

به نام خدا

## گزارش دوم آزمایشگاه سیستم عامل

سید علیرضا میرشفیعی - ۸۱۰۱۰۱۵۳۲

امیر حسین صفری - ۸۱۰۱۰۱۵۷۱

محمد صدرا عباسی - ۸۱۰۱۰۱۴۶۹

# بخش تشریحی

## سوال ۱

چگونگی پنهان سازی جزئیات فراخوانی های سیستمی:

با توجه به تعریف ماکروی `SYSCALL` در فایل `usys.S`، جزئیات سطح پایین مربوط به پیاده سازی یک فراخوانی سیستمی، از جمله:

- قرار دادن شماره ی `system call` مربوطه در رجیستر `eax`
- پرتاب وقفه نرم افزاری (`int $T_SYSCALL`) به هسته برای انتقال کنترل
- و دریافت مقدار بازگشتی از هسته و ادامه اجرای برنامه سطح کاربر پس از بازگشت

همگی از دید برنامه نویس پنهان می شوند. بنابراین، برنامه نویس صرفاً با فرض انجام صحیح این مراحل در پس زمینه، می تواند بر پیاده سازی منطق `system call` در فایل هایی مانند `sysproc.c` و `proc.c` تمرکز کند، بدون نیاز به درگیر شدن با جزئیات انتقال کنترل بین `user space` و `kernel space`.

همچنین پیاده سازی توابع پایه ای که در بسیاری از برنامه های سطح کاربر کاربرد دارند در فایل `ulib.c` آمده است.

دلایل استفاده از `abstraction`:

- از اونجایی که جزئیات پیاده سازی سطح پایین از دید برنامه نویس پنهان است قابلیت توسعه آن افزایش میابد چون بیشتر درگیر پیاده سازی منطق اصلی برنامه می شود
- به دلیل عدم دسترسی مستقیم برنامه نویس به رجیستر ها احتمال باگ کمتر می شود
- اگر بخواهیم در معماری سیستم عامل تغییر اساسی ایجاد کنیم لازم است تنها فایل های سطح پایین رو تغییر دهیم و منطق اصلی برنامه های سطح کاربر تغییر نمی کند

## سوال ۲

- `Vsyscall`

بعضی از توابع پرکاربرد در جای خاصی از حافظه قرار

می گیرند که توسط کرنل مقدار دهی می شود اما برنامه های سطح کاربر هم می توانند مستقیم آن را بخوانند و این فرایند بدون تغییر مد از کاربر به کرنل انجام می شود که به دلیل عدم سربار باعث سریع بودن سیستم کال مجازی می شود

- Signals

کرنل می تواند سیگنال به برنامه ها بفرستد و برنامه ها نیز می توانند سیگنال هندلر تعریف کنند. سیگنال ها می توانند هر زمانی و به صورت آسنکرون اجرا شوند

- sys/ و proc/

هر دو سیستم فایل های مجازی اند که حاوی اطلاعات واقعی کرنل می باشند. می توان بدون سیستم کال این فایل ها را بازکرد، خواند و در آنها نوشت

- Shared memory

برای رد و بدل کردن دیتا بین کرنل و فضای کاربر می توان بخشی از حافظه رو به صورت مشترک بین هر دو استفاده کرد که امکان دسترسی مستقیم به دیتا را در هر دو بخش فراهم می کند

- Sockets

برنامه ها می توانند به یکدیگر یا کرنل از طریق سوکت ها پیام ارسال کنند

## سوال ۳

تنها تله ای که به صورت مستقیم توسط برنامه های فضای کاربر فعال می شود، system call است. در این حالت، برنامه ای کاربر با فعال سازی یک Gate از نوع Trap که دارای سطح دسترسی `DPL_USER = 3` است، درخواست خود را به کرنل ارسال می کند اما وقفه های سخت افزاری و استثناءها (Exceptions) توسط سخت افزار یا پردازنده به طور خودکار ایجاد می شوند و نباید توسط فضای کاربر صدا زده شوند. به همین دلیل، Gate های مربوط به این تله ها دارای `DPL = 0` هستند تا فقط از سمت کرنل یا CPU قابل فعال سازی باشند. در غیر این صورت امنیت و ثبات کرنل به خطر می افتد.

## سوال ۴

این کار به این دلیل انجام می شود که در صورت تغییر privilege level، لازم است پردازنده از یک پشته ی جدید (مختص سطح جدید) استفاده کند و برای بازگشت صحیح، آدرس پشته ی قبلی (ss و esp) نیز حفظ شود. اما اگر سطح دسترسی تغییر نکند، نیازی به تغییر پشته نیست و بنابراین push این مقادیر ضرورتی ندارد.

## سوال ۵

`argint(int n, int *ip)`

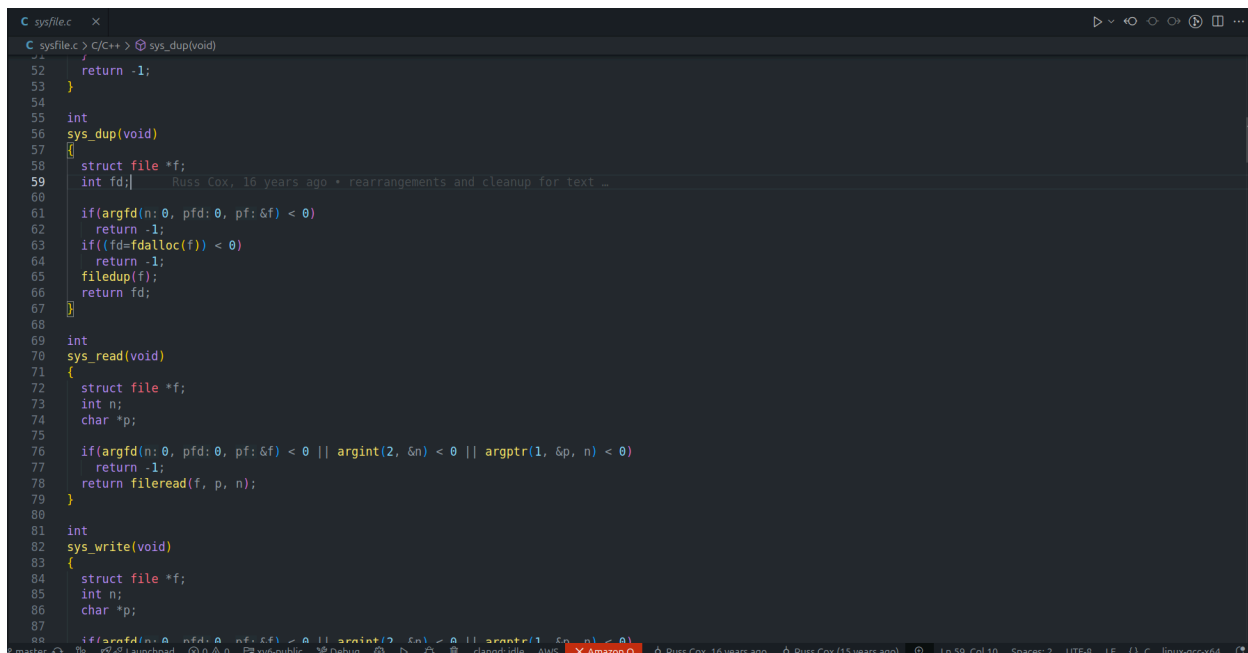
از این تابع برای گرفتن آرگومان های `integer` یک `syscall` استفاده می شود.  
در تابع مشخص شده `n` نشان دهنده شماره پارامتری است که باید دریافت شود.  
این تابع برای دریافت آرگومان ها از `user stack` از رجیستری به نام `esp` استفاده میکند. این رجیستر به بالاترین نقطه استک اشاره می کند. نحوه بدست آوردن مکان دقیق پارامتر مورد نظر نیز بصورت `esp+4+4*n` است.

