1. Compiles correctly with CS333_P4 flag set to 0.

This section will show that my project four code compiles with the project flag set to “0”. Compile with CS333 PROJECT set to 0. Since the listing is so long, this will require two screen shots.

In the first screen shot, the current date and time is displayed as well as the value for CS333 PROJECT, verifying that it is set to 0. In the second screen show, the same informa- tion is displayed. This is used to show that the two screen shots are from the same compilation sequence.

The expected outcome is that the compilation step will correctly compile with the project flag set to “0”.

```
jiacheng@babbage:~/foo/xv6-pdx$ date
Sun Jun  2 19:51:47 PDT 2019
jiacheng@babbage:~/foo/xv6-pdx$ grep "CS333_PROJECT ?=" Makefile
CS333_PROJECT ?= 0
jiacheng@babbage:~/foo/xv6-pdx$ make
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O1 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -DPDX_XV6 -fno-pie -no-pie -fnc
nostdinc -I. -c bootmain.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O1 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -DPDX_XV6 -fno-pie -no-pie -fnc
tdinc -I. -c bootasm.S
ld -m    elf_i386 -N -e start -Ttext 0x7C00 -o bootblock.o bootasm.o bootmain.o
objdump -S bootblock.o > bootblock.asm
objcopy -S -O binary -j .text bootblock.o bootblock
./sign.pl bootblock
```

Figure 1: Compilation with CS333 PROJECT set to 0.

```
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB, 4.9 MiB) copied, 0.129619 s, 39.5 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.010404 s, 49.2 kB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
322+1 records in
322+1 records out
165052 bytes (165 kB, 161 KiB) copied, 0.0123451 s, 13.4 MB/s
jiacheng@babbage:~/foo/xv6-pdx$ grep "CS333_PROJECT ?=" Makefile
CS333_PROJECT ?= 0
jiacheng@babbage:~/foo/xv6-pdx$ date
Sun Jun  2 19:52:02 PDT 2019
```

Figure 2: Compilation with CS333 PROJECT set to 0.

The date in the first and second figures show about 49 seconds of elapsed time. This shows that the two date commands occurred close in time. The grep commands before and after the compilation show that the project flag in the Makefile is set to “0”. The date commands are executed close in time, the project flag shows the same value before and after the compilation, and the compilation shows no errors. This leads to the conclusion that the project code correctly compiles with the project flag turned off.

This test **PASSES** .

2. MAXPRIO = 0.

(a) Show that the scheduler operates as a single round robin queue, as it did in project 3.
In this test, I use p4-test.c which is provided by professor. The scheduler should pick the first process and move it to the CPU. After running from the CPU, it will be moved back to the end of the ready lis

```
51      p4-test 0       0       0       0       41.87   10005.180       runble  12288
52      p4-test 0       0       0       0       41.81   9927.751        runble  12288
53      p4-test 0       0       0       0       40.677  9889.490        runble  12288
54      p4-test 0       0       0       0       40.470  9890.290        runble  12288
55      p4-test 0       0       0       0       39.200  9644.740        runble  12288
56      p4-test 0       0       0       0       39.194  9469.471        sleep   12288    80103f0f 80105a3b 80104d99 80105d7f 80105c9b
57      p4-test 0       0       0       0       38.967  9609.150        runble  12288
58      p4-test 0       0       0       0       38.961  9565.500        runble  12288
59      p4-test 0       0       0       0       38.735  9523.490        runble  12288
60      p4-test 0       0       0       0       36.640  9180.250        runble  12288
61      p4-test 0       0       0       0       36.634  9138.350        runble  12288
62      p4-test 0       0       0       0       36.382  9052.351        runble  12288
63      p4-test 0       0       0       0       35.882  8837.534        sleep   12288    80103f0f 80105a3b 80104d99 80105d7f 80105c9b
64      p4-test 0       0       0       0       35.376  8998.330        runble  12288
Ready List Processes:
(31,10000)->(34,10000)->(33,10000)->(36,10000)->(38,10000)->(37,10000)->(40,10000)->(43,10000)->(44,10000)->(41,10000)->(45,10000)->(47,10000)->(60,10000)->(48,10000)->(61,1
0000)->(64,10000)->(62,10000)->(42,10000)->(7,10000)->(21,10000)->(49,10000)->(28,10000)->(56,10000)->(14,10000)->(63,10000)->(4,10000)->(35,10000)->(55,10000)->(30,10000)->
(50,10000)->(59,10000)->(58,10000)->(52,10000)->(51,10000)->(53,10000)->(54,10000)->(57,10000)->(46,10000)->(39,10000)->(23,10000)->(24,10000)->(16,10000)->(17,10000)->(19,1
0000)->(25,10000)->(18,10000)->(26,10000)->(27,10000)->(29,10000)->(13,10000)->(8,10000)->(6,10000)->(5,10000)->(9,10000)->(10,10000)->(11,10000)->(12,10000)->(20,10000)->(1
5,10000)
```

