آزمایشگاه سیستم عامل

پروژه دوم: فراخوانی سیستم تاریخ تحویل:23 آبان

مقدمه

1)

فایل های تشکیل دهنده ULIB عبارتند از:

- Usys.S
- Printf.c
- Ulib.c
- Umalloc.c

از بین فایل های ذکر شده تنها usys.S می باشد که در فراخوانی سیستمی به کار می رود.

2)

سه مورد وجود دارد که در این موارد باید کنترل از سطح کاربر به هسته برود:

1. **فراخوانی سیستمی** (system call): وقتی که برنامه ی سطح کاربر از سیستم عامل درخواست سرویس می کند.
2. **استثنأ** (exception) **و وقفه** (interrupt): وقتی یک برنامه عمل غیرمجازی را انجام می دهد. مانند تقسیم بر صفر. همچنین وقتی که یک دستگاه(device) سیگنالی تولید می کند تا نشان دهد نیاز به توجه سیستم عامل دارد. برای مثال یک clock chip ممکن است هر 100 میلی ثانیه یک وقفه تولید کند تا به هسته قابلیت زمانبندی (time sharing) را بدهد.
3. **شبه‌فایل‌ها(pseudo-files)**: شبه‌فایل‌ها توسط کامند لاین یا اسکریپت مورداستفاده قرار می‌گیرند. مثل proc/ و sys/ و dev/

شبه فایل ها برای پارامتر های تنظیمی و سایرapi ها مناسب هستند.

سازوکار اجرای فراخوانی سیستمی در xv6

3)

خیر. به علت حفظ امنیت سیستم، سیستم عامل xv6 اجازه نمی دهد که پردازه ها بقیه ی تله ها مانند deviceها interrupt را با int فعال کنند و اگه تلاش به چنین کاری کنند با یک استثنای عمومی محافظتی مواجه می شوند که به vector 13 می رود. اگر DPL_USER به تله ها دسترسی داشت، این یعنی کاربر به تله ها دسترسی دارد که این امنیت سیستم را به خطر می اندازد.

4)

می‌دانیم ss پوینتر به بخش استک و esp سر استک را نگه می‌دارند. هنگامی که تغییر دسترسی نیاز باشد(از سطح کاربر به سطح کرنل) این دو ثبات به استک اضافه می‌شوند تا دوباره هنگامی که بخواهیم به سطح قبلی بازگردیم، لازم است این مقادیر این دو ثبات را داشته باشیم، حال هنگامی که تغییر دسترسی رخ نمی‌دهد، نیازی به اضافه کردن این دو ثبات نیست.

5)

1. **argint:** برای دریافت ورودی int می باشد که دو آرگومان دارد:
   اولی شماره پارامتر تابع و دومی آدرس متغیر int
2. **argstr:** برای دریافت ورودی string می باشد که دو آرگومان دارد:
   اولی شماره پارامتر تابع و دومی آدرس متغیر char.
3. **argptr:** برای دریافت ورودی pointer می باشد که سه آرگومان دارد:
   اولی شماره پارامتر تابع ، دومی آدرس اشاره گر متغیر پونتر و سومی اندازه مقداری که قرار است پوینتر بخواند را ورودی میگیرد.
4. **argfd:** سه آرگومان می گیرد:
   اولی شماره پارامتر تابع، دومی file descriptor و سومی struct file. که دو آرگومان آخر را خودش مقدار دهی می کند.

5. **Fetchint**: این تابع بررسی می کند که آدرس داده شده در محدوده معین باشد.

چرا در argptr()بازه آدرسها بررسی میگردد؟ تا بررسی گردد که آیا یک پوینتر قابل قبول به فضای کاربر است یا نه چرا که ممکن است به اطلاعات غلطی دسترسی پیدا کنیم.

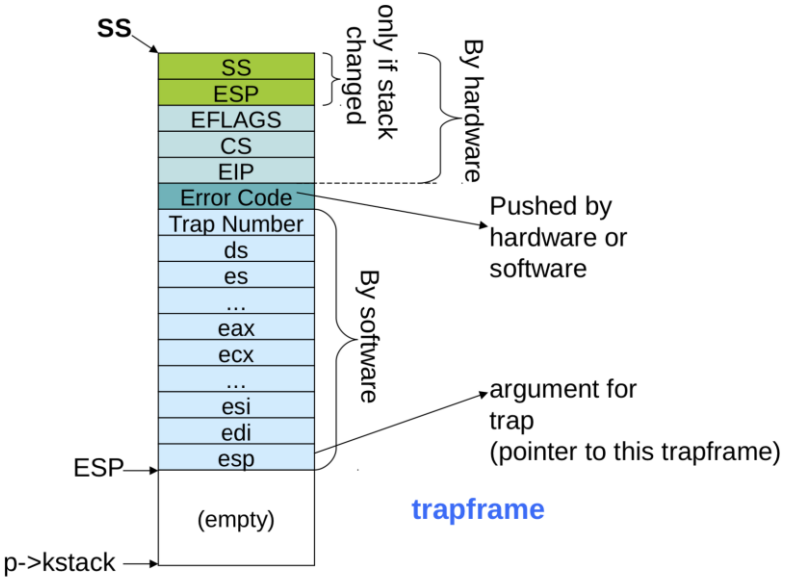
تجاوز از بازه معتبر، چه مشکل امنیتی ایجاد میکند؟اگر یک برنامه سطح کاربر تلاش کند که از بازه ی نامعتبر بخواند یا در آن بنویسد processor یک تله ی segmentation ایجاد می کند و آن تله process را kill می کند.

هنگام سیستم کال sys_read() که برای خواندن ورودی از کاربر می باشد، ممکن است به قسمت هایی از حافظه که متعلق به آن ورودی نباشد دسترسی غیر قابل قبول و بیش از حد مجاز پیدا کند و باعث مختل شدن اجرای سیستم شود.

(6

برای ادامه دادن از جای متوقف شده در برنامه بعد از رفتن از هسته به سطح کاربر از ساختار داده ای به نام trapframe استفاده می کنیم. که این ساختار داده دارای رجیستر هایی هستند که بعضی از آن ها توسط نرم افزار و بعضی توسط سخت افزار مقدار می گیرند و پس از برگشتن به سطح کاربر با استفاده از این رجیسترها می توان state های قبل را بازیابی کرد.

