Operating Systems – Assignment 1

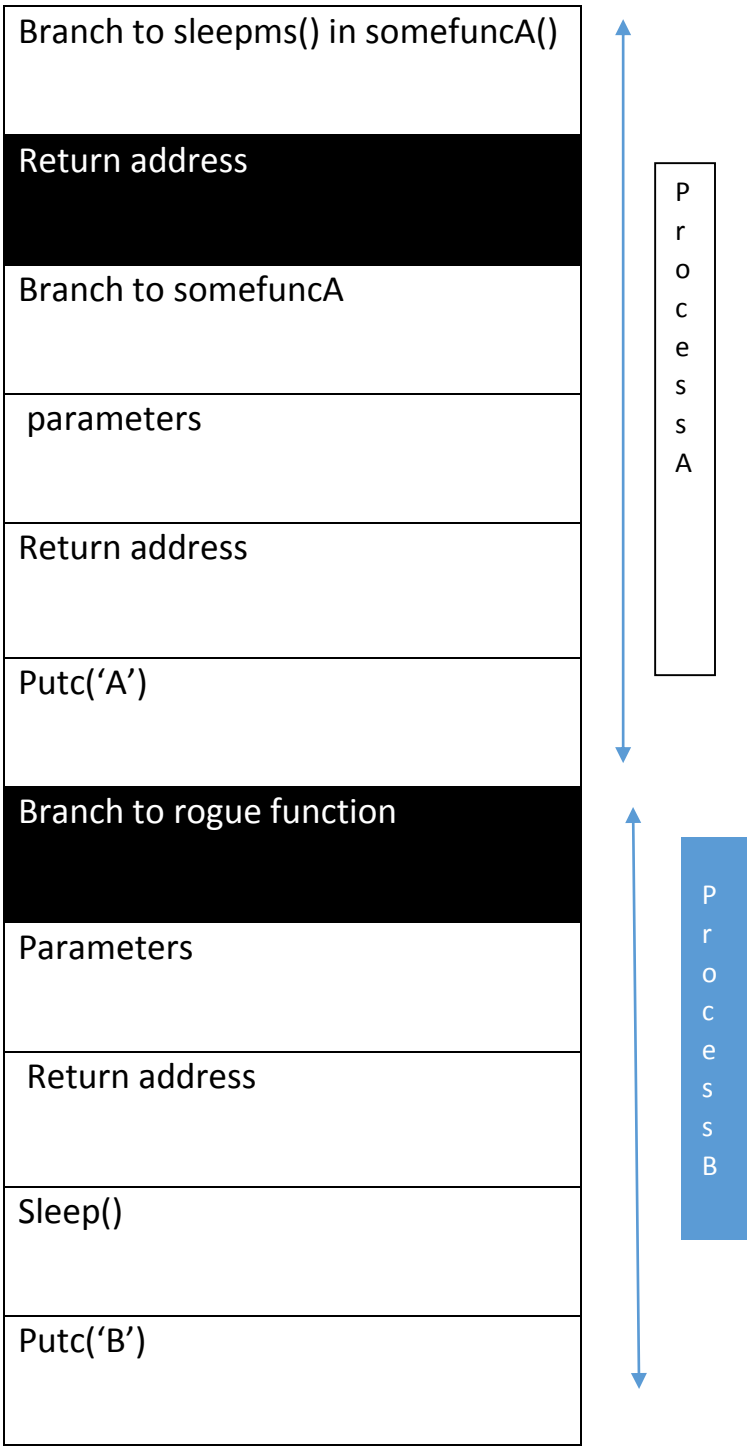
Problem 5:

I have written the rogue function, wherein it first prints ‘B’ and then there is a sleep. A context switch to Process A happens and it prints ‘A’ and then it encounters sleep. Now Process B runs again where there is a nested call. I have written this nested function, which terminates after 50 iterations. This overwrites the stack of process A and corrupts its data, due to which I am getting a general violation protection trap.

Process B stack	Process A stack	

Due to the evergrowing stack size of Process B due to nested function calls, it eventually overwrites the Process A stack and its data is corrupted as shown above.

Problem 6:


Branch to sleepms() in somefuncA()
Return address
Branch to somefuncA
parameters
Return address
Putc('A')
Branch to rogue function
Parameters
Return address
Sleep()
Putc('B')
Process A
Process B

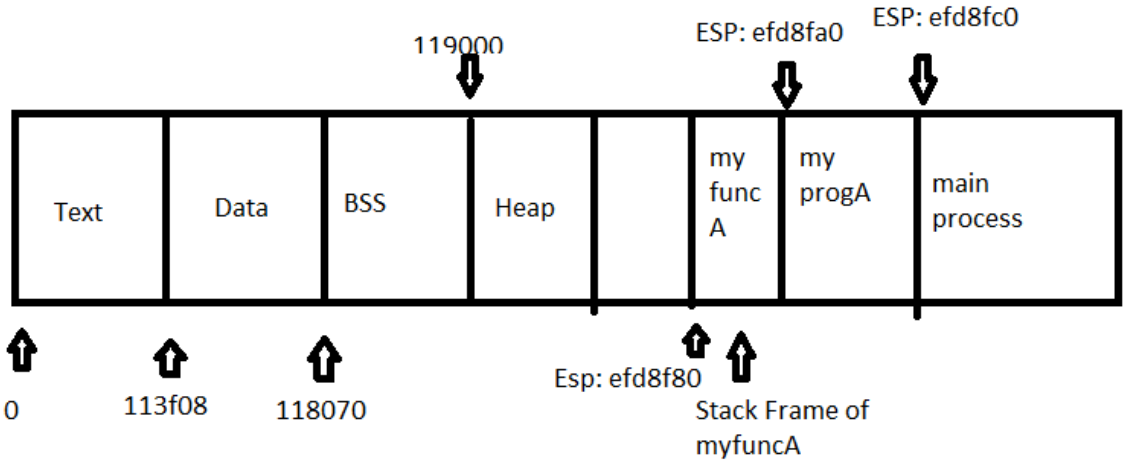
The stack for the two processes looks roughly as shown above. So in task 2 , in order to hijack the Process A, we have to overwrite the return address for Process A with the address of roguefuncB.

We have Process B which first prints B and then does a sleep. It is context switched and Process A starts to run. It prints A and calls somefuncA. Now you have a sleep in somefuncA. The return address is pushed onto the stack so it can resume and print the character ‘a’. We have to overwrite this return address with the address of the roguefuncB. So it will branch out to roguefuncB and executes that code, thereby not printing the characters ‘a’ and ‘A’.

I have used the stack pointer address after the sleep in somefuncA, which gives where the return address for Process A is stored. At this location we have to overwrite it with the address of rogue function. The address of rogue function is found by printing the stack pointer at the starting of the roguefunctionB.

Problem 4

4.1



4.2