Figure 3: Round robin

```
56      p4-test 0       0       0       0       28.695  6262.181        runble  12288
57      p4-test 0       0       0       0       28.468  6156.380        runble  12288
58      p4-test 0       0       0       0       28.462  6113.750        runble  12288
59      p4-test 0       0       0       0       28.236  6071.960        runble  12288
60      p4-test 0       0       0       0       26.141  5730.140        runble  12288
61      p4-test 0       0       0       0       26.135  5687.830        runble  12288
62      p4-test 0       0       0       0       25.883  5601.421        runble  12288
63      p4-test 0       0       0       0       25.383  5628.144        runble  12288
64      p4-test 0       0       0       0       24.877  5467.910        runble  12288
Ready List Processes:
(37,10000)->(40,10000)->(43,10000)->(28,10000)->(44,10000)->(41,10000)->(45,10000)->(47,10000)->(60,10000)->(48,10000)->(56,10000)->(61,10000)->(64,10000)->(62,10000)->(55,1
0000)->(50,10000)->(59,10000)->(58,10000)->(51,10000)->(52,10000)->(53,10000)->(14,10000)->(63,10000)->(54,10000)->(57,10000)->(46,10000)->(39,10000)->(4,10000)->(23,10000)-
>(24,10000)->(35,10000)->(16,10000)->(17,10000)->(19,10000)->(25,10000)->(18,10000)->(26,10000)->(27,10000)->(29,10000)->(42,10000)->(13,10000)->(8,10000)->(6,10000)->(5,100
00)->(9,10000)->(10,10000)->(11,10000)->(12,10000)->(20,10000)->(15,10000)->(7,10000)->(21,10000)->(30,10000)->(22,10000)->(32,10000)->(31,10000)->(49,10000)->(34,10000)->(3
3,10000)
```

Figure 3: Round robin

After input the “p4-test” for round robin test, I use ctrl-p and ctrl-r several times. In figure 30, 31 should be the next to run, and 15 is the last one. After the second ctrl-p and ctrl-r, 33 followed by 21 and 7. It means that 21 and 7 was in the running state. 21 and 7 followed by 15, which is the end of the first ctrl-r. When I repeat the ctrl-r, same pattern occurred. This shows the ready list follow the round robin pattern. This Test **Pass.**

(b) Show that setpriority() for any value other than 0 fails.

```
cpu1: starting 1
cpu0: starting 0
sb: size 2000 nblocks 1941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ p4-priority
MAXPRIO is 0. Change MAXPRIO and try again
```

Because the p4-priority not work when set MAXPRIO = 0. My test is **Fail.**

(c) Code should not attempt promotion or demotion when MAXPRIO == 0.
In this test, I use p4-test.c which is provided by professor. It contains six times ctrl-p and ctrl-r. My program should run p4-test correctly.

```
Ready List Processes:
(60,10000)->(61,10000)->(40,10000)->(62,10000)->(55,10000)->(49,10000)->(54,10000)->(58,10000)->(57,10000)->(50,10000)->(51,10000)->(52,10000)->(53,10000)->(41,10000)->(56,1
0000)->(46,10000)->(44,10000)->(7,10000)->(47,10000)->(45,10000)->(48,10000)->(16,10000)->(20,10000)->(43,10000)->(63,10000)->(23,10000)->(3,10000)->(22,10000)->(21,10000)->
(17,10000)->(15,10000)->(24,10000)->(18,10000)->(8,10000)->(25,10000)->(27,10000)->(26,10000)->(28,10000)->(6,10000)->(5,10000)->(4,10000)->(9,10000)->(14,10000)->(11,10000)
->(12,10000)->(35,10000)->(13,10000)->(10,10000)->(19,10000)->(29,10000)->(30,10000)->(31,10000)->(32,10000)->(33,10000)->(34,10000)->(37,10000)->(42,10000)->(38,10000)->(36
,10000)
```

Figure 6: The first time to ctrl-r.