همانطور که در سوال 4 نیز گفته شد، دو ثبات ss و esp هنگام تغییر دسترسی از سطح کاربر به کرنل به استک اضافه می شدند به همین دلیل برای بازگشت به سطح کاربر از اطلاعات این دو ثبات استفاده می کنیم.



ref : struct trapframe in x86.h (0602 [06])

(7

عدم توالی دادههای یک فایل روی دیسک، باعث می شود که ندانیم هر سکتور برای کدام داده است. چراکه هنگام بازیابی اطلاعات،سکتور محتوای دیسک را می‌خواند، و برای هر بخش بررسی می کند که آیا این بخش همان سر فایل است یا خیر. به عبارتی ما فقط به سر و ته داده ها دسترسی داریم. پس اگر داده های یک فایل به صورت متوالی باشند، با دانستن مکان اولین و آخرین این داده ها به داده های وسطی نیز دسترسی خواهیم داشت

ارسال آرگومان های فراخوانی های سیستمی

برای اضافه کردن فراخوانی سیستمی، باید فایل های زیر تغییر کنند.

- syscall.h
- syscall.c
- sysproc.c
- usys.S
- user.h
- proc.c

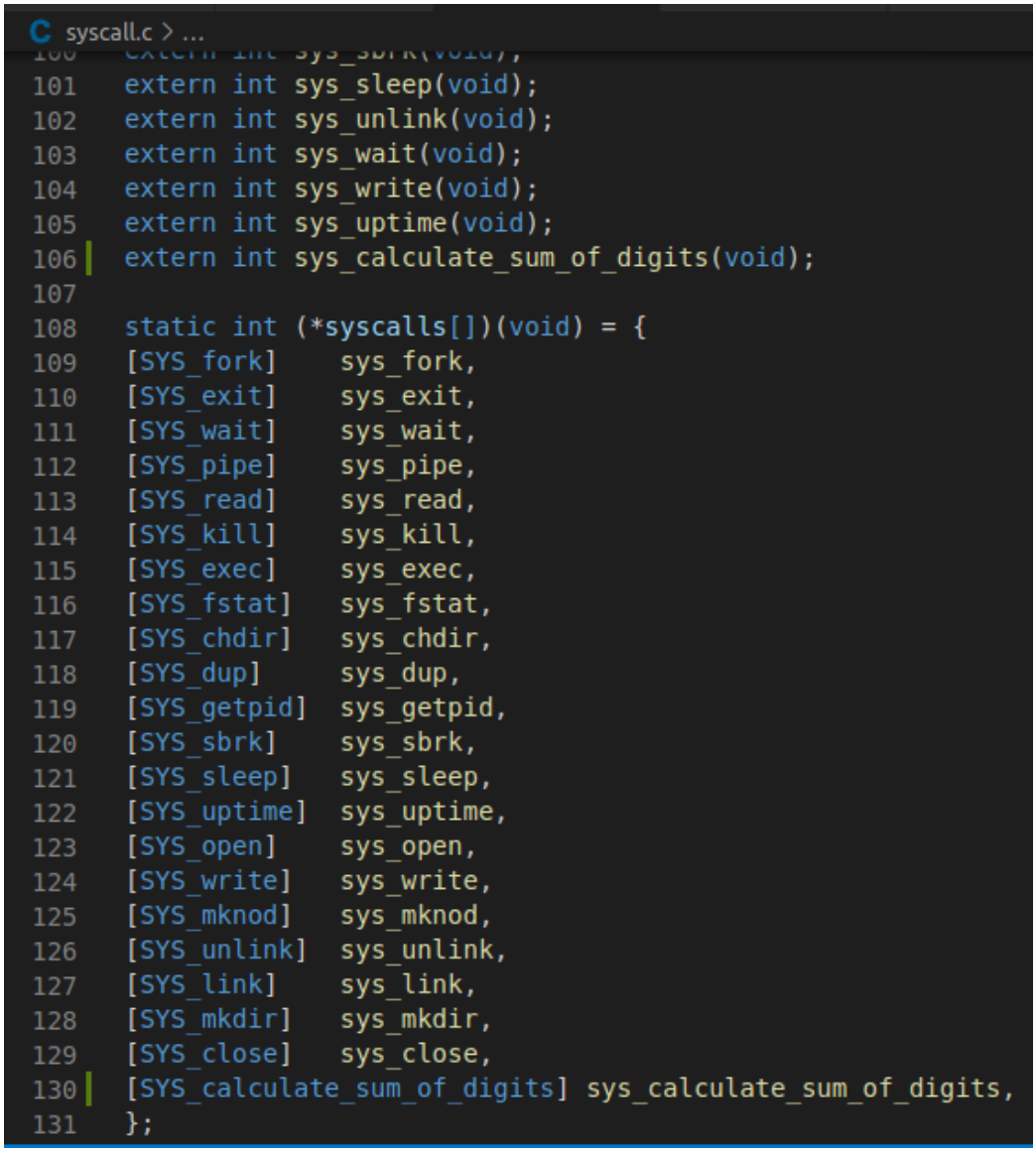
1. syscall.h

```h
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
#define SYS_mknod  17
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
#define SYS_calculate_sum_of_digits 22

```

رزرو شماره 22 فراخوانی سیستم(1تا 21 پیش فرض پر هستند)

2.

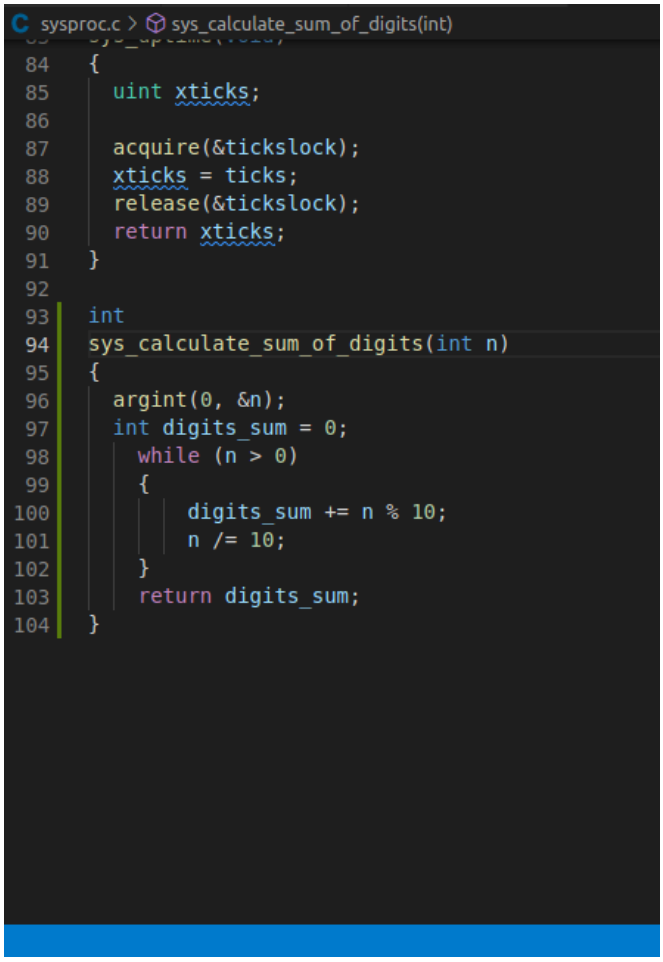
syscall.c



اضافه کردن یک پوینتر به آرایه function pointers در ایندکس تخصیص داده شده

3.

