Project One Report
Introduction to Operating Systems
New Beginnings Spring 2018

Gavin Megson

7 April 2018

Description

For this assignment, I learned about the flow of control for system calls in xv6; how to add a new system call; how to access specific information for each active process; and how to use conditional compilation to enable and disable kernel features.

Deliverables

The following features were added to xv6:

- A system call tracing facility that, when enabled, prints the following information to the console:

  ```
  <system call name> -> <system call return code>
  ```

- A new system call, `date()`, that returns the current UTC date and time.
- A new user command, `date`, that prints the current UTC date and time to standard output.
- Each process now records the value of the `ticks` global variable when that process is created. This value is used to calculate *elapsed time* for each process.
- A modification to the existing control–p mechanism, which displays debugging information, to include elapsed time for each process.

Implementation

System Call Tracing Facility

All the code for the system call tracing facility was conditionally compiled using the `PRINT_SYSCALLS` flag in the `Makefile` (line 07). The implementation modified `syscall.c` as follows:

- Lines 137 – 166 define an array of system call names, `syscallnames[]`, indexed by system call number as defined in `syscall.h`.
- Lines 178 – 184 prints the name of the system call and the corresponding return value.

Date System Call

The following files were modified to add the `date()` system call.

- `user.h`. The user-side function prototype for the `date()` system call was added (line 27). The system call takes a pointer to a user-defined `rtctime struct`. The prototype is:
  `int date(struct rtcdate*);`
  The file `date.h` contains the `rtcdate` definition.
- `syscall.h`. The `date()` system call number was created by appending to the existing list (line 25).
- `syscall.c`. Modified to include the kernel-side function prototype (line 102); an entry in the function dispatch table `syscalls[]` (line 129); and an entry into the `syscallnames[]` array to print the system call name when the `PRINT_SYSCALLS` flag is defined. All prototypes here are defined as taking a *void* parameter as the function call arguments are passed into the kernel on the stack. Each implementation (e.g., `sys_date()`) retrieves the arguments from the stack according to the syntax of the system call.

- usys.S. The user-side stub for the new system call was added (line 33). This stub uses a macro that essentially just traps into kernel-mode.
- sysproc.c. Contains the kernel-side implementation of the system call in sys_date() (lines 96 – 108). This routine removes the pointer argument from the stack and passes it to the existing routine cmostime() in lapic.c (line 206). The pointer argument is expected to be a struct rtctime*. The routine cmostime() cannot fail so a success code is returned by sys_date().

Date User Command

The date user command is implemented in the file date.c. This command invokes the new date() system call to fill in the supplied rtcdate struct; passed by reference. The command then displays the date and time information on standard output. The return code from the system call is checked and handled as a user program does not know if a system call will succeed or fail.

control–p Modifications and Elapsed Time

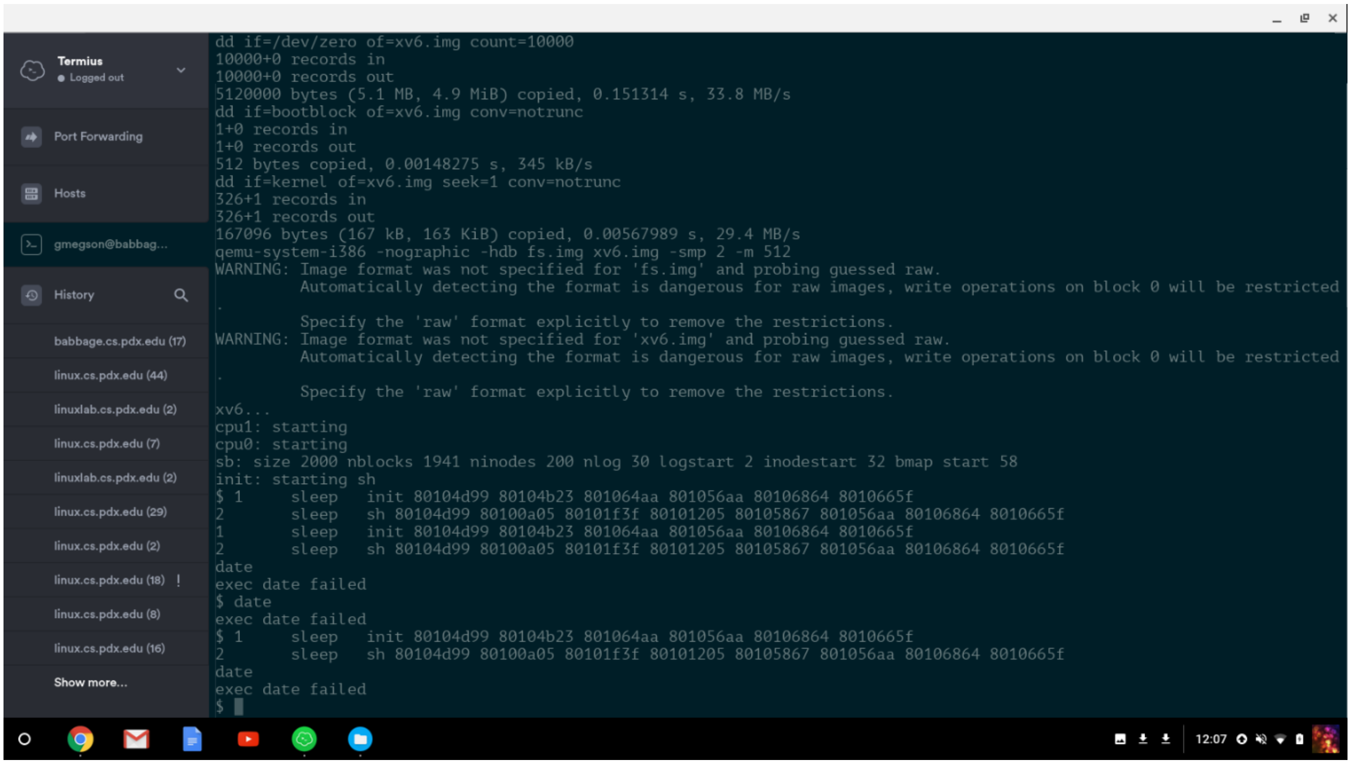
The control–p console command prints debugging information to the console. The following modifications were made to capture and display elapsed time as part of the existing control–p debugging information.