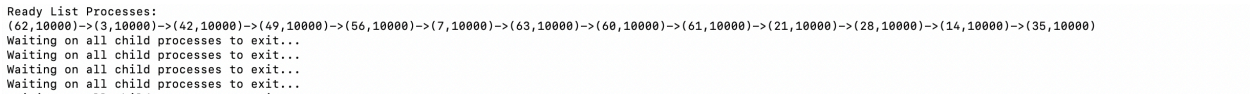

Figure 6: The first time to ctrl-r.

For these two figures, code does not attempt promotion or demotion when MAXPRIO = 0. This Test **Pass.**

3**. For MAXPRIO = 2.**

(a). Show that the scheduler always selects the first process on the highest priority nonempty list.
In this test, I set both the BUDGET to 100000 and TICKS_TO_PROMOTE to 5000. I set MAXPRIO to 2 and run the p4-test. It is expected that scheduler selects the first process on the highest priority.

```
54      p4-test 0       0       0       2       11.560  2613.651        runble  12288
55      p4-test 0       0       0       2       10.72   2256.220        runble  12288
56      p4-test 0       0       0       2       10.65   2262.250        runble  12288
57      p4-test 0       0       0       2       9.838   2152.781        runble  12288
58      p4-test 0       0       0       2       9.619   2048.340        runble  12288
59      p4-test 0       0       0       2       9.156   1943.40 runble  12288
60      p4-test 0       0       0       2       6.341   1406.460        runble  12288
61      p4-test 0       0       0       2       6.95    1233.370        run     12288
62      p4-test 0       0       0       2       5.609   1007.220        run     12288
63      p4-test 0       0       0       2       4.611   1011.98 runble  12288
64      p4-test 0       0       0       2       4.349   835.0   runble  12288
Ready List Processes:
(27,100000)->(8,100000)->(11,38050)->(9,38040)->(5,38040)->(6,100000)->(10,100000)->(14,38030)->(13,100000)->(12,100000)->(20,38010)->(15,38010)->(31,100000)->(30,38000)->(3
3,37990)->(37,100000)->(35,37990)->(21,37990)->(63,37989)->(36,100000)->(22,37980)->(38,37970)->(39,100000)->(41,37960)->(43,37960)->(4,37960)->(44,37950)->(47,37950)->(42,3
7950)->(45,37940)->(60,100000)->(61,100000)->(62,100000)->(48,100000)->(64,37920)->(55,37910)->(58,37910)->(57,37900)->(59,37900)->(28,100000)->(52,37890)->(46,100000)->(50,
100000)->(54,100000)->(49,100000)->(53,100000)->(51,37870)->(7,37870)->(34,37860)->(40,100000)->(32,100000)->(24,37850)->(56,37850)->(23,100000)->(17,37840)->(16,100000)->(2
5,37830)->(19,100000)->(26,37820)
```

Figure 8: Scheduler selects the first process
In this screenshot, the first time ctrl-r shows that process on the highest priority which is the MAXPRIO 2.
This Test **Pass.**

(b). Show that promotion correctly moves processes on the ready lists to the next higher priority list (if one exists) and maintains correct ordering.
In this test, I set both the BUDGET to 100000 and TICKS_TO_PROMOTE to 50000. I set MAXPRIO to 2 and run the p4-test. It is expected that code demotion correctly.

```
56      p4-test 0       0       0       2       10.65   2262.250        runble  12288
57      p4-test 0       0       0       2       9.838   2152.781        runble  12288
58      p4-test 0       0       0       2       9.619   2048.340        runble  12288
59      p4-test 0       0       0       2       9.156   1943.40 runble  12288
60      p4-test 0       0       0       2       6.341   1406.460        runble  12288
61      p4-test 0       0       0       2       6.95    1233.370        run     12288
62      p4-test 0       0       0       2       5.609   1007.220        run     12288
63      p4-test 0       0       0       2       4.611   1011.98 runble  12288
64      p4-test 0       0       0       2       4.349   835.0   runble  12288
Ready List Processes:
(27,100000)->(8,100000)->(11,38050)->(9,38040)->(5,38040)->(6,100000)->(10,100000)->(14,38030)->(13,100000)->(12,100000)->(20,38010)->(15,38010)->(31,100000)->(30,38000)->(3
3,37990)->(37,100000)->(35,37990)->(21,37990)->(63,37989)->(36,100000)->(22,37980)->(38,37970)->(39,100000)->(41,37960)->(43,37960)->(4,37960)->(44,37950)->(47,37950)->(42,3
7950)->(45,37940)->(60,100000)->(61,100000)->(62,100000)->(48,100000)->(64,37920)->(55,37910)->(58,37910)->(57,37900)->(59,37900)->(28,100000)->(52,37890)->(46,100000)->(50,
100000)->(54,100000)->(49,100000)->(53,100000)->(51,37870)->(7,37870)->(34,37860)->(40,100000)->(32,100000)->(24,37850)->(56,37850)->(23,100000)->(17,37840)->(16,100000)->(2
5,37830)->(19,100000)->(26,37820)
Nothing !!!
Nothing !!!
```