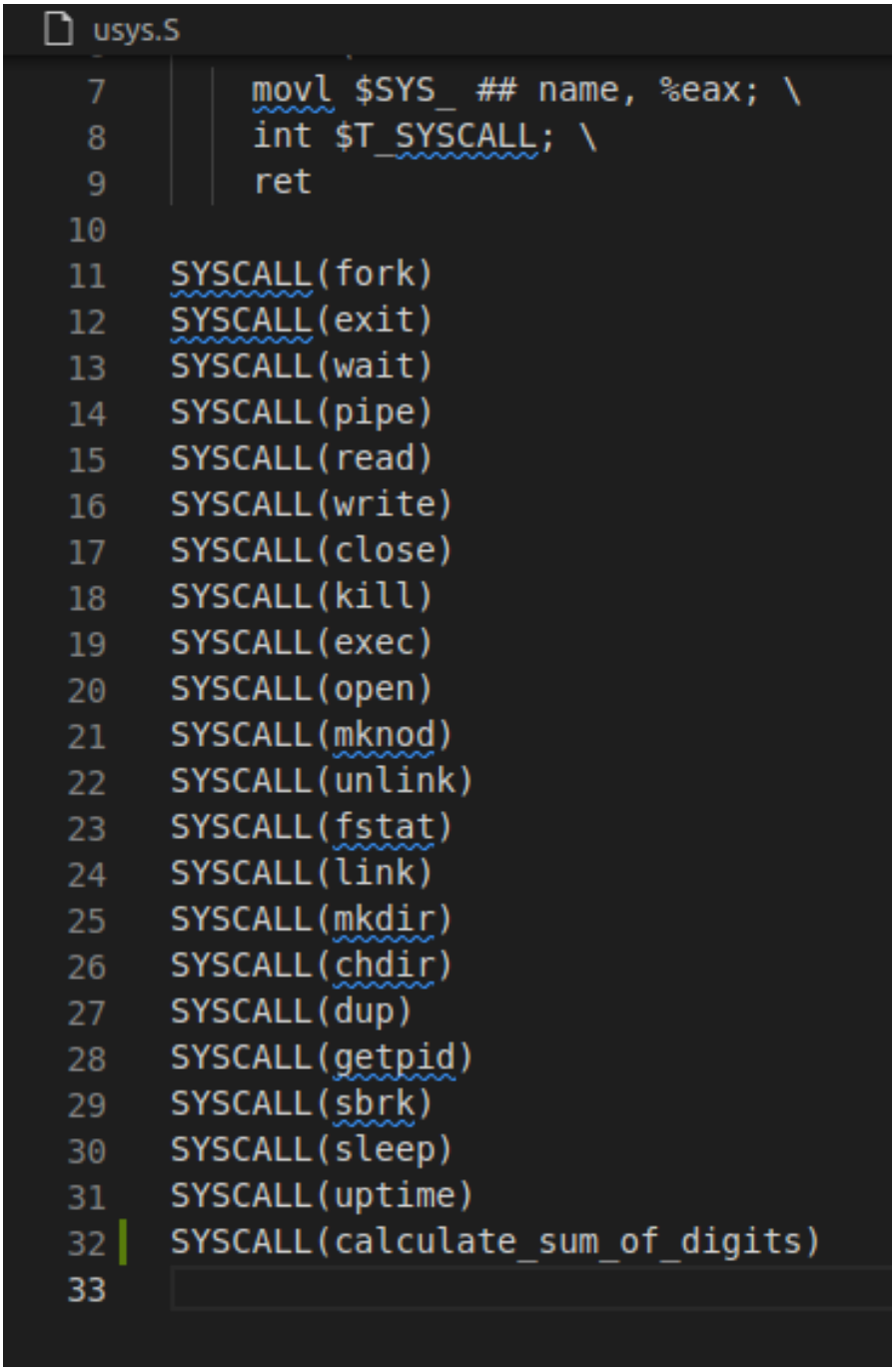
sysproc.c



Function implementation

4.

usys.S



اینترفیس لازم برای دسترسی کاربر به فراخوانی سیستمی

(user interface)

5.

user.h

```h
struct stat;
struct rtcdate;

// system calls
int fork(void);
int exit(void) __attribute__((noreturn));
int wait(void);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int calculate_sum_of_digits(int);

// ulib.c
int stat(const char*, struct stat*);
char* strcpy(char*, const char*);
void *memmove(void*, const void*, int);
```

تابعی که قرار است کاربر آن را صدا بزند را اضافه می کنیم. (user interface)

کد تست به صورت رو به روست:

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

int my_stoi(char *inp){…

int
main(int argc, char *argv[])
{
    int num = my_stoi(argv[1]);

    int sum = calculate_sum_of_digits(num);

    printf(1, "Sum of digits is: %d\n", sum);

    exit();
}
```

تغییرات زیر در Makefile صورت گرفته است:

Makefile

```
# Prevent deletion of intermediate files, e.g. cat.o, after first build, so
# that disk image changes after first build are persistent until clean.  More
# details:
# http://www.gnu.org/software/make/manual/html_node/Chained-Rules.html
.PRECIOUS: %.o

UPROGS=\
    _cat\
    _echo\
    _forktest\
    _grep\
    _init\
    _kill\
    _ln\
    _ls\
    _mkdir\
    _rm\
    _sh\
    _stressfs\
    _usertests\
    _wc\
    _zombie\
    _Test_CSOD\

fs.img: mkfs README $(UPROGS)
    ./mkfs fs.img README $(UPROGS)

