Lab 3: Monitoring and Scheduling of XUNU Processes

Problem 3

Test1
Having 3 same processes with same priority 20, since they have same priority as main process. These 3 processes have around the same CPU usage.

Test2
Having 3 same processes with different priorities, let's say P1 has priority 30, P2 has priority 40 and P3 has priority 50. There are no sleep() in processes. When P1 resumes, P2 and P3 have no chance to resume, because P1 has higher priority than Main Process. Therefore, only P1 consumes the CPU.

Test 3
Having 3 same process with different priorities, let's say P1 has priority 20, P2 has priority 30, and P3 has priority 40. There are no sleep() in processes. When P1 resumes, since it has the same priority as Main Process, main process still got a chance to resume P2. After P2 was resumed, P1, P3 and main process had no chance to take CPU usage anymore, because P2 always has the highest priority among these processes.

Test4
Having 3 same processes with same priority 30. There are sleepms(1) in every process. By observation, these 3 processes have around same CPU usage.

Test5
Having 3 same processes with same priority 30. However, in P1 and P2 only sleepms(1), and in P3 sleepms(10). The ratio of CPU usage between these 3 processes is 1 : 1 : 3

Test 6
Having 3 same processes with same priority 30. However, P1 sleepms(1), P2 sleepms(5), and P3 sleepms(10). The ratio of CPU usage between these 3 processes is 10 : 6 : 3

Test 7
Having 3 same processes with same priority 20. However, P1 doesn't sleep, P2 sleepms(5), and P3 sleepms(10). The ratio of CPU usage between these 3 processes is 6 : 2 : 1

Problem 4

4.3.1
The output shows that all processes share the same CPU usage.
The 4 processes run in turn for a while. The running order is process 1, process 2, process 3, process 4, process 3, process 2, process 1, process 2…Every process runs until its time slice becomes 0, unless it is still the highest priority process. For example, when process 4 runs, its time slice becomes 0 but its priority is still the highest, so its time slice resets and keeps running until the second time slice becomes zero. Then the next highest priority process will begin to run until all processes finished.

4.3.2
The output shows that all processes share about the same CPU usage.
The processes didn't use any time slice, they voluntarily give up CPU.
Although it seems that some processes will have slightly higher cpu usage than other processes, they share the CPU about equally.

4.3.3
When running cpu-bond and io-bond processes together. CPU-bond processes share the CPU usage equally among themselves. So are io-bond processes.
Among CPU-bond and io-bond processes, CPU-bond processes will run first and more frequently at the beginning, because io-bond processes always voluntarily give up CPU usage to let CPU-bond processes run first.
The CPU-bond processes seems to take more cpu usage than io-bond processes. Maybe it is because io-bond sleeps more, and during sleep, it doesn't take CPU usage.

Problem 5

First, create stacksmashA process. Second, create stacksmashV process. This will make Attacker Process stack on the top of Victim Process stack. Attacker's ESP is Victim's EBP. Then, Resume Victim Process and then resume Attacker Process. When Victim printed the first 'V' and goes to sleep, Attacker process starts to run. Attacker Process gets its ESP first, and then minus its ESP with 1024 bytes to get Victim Process Stack's ESP. Use a unsigned long pointer, pt, to record Victim Process Stack's ESP address. Then Replace value of (pt-1), which is sleepms's return address, with takeover's address. After Sleepms returns, it will return to takeover to run the takeover function.

My approach could be defended by memory protection. If memory boundary checking are done during attack process, the attack will fail.

Bonus Problem

In the process table, there should be a property indicating that whether this process is TS or RT. If the process is RT, then in the reschedule process, its cpu usage (priority) should not increase(decrease), so that it always has the highest priority when comparing to other TS processes because TS will decrease their priority while running.