Figure : Ctrl-r before verify promotion

```
58      p4-test 0       0       0       2       20.599  4926.720        runble  12288
59      p4-test 0       0       0       2       20.136  4821.851        runble  12288
60      p4-test 0       0       0       2       17.321  4294.530        runble  12288
61      p4-test 0       0       0       2       17.75   4110.670        run     12288
62      p4-test 0       0       0       2       16.589  3884.740        run     12288
63      p4-test 0       0       0       2       15.591  3965.170        sleep   12288   80103ff2 80105bde 80104f3c 80105f22 80105e3e
64      p4-test 0       0       0       2       15.329  3712.950        runble  12288
Ready List Processes:
(55,27190)->(58,27180)->(57,27180)->(4,100000)->(59,27170)->(52,27170)->(42,100000)->(46,100000)->(50,100000)->(54,100000)->(51,27150)->(53,100000)->(34,27140)->(40,100000)-
>(32,100000)->(28,27130)->(24,27120)->(23,100000)->(17,27110)->(16,100000)->(49,27110)->(25,27100)->(19,100000)->(7,100000)->(26,27090)->(18,27090)->(29,27080)->(27,27080)->
(56,100000)->(8,27070)->(11,100000)->(9,100000)->(5,100000)->(6,27050)->(10,27050)->(13,27040)->(12,27040)->(20,100000)->(15,100000)->(31,27020)->(30,100000)->(33,100000)->(
37,27010)->(14,27010)->(36,27000)->(22,100000)->(38,100000)->(39,26990)->(41,100000)->(43,100000)->(44,100000)->(47,100000)->(35,26970)->(21,26970)->(63,26970)->(45,100000)-
>(60,26960)->(61,26950)->(62,26950)
Nothing !!!
Nothing !!!
```

Figure : Ctrl-r to verify promotion
In two screenshot, I can't verify the promotion, because my control-r not shows the different in priority 1 and 2. My test is **Fail.**

(c). Show that demotion correctly moves a process to the next lower priority list (if one exists) when the processes budget is used up.

In this test, I set both the BUDGET to 100000 and TICKS_TO_PROMOTE to 200000 and will observe how priority demotion work. I set MAXPRIO to 2 and run the p4-test. It is expected that code demotion correctly.

```
54      p4-test 0       0       0       2       53.494  17942.113       runble  12288
55      p4-test 0       0       0       2       52.6    17578.410       run     12288
56      p4-test 0       0       0       2       51.999  17841.721       runble  12288
57      p4-test 0       0       0       2       51.772  17476.793       runble  12288
58      p4-test 0       0       0       2       51.553  17371.413       run     12288
59      p4-test 0       0       0       2       51.90   17268.183       runble  12288
60      p4-test 0       0       0       2       48.275  16765.340       runble  12288
61      p4-test 0       0       0       2       48.29   16655.350       runble  12288
62      p4-test 0       0       0       2       47.543  16430.120       runble  12288
63      p4-test 0       0       0       2       46.545  16610.237       runble  12288
64      p4-test 0       0       0       2       46.283  16259.732       runble  12288
Ready List Processes:
(56,100000)->(57,100000)->(46,100000)->(59,100000)->(50,100000)->(14,100000)->(54,100000)->(52,100000)->(35,100000)->(51,100000)->(53,100000)->(21,100000)->(40,100000)->(41,
100000)->(63,100000)->(39,100000)->(43,100000)->(44,100000)->(42,100000)->(47,100000)->(4,100000)->(28,100000)->(60,100000)->(45,100000)->(49,100000)->(61,100000)->(62,10000
0)->(7,100000)->(48,100000)->(64,100000)
Nothing !!!
Nothing !!!
```