-include *.d

clean:
    rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
```

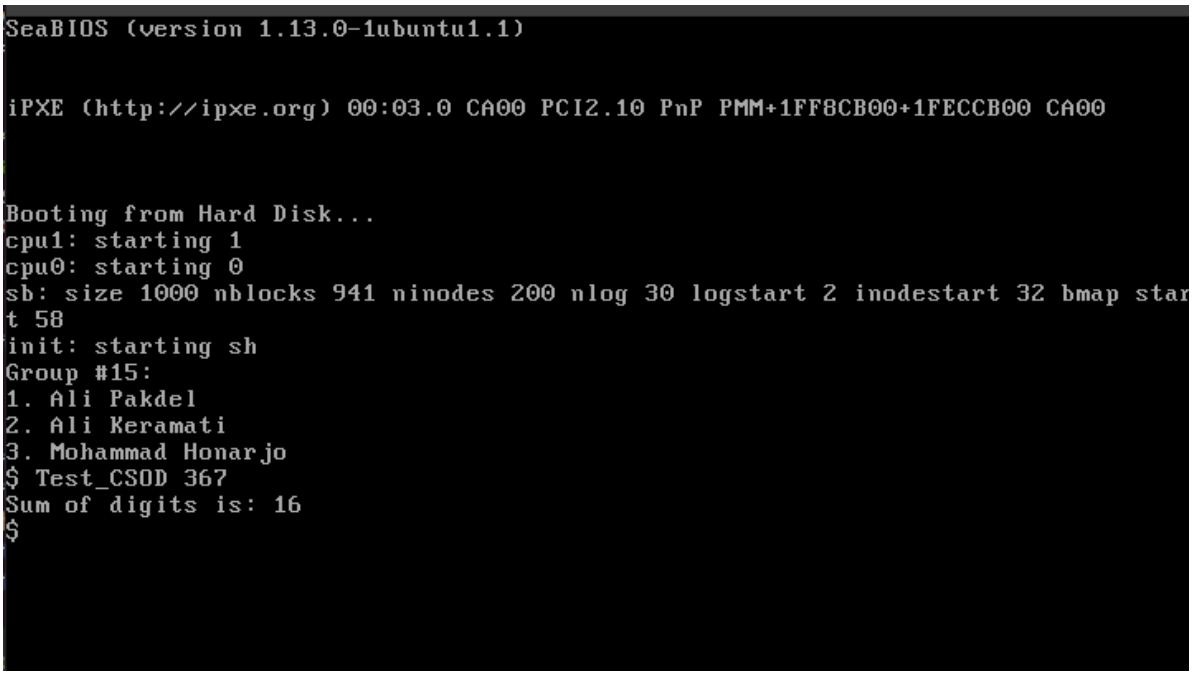
اضافه کردن _Test_CSOD

Makefile

```
qemu-nox-gdb: fs.img xv6.img .gdbinit
    @echo "*** Now run 'gdb'." 1>&2
    $(QEMU) -nographic $(QEMUOPTS) -S $(QEMUGDB)

# CUT HERE
# prepare dist for students
# after running make dist, probably want to
# rename it to rev0 or rev1 or so on and then
# check in that version.

EXTRA=\
    mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
    ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c Test_CSOD.c\
    printf.c umalloc.c\
    README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
    .gdbinit.tmpl gdbutil\

dist:
    rm -rf dist
    mkdir dist
    for i in $(FILES); \
    do \
        grep -v PAGEBREAK $$i >dist/$$i; \
    done
    sed '/CUT HERE/,$$d' Makefile >dist/Makefile
    echo >dist/runoff.spec
    cp $(EXTRA) dist

