To obtain the LATEX source for this document, change the file extension to ".tex" in the url.

Testing

Compilation Test

This section will show that my project one code compiles with the project flag set to "1" and also set to "0".

Subtest 1: 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 May 12 10:59:06 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-pr
otector -DPDX_XV6 -fno-pie -no-pie -fno-pic -O -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-pr
otector -DPDX_XV6 -fno-pie -no-pie -fno-pic -nostdinc -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.

```
objdump -S kernel > kernel.asm
objdump -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kernel.sym
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB, 4.9 MiB) copied, 0.151865 s, 33.7 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.00822425 s, 62.3 kB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
322+1 records in
322+1 records out
165092 bytes (165 kB, 161 KiB) copied, 0.0143556 s, 11.5 MB/s
jiacheng@babbage:~/foo/xv6-pdx$ grep "CS333_PROJECT ?=" Makefile
CS333_PROJECT ?= 0
jiacheng@babbage:~/foo/xv6-pdx$ date
Sun May 12 10:59:28 PDT 2019
```

Figure 2: Compilation with CS333 PROJECT set to 0.

The date in the first and second figures show about 12 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.

Subtest 2: Boot compile with CS333 PROJECT set to 3. 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 3. In the second screen show, the same information 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 “3”.

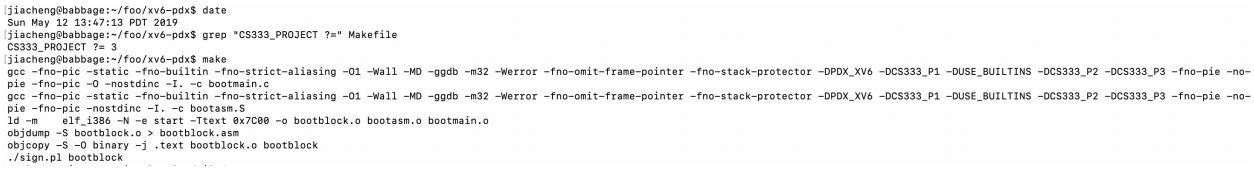


Figure 3: Compilation with CS333 PROJECT set to 3.

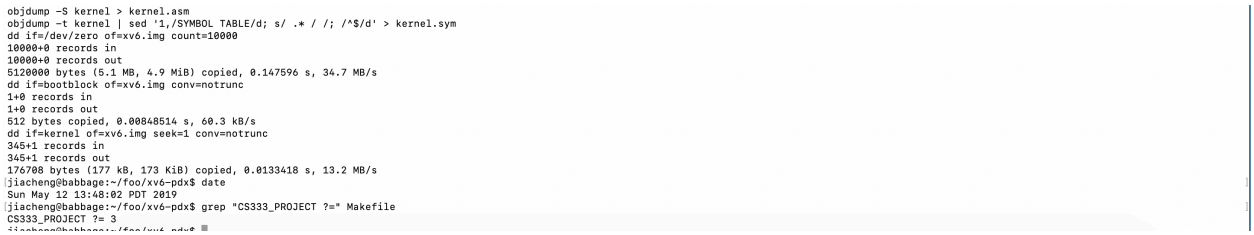


Figure 4: Compilation with CS333 PROJECT set to 3.

The date in the first and second figures show about 10 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 “3”. 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 subtest PASSES.

Since each subtest passes and the subtests fully test the objectives, this test PASSES.

Usertests and forktest run with CS333 P3 flag turned off

I tested that xv6 correctly compiles and runs with the CS333 P3 flag disabled. I set the CS333 PROJECT value in the Makefile to 0. I then verified this setting and compiled xv6.
It is expected that xv6 will boot normally and both usertests and forktest will successfully execute. Since forktest is executed as part of usertests, only usertests will be executed.

```
$ usertests
usertests starting
arg test passed
createdelete test
createdelete ok
linkunlink test
linkunlink ok
concreate test
concreate ok
fourfiles test
```

