3.1C

Create sets main to return to the address of userret(), which then kills the current process. According to the textbook, “The function to which a user process returns if the process exits. Userret must never return. It must kill the process that executed it because the stack does not contain a legal frame or return address.”

3.2A

Create()	Fork()
Create() allows to start a new process, which runs the code of a function specified by a function pointer in create() arguments. The create() function has an argument that allows us to specify the priority of a function. The initial state for a process generated from create () is a suspended state. An additional difference between create() and fork is that for create() you must also specify the stack size required in bytes as an argument.	Fork() creates a child process which is a copy of the parent process that originally called fork(). The child process created by fork() maintains the same priority that its parent process had. The initial state for a child process created by fork() is the same state of the parent process that created it.

3.2B

Priority of child process running main = 23
Priority of sndA and sndB = 20
Output:

```
AAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBAAAAAAAAAAAAAA
AAAAAAAAABBBBBBBBBBBBBBBBBBBBBBB
```

Priority of child process running main = 15
Priority of sndA and sndB = 20

Output:
```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

When the priority of the child process running main() is higher than the priority of sndA() and sndB(), you can see that sndA() and sndB() alternate running. However, when the priority of sndA() and sndB() is higher than the priority of the child process running main(), main() is never able to run, preventing the OS from switching to sndB(), so sndA runs permanently.

4.1

In order to test section 4.1, I printed out msclkcounter1 and msclkcounter2 at the same time in main(), then called sleep(5), and then printed them both out again. The first time I printed them out, both values were 4, and the second time, both values were 5008. Therefore we can confirm that the code for this section functions as expected.

4.2

QUANTUM = 2
Priority of child process running main = 23
Priority of sndA and sndB = 20
Output:
```
AAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBAAAAAAAAAAAAAA
AAAAAAAAABBBBBBBBBBBBBBBBBBBBBBB
```

QUANTUM = 15
Priority of child process running main = 23
Priority of sndA and sndB = 20
Output:
```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
```
I did not test the scenario in which the priority of sndA and sndB is higher than the priority of main, because we know from section 3.2 that sndA will run continuously because main() never gets run.

However, you can see in the scenario where the priority of the child process running main() is higher than the priorities of sndA() and sndB(), the outputs of sndA and sndB are both alternating, indicating that the OS is switching off between running them. You can see that when QUANTUM = 2, that sndA and sndB switch more frequently then when QUANTUM = 15, because of the shorter span of letters between switching when QUANTUM = 2. This makes sense because QUANTUM increases the time slice that each process gets to run before switching off with a process of equal priority.

Bonus