- proc.h. A new field was added to struct proc named uint start_ticks for storing the time of creation (in *ticks*) for each process (line 69).
- proc.c. The routine allocproc() (line 42) was modified to correctly set start_ticks on process creation.
- procdump(). This routine in proc.c was modified to:
  - Print a header (line 512) to the console.
  - Calculate the *elapsed time* since process creation (lines 528 – 530). This section calculates elapsed time as seconds and hundredths of seconds as the granularity of the ticks variable is at thousandths of a second.
  - Include the elapsed time in the display of process information on the console (line 531).
  - Minor changes to the formatting of the existing information displayed (line 524)

Testing

Previous Output

For reference, the output when the project flag is set to 0 is included here:

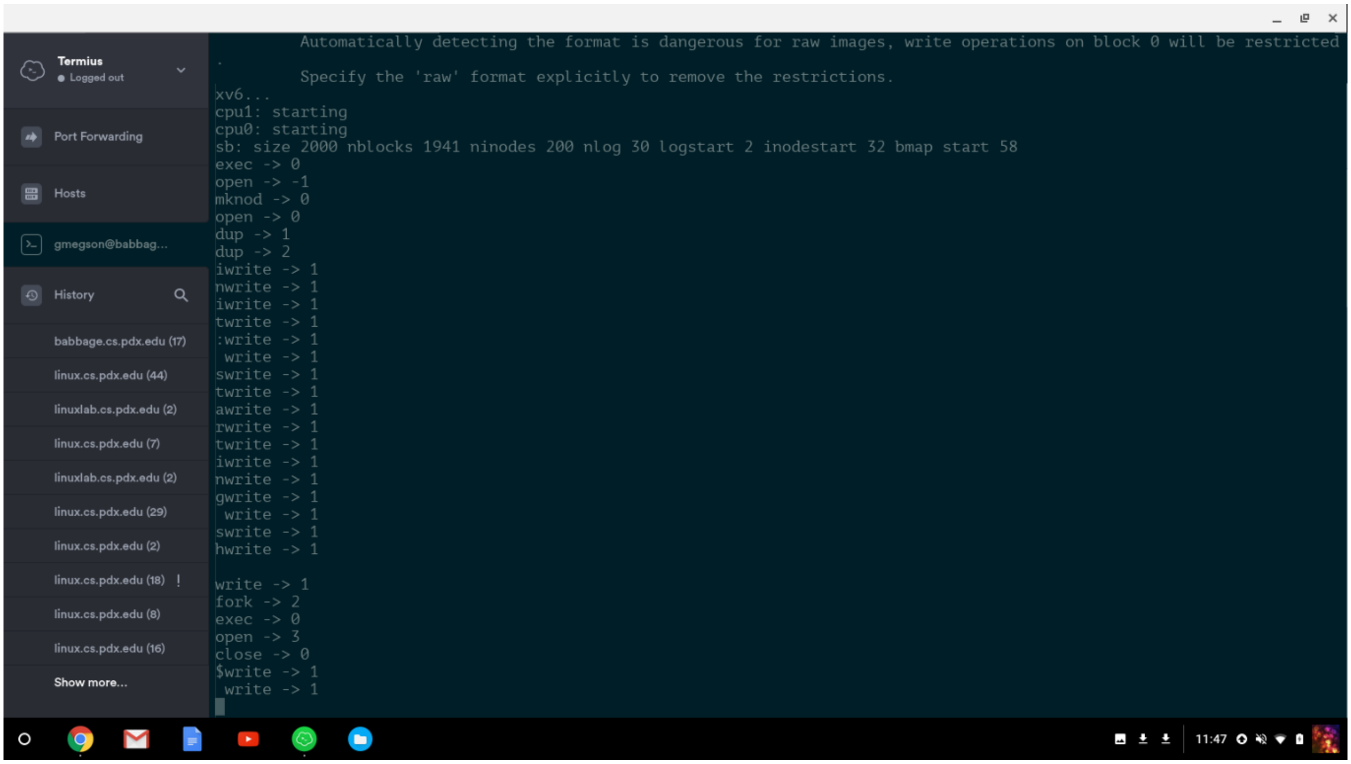


```
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB, 4.9 MiB) copied, 0.151314 s, 33.8 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.00148275 s, 345 kB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
326+1 records in
326+1 records out
167096 bytes (167 kB, 163 KiB) copied, 0.00567989 s, 29.4 MB/s
qemu-system-i386 -nographic -hdb fs.img xv6.img -smp 2 -m 512
WARNING: Image format was not specified for 'fs.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted
.
         Specify the 'raw' format explicitly to remove the restrictions.
WARNING: Image format was not specified for 'xv6.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted
.
         Specify the 'raw' format explicitly to remove the restrictions.
xv6...
cpu1: starting
cpu0: starting
sb: size 2000 nblocks 1941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ 1     sleep   init 80104d99 80104b23 801064aa 801056aa 80106864 8010665f
2       sleep   sh 80104d99 80100a05 80101f3f 80101205 80105867 801056aa 80106864 8010665f
1       sleep   init 80104d99 80104b23 801064aa 801056aa 80106864 8010665f
2       sleep   sh 80104d99 80100a05 80101f3f 80101205 80105867 801056aa 80106864 8010665f
date
exec date failed
$ date
exec date failed
$ 1     sleep   init 80104d99 80104b23 801064aa 801056aa 80106864 8010665f
2       sleep   sh 80104d99 80100a05 80101f3f 80101205 80105867 801056aa 80106864 8010665f
date
exec date failed
$ 
```

