

Description

In this assignment, I learned how the system calls flow in xv6, how to use preprocessor directives to control conditional compilation for kernel features, how to trace system call, and how to add a new system call.

Deliverables

The following features were added into the system of xv6:

1. A system call tracing facility that prints out the system call name and system call return code when enabled in the following format:
<system call name> → <system call return code>
2. A new system call called date() was added and it returns the current date and time in UTC format.
3. A new user command called date was added and it prints the current date and time in UTC format.
4. Processes are now recorded using the ticks global variable when that process is initiated. The value of ticks is used to calculate for the Elapsed time for each process.
5. Control-p sequence was modified to display the additional elapsed time for each process on the console.

Implementation

System Call Tracing

All the code for system call tracing was wrapped around preprocessor directives and can only be conditionally compiled using PRINT_SYSCALLS flag in the Makefile (line 7). The implementation that modified the file syscall.c as follows:

- Lines 136 – 162 define names of system calls in an array called syscallnames[] that are indexed by system call number as defined in syscall.h.
- Lines 173 – 176 print the name of the system call and its corresponding return value.

System Call date()

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

- user.h – The prototype function for the date() system call for user-side was added (line 28). The system call takes a pointer to a user-defined rtcdate struct. The prototype is:
int date(struct rtcdate*);
The definition of the rtc structure was defined in date.h file.

- `syscall.h` – The system call number for the `date()` system call was added by appending to the existing list (line 25).
- `syscall.c` – modified code included kernel-side function prototype (line 102 – 104), an entry in the function dispatch table `syscalls[]` (line 129 – 131), and an entry to the `syscallnames[]` array to print out the system call name and return code on the console when the `PRINT_SYSCALLS` flag is turned on. The system call prototypes take void parameters because function call arguments are passed into the kernel on the stack. Each defined function such as `sys_date()` gets 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). As we can see, there are no preprocessor directives used in this file, as it is an assembly file hence it will be assembled instead of being compiled when we run the program. This stub uses a macro that traps it into kernel-mode.
- `sysproc.c` – Contains the implementation of kernel-side system call `sys_date()` (line 97 – 108). This routine removes pointer argument from the stack and passes it to the existing routine `cmostime()` (line 105). The `cmostime()` routine is defined in `lapic.c` (line 206 – 239). The pointer argument is expected to be a `struct rtctime*`. The routine `cmostime()` cannot fail so a success code is returned by `sys_date()`.

The date User Command

The implementation of the date user command is in the file `date.c`. This command invokes the `date()` system call to filled in the supplied `rtctime` struct, passed by reference. The command will then print the date and time information in UTC on the console. The return code of the system call is checked and handled as the user program does not know if a system call will succeed or fail.

Control – P Modifications and Elapsed Time

The `control – p` command prints debugging information to the console. The modifications made in the following files were to display elapsed time as part of the already existed `control – p` debugging information.

- `proc.h` – A new field called `unit start_ticks` was added to the struct `proc` to store time of creation (in ticks) for each process (line 69).
- `proc.c` – The function `allocproc()` (line 47) was modified (line 85 – 87) to correctly set `start_ticks` on process creation.
- `Procdump()` – This function in `proc.c` was modified to:
 - Print a header (line 518) to the console.
 - Calculate the elapsed time since process creation (line 532 – 535). This section calculates elapsed time as seconds, hundredths of seconds, and

thousandths of seconds as the granularity of the ticks variable is at hundredths of seconds.

- Include the elapsed time in the display of information on the console (line 536)

Testing

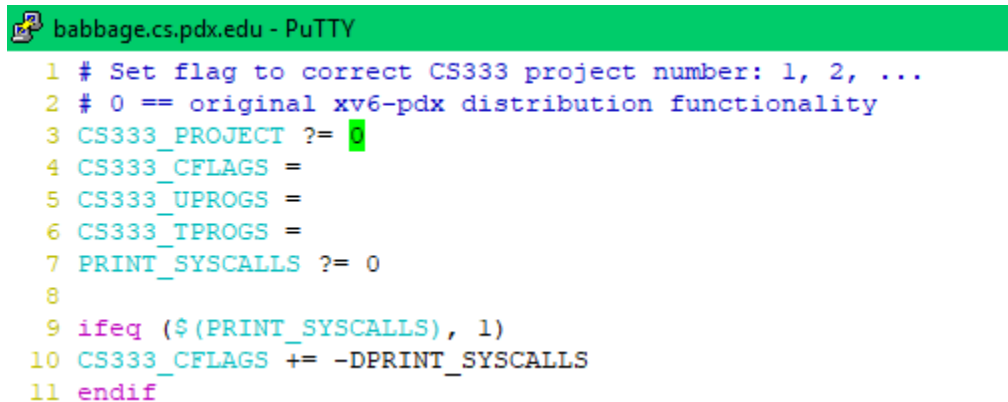
With PRINT_SYSCALLS flag turned off

This test is to verify that my kernel compiles correctly with PRINT_SYSCALLS turned off by set the flag to 0 in the Makefile. This test includes two additional subtests:

1. Kernel compiles correctly and boots with CS333_P1 flag turned off by set CS333_PROJECT to 0.
2. Kernel compiles correctly and boots with CS333_P1 flag turned on by set CS333_PROJECT to 1.

Since the PRINT_SYSCALLS causes all system calls to be displayed along with their return code to the console, booting to the shell is enough to prove that they will work since that process will invoke many system calls. With this, information of the system calls is not expected to be printed to the console.

Subtest 1: CS333_PROJECT and PRINT_SYSCALLS are set to 0.



```
babbage.cs.pdx.edu - PuTTY
1 # Set flag to correct CS333 project number: 1, 2, ...
2 # 0 == original xv6-pdx distribution functionality
3 CS333_PROJECT ?= 0
4 CS333_CFLAGS =
5 CS333_UPROGS =
6 CS333_TPROGS =
7 PRINT_SYSCALLS ?= 0
8
9 ifeq ($(PRINT_SYSCALLS), 1)
10 CS333_CFLAGS += -DPRINT_SYSCALLS
11 endif
```

Figure 1: Makefile with CS333_PROJECT and PRINT_SYSCALLS set to 0

```

thon@babbage:~/CS333/xv6-pdx$ make qemu-nox
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, 0.141237 s, 36.3 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.0012164 s, 421 kB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
326+1 records in
326+1 records out
167248 bytes (167 kB, 163 KiB) copied, 0.00639294 s, 26.2 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
$ █

```

Figure 2: Boot with CS333_PROJECT and PRINT_SYSCALLS set to 0

No system information displayed when boot with CS333_PROJECT and PRINT_SYSCALLS set to 0 as expected hence this subtest **PASSES** and is a **SUCCESS**.

Subtest 2: CS333_PROJECT set to 1 and PRINT_SYSCALLS are set to 0.

```

1 # Set flag to correct CS333 project number: 1, 2, ...
2 # 0 == original xv6-pdx distribution functionality
3 CS333_PROJECT ?= 1
4 CS333_CFLAGS =
5 CS333_UPROGS =
6 CS333_TPROGS =
7 PRINT_SYSCALLS ?= 0
8
9 ifeq ($(PRINT_SYSCALLS), 1)
10 CS333_CFLAGS += -DPRINT_SYSCALLS
11 endif
12

```

Figure 3: Makefile with CS333_PROJECT set to 1 and PRINT_SYSCALLS set to 0

```
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, 0.131677 s, 38.9 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.00138362 s, 370 kB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
327+1 records in
327+1 records out
167880 bytes (168 kB, 164 KiB) copied, 0.00614613 s, 27.3 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
$ █
```

