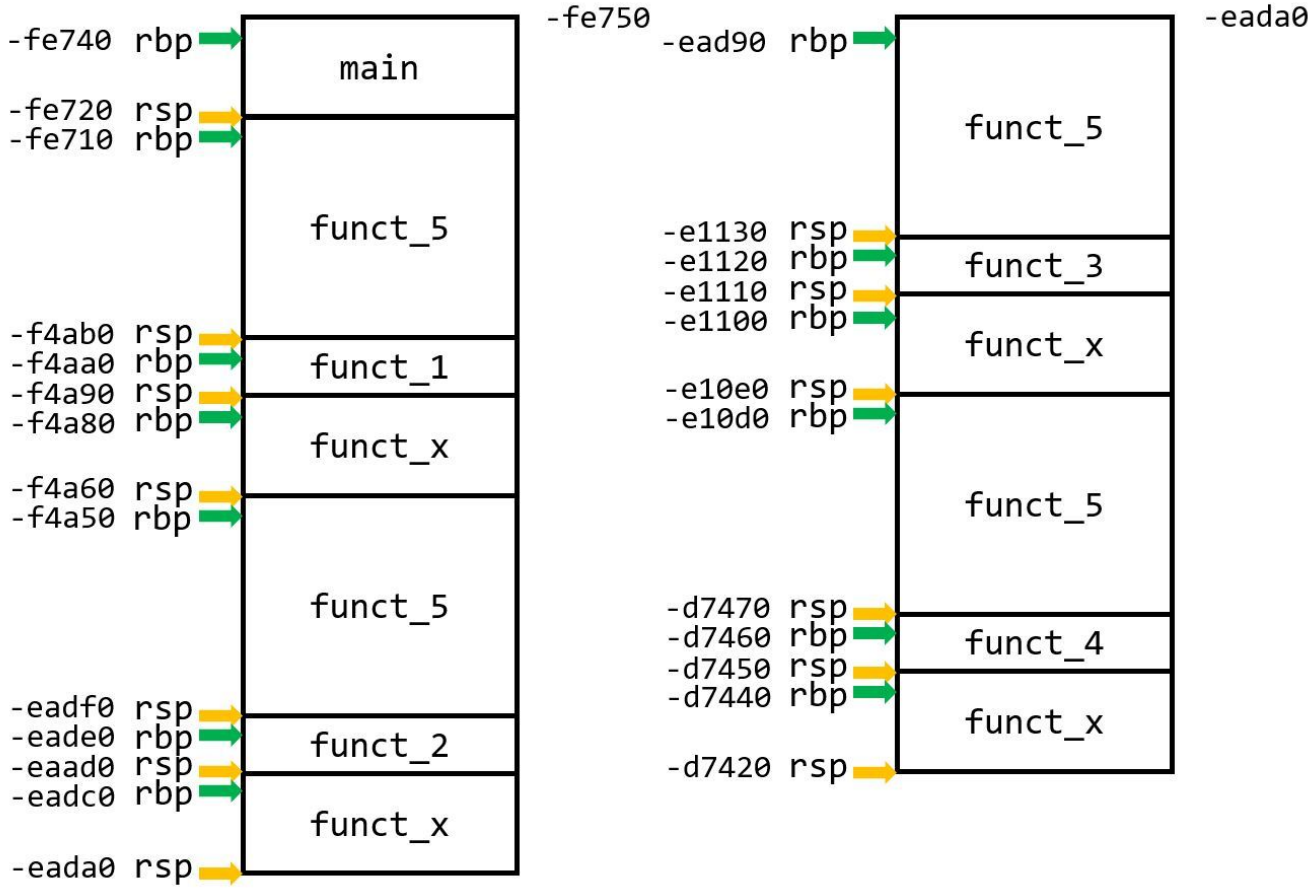
SP Homework #6

ID: B07902070 Name: 陳昱妤

Answer

a. 以下為各 function 的 stack frame 分布，
-xxxxx 代表 0x7fffffxxxxx。
因為 funct_1/2/3/4 重複的部分很多，故呼叫 funct_x 以執行之後動作。



b. 是。因為 FCB 中的 Environment 記錄目前的 stack frame 環境，
因此可還原 local variables。

c. 避免 funct_x 因執行其它動作而增長其 stack frame，
因而改動增長方向之其它 stack frame 內部值，造成錯誤。

d. 不行。因為 funct_x(3) 增長，
改到 funct_5 的 rbp 指向位置: 儲存 funct_x(3) 之 rbp。
因此不能以 funct_5 的 rbp 回到正確的 funct_x(3) 之 stack frame。

