

4.2

Top of stack is the ESP register. Value of ESP register were as follows:

```
[1] Before create() : sp [EFD8FC0] : 111A8C
[2] After create() & before resume() : sp [EFD8FC0] : 0
[3] in app1() before func1() : sp [FDEFFE4] : 0
[4] before func1() returns : sp [FDEFFC4] : 0
[5] after func1() returns: sp [FDEFFE4] : 0
r resume() : sp [EFD8FC0] : 3
```

(The last line was intended to print : [6] After resume() in main() : sp [EFD8FC0] : 3)

Format: sp[value of sp] : value at sp

The print statement did not correctly output the last line due to a context switch.

- [1] The top of the run time stack is the top of the stack for function main().
- [2] The process is not yet resumed so run time stack still belongs to main()
- [3] The runtime stack belongs to app1() and ESP is at the address of the top of app1's stack
- [4] This is printed within func1() and SP points to the top of Func1's stack
- [5] func1 has returned and ESP is back to pointing at app1()'s top of sack
- [6] app1() has been context switched out and ESP points to main()'s top of stack

4.3

Actual output:

```
Printing top of stack frame (ESP)
[1] Before create() : sp [EFD8FC0] : 111B38
[A]proctab->prstkptr before create(): sp [EFD8FCC] : 0
[2a] After create() & before resume() : sp [EFD8FC0] : 0
[B]proctab->prstkptr after create(): sp [EFD8FCC] : 0
[3] in app1() before[6a] After resume() : s func1() : sp [FDEFFE4]p [EFD8FC0] : 3
: 0
[C]proctab->prstkptr before func1(): sp [FDEFEAC] : 0
[4] in func1() before it returns : sp [FDEFFC4] : FDEF1C4
[5] after func1() returns: sp [FDEFFE4] : 0
```

Output adjusted for context switching:

Printing top of stack frame (ESP)

[1] Before create() : sp [EFD8FC0] : 111B38

[A] proctab->prstkptr before create(): sp [EFD8FCC] : 0

The values of ESP and the SP entry stored in the process table are not the same before calling create() on app1().

[2] After create() & before resume(app1) : sp [EFD8FC0] : 0

[B] proctab->prstkptr after create(): sp [EFD8FCC] : 0

```
[3] In app1() before func1() : sp [FDEFFE4] : 0
```

```
[C] proctab->prstkptr In app1() before func1(): sp [FDEFEAC] : 0
```

The values of ESP and the SP entry stored in the process table are not the same int `app1()` before calling `func1()`.

The values for the entries under consideration are not the same because the current ESP may be modified by the running process. The stored value is the value which needs to be restored to continue the execution after a context switch.

5.1

Actual output:

```
val: 300 count: 8625
val: 100 count: 8625
val: 200 count: 862: 8626
val: 300 count: 8626
val: 100 count: 26
val: 200 count: 862: 8627
val: 300 count: 8627
val: 100 count: 7
val: 200 count: 8628 8628
val: 300 count: 8628
val: 100 count: 8
val: 200 count: 8629
val: 300 count: 8629
val: 200 count: 8630
630
val: 300 count: 8630
val: 100 count: 863val: 200 count: 8631
v31
val: 300 count: 8631
val: 100 count: 863a1: 200 count: 8632
va2
val: 300 count: 86332
val: 100 count: 86331: 200 count: 8633
val
val: 300 count: 8634
val: 100 count: 8634
: 200 count: 8634
val: 300 count: 8635
val: 100 count: 8635
200 count: 8635
val: val: 300 count: 8636
vval: 100 count: 8636
v200 count: 8636
val: 2a1: 300 count: 8637
vaal: 100 count: 8637
va00 count: 8637
val: 201: 300 count: 8638
vall: 100 count: 8638
val0 count: 8638
val: 200: 300 count: 8639
val:: 100 count: 8639
val: count: 8639
val: 200 300 count
(command-mode)
```

- The count against each “val” is roughly the same when the processes are stopped at an arbitrary point. Therefore, all children receive approximately the same number of cpu cycles when priority is equal.

- “Main” and “MAIN” are printed before “count” for any process gets a count higher than 10 so the main process with priority 20 gets CPU cycles as soon as it is ready.