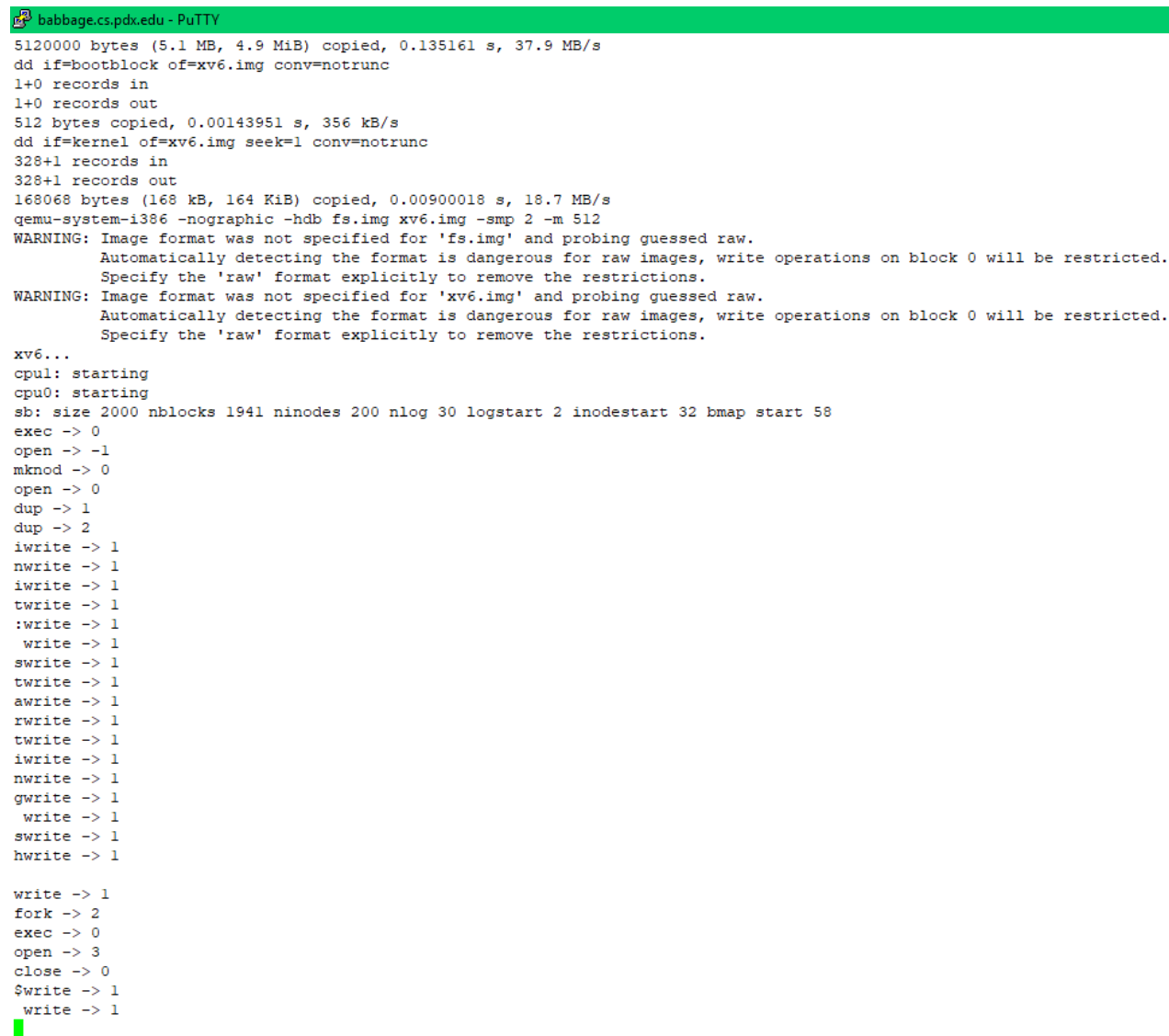
Figure 4: Boot with CS333_PROJECT set to 1 and PRINT_SYSCALLS set to 0

No system information displayed when boot with CS333_PROJECT set to 1 and PRINT_SYSCALLS set to 0 as expected hence this subtest **PASSES** and is a **SUCCESS**.

Both of the subtests pass hence this test **PASSES** and is a **SUCCESS**.

With PRINT_SYSCALLS turned on to test system call tracing

This test verifies that kernel compiles as intended with PRINT_SYSCALLS turns on by setting it to 1 in the Makefile. This test boots the kernel and the output should contain information from the PRINT_SYSCALLS turned off test and the system call names and their return code. The list below should closely match the output that is shown in the project description.



```

babbage.cs.pdx.edu - PuTTY
5120000 bytes (5.1 MB, 4.9 MiB) copied, 0.135161 s, 37.9 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.00143951 s, 356 kB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
328+1 records in
328+1 records out
168068 bytes (168 kB, 164 KiB) copied, 0.00900018 s, 18.7 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...
cpu0: 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 5: Boot with CS333_PROJECT set to 1 and PRINT_SYSCALLS set to 1

The system call trace displays the correct information which is the invoked system calls and their return code. The output in figure 5 matches the reference output from the description of project 1. The standard output is interleaved with the trace output as expected.

This test **PASSES** and is a **SUCCESS**.

Usertests and forktest run with CS333_P1 turned off

In the previous tests, I verified that xv6 compiled and ran as intend with the CS333_P1 flag disabled. This was achieved by set CS333_PROJECT to 0.

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.