dist-test:
    rm -rf dist
```

اضافه کردن Test_CSOD.c\

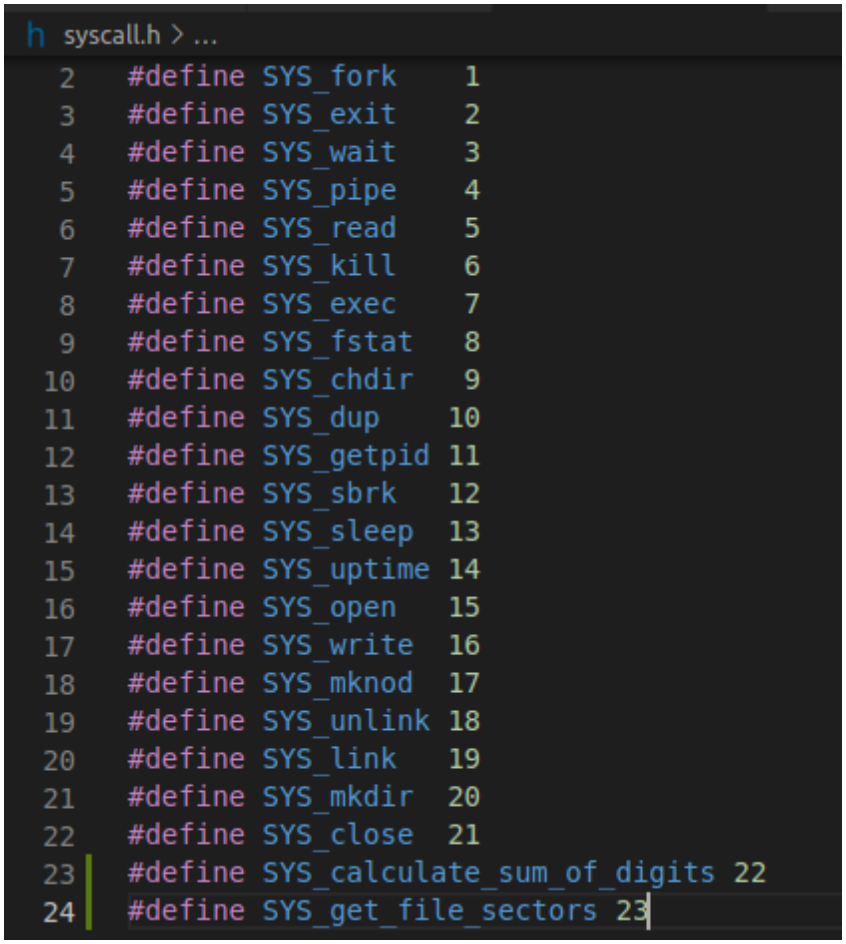
در نهایت خروجی فراخوانی سیستمی ما به صورت زیر است:



پیاده سازی فراخوانی سیستمی پیدا کردن آدرس سکتور های فایل

1.

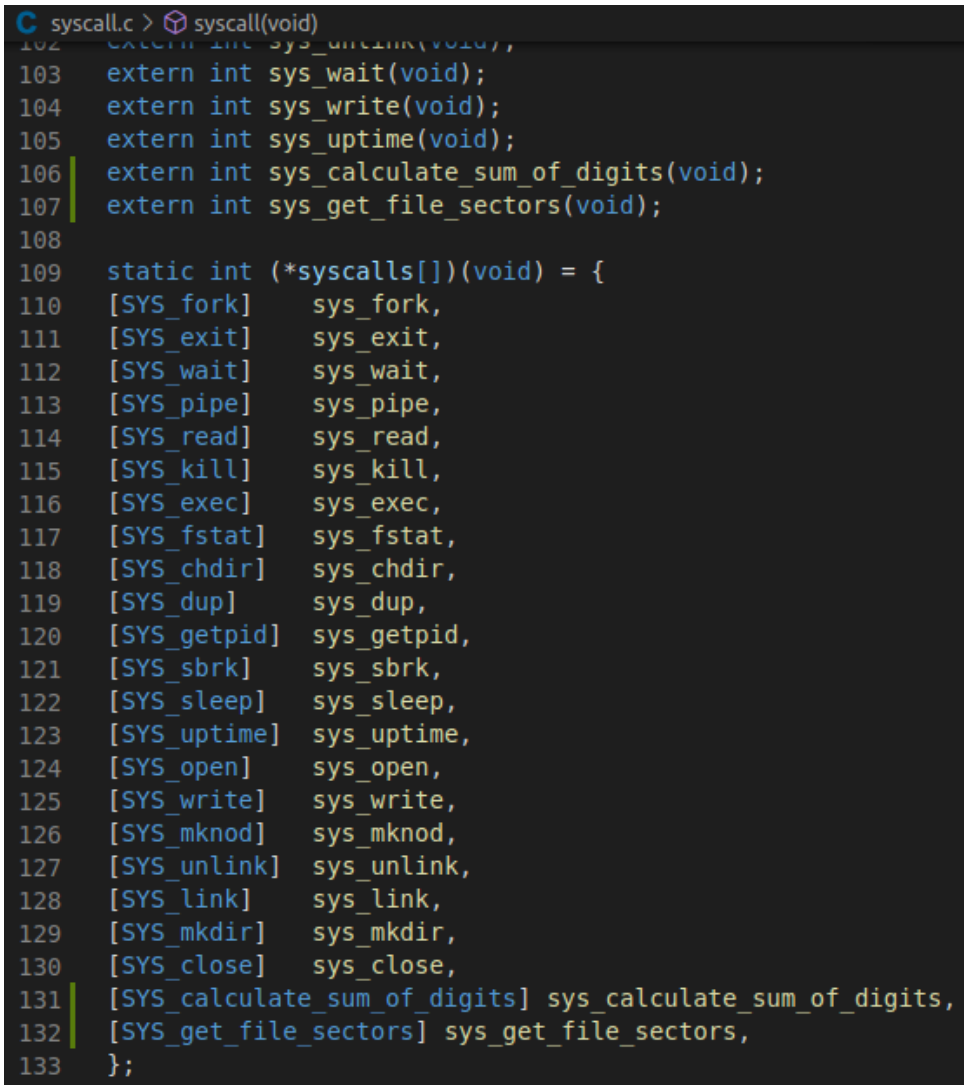
syscall.h



رزرو شماره 23 فراخوانی سیستم

2.

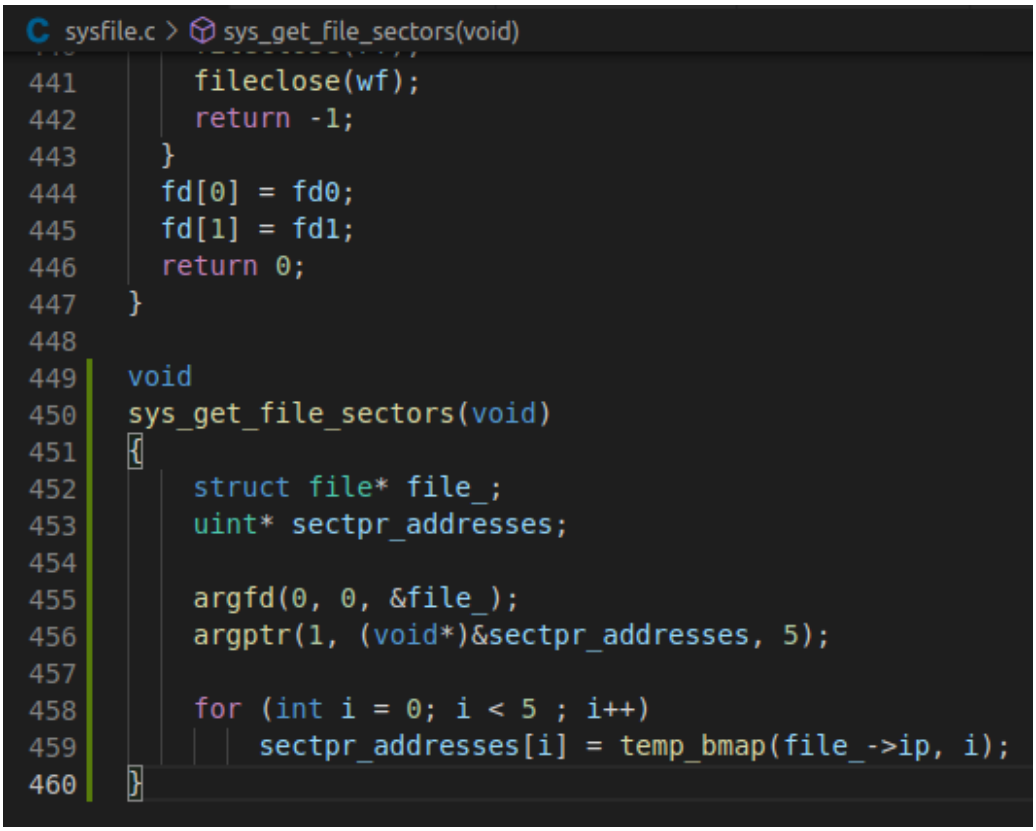
syscall.c



اضافه کردن یک پوینتر به آرایه function pointers در ایندکس تخصیص داده شده

3.

sysfile.c



Function implementation

4.

usys.S

usys.S

```
ret

SYSCALL(fork)
SYSCALL(exit)
SYSCALL(wait)
SYSCALL(pipe)
SYSCALL(read)
SYSCALL(write)
SYSCALL(close)
SYSCALL(kill)
SYSCALL(exec)
SYSCALL(open)
SYSCALL(mknod)
SYSCALL(unlink)
SYSCALL(fstat)
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(calculate_sum_of_digits)
SYSCALL(get_file_sectors)
```

اینترفیس لازم برای دسترسی کاربر به فراخوانی سیستمی

(user interface)

5.

user.h