Figure 10: CS333 P3 disabled 3 Some output has been elided.

```
fork test
fork test OK
bigdir test
bigdir ok
uio test
pid 591 usertests: trap 13 err 0 on cpu 0 eip 0x340d addr 0x800249f0--kill proc
uio test done
exec test
ALL TESTS PASSED
$ QEMU: Terminated
make[1]: Leaving directory '/u/jiacheng/foo/xv6-pdx'
jiacheng@babbage:~/foo/xv6-pdx$ grep "CS333_PROJECT ?=" Makefile
CS333_PROJECT ?= 0
```

Figure 10: CS333 P3 disabled 3

From both figures, we can see that all usertests have passed. Further, since forktest is run as a part of usertests, we know that forktest has passed.
This test PASSES.

Usertests and forktest run with CS333 P3 flag turned on

I tested that xv6 correctly compiles and runs with the CS333 P3 flag enabled. I set the CS333 PROJECT value in the Makefile to 3, compiled and booted xv6 using make qemu-nox, and then ran usertests. As mentioned above, this is an acceptable test for both usertests and forktest since forktest is run as part of usertests.
It is expected that all tests from usertests will pass.

```
fork test
fork test OK
bigdir test
bigdir ok
uio test
pid 591 usertests: trap 13 err 0 on cpu 1 eip 0x340d addr 0x801dc130--kill proc
uio test done
exec test
ALL TESTS PASSED
$ QEMU: Terminated
make[1]: Leaving directory '/u/jiacheng/foo/xv6-pdx'
jiacheng@babbage:~/foo/xv6-pdx$ grep "CS333_PROJECT ?=" Makefile
CS333_PROJECT ?= 3
jiacheng@babbage:~/foo/xv6-pdx$ date
Sun May 12 13:51:27 PDT 2019
```

Figure 12: Usertests with CS333 P3 enabled

From the above figure, we can see that all usertest tests pass. This test PASSES.

Free list is initialized after xv6 is fully booted.

I'm do the test to prove after xv6 fully booted, the free list is correct initialized.

I will boot the xv6 and use control-p to show there are 2 active processes and use control-f to show number of the rest of the free list. In the beginning, there are 64 processes totally. But when xv6 fully boot, 2 processes is already used. So control-f should only show the 62 process in the free list. My test will pass, if control-p shows 2 processes is active and control-f shows 62 processes is in free list.

```
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
$
PID     Name    UID     GID     PPID    Elapsed CPU     State   Size    PCs
1       init    0       0       0       2.102   0.623   sleep   12288   80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       2.60    1.0     sleep   16384   80103de8 801002b1 80101914 80100f0b 80104d82 80104aa7 801059d0 801058ec
Free List Size:  62 processes
halt
Shutting down ...
make[1]: Leaving directory '/u/jiacheng/foo/xv6-pdx'
```

Figure: free list initialized

Based on my result, control-p shows 2 processes is active and control-f shows 62 processes is in free list. The free list is initialized.

This test PASSES.

Free list is updated when a process is allocated and deallocated.

I'm do the test to prove after a process is allocated and deallocated, the free is updated correctly.

I will boot xv6 and use control-p to show there are 2 active process initialized. And allocated one more process, do control-p again. Deallocated one process, do control-p. My test will pass, if after allocated one process control-p show one more process in it. And after deallocated one process control-p show one less processes in it.

If my test passed

```
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
$
PID     Name    UID     GID     PPID    Elapsed CPU     State   Size    PCs
1       init    0       0       0       32.337  0.538   sleep   12288   80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       32.301  0.868   sleep   16384   80103de8 801002b1 80101914 80100f0b 80104d82 80104aa7 801059d0 801058ec
Free List Size:  62 processes
sh
$
PID     Name    UID     GID     PPID    Elapsed CPU     State   Size    PCs
1       init    0       0       0       47.125  0.538   sleep   12288   80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       47.89   88.913  sleep   16384   80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
3       sh      0       0       0       3.106   88.70   sleep   16384   80103de8 801002b1 80101914 80100f0b 80104d82 80104aa7 801059d0 801058ec
Free List Size:  61 processes
kill 3
$
PID     Name    UID     GID     PPID    Elapsed CPU     State   Size    PCs
1       init    0       0       0       81.684  0.538   sleep   12288   80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       81.648  154.829 sleep   16384   80103de8 801002b1 80101914 80100f0b 80104d82 80104aa7 801059d0 801058ec
Free List Size:  62 processes
```

Figure: free list allocated and deallocated

Based on my result, control-p shows the correct way. After add one more sh in it. It shows one more sh. After kill one sh, it shows one less sh. The free list is updated.