Figure : The third ctrl-r in p4-test

```
53      p4-test 0       0       0       2       64.234  27391.487       zombie  12288
54      p4-test 0       0       0       2       64.27   26837.362       zombie  12288
55      p4-test 0       0       0       2       62.539  31466.756       zombie  12288
56      p4-test 0       0       0       2       62.532  43914.917       runble  12288
57      p4-test 0       0       0       2       62.305  31366.425       zombie  12288
58      p4-test 0       0       0       2       62.86   33279.707       zombie  12288
59      p4-test 0       0       0       2       61.623  33178.209       zombie  12288
60      p4-test 0       0       0       2       58.808  41510.392       run     12288
61      p4-test 0       0       0       2       58.562  41289.261       run     12288
62      p4-test 0       0       0       2       58.76   41177.257       runble  12288
63      p4-test 0       0       0       2       57.78   42575.798       runble  12288
64      p4-test 0       0       0       2       56.816  41008.491       runble  12288
Ready List Processes:
(35,100000)->(42,100000)->(21,100000)->(28,100000)->(60,100000)->(61,100000)->(49,100000)->(63,100000)->(56,100000)->(7,100000)->(4,100000)->(14,100000)
Nothing !!!
Nothing !!!
```

Figure : Demotion incorrectly

In my screenshot, it show priority 1 and 0 both nothing in it. It should have list remove priority 1 to priority 0. I can't verify the demotion. My test is **Fail.**

4**. For MAXPRIO = 6.**

(a). Show that the scheduler always selects the first process on the highest priority nonempty list.
In this test, I set both the BUDGET to 100000 and TICKS_TO_PROMOTE to 5000. I set MAXPRIO to 6 and run the p4-test. It is expected that scheduler selects the first process on the highest priority.

```
50      p4-test 0       0       0       6       9.613   12315.640       run     12288
51      p4-test 0       0       0       6       9.420   12004.620       runble  12288
52      p4-test 0       0       0       6       9.215   11693.180       runble  12288
53      p4-test 0       0       0       6       9.7     11381.640       runble  12288
54      p4-test 0       0       0       6       8.796   11069.340       runble  12288
55      p4-test 0       0       0       6       7.103   8884.20 runble  12288
56      p4-test 0       0       0       6       6.878   8568.730        runble  12288
57      p4-test 0       0       0       6       6.648   8256.50 runble  12288
58      p4-test 0       0       0       6       6.427   7306.250        run     12288
59      p4-test 0       0       0       6       5.971   6991.500        runble  12288
60      p4-test 0       0       0       6       4.103   5100.100        runble  12288
61      p4-test 0       0       0       6       4.97    4783.430        runble  12288
62      p4-test 0       0       0       6       3.849   4466.390        runble  12288
63      p4-test 0       0       0       6       3.601   4465.990        runble  12288
64      p4-test 0       0       0       6       3.599   4466.470        runble  12288
Ready List Processes:
6: (18,100000)->(26,100000)->(29,100000)->(28,100000)->(27,100000)->(9,100000)->(6,100000)->(5,100000)->(10,100000)->(12,100000)->(13,100000)->(15,100000)->(20,100000)->(22,
100000)->(56,100000)->(30,100000)->(23,100000)->(31,100000)->(33,100000)->(34,100000)->(36,100000)->(37,100000)->(32,100000)->(38,100000)->(40,100000)->(63,100000)->(41,1000
00)->(43,100000)->(4,100000)->(44,100000)->(39,100000)->(35,100000)->(45,100000)->(46,100000)->(47,100000)->(60,100000)->(55,100000)->(61,100000)->(62,100000)->(64,100000)->
(21,100000)->(57,100000)->(58,100000)->(42,100000)->(50,100000)->(59,100000)->(48,100000)->(51,100000)->(52,100000)->(54,100000)->(7,100000)->(14,100000)->(53,100000)->(24,1
00000)->(16,100000)->(17,100000)->(11,100000)->(19,100000)->(49,100000)
5: Nothing !!!
4: Nothing !!!
```

Figure 13: Scheduler selects the first process

In this screenshot, the first time ctrl-r shows that process on the highest priority which is the MAXPRIO 6. This test **PASSES** .