Figure 1: Original Output

System Call Tracing Facility

I tested this feature by modifying the `Makefile` to turn on `PRINT_SYSCALLS` flag, then booting the xv6 kernel, and observing the following output:



```
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted
.
         Specify the 'raw' format explicitly to remove the restrictions.
xv6...
cpu1: starting
cpu0: starting
sb: size 2000 nblocks 1941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
exec -> 0
open -> -1
mknod -> 0
open -> 0
dup -> 1
dup -> 2
iwrite -> 1
nwrite -> 1
iwrite -> 1
twrite -> 1
:write -> 1
 write -> 1
swrite -> 1
twrite -> 1
awrite -> 1
rwrite -> 1
twrite -> 1
iwrite -> 1
nwrite -> 1
gwrite -> 1
 write -> 1
swrite -> 1
hwrite -> 1

write -> 1
fork -> 2
exec -> 0
open -> 3
close -> 0
$write -> 1
 write -> 1
```

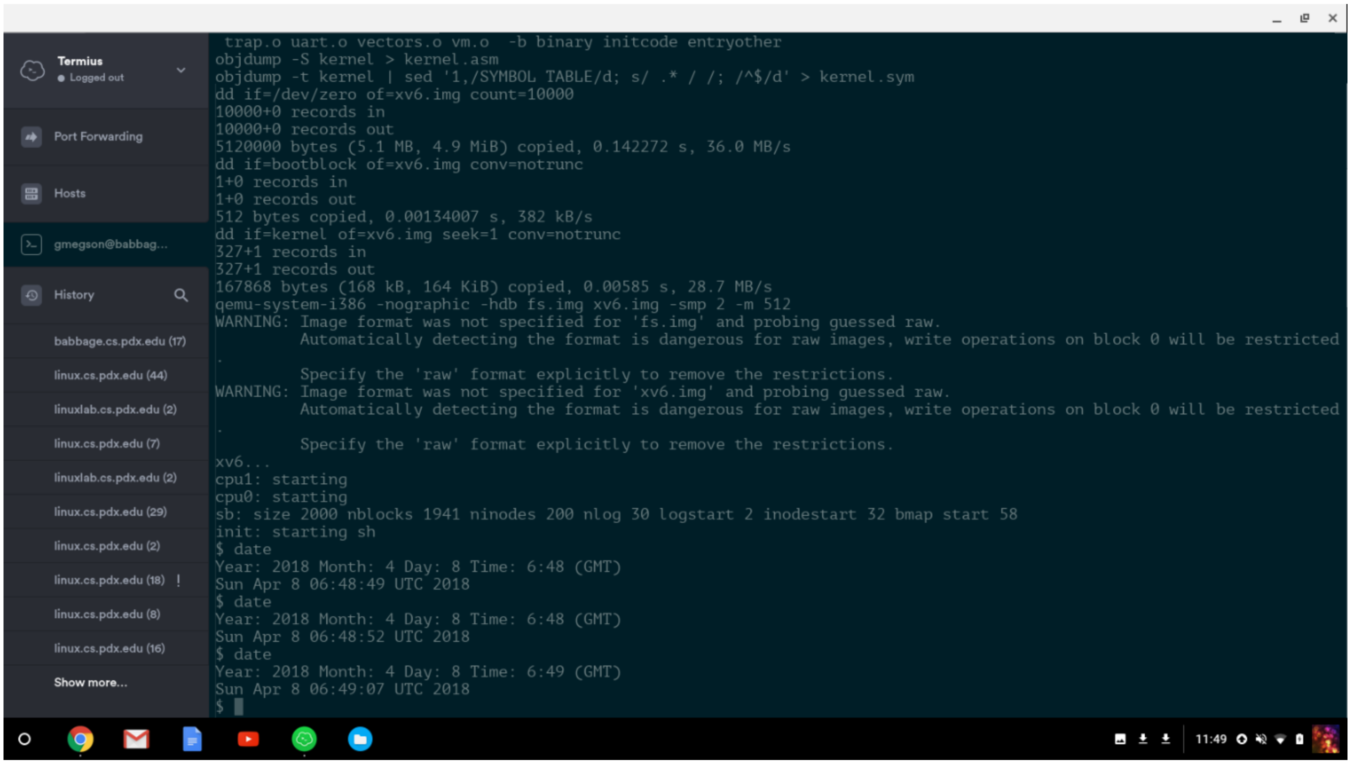
Figure 2: System Call Tracing Facility

The system call trace correctly displays invoked system calls. The standard output is interleaved with the trace output. The output "init: starting sh" is printed by the `init()` process (`init.c`) and the "$" is printed by the shell process (`sh.c`).

This test `PASSES`.

Date System Call and User Command

I am going to use the `date` command to test both the `date()` system call and date command, as I can't directly invoke a system call from the shell. My testing will invoke my `date` command in xv6 and then invoke the corresponding Linux `date` command and see if the former closely matches the latter.



```
 trap.o uart.o vectors.o vm.o  -b binary initcode entryother
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, 0.142272 s, 36.0 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.00134007 s, 382 kB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
327+1 records in
327+1 records out
167868 bytes (168 kB, 164 KiB) copied, 0.00585 s, 28.7 MB/s
qemu-system-i386 -nographic -hdb fs.img xv6.img -smp 2 -m 512
WARNING: Image format was not specified for 'fs.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted
.
         Specify the 'raw' format explicitly to remove the restrictions.
WARNING: Image format was not specified for 'xv6.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted
.
         Specify the 'raw' format explicitly to remove the restrictions.
xv6...
cpu1: starting
cpu0: starting
sb: size 2000 nblocks 1941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ date
Year: 2018 Month: 4 Day: 8 Time: 6:48 (GMT)
Sun Apr 8 06:48:49 UTC 2018
$ date
Year: 2018 Month: 4 Day: 8 Time: 6:48 (GMT)
Sun Apr 8 06:48:52 UTC 2018
$ date
Year: 2018 Month: 4 Day: 8 Time: 6:49 (GMT)
Sun Apr 8 06:49:07 UTC 2018
$ 
```

Figure 3: Date Test

The output from my `date` command closely matches the output of the Linux `date` command, except for a discrepancy in the precision. The struct rtcdate does not have fields more precise than seconds. This test shows that the `date` command works correctly, along with the date system call, since the command prints out all of the information extracted by the system call.

This test `PASSES`.