```h
// system calls
int fork(void);
int exit(void) __attribute__((noreturn));
int wait(void);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int calculate_sum_of_digits(int);
int get_file_sectors(int, uint*, int);
```

تابعی که قرار است کاربر آن را صدا بزند را اضافه می کنیم. (user interface)

کد تست به صورت رو به روست:

```c
}

int
main(int argc, char *argv[])
{
    // char* buff = "Fateh Garh Resort  5   Sit

    // if (open(argv[2], O_RDONLY) >= 0)
    //     unlink(argv[2]);

    // int num = my_stoi(argv[1]);
    // uint sector_addresses[num];
    // num *= 512;

    // int fd = open(argv[2], O_CREATE | O_WRONL
    // write(fd, buff, sizeof(buff));
    // close(fd);

    int fd = open("test2.txt", O_RDONLY);
    uint sector_addresses[5];
    get_file_sectors(fd, sector_addresses);
    close(fd);

    for (int i = 0; i < 5; i++)
    {
        printf(1, "%d ", sector_addresses[i]);
    }

    exit();
}
```

تغییرات زیر در Makefile صورت گرفته است:

Makefile

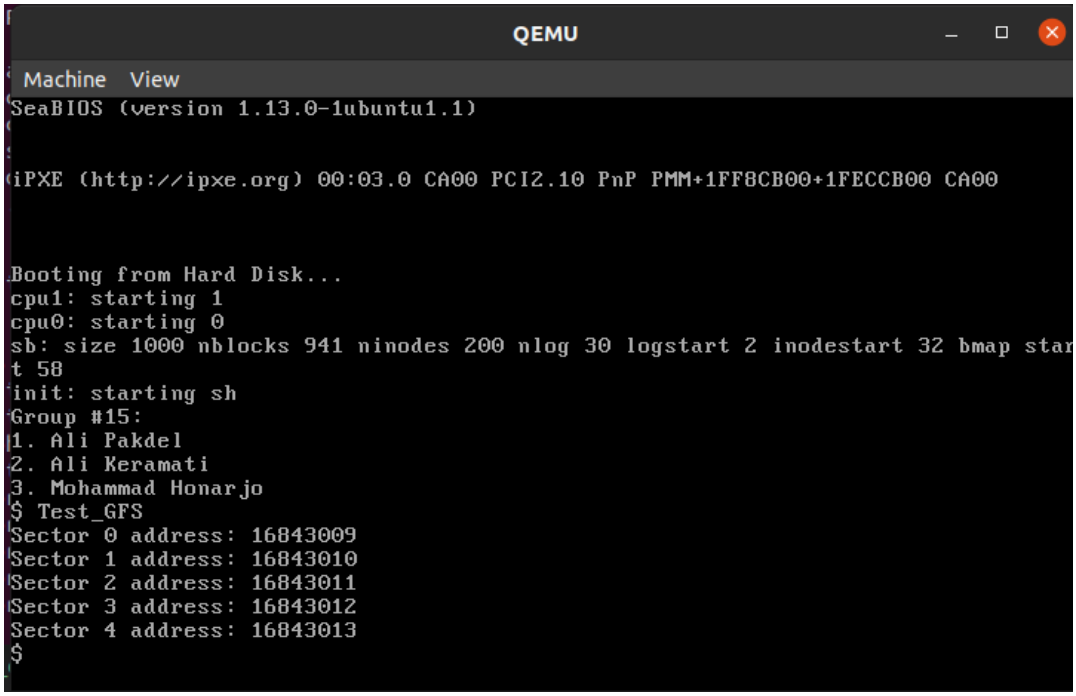
```
161
162  # Prevent deletion of intermediate files, e.g. cat.o, after first build, s
163  # that disk image changes after first build are persistent until clean.  M
164  # details:
165  # http://www.gnu.org/software/make/manual/html_node/Chained-Rules.html
166  .PRECIOUS: %.o
167
168  UPROGS=\
169      _cat\
170      _echo\
171      _forktest\
172      _grep\
173      _init\
174      _kill\
175      _ln\
176      _ls\
177      _mkdir\
178      _rm\
179      _sh\
180      _stressfs\
181      _usertests\
182      _wc\
183      _zombie\
184      _Test_CSOD\
185      _Test_GFS\
186
```

اضافه کردن _Test_GFS\