`argptr(int n, char **pp, int size)`

از این تابع برای گرفتن `n` امین آرگومان `syscall` که `pointer argument` است استفاده می شود.

`argstr(int n, char **pp)`

از این تابع برای گرفتن `n` امین آرگومان `syscall` که اشاره گری به یک `string` است استفاده می شود. وظیفه دیگر تابع این است که `string` موردنظر `valid` است و به `null` ختم می شود.  
دلیلی که `argptr` بازه های آدرس ورودی را بررسی می کند این است که اشاره گر مورد نظر در محدوده اختصاص یافته به `process` حضور داشته باشد. از طرف دیگر بررسی می کند که اشاره گر به بخش معتبری از حافظه با اندازه مشخص دسترسی پیدا کند. این مرحله اطمینان حاصل می کند که `user process` به `kernel memory` دسترسی پیدا نکند.



```
1 //
2
3 #include <sys/types.h>
4 #include <unistd.h>
5 #include <fcntl.h>
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <string.h>
9 #include <sys/syscall.h>
10
11 #define ARGFD(n, pfd, pf, &f) < 0
12 #define ARGPTR(n, &p, n) < 0
13 #define ARGSTR(n, &p) < 0
14
15 int
16 sys_dup(void)
17 {
18     struct file *f;
19     int fd;
20
21     if (argfd(1, &fd, &f) < 0)
22         return -1;
23     if ((fd = fdopen(f)) < 0)
24         return -1;
25     fileread(f);
26     return fd;
27 }
28
29 int
30 sys_read(void)
31 {
32     struct file *f;
33     int n;
34     char *p;
35
36     if (argfd(1, &fd, &f) < 0 || argptr(2, &n, n) < 0 || argptr(3, &p, n) < 0)
37         return -1;
38     return fileread(f, p, n);
39 }
40
41 int
42 sys_write(void)
43 {
44     struct file *f;
45     int n;
46     char *p;
47
48     if (argfd(1, &fd, &f) < 0 || argptr(2, &n, n) < 0 || argptr(3, &p, n) < 0)
```

در `sys_read` در صورتی که `argptr` بازه آدرس ورودی را بررسی نمی کرد، `user program` مقدار `p` را می توانست اشاره گر به یک آدرس غیرقابل قبول یا اشاره گر به `kernel memory` انتخاب کند.

# بررسی گام های اجرای فراخوانی سیستمی در سطح کرنل توسط gdb

ابتدا یک برنامه کاربر ساده می نویسیم که سیستم کال انجام دهد:

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

Windsurf: Refactor | Explain | Generate Function Comment | ×
int main(int argc, char* argv[]) {

    int pid = getpid();

    printf(1, "PID result : (%d)\n", pid);
    exit();
}
```

سپس سیستم عامل را در حالت دیباگ اجرا کرده و gdb را به آن متصل می کنیم. برای این کار، ابتدا این دستور را اجرا می کنیم: `make qemu-gdb`  
سپس در یک ترمینال جدا، gdb را اجرا کرده و به پورت دیباگ متصل می شویم:

`gdb kernel`

`(gdb) target remote :26000`

```
sadra@sadra-ThinkPad-X1-Yoga-4th:~/Documents/Codes/xv6-public$ make qemu-gdb
*** Now run 'gdb'.
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512 -S -gdb tcp::26000
(gdb) target remote :26000
Remote debugging using :26000
0x00000000 in ?? ()
(gdb)
```

سپس روی `syscall` یک `break point` ست می کنیم

```
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
    info "(gdb)Auto-loading safe path"
(gdb) break syscall
Breakpoint 1 at 0x80105820: file syscall.c, line 181.
(gdb)
```

اجرا را تا رسیدن به نقطه مورد نظر ادامه می دهیم سپس برنامه کاربر را در کنسول فراخوانی می کنیم و با یک `bt` خروجی به صورت زیر خواهد بود:

```
(gdb) bt
#0  syscall () at syscall.c:181
#1  0x801069ed in trap (tf=0x8dffe4b4) at trap.c:43
#2  0x8010678c in alltraps () at trapasm.S:20
#3  0x8dffe4b4 in ?? ()
Backtrace stopped: previous frame inner to this frame (corrupt stack?)
```

- دستور `bt`

در GDB برای مشاهده ی پشته ی فراخوانی (`call stack`) استفاده می شود. با اجرای این دستور، می توان توالی توابعی را که منجر به وضعیت فعلی اجرای برنامه شده اند را مشاهده کرد.

- تحلیل خروجی

خروجی نشان می دهد که فراخوانی ابتدا از فضای کاربر آغاز شده، سپس وارد مسیر `alltraps` در کد `assembly` شده و از آنجا به تابع `trap` در کرنل منتقل شده است. در `trap`، پس از ساخت `trapframe` و تشخیص شماره تله برابر با `T_SYSCALL`، کنترل به تابع `syscall` منتقل می شود. در `syscall`، با استفاده از شماره `system call`

(مقدار `eax`) تابع متناظر از آرایه `syscalls` فراخوانی می‌شود. این خروجی نحوه ورود از فضای کاربر به هسته و مسیر اجرای سیستم کال را نشان می‌دهد

• کاربرد دستور `down` و `up`:

دستور `down` در GDB برای حرکت به فریم پایین‌تر در پشته‌ی فراخوانی استفاده می‌شود؛ یعنی به تابعی که تابع فعلی را فراخوانی کرده است. این دستور برای بررسی `context` توابع قبلی، آرگومان‌های ورودی و محل دقیق فراخوانی آن‌ها کاربرد دارد. دستور `up` در مقابل، برای بازگشت به فریم بالاتر در `stack` به‌کار می‌رود؛ یعنی به تابعی که توسط فریم فعلی فراخوانی شده است. این ابزار کمک می‌کند در پشته جابه‌جا شویم و وضعیت اجرای برنامه در سطوح مختلف را تحلیل کنیم.

در ادامه باید از دستور `down` استفاده کرده و محتوای رجیستر `eax` را تحلیل کنیم:

```
(gdb) down
Bottom (innermost) frame selected; you cannot go down.
(gdb) █
```

این پیغام نشان می‌دهد که در پایین‌ترین سطح پشته قرار داریم و امکان رفتن به فریم پایین‌تر وجود ندارد. بنابراین با اجرای دستور `up` یک سطح به بالا برمی‌گردیم و وارد فریم مربوط به تابع `trap` می‌شویم. در این فریم، به ساختار `trapframe` دسترسی داشته و با استفاده از دستور `p tf->eax` مقدار شماره `system call` را استخراج می‌کنیم:

```
(gdb) up
#1  0x801069ed in trap (tf=0x8dffefb4) at trap.c:43
43      syscall();
(gdb) p tf->eax
$1 = 5
(gdb) █
```

مقدار شماره سیستم‌کال برابر ۵ است و این به این دلیل است که قبل از اجرای اصلی برنامه کاربر، هسته‌ی سیستم‌عامل ابتدا نیاز دارد ورودی کاربر را پردازش کند. این ورودی خوانده می‌شود و برای این کار، `system call` شماره ۵ که مربوط به `read` است، فراخوانی می‌شود. پس از آن، سیستم کال‌های دیگری مانند `fork`، `exec`، و در نهایت `getpid` اجرا خواهند شد. بنابراین مشاهده‌ی `system call read` پیش از `getpid` در مسیر اجرای پشته (`call stack`) نشان می‌دهد که هنوز به نقطه‌ای نرسیده‌ایم که `getpid` فراخوانی شده باشد.

اجرا را ادامه می‌دهیم تا شماره سیستم کال برابر `getpid` یا همان ۱۱ شود:

```

#1 0x801069ed in trap (tf=0x8dffefb4) at trap.c:43
43      syscall();
(gdb) p tf->eax
$6 = 5
(gdb) c
Continuing.

Thread 1 hit Breakpoint 1, syscall () at syscall.c:181
181      struct proc *curproc = myproc();
(gdb) up
#1 0x801069ed in trap (tf=0x8dffefb4) at trap.c:43
43      syscall();
(gdb) p tf->eax
$7 = 1
(gdb) c
Continuing.

Thread 1 hit Breakpoint 1, syscall () at syscall.c:181
181      struct proc *curproc = myproc();
(gdb) up
#1 0x801069ed in trap (tf=0x8dffefb4) at trap.c:43
43      syscall();
(gdb) p tf->eax
$8 = 3
(gdb) c
Continuing.

Thread 1 hit Breakpoint 1, syscall () at syscall.c:181
181      struct proc *curproc = myproc();
(gdb) up
#1 0x801069ed in trap (tf=0x8df1bfb4) at trap.c:43
43      syscall();
(gdb) p tf->eax
$9 = 12
(gdb) c
Continuing.

Thread 1 hit Breakpoint 1, syscall () at syscall.c:181
181      struct proc *curproc = myproc();
(gdb) up
#1 0x801069ed in trap (tf=0x8df1bfb4) at trap.c:43
43      syscall();
(gdb) p tf->eax
$10 = 7
(gdb) c
Continuing.

Thread 1 hit Breakpoint 1, syscall () at syscall.c:181
181      struct proc *curproc = myproc();
(gdb) up
#1 0x801069ed in trap (tf=0x8df1bfb4) at trap.c:43
43      syscall();
(gdb) p tf->eax
$11 = 11
(gdb) █

```



# ارسال آرگومان های فراخوانی های سیستمی

برای پیاده سازی systemcall خواسته شده ابتدا منطق آنرا در proc.c پیاده سازی می کنیم:

```
C proc.c > clangd > set_sleep_syscall
522     if(p->state >= 0 && p->state < NELEM(states) && states[p->state])
523         state = states[p->state];
524     else
525         state = "???";
526     cprintf("%d %s %s", p->pid, state, p->name);
527     if(p->state == SLEEPING){
528         getcallerpcs((uint*)p->context->ebp+2, pc); Cast to 'uint *' (aka 'unsigned int *') from smaller integer type 'uint' (aka 'unsigned int')
529         for(i=0; i<10 && pc[i] != 0; i++)
530             cprintf(" %p", pc[i]);
531     }
532     cprintf("\n");
533 }
534 }
535
536
537 void
538 next_palindrome(int num)
539 {
540     int i=num+1;
541     while(1)
542     {
543         int number=i,reverse=0;
544         while(number>0)
545         {
546             int digit=number%10;
547             reverse=reverse*10+digit;
548             number/=10;
549         }
550         if(reverse==i)
551             break;
552         i++;
553     }
554     cprintf("result: %d\n",i);
555 }
556
557
```

سپس در مرحله بعدی برنامه را در sysproc.c پیاده می کنیم:

