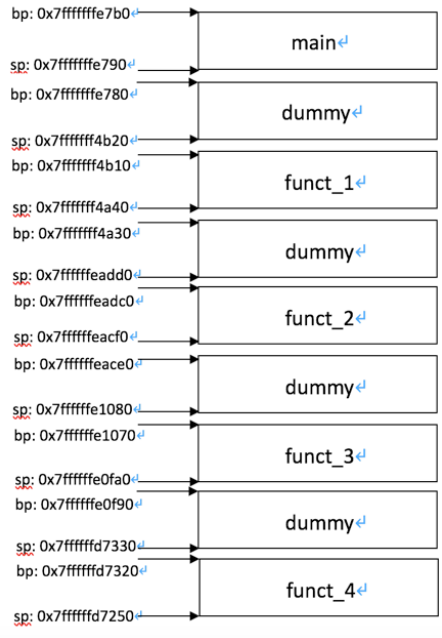
SP Programming HW3 Report

B07902031 資工二 黃永雯

Problem a

This is the stack frame of funct1, funct2, funct3, funct4 and dummy, bp represents the base pointer of the function and sp represents the stack pointer of the function. The commands are as follows:

```
info registers rbp rsp
display $rbp
display $rsp
```



Problem b

The value will be the same when the scheduler jump back to the same function again, since we use setjmp to store the environment of the function before longjmp to the scheduler, when scheduler longjmp back, the environment will be the same as that before longjmp. Moreover, local variable will not be changed in other functions, it can make sure that there are no other functions changing the variable.

Problem c

The usage of dummy function is to make sure that when doing other operations such as calling Scheduler, receiving signals, calling signal handler will not cover the stack frame of function1, 2, 3, 4. Since dummy function allocates a variable which needs a lot of space, it can be used as a buffer.

Problem d

It cannot follow the path and return to main when Scheduler switch to function4 and call return in function4. Since after initialization, funct4 will longjmp to main and main will call Scheduler(). Since Scheduler may cover the space of dummy function, we cannot return back to the origin dummy function as described in the problem. I use gdb to test it and get stack smash.

```
child switch to funct_2
child switch to funct_3
child switch to funct_4
*** stack smashing detected ***: <unknown> terminated

Thread 2.1 "hw3" received signal SIGABRT, Aborted.
0x00007ffff7de0f25 in raise () from /usr/lib/libc.so.6
1: $rbp = (void *) 0x7fffffffd7300
2: $rsp = (void *) 0x7fffffffd6f20
(gdb)
Single stepping until exit from function raise,
which has no line number information.

Program terminated with signal SIGABRT, Aborted.
The program no longer exists.
```

Problem e

For main.c I use sigemptyset, sigaddset, sigprocmask to set up the signals then construct a pipe. Next, use fork and exec to switch to the child process, and parent process send signal to child process. For hw3.c, I initialize the linked-list and the stack frame to set setjmp first. For task 1, run the for loop and append number to arr as description, then call longjmp(SCHEDULER, -2) to go back to the Scheduler. For task

2, add a variable mutex to represent the lock. If it is locked by other function, longjmp back to Scheduler. Otherwise, append the number and check whether the lock can be release. For task 3, it needs more initialization, I use signal, sigemptyset, sigaddset to set the signals. I write a function handler to write ACK message and to longjmp to Scheduler, and a function unblock to unblock the signal, since the signals are block first, it can go to the signal handler and be delivered after unblock.