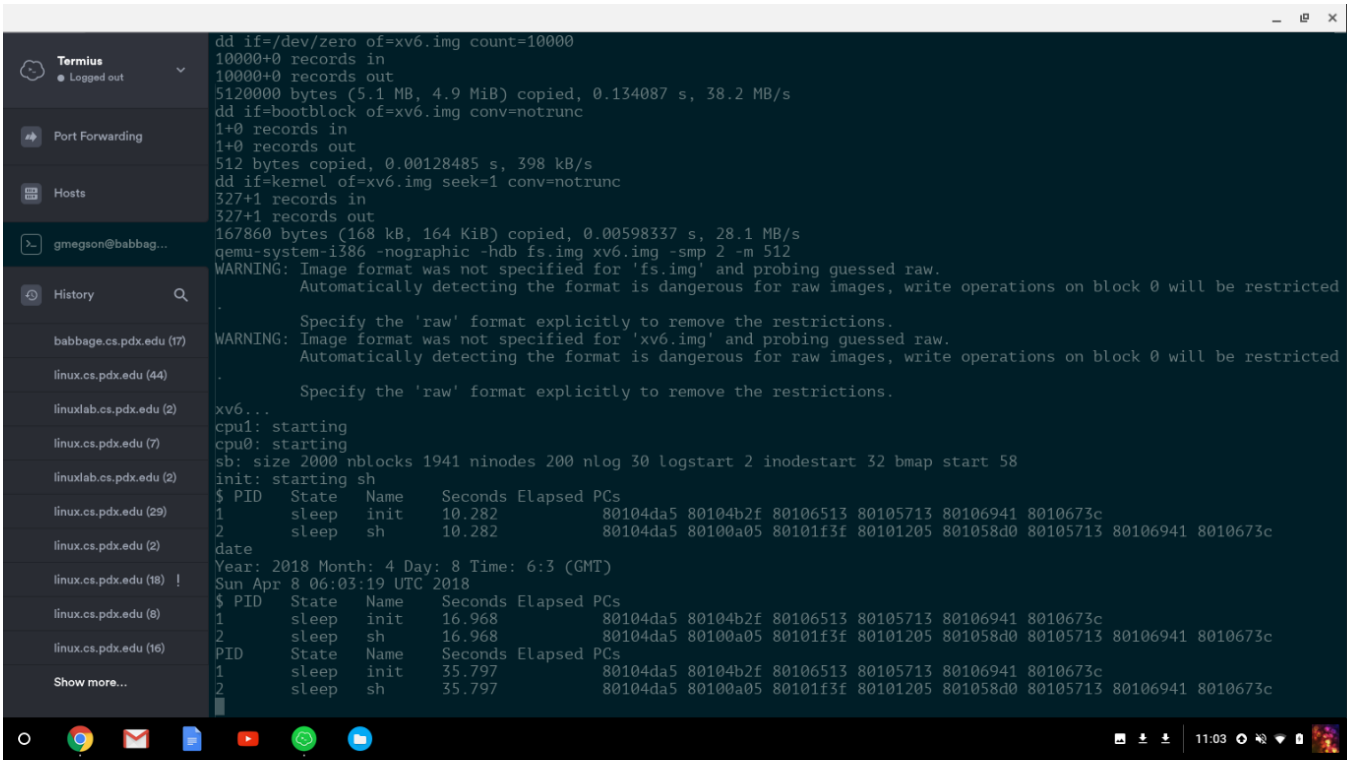
control–p and Elapsed Time

The test for these will be split into two phases. My first test will show that control–p is outputting the correct information, while my second test will use the first test to show that the elapsed time is correct.

Here is the output of the first test:

The control–p output indicates that there are two processes running in xv6. This is correct. The first process is the initial process, here named "init", with a PID of 1. The second process is the shell, named "sh", with a PID of 2 (as it is the second process created). Note that the PCs appear to be correct, as they correspond to valid addresses in the kernel.asm file and the code for printing this information was not modified.

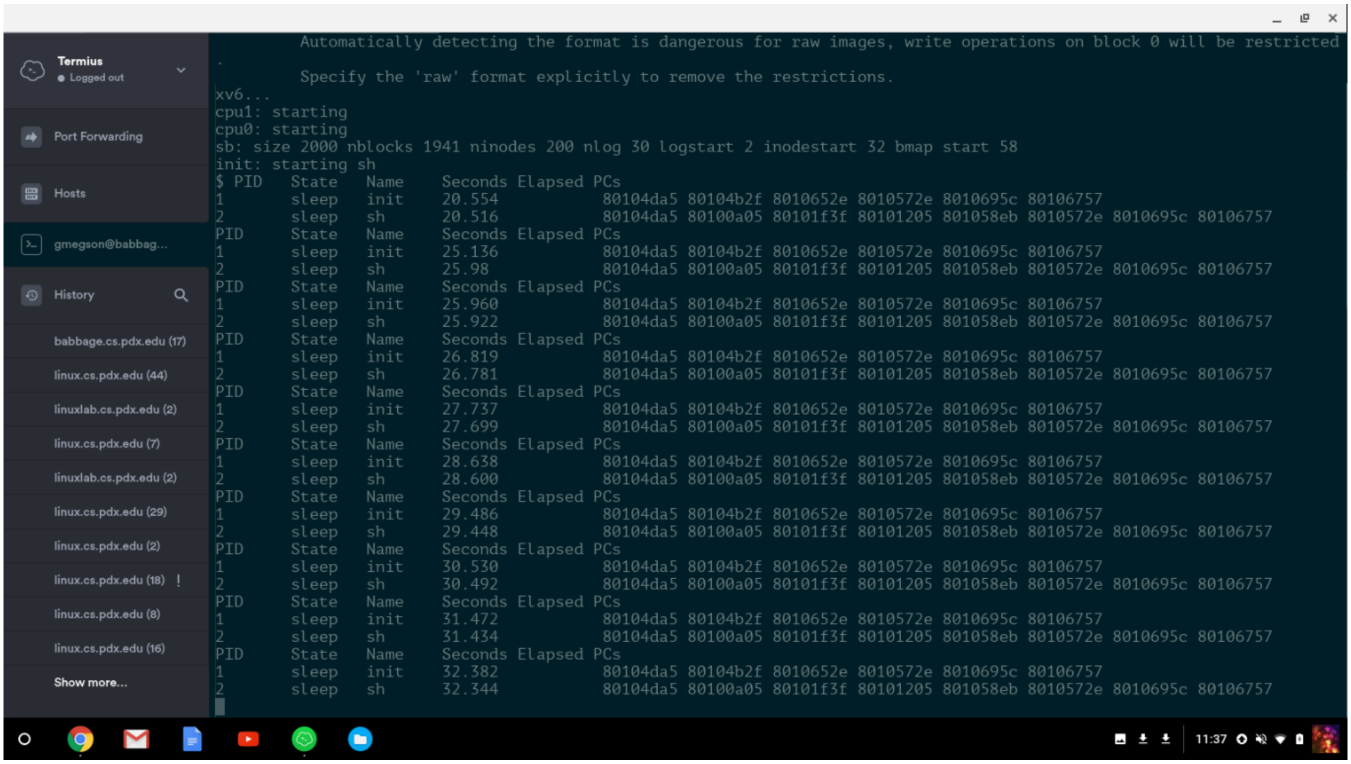
This sub-test `PASSES`.



```
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, 0.134087 s, 38.2 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.00128485 s, 398 kB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
327+1 records in
327+1 records out
167860 bytes (168 kB, 164 KiB) copied, 0.00598337 s, 28.1 MB/s
qemu-system-i386 -nographic -hdb fs.img xv6.img -smp 2 -m 512
WARNING: Image format was not specified for 'fs.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted
.
         Specify the 'raw' format explicitly to remove the restrictions.
WARNING: Image format was not specified for 'xv6.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted
.
         Specify the 'raw' format explicitly to remove the restrictions.
xv6...
cpu1: starting
cpu0: starting
sb: size 2000 nblocks 1941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ PID   State   Name    Seconds Elapsed PCs
1       sleep   init    10.282          80104da5 80104b2f 80106513 80105713 80106941 8010673c
2       sleep   sh      10.282          80104da5 80100a05 80101f3f 80101205 801058d0 80105713 80106941 8010673c
date
Year: 2018 Month: 4 Day: 8 Time: 6:3 (GMT)
Sun Apr 8 06:03:19 UTC 2018
$ PID   State   Name    Seconds Elapsed PCs
1       sleep   init    16.968          80104da5 80104b2f 80106513 80105713 80106941 8010673c
2       sleep   sh      16.968          80104da5 80100a05 80101f3f 80101205 801058d0 80105713 80106941 8010673c
PID     State   Name    Seconds Elapsed PCs
1       sleep   init    35.797          80104da5 80104b2f 80106513 80105713 80106941 8010673c
2       sleep   sh      35.797          80104da5 80100a05 80101f3f 80101205 801058d0 80105713 80106941 8010673c
```