This test PASSES.

Demonstrate that round-robin scheduling is maintained.

I'm do the test to prove after xv6 fully booted, the free list is correct initialized. The round-robin scheduling is that remove the first one from the front of ID list, and put it in other place.

I will boot xv6 and use control-r to test if its remove the first one from the front of ID list, and put it in other place. There are some Child process will be created and loop forever. My test will pass, when it remove the first one from the front of ID list, and put it in other place.

```
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
$ p3-test 7

---------- TEST 7 Round Robin Test ----------
Ready List Processes:
18->19->20->21->11->22->3->5->6->7->8->9->4->10->12->13->14->15
20 Child Processes Created and are looping forever. Parent is now sleeping for 30 sec. Use control-r rapidlyReady List Processes:
3->6->5->7->8->9->4->10->12->13->14->15->16->17->18->19->20->21->11
Ready List Processes:
7->8->9->4->10->12->13->14->15->16->17->18->19->20->21->11->22->23
Ready List Processes:
22->23->3->6->5->7->8->9->4->10->12->13->14->15->16->17->18->19->20
Ready List Processes:
17->18->19->20->21->11->22->23->3->6->5->7->8->9->4->10->12->13->14
Ready List Processes:
13->14->15->16->17->18->19->20->21->11->22->23->3->6->5->7->8->9->4
Ready List Processes:
17->Ready List Processes:
10->17->18->19->20->21->11->22->3->23->6->5->7->8->4->9->12->13->14
Ready List Processes:
9->12->13->14->15->16->10->17->18->19->20->21->11->22->3->23->6->5->7
Wait() has been called on Child Process.

---------- TEST 7 COMPLETE ----------
$ zombie!
```

Figure: round-robin scheduling test

Base on my result. It easily to see that for every ready process list, it remove the first one from the front of ID list, and put it in other place. The round-robin scheduling is maintained.

This test PASSES.

Sleep list is updated when a process sleeps and is woken up.

I'm do the test to prove after xv6 fully booted, the free list is correct initialized.

I will boot xv6 and use control-s to show the sleeping list. I will changed the state of on process from sleep to run. My test will pass, when the sleep list have same item with control-p.

```
$ Sleep List Processes:
1 -> 2
p3-test 6

---------- TEST 6 Sleep/Wake Test ----------
Child Process is now sleeping for 5 seconds. Use Control-p followed by Control-s within 5 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       536.195 0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       536.158 3123.492        sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
14      p3-test 0       0       0       1.769   472146.279      sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
15      p3-test 0       0       0       1.745   461457.890      sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
Sleep List Processes:
1 -> 2 -> 15 -> 14
Child Process is looping forever. Use Control-p and r to show this.
After 5 seconds, the process will be killed.

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       541.426 0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       541.389 3123.492        sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
14      p3-test 0       0       0       7.0     2486997.110     sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
15      p3-test 0       0       0       6.976   1448896.680     run     16384
Child Process 15 has been killed. Use control-p and z to show that its on the zombie list. You have 5 sec
Sleep List Processes:
1 -> 2 -> 14
Wait() has been called on Child Process 15. Use control-p, z, f to show that is removed from zombie list and added to unused.
You have 10 sec

---------- TEST 6 COMPLETE ----------
$ Sleep List Processes:
1 -> 2
```

Figure: sleep list test

Based on my result, at the beginning do this test, its have 4 sleep state processes. The sleep list show the 4 items. After changed one process to run state. The sleep list show 3 rest item in the list. The sleep list is updated.

This test PASSES.

