SYSTEM PROGRAMMING
PROGRAMMING ASSIGNMENT #3
REPORT

B07902143 陳正康

a. Using gdb's si, ni, and p $rbp, p $rsp:

	Stack frame (first call)	Function	Stack frame (after longjmp to main)	Function
High Addr.	rbp=0x7fffffffe150 rsp=0x7fffffffe140	main	(same as left)	main
	rbp=0x7fffffffe130 rsp=0x7ffffff44d0	funct_5 (dummy)	rbp=0x7fffffffe130 rsp=0x7fffffffe120	Scheduler
	rbp=0x7ffffff44c0 rsp=0x7ffffff4410	funct_1	(same as left)	funct_1
	rbp=0x7ffffff4400 rsp=0x7fffffea7a0	funct_5 (dummy)	Preserved area	Functions called by funct_1
	rbp=0x7fffffea790 rsp=0x7fffffea6d0	funct_2	(same as left)	funct_2
	rbp=0x7fffffea6c0 rsp=0x7fffffe0a60	funct_5 (dummy)	Preserved area	Functions called by funct_2
	rbp=0x7fffffe0a50 rsp=0x7fffffe0990	funct_3	(same as left)	funct_3
	rbp=0x7fffffe0980 rsp=0x7fffffd6d20	funct_5 (dummy)	Preserved area	Functions called by funct_3
	rbp=0x7fffffd6d10 rsp=0x7fffffd6c50	funct_4	(same as left)	funct_4
Low Addr.	(unused stack space)			Functions called by funct_4

b. The answer depends.
Local variables remain the same value if they are stored in stack, since stack frame will be restored when long-jumping back.
They won't if they are stored in registers (due to optimization), since registers may be put to other use and their values may change.

c. As shown in a., the dummy function called after main() is to provide a memory space for scheduler()'s stack frame, the other times it's called is to provide space for stack frame of possible function calls inside funct_n().

d. True. Since old rbp value and return address remains the old value (even after a series of longjmp and scheduling) inside the 0x10 gap between each stack frames, it's possible to return all way to main().

Using gdb:

```
# break at the final longjmp in funct_4:
0x0000555555556341 <funct_4+845>:    be fe ff ff ff   mov    esi,0xfffffffe
0x0000555555556346 <funct_4+850>:    48 8d 3d f3 2d 00 00    lea    rdi,[rip+0x2df3]        # 0x555555559140 <SCHEDULER>
=> 0x000055555555634d <funct_4+857>:    e8 ce ed ff ff   call 0x555555555120 <longjmp@plt>

# this shows the gap between funct_3 and funct_4 stack frames
# previous rbp and return address
(gdb) x/2gx $rbp
0x7ffffffd6d10: 0x00007fffffe0980      0x00005555555563f0

# these 2 are same as "leave" in assembly code
(gdb) set $rsp=$rbp+0x10
(gdb) set $rbp=0x00007fffffe0980

# this is same as "ret" in assembly code
(gdb) set $rip= 0x00005555555563f0

(gdb) ni
# successfully return to funct_5
# the similar flow follows, until arriving main()
```

e. Signal handling: At the beginning of main(), SIGUSR1 ~ 3 is blocked, only after one round of small loop will it check for pending signal and unblock it.

   Waiting queue: It uses a boolean array to store which function is in queue. That prevents re-entering the queue.

   funct_frag.h: since funct_1 to funct_4 has so many duplicate code, they are extracted as a shared file to reduce time for re-writing.

   Restore loop counter after schedule back: After context-switch and switching back, the loop counter should continue with previous value. The program store this as a local value "saved_j" (declared as volatile), which is at stack will be restored after longjmp().

   Pipe: main and hw3 use TWO pipe for communication: pmsg and pdata. "pmsg" is for sending ACK message from hw3 to main, "pdata" is for hw3 to send back who's in queue (after receiving SIGUSR3) and final output. "pdata" is redirected as STDOUT of hw3, while file descriptor of "pmsg" is forward to hw3 as the 4-th argument (not used in task 3) when executed.