Figure 4: Control–p Test

For the second test, I will restart the kernel, and then press control–p several times, each press being within one second of the other. The results are shown below:



```
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted
.
         Specify the 'raw' format explicitly to remove the restrictions.
xv6...
cpu1: starting
cpu0: starting
sb: size 2000 nblocks 1941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ PID   State   Name    Seconds Elapsed PCs
1       sleep   init    20.554          80104da5 80104b2f 8010652e 8010572e 8010695c 80106757
2       sleep   sh      20.516          80104da5 80100a05 80101f3f 80101205 801058eb 8010572e 8010695c 80106757
PID     State   Name    Seconds Elapsed PCs
1       sleep   init    25.136          80104da5 80104b2f 8010652e 8010572e 8010695c 80106757
2       sleep   sh      25.98           80104da5 80100a05 80101f3f 80101205 801058eb 8010572e 8010695c 80106757
PID     State   Name    Seconds Elapsed PCs
1       sleep   init    25.960          80104da5 80104b2f 8010652e 8010572e 8010695c 80106757
2       sleep   sh      25.922          80104da5 80100a05 80101f3f 80101205 801058eb 8010572e 8010695c 80106757
PID     State   Name    Seconds Elapsed PCs
1       sleep   init    26.819          80104da5 80104b2f 8010652e 8010572e 8010695c 80106757
2       sleep   sh      26.781          80104da5 80100a05 80101f3f 80101205 801058eb 8010572e 8010695c 80106757
PID     State   Name    Seconds Elapsed PCs
1       sleep   init    27.737          80104da5 80104b2f 8010652e 8010572e 8010695c 80106757
2       sleep   sh      27.699          80104da5 80100a05 80101f3f 80101205 801058eb 8010572e 8010695c 80106757
PID     State   Name    Seconds Elapsed PCs
1       sleep   init    28.638          80104da5 80104b2f 8010652e 8010572e 8010695c 80106757
2       sleep   sh      28.600          80104da5 80100a05 80101f3f 80101205 801058eb 8010572e 8010695c 80106757
PID     State   Name    Seconds Elapsed PCs
1       sleep   init    29.486          80104da5 80104b2f 8010652e 8010572e 8010695c 80106757
2       sleep   sh      29.448          80104da5 80100a05 80101f3f 80101205 801058eb 8010572e 8010695c 80106757
PID     State   Name    Seconds Elapsed PCs
1       sleep   init    30.530          80104da5 80104b2f 8010652e 8010572e 8010695c 80106757
2       sleep   sh      30.492          80104da5 80100a05 80101f3f 80101205 801058eb 8010572e 8010695c 80106757
PID     State   Name    Seconds Elapsed PCs
1       sleep   init    31.472          80104da5 80104b2f 8010652e 8010572e 8010695c 80106757
2       sleep   sh      31.434          80104da5 80100a05 80101f3f 80101205 801058eb 8010572e 8010695c 80106757
PID     State   Name    Seconds Elapsed PCs
1       sleep   init    32.382          80104da5 80104b2f 8010652e 8010572e 8010695c 80106757
2       sleep   sh      32.344          80104da5 80100a05 80101f3f 80101205 801058eb 8010572e 8010695c 80106757
```