```
C sysproc.c > clangd > sys_get_system_time
60 sys_sleep(void)
61 {
62     while(ticks - ticks0 < n){
63         if(myproc()->killed){
64             release(&tickslock);
65             return -1;
66         }
67         sleep(&ticks, &tickslock);
68     }
69     release(&tickslock);
70     return 0;
71 }
72
73 // return how many clock tick interrupts have occurred
74 // since start.
75 int
76 sys_uptime(void)
77 {
78     uint xticks;
79     acquire(&tickslock);
80     xticks = ticks;
81     release(&tickslock);
82     return xticks;
83 }
84
85 int
86 sys_next_palindrome(void)
87 {
88     int in_num=myproc()->tf->ebx;
89     next_palindrome(in_num);
90     return 0;
91 }
92
93
94
95
```

در مرحله بعد شماره systemcall را به syscall.h اضافه می کنیم:

```
C syscall.h > SYS_get_system_time
You, 2 weeks ago | 3 authors (Frans Kaashoek and others)
1 // System call numbers
2 #define SYS_fork 1
3 #define SYS_exit 2
4 #define SYS_wait 3
5 #define SYS_pipe 4
6 #define SYS_read 5
7 #define SYS_kill 6
8 #define SYS_exec 7
9 #define SYS_fstat 8
10 #define SYS_chdir 9
11 #define SYS_dup 10
12 #define SYS_getpid 11
13 #define SYS_sbrk 12
14 #define SYS_sleep 13
15 #define SYS_uptime 14
16 #define SYS_open 15
17 #define SYS_write 16
18 #define SYS_mknod 17
19 #define SYS_unlink 18
20 #define SYS_link 19
21 #define SYS_mkdir 20
22 #define SYS_close 21
23 #define SYS_next_palindrome 22
24 #define SYS_set_sleep_syscall 23
25 #define SYS_get_system_time 24 | You, 2 weeks ago • implementation of set_sleep_syscall ...

master Launchpad 0 28 xv6-public Debug clangd: idle AWS Amazon Q amirhosseinas (1 week ago) Ln 25, Col 31 Spaces: 4 UTF-8 LF C++ linux-gcc-x64
```

در syscall.c تمام declaration های مربوطه را انجام می دهیم:

```
C syscall.c > clangd > argptr
101 extern int sys_sleep(void);
102 extern int sys_unlink(void);
103 extern int sys_wait(void);
104 extern int sys_write(void);
105 extern int sys_uptime(void);
106 extern int sys_next_palindrome(void);
107 extern int sys_set_sleep_syscall(void);
108 extern int sys_get_system_time(void);
109 static int (*syscalls[])(void) = {
110 [SYS_fork] sys_fork, Use of GNU 'missing =' extension in designator (fix available)
111 [SYS_exit] sys_exit, Use of GNU 'missing =' extension in designator (fix available)
112 [SYS_wait] sys_wait, Use of GNU 'missing =' extension in designator (fix available)
113 [SYS_pipe] sys_pipe, Use of GNU 'missing =' extension in designator (fix available)
114 [SYS_read] sys_read, Use of GNU 'missing =' extension in designator (fix available)
115 [SYS_kill] sys_kill, Use of GNU 'missing =' extension in designator (fix available)
116 [SYS_exec] sys_exec, Use of GNU 'missing =' extension in designator (fix available)
117 [SYS_fstat] sys_fstat, Use of GNU 'missing =' extension in designator (fix available)
118 [SYS_chdir] sys_chdir, Use of GNU 'missing =' extension in designator (fix available)
119 [SYS_dup] sys_dup, Use of GNU 'missing =' extension in designator (fix available)
120 [SYS_getpid] sys_getpid, Use of GNU 'missing =' extension in designator (fix available)
121 [SYS_sbrk] sys_sbrk, Use of GNU 'missing =' extension in designator (fix available)
122 [SYS_sleep] sys_sleep, Use of GNU 'missing =' extension in designator (fix available)
123 [SYS_uptime] sys_uptime, Use of GNU 'missing =' extension in designator (fix available)
124 [SYS_open] sys_open, Use of GNU 'missing =' extension in designator (fix available)
125 [SYS_write] sys_write, Use of GNU 'missing =' extension in designator (fix available)
126 [SYS_mknod] sys_mknod, Use of GNU 'missing =' extension in designator (fix available)
127 [SYS_unlink] sys_unlink, Use of GNU 'missing =' extension in designator (fix available)
128 [SYS_link] sys_link, Use of GNU 'missing =' extension in designator (fix available)
129 [SYS_mkdir] sys_mkdir, Use of GNU 'missing =' extension in designator (fix available)
130 [SYS_close] sys_close, Use of GNU 'missing =' extension in designator (fix available)
131 [SYS_next_palindrome] sys_next_palindrome, Use of GNU 'missing =' extension in designator (fix available)
132 [SYS_set_sleep_syscall] sys_set_sleep_syscall, Use of GNU 'missing =' extension in designator (fix available)
133 [SYS_get_system_time] sys_get_system_time, Use of GNU 'missing =' extension in designator (fix available)
134 };
135
136 void
137 syscall(void)

master Launchpad 0 28 xv6-public Debug clangd: idle AWS Amazon Q rsc 18 years ago rsc (17 years ago) Ln 59, Col 1 Spaces: 2 UTF-8 LF C linux-gcc-x64
```

در usys.s کار declaration را انجام می دهیم:

```
usys.S
You, 2 weeks ago | 3 authors (rsc and others)
1 #include "syscall.h"
2 #include "traps.h"
3
4 #define SYSCALL(name) \
5     .globl name; \
6     name: \
7         movl $SYS_ ## name, %eax; \
8         int $T_SYSCALL; \
9         ret
10
11 SYSCALL(fork)
12 SYSCALL(exit)
13 SYSCALL(wait)
14 SYSCALL(pipe)
15 SYSCALL(read)
16 SYSCALL(write)
17 SYSCALL(close)
18 SYSCALL(kill)
19 SYSCALL(exec)
20 SYSCALL(open)
21 SYSCALL(mknod)
22 SYSCALL(unlink)
23 SYSCALL(fstat)
24 SYSCALL(link)
25 SYSCALL(mkdir)
26 SYSCALL(chdir)
27 SYSCALL(dup)
28 SYSCALL(getpid)
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(next_palindrome)
33 SYSCALL(set_sleep_syscall)
34 SYSCALL(get_system_time) You, 2 weeks ago • implementation of set_sleep_syscall ...
```

اضافه کردن declaration در user.h:

```
C user.h
You, 2 hours ago | 7 authors (rsc and others)
1 struct stat;
2 struct rtcdate;
3
4 // system calls
5 int fork(void);
6 int exit(void) __attribute__((noreturn));
7 int wait(void);
8 int pipe(int*);
9 int write(int, const void*, int);
10 int read(int, void*, int);
11 int close(int);
12 int kill(int);
13 int exec(char*, char**);
14 int open(const char*, int);
15 int mknod(const char*, short, short);
16 int unlink(const char*);
17 int fstat(int fd, struct stat*);
18 int link(const char*, const char*);
19 int mkdir(const char*);
20 int chdir(const char*);
21 int dup(int);
22 int getpid(void);
23 char* sbrk(int);
24 int sleep(int);
25 int uptime(void);
26 void next_palindrome(int num);
27 int set_sleep_syscall(int tick);
28 int get_system_time(struct rtcdate*);
29
30 // ulib.c
31 int stat(const char*, struct stat*);
32 char* strcpy(char*, const char*);
33 void *memmove(void*, const void*, int);
34 char* strchr(const char*, char c);
35 int strcmp(const char*, const char*);
36 void printf(int, const char*, ...);
```

اضافه کردن declaration در defs.h:

```
C defsh 9+ X
C defsh > C/C++ > allocvm(pde_t*, uint, uint)
99 int pipealloc(struct file**, struct file**);
100 void pipeclose(struct pipe*, int);
101 int piperead(struct pipe*, char*, int);
102 int pipewrite(struct pipe*, char*, int);
103
104 //PAGEBREAK: 16
105 // proc.c
106 int cpuid(void);
107 void exit(void);
108 int fork(void);
109 int growproc(int);
110 int kill(int);
111 struct cpu* mycpu(void);
112 struct proc* myproc();
113 void pinit(void);
114 void procdump(void);
115 void scheduler(void) __attribute__((noreturn));
116 void sched(void);
117 void setproc(struct proc*);
118 void sleep(void*, struct spinlock*);
119 void userinit(void);
120 int wait(void);
121 void wakeup(void*);
122 void yield(void);
123 void next_palindrome(int);
124 int set_sleep_syscall(int);
125 int get_system_time(struct rtcdate*);
126
127 // swtch.S
128 void swtch(struct context**, struct context*);
129
130 // spinlock.c
131 void acquire(struct spinlock*);
132 void getcallerpcs(void*, uint*); Unknown type name 'uint'; did you mean 'int'? (fixes available)
133 int holding(struct spinlock*);
134 void initlock(struct spinlock*, char*);
135 void release(struct spinlock*);
```

تغييرات :makefile

```
M Makefile X
M Makefile
229 qemu: fs.img xv6.img
231
232
233 qemu-memfs: xv6memfs.img
234 $(QEMU) -drive file=xv6memfs.img,index=0,media=disk,format=raw -smp $(CPUS) -m 256
235
236 qemu-nox: fs.img xv6.img
237 $(QEMU) -nographic $(QEMUOPTS)
238
239 .gdbinit: .gdbinit.tmpl
240 sed "s/localhost:1234/localhost:$(GDBPORT)/" < $^ > $@
241
242 qemu-gdb: fs.img xv6.img .gdbinit
243 @echo "**** Now run 'gdb'." 1>&2
244 $(QEMU) -serial mon:stdio $(QEMUOPTS) -S $(QEMUGDB)
245
246 qemu-nox-gdb: fs.img xv6.img .gdbinit
247 @echo "**** Now run 'gdb'." 1>&2
248 $(QEMU) -nographic $(QEMUOPTS) -S $(QEMUGDB)
249
250 # CUT HERE
251 # prepare dist for students
252 # after running make dist, probably want to
253 # rename it to rev0 or rev1 or so on and then
254 # check in that version.
255
256 EXTRA=\
257 mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
258 ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
259 printf.c umalloc.c user_curly_brace_correction_check.c\ next_palindrome.c\ set_sleep_syscall.c\
260 README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
261 .gdbinit.tmpl gdbutil\
262
263 dist:
264 rm -rf dist
265 mkdir dist
266 for i in $(FILES); \
267 do \
268
master  Launchpad  xv6-public  Debug  AWS  Amazon Q  rsc, 19 years ago  rsc (18 years ago)  Ln 287, Col 22  Tab Size: 4  UTF-8  LF  Makefile
```

```
M Makefile X
M Makefile
155 forktest: forktest.o $(ULIB)
161 mkfs: mkfs.c fs.h
162 gcc -Werror -Wall -o mkfs mkfs.c
163
164 # Prevent deletion of intermediate files, e.g. cat.o, after first build, so
165 # that disk image changes after first build are persistent until clean. More
166 # details:
167 # http://www.gnu.org/software/make/manual/html_node/Chained-Rules.html
168 .PRECIOUS: %.o
169
170 UPROGS=\
171 _cat\
172 _echo\
173 _forktest\
174 _grep\
175 _init\
176 _kill\
177 _ln\
178 _ls\
179 _mkdir\
180 _rm\
181 _sh\
182 _stressfs\
183 _usertests\
184 _wc\
185 _zombie\
186 _user_curly_brace_correction_check\
187 _next_palindrome\
188 _set_sleep_syscall\
189
190 fs.img: mkfs README $(UPROGS)
191 ./mkfs fs.img README $(UPROGS)
192
193 -include *.d
194
195 clean:
196 rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
197
master  Launchpad  xv6-public  Debug  AWS  Amazon Q  rsc, 19 years ago  rsc (18 years ago)  Ln 250, Col 28  Tab Size: 4  UTF-8  LF  Makefile
```

## پیاده سازی user program:

```
C next_palindrome.c x
C next_palindrome.c > clangd > main
You, 22 hours ago | 1 author (You)
1 #include "types.h"
2 #include "user.h"
3 #define WRONGINPUTTRYAGAIN "invalid command!try again"
4 int main(int argc, char *argv[])
5 {
6     if(argc!=2)
7     {
8         printf(1,"%s\n",WRONGINPUTTRYAGAIN);
9         exit();
10    }
11    int num=atoi(argv[1]),prev_val_ebx;
12    asm volatile(
13        "movl %%ebx, %0\n\t"
14        "movl %1, %%ebx"
15        : "=r" (prev_val_ebx)
16        : "r" (num)
17        : "ebx"
18    );
19    next_palindrome(num);
20    asm volatile(
21        "movl %0, %%ebx"
22        :
23        : "r" (prev_val_ebx)
24        : "ebx"
25    );
26    exit();
27 }
```

توضیحات مربوط به پیاده سازی:

برای پیاده سازی این بخش در `proc.c` منطق `syscall` را پیاده سازی کردیم که در آن برای هر عدد `reverse` آن را ساخته و بررسی می کنیم که اگر `reverse` عدد با خود عدد برابر بود، پس عدد `palindrome` را یافته و آنرا `print` می کنیم. در برنامه کاربر نیز ابتدا مقدار `ebx` را در متغیر `prev_val_ebx` ذخیره کرده و سپس عدد ورودی کاربر را در آن ذخیره می کنیم و `syscall` را صدا زده و پس از پایان `syscall` مقدار قبلی رجیستر `ebx` را دوباره در آن ذخیره می کنیم. نتیجه اجرا:

```
QEMU
Machine View
SeaBIOS (version 1.16.3-debian-1.16.3-2)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1EFCB050+1EF0B050 CA00

Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
o- SeyedAlirezaMirshafiee
o- SadraAbbasi
o- AmirSafari
init: starting sh
$ next_palindrome 1200
result: 1221
$ _
```

## پیاده سازی فراخوانی سیستمی خواباندن پردازنده

برای پیاده سازی systemcall خواسته شده ابتدا منطق آنرا در proc.c پیاده سازی می کنیم:

```
C proc.c 9+ X
C proc.c > clangd > set_sleep_syscall

558
559
560 int
561 set_sleep_syscall(int input_tick)
562 {
563     int current_time;
564     acquire(&tickslock);
565     current_time=ticks;
566     release(&tickslock);
567     while(1)
568     {
569         acquire(&tickslock);
570         if(ticks-current_time==input_tick)
571         {
572             release(&tickslock);
573             break;
574         }
575         release(&tickslock);
576     }
577     return 0;
578 }
```

سپس در مرحله بعدی برنامه را در sysproc.c پیاده می کنیم:

```
C sysproc.c > clangd > sys_get_system_time
100     next_palindrome(in_num);
101     return 0;
102 }
103
104
105
106 int
107 sys_set_sleep_syscall(void)
108 {
109     int input_tick;
110     if(argptr(0,&input_tick)<0)
111         return -1;
112     if(set_sleep_syscall(input_tick)==-1)
113         return -1;
114     return 0;
115 }
116
117
118
119 int
120 sys_get_system_time(void)
121 {
122     struct rtcdate* current_time;
123     if(argptr(0,(void*)&current_time,sizeof(*current_time))<0)
124         return -1;
125     cmostime(r: current_time);
126     return 0;
127 }
```

در مرحله بعد شماره systemcall را به syscall.h اضافه می کنیم:

```
C syscall.h > SYS_get_system_time
You, 2 weeks ago | 3 authors (Frans Kaashoek and others)
1 // System call numbers
2 #define SYS_fork 1
3 #define SYS_exit 2
4 #define SYS_wait 3
5 #define SYS_pipe 4
6 #define SYS_read 5
7 #define SYS_kill 6
8 #define SYS_exec 7
9 #define SYS_fstat 8
10 #define SYS_chdir 9
11 #define SYS_dup 10
12 #define SYS_getpid 11
13 #define SYS_sbrk 12
14 #define SYS_sleep 13
15 #define SYS_uptime 14
16 #define SYS_open 15
17 #define SYS_write 16
18 #define SYS_mknod 17
19 #define SYS_unlink 18
20 #define SYS_link 19
21 #define SYS_mkdir 20
22 #define SYS_close 21
23 #define SYS_next_palindrome 22
24 #define SYS_set_sleep_syscall 23
25 #define SYS_get_system_time 24
```

در syscall.c تمام declaration های مربوطه را انجام می دهیم:



```
C syscall.c > clangd > argptr
101 extern int sys_sleep(void);
102 extern int sys_unlink(void);
103 extern int sys_wait(void);
104 extern int sys_write(void);
105 extern int sys_uptime(void);
106 extern int sys_next_palindrome(void);
107 extern int sys_set_sleep_syscall(void);
108 extern int sys_get_system_time(void);
109 static int (*syscalls[])(void) = {
110 [SYS_fork] sys_fork, Use of GNU 'missing =' extension in designator (fix available)
111 [SYS_exit] sys_exit, Use of GNU 'missing =' extension in designator (fix available)
112 [SYS_wait] sys_wait, Use of GNU 'missing =' extension in designator (fix available)
113 [SYS_pipe] sys_pipe, Use of GNU 'missing =' extension in designator (fix available)
114 [SYS_read] sys_read, Use of GNU 'missing =' extension in designator (fix available)
115 [SYS_kill] sys_kill, Use of GNU 'missing =' extension in designator (fix available)
116 [SYS_exec] sys_exec, Use of GNU 'missing =' extension in designator (fix available)
117 [SYS_fstat] sys_fstat, Use of GNU 'missing =' extension in designator (fix available)
118 [SYS_chdir] sys_chdir, Use of GNU 'missing =' extension in designator (fix available)
119 [SYS_dup] sys_dup, Use of GNU 'missing =' extension in designator (fix available)
120 [SYS_getpid] sys_getpid, Use of GNU 'missing =' extension in designator (fix available)
121 [SYS_sbrk] sys_sbrk, Use of GNU 'missing =' extension in designator (fix available)
122 [SYS_sleep] sys_sleep, Use of GNU 'missing =' extension in designator (fix available)
123 [SYS_uptime] sys_uptime, Use of GNU 'missing =' extension in designator (fix available)
124 [SYS_open] sys_open, Use of GNU 'missing =' extension in designator (fix available)
125 [SYS_write] sys_write, Use of GNU 'missing =' extension in designator (fix available)
126 [SYS_mknod] sys_mknod, Use of GNU 'missing =' extension in designator (fix available)
127 [SYS_unlink] sys_unlink, Use of GNU 'missing =' extension in designator (fix available)
128 [SYS_link] sys_link, Use of GNU 'missing =' extension in designator (fix available)
129 [SYS_mkdir] sys_mkdir, Use of GNU 'missing =' extension in designator (fix available)
130 [SYS_close] sys_close, Use of GNU 'missing =' extension in designator (fix available)
131 [SYS_next_palindrome] sys_next_palindrome, Use of GNU 'missing =' extension in designator (fix available)
132 [SYS_set_sleep_syscall] sys_set_sleep_syscall, Use of GNU 'missing =' extension in designator (fix available)
133 [SYS_get_system_time] sys_get_system_time, Use of GNU 'missing =' extension in designator (fix available)
134 };
135
136 void
137 syscall(void)
```

در `usys.s` کار declaration را انجام می دهیم:

```
usys.S
You, 2 weeks ago | 3 authors (rsc and others)
1 #include "syscall.h"
2 #include "traps.h"
3
4 #define SYSCALL(name) \
5 .globl name; \
6 name: \
7 movl $SYS_ ## name, %eax; \
8 int $T_SYSCALL; \
9 ret
10
11 SYSCALL(fork)
12 SYSCALL(exit)
13 SYSCALL(wait)
14 SYSCALL(pipe)
15 SYSCALL(read)
16 SYSCALL(write)
17 SYSCALL(close)
18 SYSCALL(kill)
19 SYSCALL(exec)
20 SYSCALL(open)
21 SYSCALL(mknod)
22 SYSCALL(unlink)
23 SYSCALL(fstat)
24 SYSCALL(link)
25 SYSCALL(mkdir)
26 SYSCALL(chdir)
27 SYSCALL(dup)
28 SYSCALL(getpid)
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(next_palindrome)
33 SYSCALL(set_sleep_syscall)
34 SYSCALL(get_system_time) You, 2 weeks ago • implementation of set_sleep_syscall
```

اضافه کردن declaration در `user.h`:

```
C user.h 5 X
C user.h > C/C++ > free(void*)
You, 2 hours ago | 7 authors (rsc and others)
1 struct stat;
2 struct rtcdate;
3
4 // system calls
5 int fork(void);
6 int exit(void) __attribute__((noreturn));
7 int wait(void);
8 int pipe(int*);
9 int write(int, const void*, int);
10 int read(int, void*, int);
11 int close(int);
12 int kill(int);
13 int exec(char*, char**);
14 int open(const char*, int);
15 int mknod(const char*, short, short);
16 int unlink(const char*);
17 int fstat(int fd, struct stat*);
18 int link(const char*, const char*);
19 int mkdir(const char*);
20 int chdir(const char*);
21 int dup(int);
22 int getpid(void);
23 char* gbrk(int);
24 int sleep(int);
25 int uptime(void);
26 void next_palindrome(int num);
27 int set_sleep_syscall(int tick);
28 int get_system_time(struct rtcdate*);
29
30 // ulib.c
31 int stat(const char*, struct stat*);
32 char* strcpy(char*, const char*);
33 void *memmove(void*, const void*, int);
34 char* strchr(const char*, char c);
35 int strcmp(const char*, const char*);
36 void printf(int, const char*, ...);
```

اضافه کردن declaration در defs.h:

```
C defs.h 9+ X
C defs.h > C/C++ > allocuvmm(pde_t*, uint, uint)
117 void getproc(struct proc*);
118 void sleep(void*, struct spinlock*);
119 void userinit(void);
120 int wait(void);
121 void wakeup(void*);
122 void yield(void);
123 void next_palindrome(int);
124 int set_sleep_syscall(int);
125 int get_system_time(struct rtcdate*);
126
127 // switch.S
128 void swtch(struct context**, struct context*);
129
130 // spinlock.c
131 void acquire(struct spinlock*);
132 void getcallerpcs(void*, uint*); Unknown type name 'uint'; did you mean 'int'? (fixes available)
133 int holding(struct spinlock*);
134 void initlock(struct spinlock*, char*);
135 void release(struct spinlock*);
136 void pushcli(void);
137 void popcli(void);
138
139 // sleeplock.c
140 void acquiresleep(struct sleeplock*);
141 void releasesleep(struct sleeplock*);
142 int holdingsleep(struct sleeplock*);
143 void initsleeplock(struct sleeplock*, char*);
144
145 // string.c
146 int memcmp(const void*, const void*, uint); Unknown type name 'uint'; did you mean 'int'? (fixes available)
147 void* memmove(void*, const void*, uint); Unknown type name 'uint'; did you mean 'int'? (fixes available)
148 void* memset(void*, int, uint); Unknown type name 'uint'; did you mean 'int'? (fixes available)
149 char* safestrcpy(char*, const char*, int);
150 int strlen(const char*);
151 int strncmp(const char*, const char*, uint); Unknown type name 'uint'; did you mean 'int'? (fixes available)
152 char* strncpy(char*, const char*, int);
153
```

تغییرات makefile:

```
M Makefile
M Makefile
227 QEMUOPTS = -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp $(CPUS) -m 512 $(QEMUEXTRA)
228
229 qemu: fs.img xv6.img
230     $(QEMU) -serial mon:stdio $(QEMUOPTS)
231
232 qemu-memfs: xv6memfs.img
233     $(QEMU) -drive file=xv6memfs.img,index=0,media=disk,format=raw -smp $(CPUS) -m 256
234
235 qemu-nox: fs.img xv6.img
236     $(QEMU) -nographic $(QEMUOPTS)
237
238 .gdbinit: .gdbinit.tmpl
239     sed "s/localhost:1234/localhost:$(GDBPORT)/" < $^ > $@
240
241 qemu-gdb: fs.img xv6.img .gdbinit
242     @echo "**** Now run 'gdb'." 1>&2
243     $(QEMU) -serial mon:stdio $(QEMUOPTS) -S $(QEMUGDB)
244
245 qemu-nox-gdb: fs.img xv6.img .gdbinit
246     @echo "**** Now run 'gdb'." 1>&2
247     $(QEMU) -nographic $(QEMUOPTS) -S $(QEMUGDB)
248
249 # CUT HERE
250 # prepare dist for students
251 # after running make dist, probably want to
252 # rename it to rev0 or rev1 or so on and then
253 # check in that version.
254
255 EXTRA=\
256     mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
257     ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
258     printf.c umalloc.c user_curly_brace_correction_check.c\ next_palindrome.c\ set_sleep_syscall.c\
259     README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
260     .gdbinit.tmpl gdbutil\
261
262 dist:
263     rm -rf dist
264
```

