Lab 2: Dynamic Behavior of XINU Processes

4.2

Before app1 process is resumed, the address and the content of the top of the run-time stack doesn't change. After app1 process is resumed, a new run-time stack is created.

Before func1 is called, the content of the top of the run-time stack is 0 because there is no new allocated value.

After func1 is called, the func1 is pushed to the stack. After func1 is returned, func1 is popped from the stack.

After app1 process terminated, the address of the top of the run-time stack become the original stack address.

4.3

Before main process creates app1(), the stored value of esp from the process table entry of the process main() is not the same as the current esp.

After process app1() is created, the stored esp value of main()'s from its process table entry and the current esp is still not the same. And both of their value didn't change.

After process app1() is created, the stored esp value of the app1() process from its process table entry are not the same as the current esp in app1() process.

5.1

Three "main"s are printed first, and then 100, 200 and 300 are printed randomly and in about same frequency. After every 3ms, "MAIN" is printed. After 10 "MAIN"s are printed every 3ms, 100, 200 and 300 are printed randomly and in about same frequency. Based on the print activity, child processes receive approximately equal share of CPU cycles. The parent process main() always has higher priority in receiving the share of CPU.

Only when higher priority process, the main() process, is not at running state, lower priority process can run.

5.2

The only difference from 5.1 is that only 200 and 300 are printed. 100 is no longer printed, because it has lower priority than other 2 looper processes. Therefore, when main process is not at running state, only 200 and 300 processes can share the CPU cycles.

5.3

After 300 is resumed, only 300 is printed. 100 and 200 are not printed at all. “MAIN” is no longer printed anymore. Because 300 has higher priority than main() process, it will not get out from the infinite while loop. Therefore, 100, 200 and main() processes will not get a chance to share the CPU cycles.

5.4

Only one “main” is printed. After 100 process is resumed, only 100 is printed. No “MAIN” or 200 or 300 is printed. Because 100 process has higher priority than main() process, so that after 100 is resumed, main() process will not be able to run. Therefore, 200 and 300 processes will not get a chance to resume. Therefore 100 process get all CPU cycles.