Process Death

Subset1: kill system call causes a process to move to the zombie list.
I am do the test to prove a process will move to zombie list, when do kill system call.
Subset2: exit system call causes a process to move to the zombie list.
I am do the test to prove a process will move to zombie list, when do exit system call.
Subset3: wait system call causes a process to move to the free list.
I am do the test to prove a process will move to the free list, when do wait system call.

By using control-p to show how many processes are active. Using control-s to show sleep list, control-r to show the free list, control-z to show zombie list. My test will be passed, sleep lists have the same number of control-p shows, after kill and exit system call the zombie list should have the process be killed. After wait function has been called on child process, control-p should not show the Child be called by wait.

```
---------- TEST 5 Kill() and Wait() ----------
Child Process is looping forever. Use Control-p and z to show this.
After 5 seconds, the process will be killed.

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       26.78   0.490   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       26.41   55.313  sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
4       p3-test 0       0       0       3.542   85439.793       sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
5       p3-test 0       0       0       3.517   8536.320        run     16384
Zombie List Processes:
Nothing !!!
Child Process 5 has been killed. Use control-p and z to show that its on the zombie list. You have 5 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       32.594  0.490   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       32.557  55.313  sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
4       p3-test 0       0       0       10.58   276015.487      sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
5       p3-test 0       0       0       10.33   21282.567       zombie  16384
Zombie List Processes:
(5, 4)
Wait() has been called on Child Process 5. Use control-p, z, f to show that is removed from zombie list and added to unused.
You have 10 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       38.614  0.490   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       38.577  55.313  sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
4       p3-test 0       0       0       16.78   489535.816      sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
Zombie List Processes:
Nothing !!!
Free List Size:  61 processes
```

Figure: kill, exit, wait system call

```
---------- TEST 6 Sleep/Wake Test ----------
Child Process is now sleeping for 5 seconds. Use Control-p followed by Control-s within 5 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       714.383 0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       714.346 5105.522        sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
16      p3-test 0       0       0       2.133   758260.473      sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
17      p3-test 0       0       0       2.104   741876.703      sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
Sleep List Processes:
1 -> 2 -> 17 -> 16
Child Process is looping forever. Use Control-p and r to show this.
After 5 seconds, the process will be killed.

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       718.605 0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       718.568 5105.522        sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
16      p3-test 0       0       0       6.355   2731713.106     sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
17      p3-test 0       0       0       6.326   1881776.241     run     16384
Ready List Processes:
Nothing !!!
Child Process 17 has been killed. Use control-p and z to show that its on the zombie list. You have 5 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       721.430 0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       721.393 5105.522        sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
16      p3-test 0       0       0       9.180   457831.360      sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
17      p3-test 0       0       0       9.151   2003363.287     zombie  16384
Zombie List Processes:
(17, 16)
Wait() has been called on Child Process 17. Use control-p, z, f to show that is removed from zombie list and added to unused.
You have 10 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       726.555 0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       726.518 5105.522        sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
16      p3-test 0       0       0       14.305  4150164.510     sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
Zombie List Processes:
Nothing !!!
Free List Size:  61 processes

---------- TEST 6 COMPLETE ----------
$ Free List Size:  62 processes
```

Figure: kill, exit, wait system call

Once use kill it will go to zombie, and also has to use exit to go to zombie. When in the zombie list, the only way to unused is using wait. The result shows the same thing with excepted result. In test, Child process 17 is killed, then the zombie list shows it. The wait has been called on child process 17, then control-p not show process 17. Kill system call is work. Exit system call is work. Wait system call is work.

All three test PASSES.