```

1 # Set flag to correct CS333 project number: 1, 2, ...
2 # 0 == original xv6-pdx distribution functionality
3 CS333_PROJECT ?= 0
4 CS333_CFLAGS =
5 CS333_UPROGS =
6 CS333_TPROGS =
7 PRINT_SYSCALLS ?= 0
8
9 ifeq ($(PRINT_SYSCALLS), 1)
10 CS333_CFLAGS += -DPRINT_SYSCALLS
11 endif
12

```

Figure 6: Makefile with CS333_P1 disabled by setting CS333_PROJECT to 0.

```

cpu0: starting
sb: size 2000 nblocks 1941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ usertests
usertests starting
createdelete test
createdelete ok
linkunlink test
linkunlink ok
concreate test
concreate ok
fourfiles test
fourfiles ok
sharedfd test
sharedfd ok
bigarg test

```

Figure 7: with CS333_P1 disabled 1

Some output has been cut and elided.

```

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
$

```

Figure 8: with CS333_P1 disabled 2

From all three figures, we can confidently say that all usertests have passed. Forktest also passed as it's part of usertests when usertests was run.

This test **PASSES** and is a **SUCCESS**.

Usertests and forktest run with CS333_P1 flag turned on

In the previous tests, I verified that xv6 compiled and ran as intend with the CS333_P1 flag enable. This was achieved by set CS333_PROJECT to 1.

This test is expected to display similar output and passes both usertests and forktest as the last test where CS333_P1 flag got turned off.

```
babbage.cs.pdx.edu - Pull Y
1 # Set flag to correct CS333 project number: 1, 2, ...
2 # 0 == original xv6-pdx distribution functionality
3 CS333_PROJECT ?= 1
4 CS333_CFLAGS =
5 CS333_UPROGS =
6 CS333_TPROGS =
7 PRINT_SYSCALLS ?= 0
8
```

Figure 9: Makefile with CS333_P1 enabled by setting CS333_PROJECT to 1.

```
Specify the raw format explicitly to remove the restrictions.
xv6...
cpul: starting
cpu0: starting
sb: size 2000 nblocks 1941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ usertests
usertests starting
createdelete test
createdelete ok
linkunlink test
linkunlink ok
concreate test
concreate ok
fourfiles test
fourfiles ok
sharedfd test
```

Figure 10: with CS333_P1 enabled 1

Some output has been cut and elided.

```
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
$
```

Figure 11: with CS333_P1 enabled 2

From all three figures, we can confidently say that all usertests have passed. Forktest also passed as it's part of usertests when usertests was run.

This test **PASSES** and is a **SUCCESS**.

The date User Command

This test is to verify that the date command works correctly. The time is tracked by the system clock in the format of universal coordinate time (UTC) hence the expected output should closely match the system clock of a Linux system date command “date -u”.

The steps for the test include:

1. Run date command in xv6
2. Exit xv6
3. Run date command from Linux prompt using “date -u”
4. Compare output

Note that the date of Linux output is expected to be delayed by a few seconds as it takes non-zero time to perform a shutdown of xv6.

```
zombie      2 19 12768
date        2 20 14244
console     3 21 0
$ date
Mon Jul 2 5:18:48 UTC 2018
$ QEMU: Terminated
thon@babbage:~/CS333/xv6-pdx$ date -u
Mon Jul 2 05:18:54 UTC 2018
thon@babbage:~/CS333/xv6-pdx$
```

Figure 12: Date command from xv6 and Linux

The date command of xv6 displays the date and time in the same format as the Linux command “date -u” with only a couple seconds delay.

This test **PASSES** and is a **SUCCESS**.

Control – P

This test is to verify that the control -p command in xv6 will processes with correct header information, the process information aligned correctly with appropriate header, and correct data is displayed.

It is expected that the header and process information will be output in a well format and each process information is aligned to correct header.

```

init: starting sh
$
PID      State   Name      Elapsed PCs
1        sleep  init      2.139  80104da5 80104b2f 80106578 80105778 8010696c 80106767
2        sleep  sh        2.110  80104da5 80100a05 80101f3f 80101205 80105935 80105778 8010696c 80106767

PID      State   Name      Elapsed PCs
1        sleep  init      3.147  80104da5 80104b2f 80106578 80105778 8010696c 80106767
2        sleep  sh        3.118  80104da5 80100a05 80101f3f 80101205 80105935 80105778 8010696c 80106767

PID      State   Name      Elapsed PCs
1        sleep  init      8.238  80104da5 80104b2f 80106578 80105778 8010696c 80106767
2        sleep  sh        8.209  80104da5 80100a05 80101f3f 80101205 80105935 80105778 8010696c 80106767

```