(b). Show that promotion correctly moves processes on the ready lists to the next higher priority list (if one exists) and maintains correct ordering.
In this test, I set both the BUDGET to 5000 and TICKS_TO_PROMOTE to 5000. I set MAXPRIO to 6 and run the p4-test. It is expected that code demotion correctly.

```
56      p4-test 0       0       0       6       6.725   1999.480        runble  12288
57      p4-test 0       0       0       6       6.495   1853.30 runble  12288
58      p4-test 0       0       0       6       6.36    1704.221        runble  12288
59      p4-test 0       0       0       6       4.178   1251.20 runble  12288
60      p4-test 0       0       0       6       4.171   1098.270        runble  12288
61      p4-test 0       0       0       6       3.684   1021.620        runble  12288
62      p4-test 0       0       0       6       3.433   944.640 runble  12288
63      p4-test 0       0       0       6       3.177   944.320 runble  12288
Ready List Processes:
6: (43,5000)->(49,5000)->(40,5000)->(31,5000)->(23,5000)->(17,5000)->(18,5000)->(4,5000)->(16,5000)->(24,5000)->(25,5000)->(26,5000)->(7,5000)->(5,5000)->(27,5000)->(56,5000
)->(6,5000)->(9,5000)->(10,5000)->(8,5000)->(11,5000)->(13,5000)->(19,5000)->(15,5000)->(28,5000)->(14,5000)->(12,5000)->(20,5000)->(29,5000)->(22,5000)->(30,5000)->(32,5000
)->(33,5000)->(34,5000)->(63,5000)->(3,5000)->(37,5000)->(39,5000)->(38,5000)->(54,5000)->(35,5000)->(50,5000)->(55,5000)->(59,5000)->(60,5000)->(57,5000)->(61,5000)->(62,50
00)->(58,5000)->(42,5000)->(51,5000)->(52,5000)->(21,5000)->(53,5000)->(45,5000)->(44,5000)->(41,5000)->(46,5000)->(47,5000)
5: Nothing !!!
4: Nothing !!!
```

Figure : Ctrl-r before verify promotion

```
54      p4-test 0       0       0       6       17.508  5651.160        runble  12288
55      p4-test 0       0       0       6       17.501  5578.351        runble  12288
56      p4-test 0       0       0       6       17.275  5676.41 runble  12288
57      p4-test 0       0       0       6       17.45   5357.430        runble  12288
58      p4-test 0       0       0       6       16.586  5209.231        runble  12288
59      p4-test 0       0       0       6       14.728  4755.10 runble  12288
60      p4-test 0       0       0       6       14.721  4602.460        runble  12288
61      p4-test 0       0       0       6       14.234  4526.220        runble  12288
62      p4-test 0       0       0       6       13.983  4449.450        runble  12288
63      p4-test 0       0       0       6       13.727  4533.480        runble  12288
Ready List Processes:
6: (42,5000)->(43,5000)->5: Nothing !!!
4: Nothing !!!
3: Nothing !!!
2: Nothing !!!
1: Nothing !!!
0: Nothing !!!
```

Figure : Promotion incorrectly

In two screenshot, I can't verify the promotion, because my control-r not shows the different in priority 5 and 6. My test is **Fail.**

(c). Show that demotion correctly moves a process to the next lower priority list (if one exists) when the processes budget is used up.
In this test, I set both the BUDGET to 15000 and TICKS_TO_PROMOTE to 25000 and will observe how priority demotion work. I set MAXPRIO to 2 and run the p4-test. It is expected that code demotion correctly.

```
53      p4-test 0       0       0       6       50.522  8972.610        runble  12288
54      p4-test 0       0       0       6       49.27   8830.320        runble  12288
55      p4-test 0       0       0       6       48.803  8811.150        runble  12288
56      p4-test 0       0       0       6       48.577  9003.983        runble  12288
57      p4-test 0       0       0       6       48.570  8793.780        runble  12288
58      p4-test 0       0       0       6       48.344  8772.180        runble  12288
59      p4-test 0       0       0       6       46.727  8614.970        runble  12288
60      p4-test 0       0       0       6       46.484  8614.150        runble  12288
61      p4-test 0       0       0       6       46.237  8593.490        runble  12288
62      p4-test 0       0       0       6       46.231  8572.631        runble  12288
63      p4-test 0       0       0       6       45.974  8703.433        runble  12288
Ready List Processes:
6: (52,15000)->(40,15000)->(44,15000)->(53,15000)->(35,15000)->(38,15000)->(46,15000)->(47,15000)->(48,15000)->(43,15000)->(45,15000)->(42,15000)->(41,15000)->(34,15000)->(3
1,15000)->(24,15000)->(49,15000)->(27,15000)->(29,15000)->(21,15000)->(56,15000)->(30,15000)->(33,15000)->(32,15000)->(36,15000)->(63,15000)->(37,15000)->(39,15000)->(3,1500
0)->(60,15000)->(59,15000)->(7,15000)->(61,15000)->(50,15000)->(28,15000)->(62,15000)->(54,15000)->(14,15000)->(55,15000)->(51,15000)
5: Nothing !!!
4: Nothing !!!
3: Nothing !!!
2: Nothing !!!
1: Nothing !!!
0: Nothing !!!
```