Control-R Format

In this test, I verified that Control-R displays the PIDs of all the processes that are currently on the ready list. I update state of processes to runnable. My test will pass, when be updated by every fork call.

```
---------- TEST 4 Control-r ----------

Fork Call # 1
Child Process 1 is running for 50 seconds. Use Control-p followed by Control-r within 5 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       1658.607        0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       1658.570        27169.248       sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
34      p3-test 0       0       0       2.41    3328463.966     sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
35      p3-test 0       0       0       2.13    333177.600      run     16384
Ready List Processes:
Nothing !!!

Fork Call # 2
Child Process 2 is running for 50 seconds. Use Control-p followed by Control-r within 5 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       1662.259        0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       1662.222        27169.248       sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
34      p3-test 0       0       0       5.693   4098594.873     runble  16384
35      p3-test 0       0       0       5.665   939234.550      run     16384
36      p3-test 0       0       0       0.653   104702.181      run     16384
Ready List Processes:
34

Fork Call # 3
Child Process 3 is running for 50 seconds. Use Control-p followed by Control-r within 5 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       1667.500        0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       1667.463        27169.248       sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
34      p3-test 0       0       0       10.934  677689.596      sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
35      p3-test 0       0       0       10.906  1766616.521     run     16384
36      p3-test 0       0       0       5.894   962104.82       runble  16384
37      p3-test 0       0       0       0.871   131707.850      runble  16384
Ready List Processes:
36->34

Fork Call # 7
Child Process 7 is running for 50 seconds. Use Control-p followed by Control-r within 5 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       1688.491        0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       1688.454        27169.248       sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
34      p3-test 0       0       0       31.925  2593116.862     runble  16384
35      p3-test 0       0       0       31.897  3355845.332     runble  16384
36      p3-test 0       0       0       26.885  2556332.73      runble  16384
37      p3-test 0       0       0       21.862  1722595.921     runble  16384
38      p3-test 0       0       0       16.839  1125070.454     run     16384
39      p3-test 0       0       0       11.804  703175.250      runble  16384
40      p3-test 0       0       0       6.760   363968.510      run     16384
41      p3-test 0       0       0       1.704   81008.160       runble  16384
Ready List Processes:
41->38->40->34->39->37

Fork Call # 8
Child Process 8 is running for 50 seconds. Use Control-p followed by Control-r within 5 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       1693.37 0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       1693.0  27169.248       sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
34      p3-test 0       0       0       36.471  2831498.947     runble  16384
35      p3-test 0       0       0       36.443  3567184.142     runble  16384
36      p3-test 0       0       0       31.431  2767670.123     run     16384
37      p3-test 0       0       0       26.408  1933933.481     run     16384
38      p3-test 0       0       0       21.385  1338099.14      runble  16384
39      p3-test 0       0       0       16.350  916205.80       runble  16384
40      p3-test 0       0       0       11.306  576997.850      runble  16384
41      p3-test 0       0       0       6.250   292347.460      runble  16384
42      p3-test 0       0       0       1.210   49081.340       runble  16384
Ready List Processes:
35->41->38->42->34->40->39

Fork Call # 9
Child Process 9 is running for 50 seconds. Use Control-p followed by Control-r within 5 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       1698.769        0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       1698.732        27169.248       sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
34      p3-test 0       0       0       42.203  3074012.663     runble  16384
35      p3-test 0       0       0       42.175  3802905.512     runble  16384
36      p3-test 0       0       0       37.163  3003390.324     runble  16384
37      p3-test 0       0       0       32.140  2169653.461     runble  16384
38      p3-test 0       0       0       27.117  1572123.4       run     16384
39      p3-test 0       0       0       22.82   1150230.850     runble  16384
40      p3-test 0       0       0       17.38   811023.210      runble  16384
41      p3-test 0       0       0       11.982  526370.280      run     16384
42      p3-test 0       0       0       6.942   283105.540      runble  16384
43      p3-test 0       0       0       1.862   67914.600       runble  16384
Ready List Processes:
37->36->35->41->38->42->40->34
```

Figure 14: Control-p output

From the above figure, we can see that for all fork call, ready list is updated. That 34, 36, 41, 35 is the PID of the first process in the ready list (the head), and the last of list is the PID of last process in the list (the tail).

This test PASSES.