```
251
252  EXTRA=\
253      mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
254      ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c Test_CSOD.c Test_GFS.c\
255      printf.c umalloc.c\
256      README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
257      .gdbinit.tmpl gdbutil\
258
259  dist:
260      rm -rf dist
261      mkdir dist
262      for i in $(FILES); \
263      do \
264          grep -v PAGEBREAK $$i >dist/$$i; \
265      done
266      sed '/CUT HERE/,$$d' Makefile >dist/Makefile
267      echo >dist/runoff.spec
```

اضافه کردن Test_GfS.c\

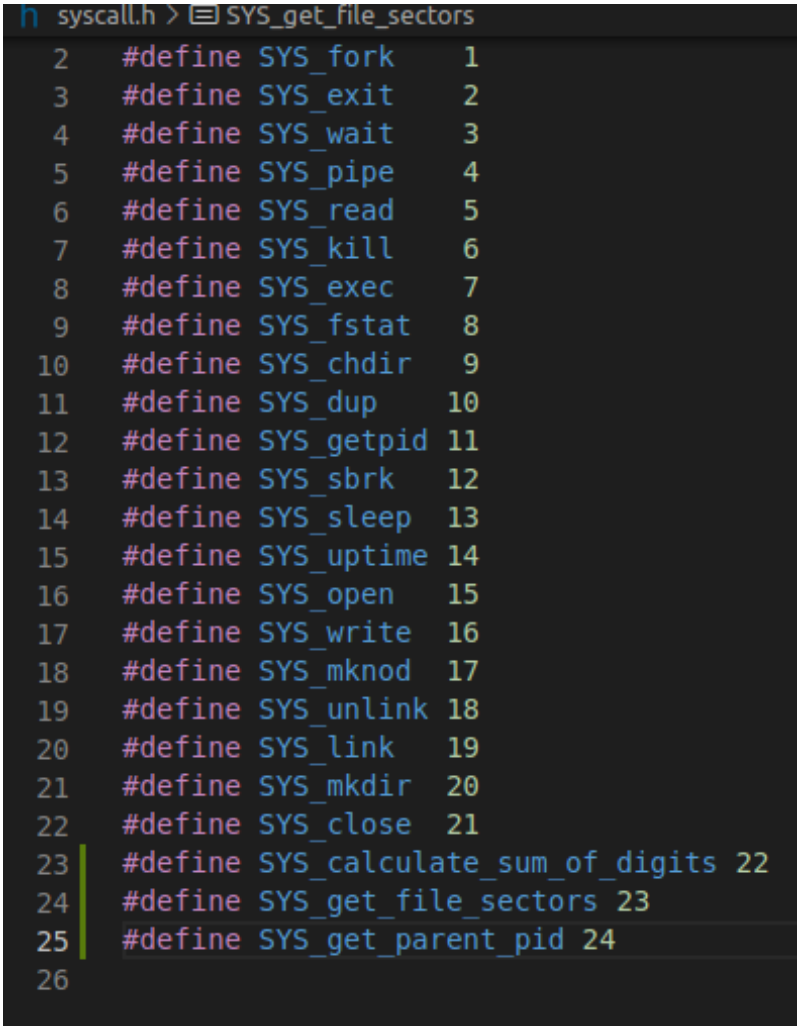
در نهایت خروجی فراخوانی سیستمی ما به صورت زیر است:



پیاده سازی فراخوانی سیستمی گرفتن پردازه پدر

1.

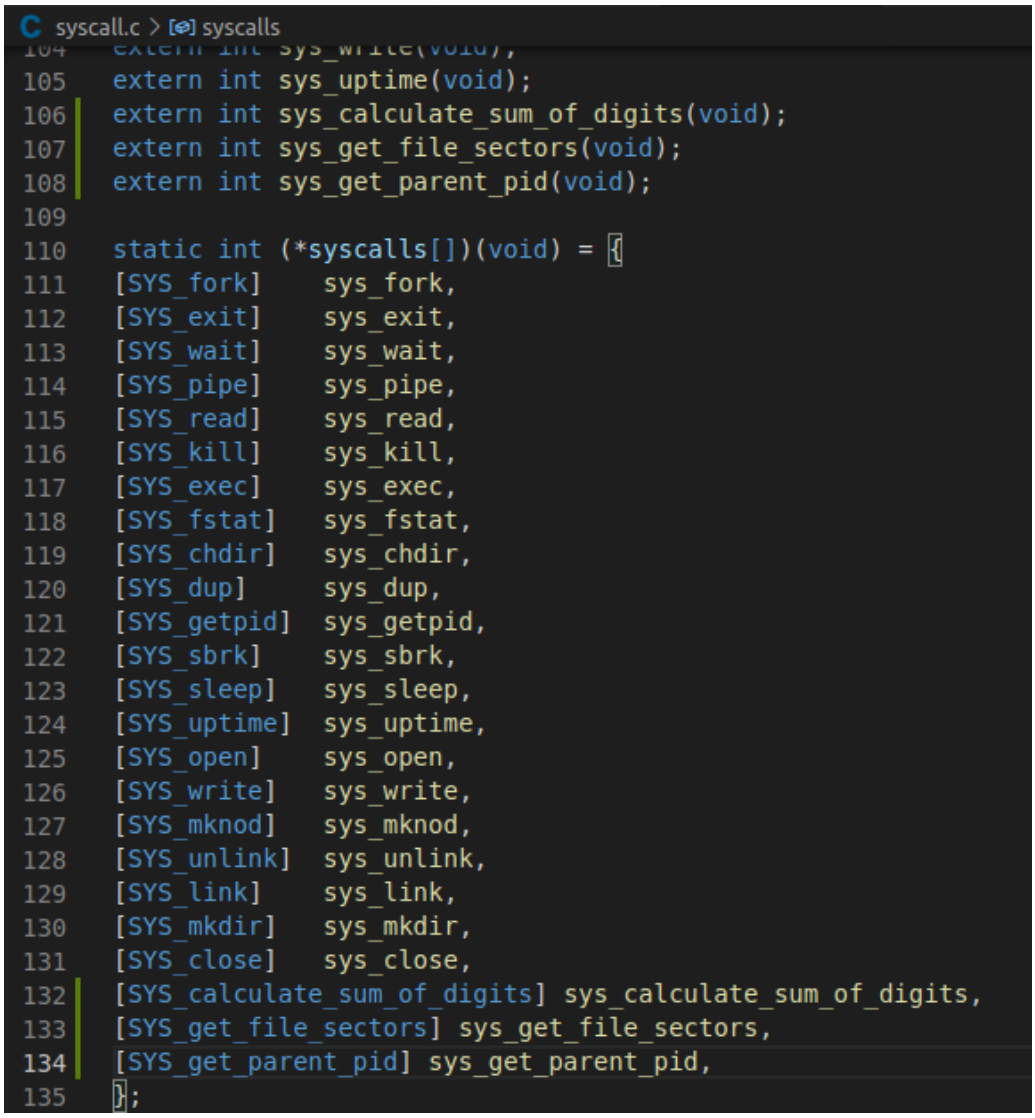
syscall.h



رزرو شماره 24 فراخوانی سیستم

2.

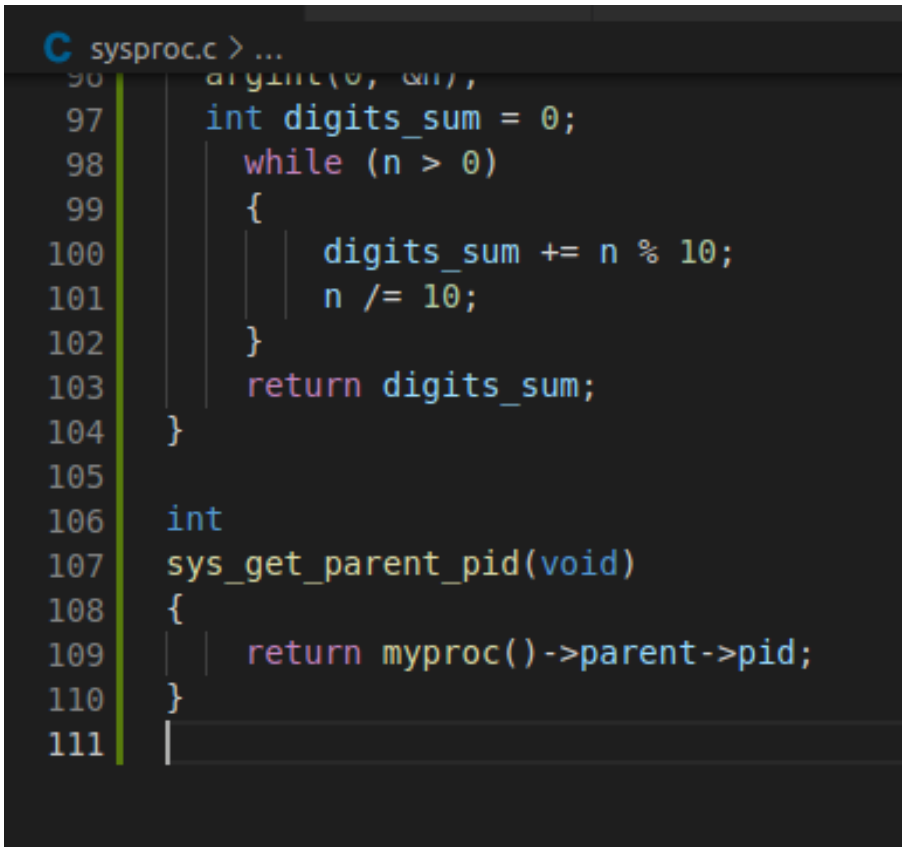
syscall.c



اضافه کردن یک پوینتر به آرایه function pointers در ایندکس تخصیص داده شده

3.

sysproc.c



Function implementation

4.

usys.S

usys.S