Figure : The third ctrl-r in p4-test

```
51      p4-test 0       0       0       6       60.670  16962.17        zombie  12288
52      p4-test 0       0       0       6       60.670  16962.17        zombie  12288
53      p4-test 0       0       0       6       60.464  16634.226       zombie  12288
54      p4-test 0       0       0       6       58.969  17272.530       run     12288
55      p4-test 0       0       0       6       58.745  17332.193       runble  12288
56      p4-test 0       0       0       6       58.519  17677.270       runble  12288
57      p4-test 0       0       0       6       58.512  17237.771       runble  12288
58      p4-test 0       0       0       6       58.286  17216.201       runble  12288
59      p4-test 0       0       0       6       56.669  17133.102       runble  12288
60      p4-test 0       0       0       6       56.426  17132.441       runble  12288
61      p4-test 0       0       0       6       56.179  17112.553       runble  12288
62      p4-test 0       0       0       6       56.173  17014.205       run     12288
63      p4-test 0       0       0       6       55.916  17446.143       runble  12288
Ready List Processes:
6: (35,15000)->(58,15000)->(57,15000)->(42,15000)->(49,15000)->(21,15000)->(60,15000)->(59,15000)->(56,15000)->(63,15000)->(7,15000)->(55,15000)->(61,15000)->(3,15000)
5: Nothing !!!
4: Nothing !!!
3: Nothing !!!
2: Nothing !!!
1: Nothing !!!
0: Nothing !!!
```

Figure : Demotion incorrectly

In my screenshot, it show priority 1 and 0 both nothing in it. It should have list remove priority 1 to priority 0. I can't verify the demotion. My test is **Fail.**

5. setpriority()

In this test, I set both the BUDGET and TICKS_TO_PROMOTE to 5000 a large that can prevent priority promotion. Set the MAXPRIO to 6. After that, run the p4-priority test.

```
init: starting sh
$ p4-priority
Testing that process starts at MAXPRIO
Priority after program start is 6
**** TEST PASSED ****


Testing that a priority cannot be set to an out of range value.
  Testing setting priority to a negative number.
  setPriority(3, -1) returned -1.
  **** TEST PASSED ****


  Testing that a priority cannot be set on a non-existent PID.
  **** TEST PASSED ****


Priority for pid 1 is 6
Press C-p to verify.


Testing promotion...

PID     Name    UID     GID     PPID    Prio    Elapsed CPU     State   Size    PCs
1       init    0       0       0       6       24.770  0.610   sleep   12288    80103fcf 8010413c 80105ad7 80104eff 80105ee5 80105e01
2       sh      0       0       0       6       24.729  20.89   sleep   16384    80103fcf 8010413c 80105ad7 80104eff 80105ee5 80105e01
3       p4-priority     0       0       0       0       5.640   121532.850      sleep   16384    80103fcf 80105ba1 80104eff 80105ee5 80105e01
Promotion has occurred.
**** TEST PASSES ****
```

Figure :p4-priority

In this figure, all tests pass. This test **PASSES** .