Figure 5: Elapsed Time Test

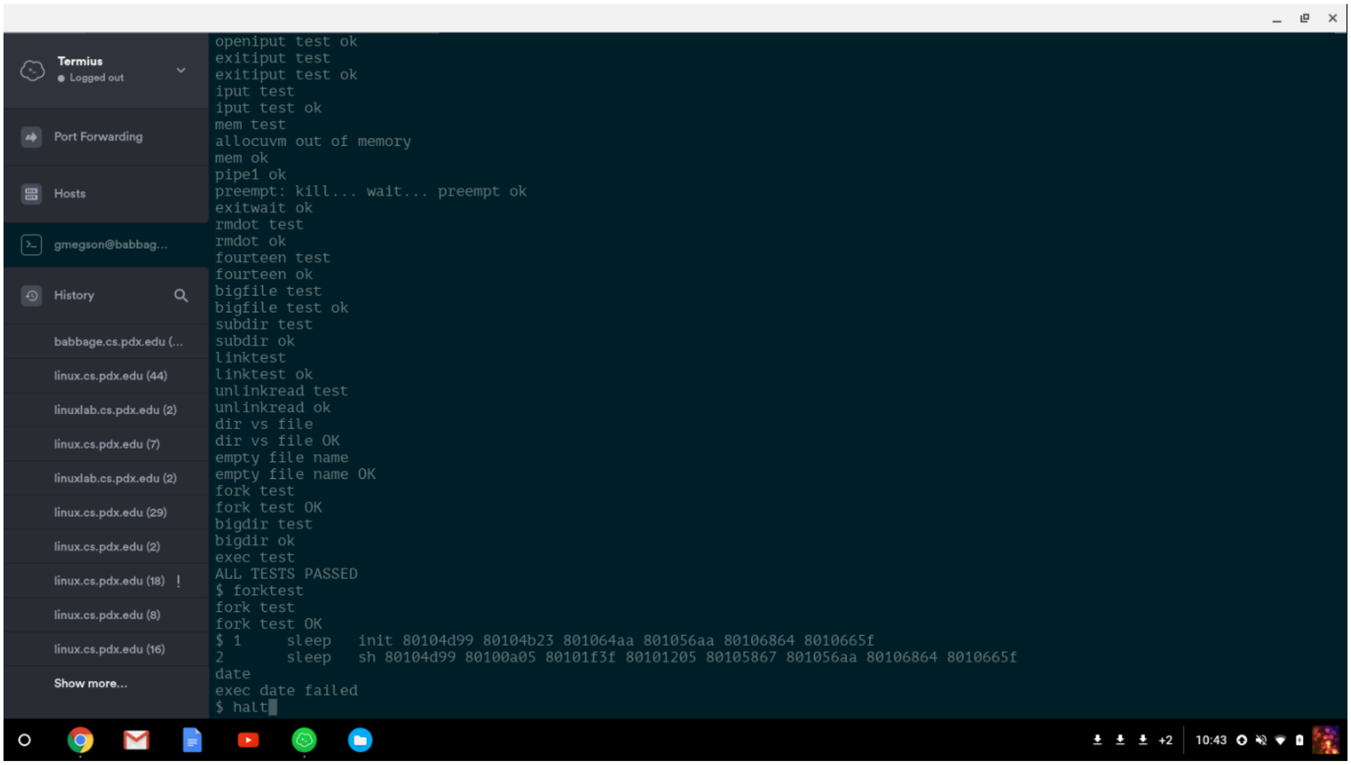
The elapsed time for the init process is 0.38 seconds higher than that of the sh process in all outputs. This makes sense as init starts before sh. Also, note that the elapsed times are steadily increasing by about one second with the same 0.38 s difference between init and sh.

This sub-test `PASSES`.

Because all sub-tests passed, this test `PASSES`.