Control-F Format

In this test, I verified that Control-F displays the number of processes that are on the free list. I update state of processes to unused. My test will pass, when be updated by every fork call.

```
Fork Call # 1
Child Process 1 is now sleeping for 20 seconds. Use Control-p followed by Control-f within 5 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       1125.190        0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       1125.153        16676.34        sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
22      p3-test 0       0       0       2.402   1344297.7       sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
23      p3-test 0       0       0       2.374   1324087.311     sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
Free List Size:  60 processes

Fork Call # 2
Child Process 2 is now sleeping for 20 seconds. Use Control-p followed by Control-f within 5 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       1130.906        0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       1130.869        16676.34        sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
22      p3-test 0       0       0       8.118   3990021.328     sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
23      p3-test 0       0       0       8.90    3976578.885     sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
24      p3-test 0       0       0       3.79    1153092.892     sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
Free List Size:  59 processes

Fork Call # 3
Child Process 3 is now sleeping for 20 seconds. Use Control-p followed by Control-f within 5 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       1134.856        0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       1134.819        16676.34        sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
22      p3-test 0       0       0       12.68   995438.861      sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
23      p3-test 0       0       0       12.40   988794.321      sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
24      p3-test 0       0       0       7.29    2459143.567     sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
25      p3-test 0       0       0       2.16    566927.995      sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
Free List Size:  58 processes
Child Process 1 has exited.
Free List Size:  59 processes
Child Process 2 has exited.
Free List Size:  60 processes
Child Process 3 has exited.
```

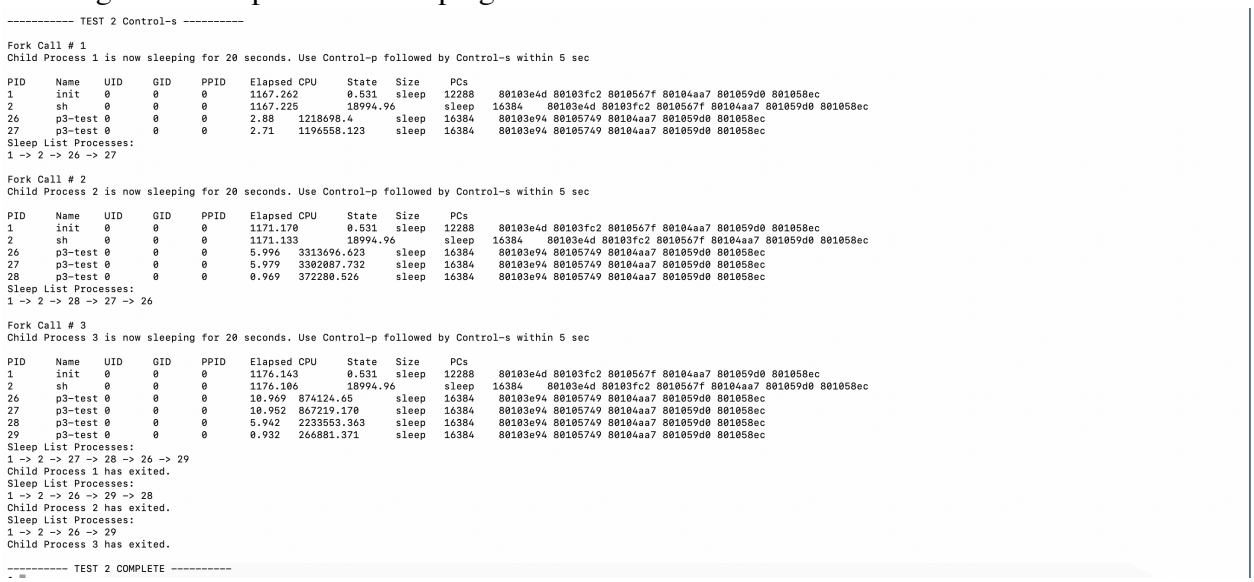
Figure 14: Control-f output

From the above figure, we can see that for process which changed to unused, free list is updated. 60, 59, 58 is the number of elements in the free list.

This test PASSES.