當 funct_x(4) 填完 arr 而欲 return 時：

```
(gdb) info stack
#0  funct_x (name=4) at test.c:159
#1  0x0000555555555792 in funct_4 (name=4) at test.c:186
#2  0x000055555555581b in funct_5 (name=4) at test.c:205
#3  0x0000555555555732 in funct_x (name=-304) at test.c:163
Backtrace stopped: previous frame inner to this frame (corrupt stack
   ?)
```

當欲從 funct_5 回到 funct_x(3) 時，會產生錯誤：

```
(gdb) c
Continuing.

Breakpoint 7, funct_5 (name=4) at test.c:210
210             return;
(gdb) c
Continuing.
*** stack smashing detected ***: <unknown> terminated

Program received signal SIGABRT, Aborted.
0x00007ffff7de0f25 in raise () from /usr/lib/libc.so.6
```

e.
- main.c
  - 使用函式：
    - err_sys(): 有錯誤時產生訊息並 exit
    - createChild(): fork 一個 child process 並執行 hw3
  - 變數介紹：
    - sig[]: 存什麼時間該傳哪種 signal
    - cpid: child process id
    - ACK[]: 儲存從 pipe 收到的 ACK
    - arr[]: 儲存從 pipe 收到的 hw3 之 arr
  - main 內容：
    - stdin 讀入所需資料

    * 產生 child process 並執行 hw3
    * 一開始 5 秒後依照 `sig[]` 發送對應之 signal，
      之後等到成功接收 ACK 後再發送下一個 signal
      (如果發出 SIGUSR3，接收 ACK 後要印出其調整後的內容至 stdout)
    * 全部 signal 發送完畢後，接收 ACK 並印出至 stdout
      (此時內容為 hw3 之 arr)
    * `wait()` child process

- hw3.c
  - 使用函式：
    * `err_sys()`: 有錯誤時產生訊息並 exit
    * `sigHandler()`: `SIGUSR1/2/3` 之 signal handler
    * `make_circular_linked_list()`: 做 circular linked list
    * `getLock()`: 取得 arr lock，取得成功回傳 true，否則回傳 false
    * `releaseLock()`: 釋放 arr lock
    * `funct_x()`: 因為 `funct_1/2/3/4` 有極大部分重複，故使用此函式

  - 變數介紹：
    * `MAIN`: jmp_buf，用於從 `funct_4()` 跳回 `main()`
    * `P`, `Q`, `task`, `runTime`: 對應 hw3 傳入 `main()` 之 argument
    * `newmask`: 可以 block `SIGUSR1/2/3` 之 mask
    * `oldmask`: 原來的 mask
    * `pendmask`: 用於記錄哪些 signal 被 pending 中
    * `unmask1/2/3`: 可以用來 unblock 特定 signal 之 mask
    * `queue[]`: 記錄哪些 function 在 queue 中

  - hw3 - main 內容：
    * 將 `SIGUSR1/2/3` 設為 catch，由 `sigHandler()` 處理
    * 將此三個 signal 設為 block
    * 由 `funct_5()` 開始，依照 spec 的順序呼叫函式或 `longjmp()`
    * 處理 FCB circular linked list
    * 開始依照 FCB list 的順序處理 function

- hw3 - funct_x 內容：
  - 處理 FCB circular linked list，
    用 `setjmp()` 將目前狀態存入 `Environment`
  - 如果由 `setjmp()` 跳回，呼叫函式或 `longjmp()`
  - 每次執行大迴圈，開頭都要加 `sigprocmask()`，
    避免從 `sigHandler()` 跳回時，
    mask 會設置成前一次呼叫 `sigprocmask()` 要求的 mask
    (`sigprocmask()` 會把 pending 且未 block 的 signal deliver 後，
    處理完再 return)
  - 如果無法成功取得 arr lock，
    將目前狀態存入 `Environment` 後跳至下一個函式，並將它放入 queue
  - 如果成功取得 arr lock，將此函式從 queue 中刪除，
    並執行小迴圈將對應字元放入 arr
  - 每次完全執行完小迴圈，依照 task 執行以下事項：
    - task 2: 依照 `runTime` 要求，在符合的時間點釋放 arr lock 後，
      將目前狀態存入 `Environment` 並跳至下一個函式
    - task 3: 查看是否有 signal 被 pending 並 unblock 它。
      如果 pending 的為 SIGUSR2，先釋放 arr lock 後再 unblock
  - 大迴圈會跑 `P + 1` 次 (為了在填完 arr 再回來後仍能擁有 lock)，
    第 `P + 1` 次在進入小迴圈前會用 `break` 跳出大迴圈
  - 完全執行完大迴圈後釋放 arr lock，並 `longjmp()` 至 `Scheduler`

- hw3 - sigHandler 內容：
  - 先將 signal mask 還原成可以 block 三個 signal 的狀態
  - 根據收到的 signal 類型，將相對應的資料輸出至 stdout，
    並調整 Current 指到的 FCB block，
    使之後可由 `Schedule()` 跳到對應的 function