5.2

Actual output:

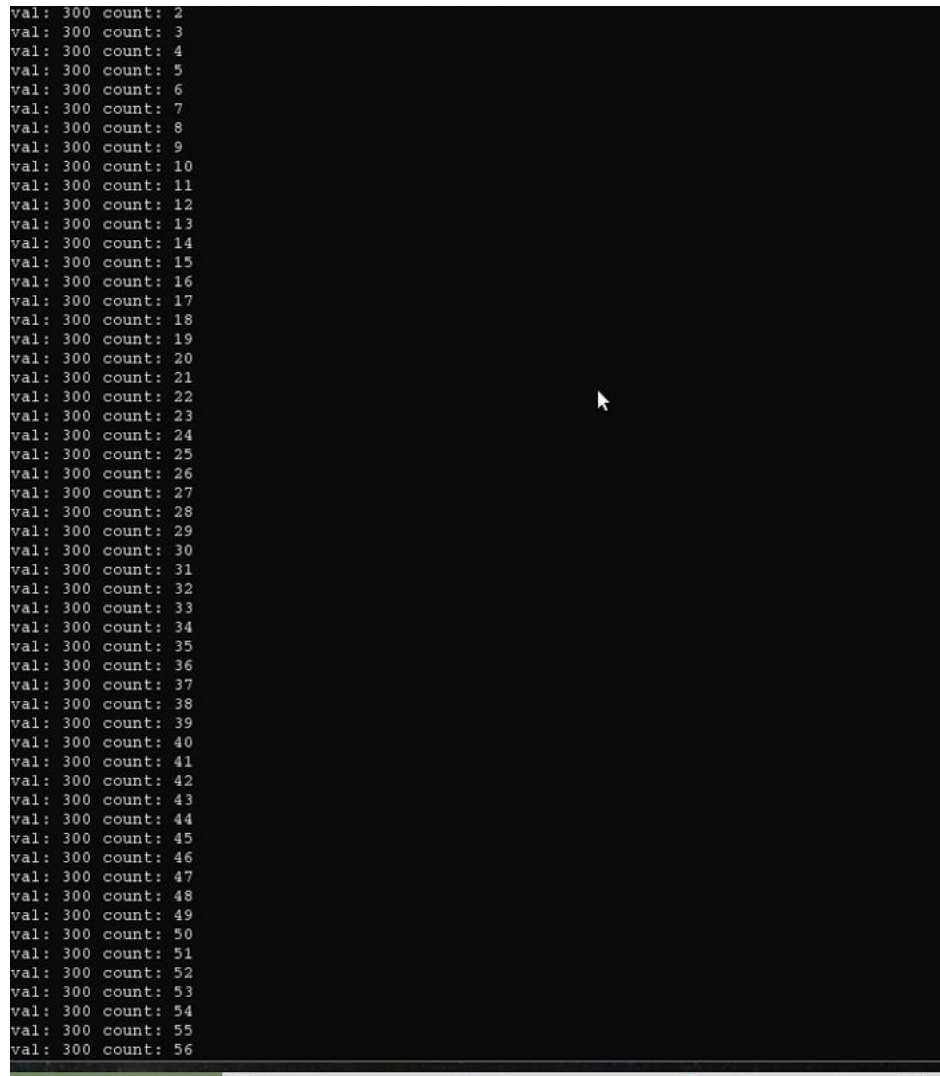
```
val: 300 cal: 200 count: 1067
vaount: 1064
val: 300 col: 200 count: 1068
valunt: 1065
val: 300 cou: 200 count: 1069
val:nt: 1066
val: 300 coun 200 count: 1070
val: t: 1067
val: 300 count200 count: 1071
val: 2: 1068
val: 300 count:00 count: 1072
val: 20 1069
val: 300 count: 0 count: 1073
val: 2001070
val: 300 count: 1 count: 1074
val: 200 071
val: 300 count: 10count: 1075
val: 200 c72
val: 300 count: 107ount: 1076
val: 200 co3
val: 300 count: 1074unt: 1077
val: 200 cou
val: 300 count: 1075
      nt: 1078
val: 300 count: 1076
t: 1079
val: 200 countval: 300 count: 1077
v: 1080
val: 200 count:al: 300 count: 1078
va 1081
val: 200 count: 1: 300 count: 1079
vall082
val: 200 count: 1: 300 count: 1080
val:083
val: 200 count: 10 300 count: 1081
val: 84
val: 200 count: 108300 count: 1082
val: 35
val: 200 count: 108600 count: 1083
val: 30
val: 200 count: 1087
      0 count: 1084
val: 200 count: 1088
count: 1085
val: 300 val: 200 count: 1089
vcount: 1086
val: 300 cal: 200 count: 1090
vaount: 1087
val: 300 col: 200 count: 1091
valunt: 1088
val: 300 cou: 200 count: 1092
valnt: 1089
val: 300 coun: 200 count: 1093
val:t: 1090
val: 300 count 200 count: 1094
val: : 1091
val: 300 count:200 count: 1095
val: 2 1092
```