Control-S Format

In this test, I verified that Control-S displays the PIDs of all the processes that are currently on the zombie list as well as their parent PID. My test will pass, when be updated by every fork call to changed state of process to sleeping. Its format will be similar to control – r above.



```
---------- TEST 2 Control-s ----------

Fork Call # 1
Child Process 1 is now sleeping for 20 seconds. Use Control-p followed by Control-s within 5 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       1167.262        0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       1167.225        18994.96        sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
26      p3-test 0       0       0       2.88    1218698.4       sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
27      p3-test 0       0       0       2.71    1196558.123     sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
Sleep List Processes:
1 -> 2 -> 26 -> 27

Fork Call # 2
Child Process 2 is now sleeping for 20 seconds. Use Control-p followed by Control-s within 5 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       1171.170        0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       1171.133        18994.96        sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
26      p3-test 0       0       0       5.996   3313696.623     sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
27      p3-test 0       0       0       5.979   3302087.732     sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
28      p3-test 0       0       0       0.969   372280.526      sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
Sleep List Processes:
1 -> 2 -> 28 -> 27 -> 26

Fork Call # 3
Child Process 3 is now sleeping for 20 seconds. Use Control-p followed by Control-s within 5 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       1176.143        0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       1176.106        18994.96        sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
26      p3-test 0       0       0       10.969  874124.65       sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
27      p3-test 0       0       0       10.952  867219.170      sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
28      p3-test 0       0       0       5.942   2233553.363     sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
29      p3-test 0       0       0       0.932   266881.371      sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
Sleep List Processes:
1 -> 2 -> 27 -> 28 -> 26 -> 29
Child Process 1 has exited.
Sleep List Processes:
1 -> 2 -> 26 -> 29 -> 28
Child Process 2 has exited.
Sleep List Processes:
1 -> 2 -> 26 -> 29
Child Process 3 has exited.

---------- TEST 2 COMPLETE ----------
```

Figure 14: Control-s output

From the above figure, we can see that for all fork call which changed state of process to sleeping, sleep list is updated.

This test PASSES.

Control-Z Format

In this test, I verified that Control-P displays the PIDs of all the processes that are currently on the zombie list as well as their parent PID. Its format will be similar to control – r above. My test will pass, when zombie list is updated follow state of process change to zombie.

```
---------- TEST 3 Control-z ----------

Fork Call # 1
Child Process 1 has exited. Use Control-p followed by Control-z within 5 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       1217.318        0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       1217.281        22620.469       sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
30      p3-test 0       0       0       1.772   2112970.306     sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
31      p3-test 0       0       0       1.742   2431.188        zombie  16384
Zombie List Processes:
(31, 30)

Fork Call # 2
Child Process 2 has exited. Use Control-p followed by Control-z within 5 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       1221.353        0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       1221.316        22620.469       sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
30      p3-test 0       0       0       5.807   2728259.96      sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
31      p3-test 0       0       0       5.777   2431.188        zombie  16384
32      p3-test 0       0       0       0.768   2441.204        zombie  16384
Zombie List Processes:
(31, 30)->(32, 30)

Fork Call # 3
Child Process 3 has exited. Use Control-p followed by Control-z within 5 sec

PID     Name    UID     GID     PPID    Elapsed CPU     State   Size     PCs
1       init    0       0       0       1226.322        0.531   sleep   12288    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
2       sh      0       0       0       1226.285        22620.469       sleep   16384    80103e4d 80103fc2 8010567f 80104aa7 801059d0 801058ec
30      p3-test 0       0       0       10.776  207319.576      sleep   16384    80103e94 80105749 80104aa7 801059d0 801058ec
31      p3-test 0       0       0       10.746  2431.188        zombie  16384
32      p3-test 0       0       0       5.737   2441.204        zombie  16384
33      p3-test 0       0       0       0.728   3676.825        zombie  16384
Zombie List Processes:
(31, 30)->(32, 30)->(33, 30)
Wait() has been called on Child Process 1.
Wait() has been called on Child Process 2.
Wait() has been called on Child Process 3.

---------- TEST 3 COMPLETE ----------
```

Figure 14: Control-z output

From the above figure, we can see that 31 is the PID of the first process on the list (the head) and 30 is the PID of the parent of process 1. The PID of the last process on the list (the tail) is 33 and 30 is the PID of parent process of process n. The zombie list is updated.

This test PASSES.