6. getpriority()

```
ps

PID     Name    UID     GID     PPID    Prio    Elapsed CPU     State   Size
1       init    0       0       1       0       16.431  0.435   sleep   12288
2       sh      0       0       1       0       16.399  17.249  sleep   16384
3       ps      0       0       2       0       0.20    114.986 run     45056
```

(a) Shows the correct priority for the current process
I can’t prove this because I don’t have the get priority system call like: getpriority 4.
My test is **Fail.**

(b) Shows the correct priority for any process other than the current process
I can’t prove this because I don’t have the get priority system call like: getpriority 1.
My test is **Fail.**

(c) Returns -1 if PID is not found
I can’t prove this because I don’t have the get priority system call like: getpriority 10.
My test is **Fail.**

7. Updated Commands

(a) Show that ps correctly displays the process priority.
In this test, I used command ps to trace process's priority.

```
$
PID     Name    UID     GID     PPID    Prio    Elapsed CPU     State   Size    PCs
1       init    0       0       0       6       32.264  0.493   sleep   12288   80103fcf 8010413c 80105ad7 80104eff 80105ee5 80105e01
2       sh      0       0       0       6       32.230  0.764   sleep   16384   80103fcf 801002b1 80101914 80100f0b 801051da 80104eff 80105ee5 80105e01
[ps                                                                                                                                                 ]

PID     Name    UID     GID     PPID    Prio    Elapsed CPU     State   Size
1       init    0       0       1       0       34.633  0.493   sleep   12288
2       sh      0       0       1       0       34.599  69.989  sleep   16384
3       ps      0       0       2       0       0.26    207.760 run     45056
$
```

Figure : ps and ctrl-p command

After booting the system, I use ctrl-p and ps command to see the process information. First ps to display the process information, then ctrl-p to display it again. The priority of init and sh are not same as the ctrl-p displayed. If is correct. For the ps process, it's priority is 6. My test is **Fail.**

(b) Show that control – p correctly displays the process priority
In this test, after booting the system, I used Ctrl-p to display the processes information.

```
$
PID     Name    UID     GID     PPID    Prio    Elapsed CPU     State   Size    PCs
1       init    0       0       0       6       32.264  0.493   sleep   12288   80103fcf 8010413c 80105ad7 80104eff 80105ee5 80105e01
2       sh      0       0       0       6       32.230  0.764   sleep   16384   80103fcf 801002b1 80101914 80100f0b 801051da 80104eff 80105ee5 80105e01
[ps                                                                                                                                                 ]
```

Figure 21: ctrl-p command

After booting the system, I use ctrl-p to see the priority of init and sh are both 6. This result is correct. This test **PASSES** .

(c) Show that control – r correctly displays all ready lists, from highest to lowest priority, and the budget for each process
In this test, I set both the BUDGET to 10000 and TICKS_TO_PROMOTE to 5000. I set MAXPRIO to 2 and run the p4-test. It is expected that ctrl-r correctly displays all ready lists.

```
54      p4-test 0       0       0       2       11.560  2613.651        runble  12288
55      p4-test 0       0       0       2       10.72   2256.220        runble  12288
56      p4-test 0       0       0       2       10.65   2262.250        runble  12288
57      p4-test 0       0       0       2       9.838   2152.781        runble  12288
58      p4-test 0       0       0       2       9.619   2048.340        runble  12288
59      p4-test 0       0       0       2       9.156   1943.40 runble  12288
60      p4-test 0       0       0       2       6.341   1406.460        runble  12288
61      p4-test 0       0       0       2       6.95    1233.370        run     12288
62      p4-test 0       0       0       2       5.609   1007.220        run     12288
63      p4-test 0       0       0       2       4.611   1011.98 runble  12288
64      p4-test 0       0       0       2       4.349   835.0   runble  12288
Ready List Processes:
(27,100000)->(8,100000)->(11,38050)->(9,38040)->(5,38040)->(6,100000)->(10,100000)->(14,38030)->(13,100000)->(12,100000)->(20,38010)->(15,38010)->(31,100000)->(30,38000)->(3
3,37990)->(37,100000)->(35,37990)->(21,37990)->(63,37989)->(36,100000)->(22,37980)->(38,37970)->(39,100000)->(41,37960)->(43,37960)->(4,37960)->(44,37950)->(47,37950)->(42,3
7950)->(45,37940)->(60,100000)->(61,100000)->(62,100000)->(48,100000)->(64,37920)->(55,37910)->(58,37910)->(57,37900)->(59,37900)->(28,100000)->(52,37890)->(46,100000)->(50,
100000)->(54,100000)->(49,100000)->(53,100000)->(51,37870)->(7,37870)->(34,37860)->(40,100000)->(32,100000)->(24,37850)->(56,37850)->(23,100000)->(17,37840)->(16,100000)->(2
5,37830)->(19,100000)->(26,37820)
```

Figure : ctrl-r command
In this figure, ctrl-r correctly displays all ready lists, from highest to lowest priority, and the budget for each process. That's correct. This test **PASSES** .