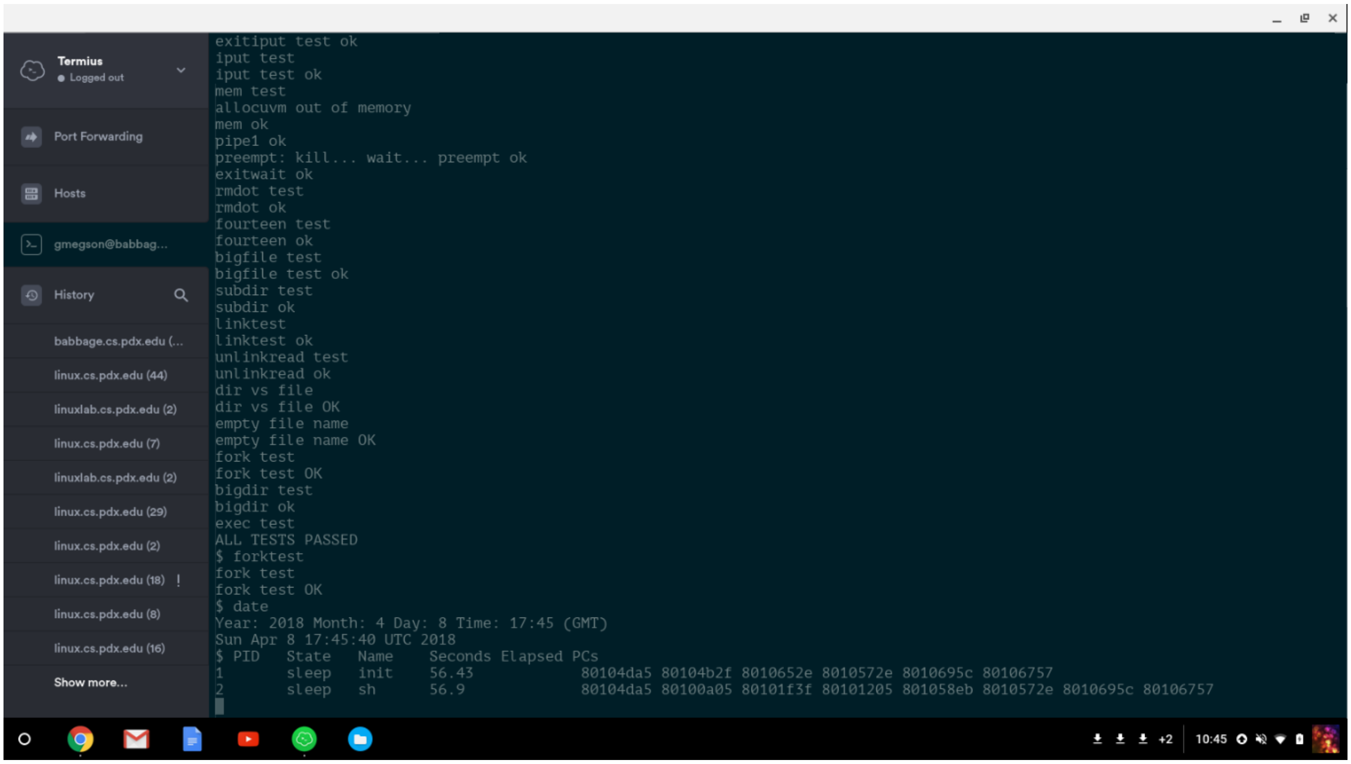
Reference: Usertests and Forktest

Inluded here are screenshots of the usertests and forktest outputs.



```
openiput test ok
exitiput test
exitiput test ok
iput test
iput test ok
mem test
allocuvm out of memory
mem ok
pipe1 ok
preempt: kill... wait... preempt ok
exitwait ok
rmdot test
rmdot ok
fourteen test
fourteen ok
bigfile test
bigfile test ok
subdir test
subdir ok
linktest
linktest ok
unlinkread test
unlinkread ok
dir vs file
dir vs file OK
empty file name
empty file name OK
fork test
fork test OK
bigdir test
bigdir ok
exec test
ALL TESTS PASSED
$ forktest
fork test
fork test OK
$ 1      sleep   init 80104d99 80104b23 801064aa 801056aa 80106864 8010665f
2        sleep   sh 80104d99 80100a05 80101f3f 80101205 80105867 801056aa 80106864 8010665f
date
exec date failed
$ halt
```

Figure 6: Project Flag to 0



```
exitiput test ok
iput test
iput test ok
mem test
allocuvm out of memory
mem ok
pipe1 ok
preempt: kill... wait... preempt ok
exitwait ok
rmdot test
rmdot ok
fourteen test
fourteen ok
bigfile test
bigfile test ok
subdir test
subdir ok
linktest
linktest ok
unlinkread test
unlinkread ok
dir vs file
dir vs file OK
empty file name
empty file name OK
fork test
fork test OK
bigdir test
bigdir ok
exec test
ALL TESTS PASSED
$ forktest
fork test
fork test OK
$ date
Year: 2018 Month: 4 Day: 8 Time: 17:45 (GMT)
Sun Apr 8 17:45:40 UTC 2018
$ PID   State   Name    Seconds Elapsed PCs
1       sleep   init    56.43           80104da5 80104b2f 8010652e 8010572e 8010695c 80106757
2       sleep   sh      56.9            80104da5 80100a05 80101f3f 80101205 801058eb 8010572e 8010695c 80106757
```

Figure 7: Project Flag to 1