Srishti Mishra
01FB15ECS310

Batch ID: 46
Project ID: PID_IOS_0817_16

Implement time command as a system program on xv6

Introduction

The xv6 kernel is a re-implementation of Dennis Ritchie's and Ken Thompson's Unix version 6 written in ANSI C and runs on multiprocessor Intel x86 machines. Xv6 was developed in MIT and aimed to unify their operating systems course around a single architecture (x86) and allowed a clean rewrite of the file system and scheduler implementations of the operating system.

A little about Unix Version 6

Unix followed an elegantly simple design philosophy since its conception in 1969 and when the Unix kernel was rewritten in C at Bell Labs in 1972, it became both portable and simple, which greatly contributed to its success. The original Unix Version 6, released in 1975, was written in pre-K&R C and ran on PDP-11 hardware, both obsolete today. Unix Version 6 was the first to be widely available outside Bell Labs.

Unix Time and Clocks

Unix time is the number of seconds that have passed since the Unix epoch; 1st January, 1970. The system time, similarly, counts the number of ticks that have occurred since the epoch. The time of a system can be found either from the hardware clock or the software clock (also called kernel clock).

Hardware clock

The hardware clock is also called the Real Time Clock, BIOS Clock or CMOS clock. It keeps time even when the system switched off ensuring that the computer always has the correct time. It can be accessed through I/O to the Real Time Clock device driver special file or from the CMOS memory registers. We will use this clock to get the system time.

Software clock

The kernel clock is started when the computer is switched on, it initializes itself from the hardware clock at boot time and then counts the time the system runs for. The kernel clock is driven by timer interrupts after it's initialized. This clock can only be found if the operating system is running.

An alternative approach to the one described below (uses the hardware clock) is to use the system clock by initializing a variable of type rtcdate at system startup with the value hardware clock and storing it. When the gettime() function is called, add the uptime of the machine to this value and return the result. However, this is liable to CPU drift and may not be as accurate.

Srishti Mishra
01FB15ECS310

Batch ID: 46
Project ID: PID_IOS_0817_16

Xv6 and CMOS time

For this task, we will use the hardware clock (CMOS clock), which is available on I/O ports. Since xv6 is programmed on Intel x86 architecture, it uses the Advanced Programmable Interrupt Controller (APIC) to handle interrupts to processors. xv6 programs the IO APIC on multiprocessor systems, and uses the LAPIC (Local Advanced Programmable Interrupt Controller) on each processor.
We will use the cmos_read() and cmostime() functions in the lapic.c file in order to access and read the values in the CMOS registers.
The CMOS clock can be accessed at ports 0x70 and 0x71.The first 14 bits correspond to the Real-time Clock, and 1 byte can be read at a time.

Implementing the system call

1. Edit the syscall.h file where a number is assigned to every system call in this xv6 system.
   syscall.h
   ```
   #define SYS_gettime  23
   ```

2. Add the system call function pointer to the system call array. The numbers of system calls defined are used as indexes for a pointer to each system call function defined elsewhere.

3. Add the function prototype, since the system call is not implemented in this file.
   syscall.h
   ```
   extern int sys_gettime(void);
   ```

4. System calls are defined in these two files in xv6 :
   sysproc.c and sysfile.c
   Since sysfile.c contains mostly file related functions, add the implementation of sys_gettime to the end of sysproc.c
   sysproc.c
   ```
   int
   sys_gettime(void){
     struct rtcdate* r;

   if(argptr(0, (void*)&r, sizeof(&r)) < 0)
       return -1;

     cmostime(r);
     return 0;
   ```

```
}
```

5. The function sys_gettime creates a variable of the type rtcdate and passes the variable to cmostime(). In order to pass variables from the user-level ot the kernel-level, the special function argptr() (provided by xv6) must be used. It checks if the process id returned is negative (not valid) and if so, it terminates.

6. The cmostime() function in the lapic.c file calls the cmos_read() and fill_rtcdate() functions which copies the values from the cmos registers into variables.
These values are BCD encoded and use the CONV function to convert these values to decimal numbers. The values are in 24-hour GMT.
lapic.c

```
  void cmostime(struct rtcdate *r)
{
  struct rtcdate t1, t2;
  int sb, bcd;

  sb = cmos_read(CMOS_STATB);

  bcd = (sb & (1 << 2)) == 0;

  // make sure CMOS doesn't modify time while we read it
  for(;;) {
    fill_rtcdate(&t1);
    if(cmos_read(CMOS_STATA) & CMOS_UIP)
        continue;
    fill_rtcdate(&t2);
    if(memcmp(&t1, &t2, sizeof(t1)) == 0)
      break;
   }

  // convert
  if(bcd) {
#define    CONV(x)     (t1.x = ((t1.x >> 4) * 10) + (t1.x &
0xf))
    CONV(second);
    CONV(minute);
    CONV(hour  );
    CONV(day   );
    CONV(month );
    CONV(year  );
#undef     CONV
  }
```

```
  *r = t1;
  r->year += 2000;
}
```

7. The usys.S and user.h files contains the interface for our user program to access the system call. Adding

usys.S

```
SYSCALL(gettime)
```

user.h

```
int gettime(struct rtcdate* );
```

This will ensure that when a user program calls gettime(), it is mapped to system call 23 and the system will handle this call.

Implementing the user program

1. The user must call the gettime() function through a user program, since the user is not supposed to directly use system programs.
Write the required c program, declare a variable of the type rtcdate to store the time. Call the system call gettime(). Add the required print statements to print out the time.

gettime.c

```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "date.h"

int main(){
      struct rtcdate r;
      printf(1, "Current time: ");
      gettime(&r);

      printf(1, "%d:%d:%d GMT\n",
         r.hour, r.minute, r.second);
  exit();
      exit();
}
```

2. The Makefile keeps track of all the files that need to be compiled and which files depend on each other. Hence, we need to add our new program to the Makefile. Add the line _gettime\ under UPROGS and gettime.c\ under EXTRA

3. Run the commands

```
$ make clean
$ make
```

Check for errors.

4. Run

```
$ make qemu
```

5. Type gettime into the terminal. You should see the current time being printed in GMT format.

```
$ gettime
Current time: 23:40:42 GMT
```

References

1. https://github.com/mit-pdos/xv6-public/
2. the XV6 book - PDOS-MIT
   https://pdos.csail.mit.edu/6.828/2016/xv6/book-rev9.pdf
3. http://wiki.osdev.org/CMOS

Abbreviations used:
APIC : Advanced Programmable Interrupt Controller
BIOS : Basic Input/Output System
CMOS : Complementary metal–oxide–semiconductor
LAPIC : Local Advanced Programmable Interrupt Controller