```
M Makefile
M Makefile
155 _forktest: forktest.o $(ULIB)
156
157 mkfs: mkfs.c fs.h
158     gcc -Werror -Wall -o mkfs mkfs.c
159
160 # Prevent deletion of intermediate files, e.g. cat.o, after first build, so
161 # that disk image changes after first build are persistent until clean. More
162 # details:
163 # http://www.gnu.org/software/make/manual/html_node/Chained-Rules.html
164 .PRECIOUS: %.o
165
166 UPROGS=\
167     _cat\
168     _echo\
169     _forktest\
170     _grep\
171     _init\
172     _kill\
173     _ln\
174     _ls\
175     _mkdir\
176     _rm\
177     _sh\
178     _stressfs\
179     _usertests\
180     _wc\
181     _zombie\
182     _user_curly_brace_correction_check\
183     _next_palindrome\
184     _set_sleep_syscall\
185
186 fs.img: mkfs README $(UPROGS)
187     ./mkfs fs.img README $(UPROGS)
188
189 -include *.d
190
191 clean:
```

پیاده سازی user program:

```
C set_sleep_syscall.c X
C set_sleep_syscall.c > ...
You, 22 hours ago | 1 author (You)
You, 2 weeks ago • implementation of set_sleep_syscall
1 #include "types.h"
2 #include "user.h"
3 #include "fcntl.h"
4 #include "date.h"
5 #define WRONGINPUTTRYAGAIN "invalid command!try again\n"
6 #define ISSUEOCCURED "issue occurred\n"
7 #define STARTEDYEARBYCMOSTIME 2000
8 int is_leap_year(int year)
9 {
10     return (year%4==0 && (year%100!=0 || year%400==0));
11 }
12 int convert_time_to_second(struct rtcdate* recorded_time)
13 {
14     int days_per_month[]={0}=0,[1]=31,[2]=28,[3]=31,[4]=30,[5]=31,[6]=30,[7]=31,[8]=31,[9]=30,[10]=31,30,31},total_second=0;
15     int day_to_second=0,year_by_cmostime=recorded_time->year,second_by_cmostime=recorded_time->second,hour_by_cmostime=recorded_time->hour,minute_by_cmostime=recorded_time->minute;
16     for (int years=STARTEDYEARBYCMOSTIME;years<year_by_cmostime;years++)
17     {
18         if(is_leap_year(year: years))
19             day_to_second+=366;
20         else
21             day_to_second+=365;
22     }
23     for (int i=1;i<recorded_time->month;i++) {
24         day_to_second+=days_per_month[i];
25         if(i==2 && is_leap_year(year: year_by_cmostime))
26             day_to_second+=1;
27     }
28     day_to_second+=recorded_time->day-1;
29     total_second=day_to_second*24*3600+hour_by_cmostime*3600 +minute_by_cmostime*60 +second_by_cmostime;
30     return total_second;
31 }
32 int main(int argc,char *argv[])
33 {
34     int difference=0,before_sleep_in_second=0,after_sleep_in_second=0;
35     struct rtcdate before_sleep,after_sleep;
36     if(argc!=2)
37         return 1;
38     if(set_sleep_syscall(argv[1],argv[2])<0)
39         return 1;
40     get_system_time(&before_sleep);
41     before_sleep_in_second=convert_time_to_second(recorded_time:&before_sleep);
42     get_system_time(&after_sleep);
43     after_sleep_in_second=convert_time_to_second(recorded_time:&after_sleep);
44     difference=after_sleep_in_second-before_sleep_in_second;
45     printf(1,"result:%d\n",difference);
46     exit(0);
47 }
```

```
C set_sleep_syscall.c X
C set_sleep_syscall.c > ...
41     day_to_second+=30;
42 }
43 for (int i=1;i<recorded_time->month;i++) {
44     day_to_second+=days_per_month[i];
45     if(i==2 && is_leap_year(year: year_by_cmostime))
46         day_to_second+=1;
47 }
48 day_to_second+=recorded_time->day-1;
49 total_second=day_to_second*24*3600+hour_by_cmostime*3600 +minute_by_cmostime*60 +second_by_cmostime;
50 return total_second;
51 }
52 int main(int argc,char *argv[])
53 {
54     int difference=0,before_sleep_in_second=0,after_sleep_in_second=0;
55     struct rtcdate before_sleep,after_sleep;
56     if(argc!=2)
57         return 1;
58     if(set_sleep_syscall(argv[1],argv[2])<0)
59         return 1;
60     printf(1,"%s\n",WRONGINPUTTRYAGAIN);
61     exit(1);
62 }
63 int input_tick=atoi(argv[1]);
64 get_system_time(&before_sleep);
65 if(set_sleep_syscall(input_tick,input_tick)<0)
66 {
67     printf(1,ISSUEOCCURED);
68     exit(1);
69 }
70 get_system_time(&after_sleep);
71 before_sleep_in_second=convert_time_to_second(recorded_time:&before_sleep);
72 after_sleep_in_second=convert_time_to_second(recorded_time:&after_sleep);
73 difference=after_sleep_in_second-before_sleep_in_second;
74 printf(1,"result:%d\n",difference);
75 exit(0);
76 }
```

منطق systemcall در proc.c پیاده شده است. در systemcall زمان به میزانی که ورودی خواسته نگه داشته می شود و در نهایت کنترل به user program می رود و در آنجا اختلاف بدست آمده به عنوان خروجی نوشته می شود.

نتیجه اجرا:

```
QEMU
Machine View
SeaBIOS (version 1.16.3-debian-1.16.3-2)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1EFCB050+1EF0B050 CA00

Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
o- SeyedAlirezaMirshafiee
o- SadraAbbasi
o- AmirSafari
init: starting sh
$ set_sleep_syscall 1000
result:10
$ _
```

پاسخ سوال مطرح شده: اولین مورد که باید به آن اشاره کرد این است که `cmostime` زمان کنونی را از `real-time clock` می خواند.

ساعت سیستم از `timer` سخت افزار کمک گرفته و با هر `timer interrupt` مقدار `tick` را افزایش می دهد.

وظیفه `rtc` نگهداری زمان واقعی است. `cmostime` زمان رو با دقت ثانیه نشان می دهد ولی زمانی که `tick` برمیگرداند با دقت ۱۰ میلی ثانیه خواهد بود. بنابراین اگر زمانی که به عنوان ورودی میدهیم در حد ۱۰ میلی ثانیه باشد زمانی که `cmostime` نشان میدهد متفاوت خواهد بود.

تأثیر خواباندن `process` برای مدت زمان معین تنها `prcess` را می خواباندن ولی ساعت سیستم به کار خود ادامه خواهد داد.