- The “val” for process 1 (priority 5) is not being printed when examining the output after stopping processes at an arbitrary point. Therefore, the process with the lowest priority does not get CPU cycles as long as higher priority processes are ready to execute.

- The main process continues to get more CPU cycles over it's child processes and executes it's code as soon as it is ready.

5.3

Actual output:

A screenshot of a terminal window with a black background and white text. The text consists of 55 lines, each starting with 'val: 300' followed by 'count:' and a number from 2 to 56. The numbers increase by 1 for each line. A mouse cursor is visible in the center of the terminal window.

```
val: 300 count: 2
val: 300 count: 3
val: 300 count: 4
val: 300 count: 5
val: 300 count: 6
val: 300 count: 7
val: 300 count: 8
val: 300 count: 9
val: 300 count: 10
val: 300 count: 11
val: 300 count: 12
val: 300 count: 13
val: 300 count: 14
val: 300 count: 15
val: 300 count: 16
val: 300 count: 17
val: 300 count: 18
val: 300 count: 19
val: 300 count: 20
val: 300 count: 21
val: 300 count: 22
val: 300 count: 23
val: 300 count: 24
val: 300 count: 25
val: 300 count: 26
val: 300 count: 27
val: 300 count: 28
val: 300 count: 29
val: 300 count: 30
val: 300 count: 31
val: 300 count: 32
val: 300 count: 33
val: 300 count: 34
val: 300 count: 35
val: 300 count: 36
val: 300 count: 37
val: 300 count: 38
val: 300 count: 39
val: 300 count: 40
val: 300 count: 41
val: 300 count: 42
val: 300 count: 43
val: 300 count: 44
val: 300 count: 45
val: 300 count: 46
val: 300 count: 47
val: 300 count: 48
val: 300 count: 49
val: 300 count: 50
val: 300 count: 51
val: 300 count: 52
val: 300 count: 53
val: 300 count: 54
val: 300 count: 55
val: 300 count: 56
```

- The process with val = 300 has priority greater than main so it gets to continue execution before “main” or “MAIN” is printed.
- The process is also always ready to execute so other processes are starved.

5.4

Actual output:

```
val: 100 count: 16929
val: 100 count: 16930
val: 100 count: 16931
val: 100 count: 16932
val: 100 count: 16933
val: 100 count: 16934
val: 100 count: 16935
val: 100 count: 16936
val: 100 count: 16937
val: 100 count: 16938
val: 100 count: 16939
val: 100 count: 16940
val: 100 count: 16941
val: 100 count: 16942
val: 100 count: 16943
val: 100 count: 16944
val: 100 count: 16945
val: 100 count: 16946
val: 100 count: 16947
val: 100 count: 16948
val: 100 count: 16949
val: 100 count: 16950
val: 100 count: 16951
val: 100 count: 16952
val: 100 count: 16953
val: 100 count: 16954
val: 100 count: 16955
val: 100 count: 16956
val: 100 count: 16957
val: 100 count: 16958
val: 100 count: 16959
val: 100 count: 16960
val: 100 count: 16961
val: 100 count: 16962
val: 100 count: 16963
val: 100 count: 16964
val: 100 count: 16965
val: 100 count: 16966
val: 100 count: 16967
val: 100 count: 16968
val: 100 count: 16969
val: 100 count: 16970
val: 100 count: 16971
val: 100 count: 16972
val: 100 count: 16973
val: 100 count: 16974
val: 100 count: 16975
val: 100 count: 16976
val: 100 count: 16977
val: 100 count: 16978
val: 100 count: 16979
val: 100 count: 16980
val: 100 count: 16981
val: 100 count: 16982
val: 100 count: 16983
val: 100 count: 16984
```

- The first process has the highest priority and continues to execute without allowing main to resume the other processes. Therefore, the process with priority 30 continues to execute, starving the others.

Bonus problem

In turnin folder.