```
ret

SYSCALL(fork)
SYSCALL(exit)
SYSCALL(wait)
SYSCALL(pipe)
SYSCALL(read)
SYSCALL(write)
SYSCALL(close)
SYSCALL(kill)
SYSCALL(exec)
SYSCALL(open)
SYSCALL(mknod)
SYSCALL(unlink)
SYSCALL(fstat)
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(calculate_sum_of_digits)
SYSCALL(get_file_sectors)
SYSCALL(get_parent_pid)
```

اینترفیس لازم برای دسترسی کاربر به فراخوانی سیستمی

(user interface)

5.

user.h

```h
// system calls
int fork(void);
int exit(void) __attribute__((noreturn));
int wait(void);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int calculate_sum_of_digits(int);
int get_file_sectors(int, uint*, int);
int get_parent_pid(void);
```

تابعی که قرار است کاربر آن را صدا بزند را اضافه می کنیم. (user interface)

کد تست به صورت زیر ست:

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

int
main(int argc, char *argv[])
{
    printf(1, "Parent pid is: %d\n", get_parent_pid());
    exit();
}
```

تغییرات زیر در Makefile صورت گرفته است:

Makefile

```
166  .PRECIOUS: %.o
167
168  UPROGS=\
169      _cat\
170      _echo\
171      _forktest\
172      _grep\
173      _init\
174      _kill\
175      _ln\
176      _ls\
177      _mkdir\
178      _rm\
179      _sh\
180      _stressfs\
181      _usertests\
182      _wc\
183      _zombie\
184      _Test_CSOD\
185      _Test_GFS\
186      _Test_GPP\
```

اضافه کردن _Test_GPP\

```
253  EXTRA=\
254      mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
255      ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c Test_CSOD.c Test_GFS.c Test_GPP.c\
256      printf.c umalloc.c\
257      README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
258      .gdbinit.tmpl gdbutil\
```

اضافه کردن Test_GPP.c\

در نهایت خروجی فراخوانی سیستمی ما به صورت زیر است:

QEMU

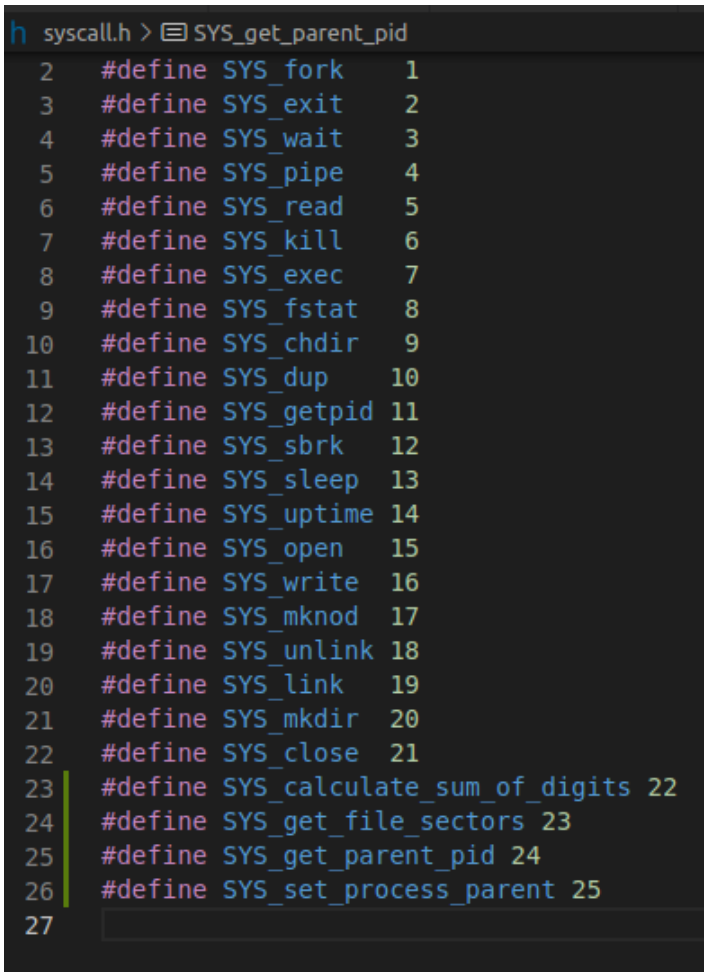
Machine View

```
SeaBIOS (version 1.13.0-1ubuntu1.1)


iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CB00+1FECCB00 CA00



Booting from Hard Disk...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group #15:
1. Ali Pakdel
2. Ali Keramati
3. Mohammad Honarjo
$ Test_GPP
Parent pid is: 2
$
```

پیاده سازی فراخوانی سیستمی تغییر پدر یک پردازه

1.

syscall.h



رزرو شماره 25 فراخوانی سیستم

2.

syscall.c

```
syscall.c > ...
106   extern int sys_calculate_sum_of_digits(void);
107   extern int sys_get_file_sectors(void);
108   extern int sys_get_parent_pid(void);
109   extern int sys_set_process_parent(void);
110
111   static int (*syscalls[])(void) = {
112   [SYS_fork]    sys_fork,
113   [SYS_exit]    sys_exit,
114   [SYS_wait]    sys_wait,
115   [SYS_pipe]    sys_pipe,
116   [SYS_read]    sys_read,
117   [SYS_kill]    sys_kill,
118   [SYS_exec]    sys_exec,
119   [SYS_fstat]   sys_fstat,
120   [SYS_chdir]   sys_chdir,
121   [SYS_dup]     sys_dup,
122   [SYS_getpid]  sys_getpid,
123   [SYS_sbrk]    sys_sbrk,
124   [SYS_sleep]   sys_sleep,
125   [SYS_uptime]  sys_uptime,
126   [SYS_open]    sys_open,
127   [SYS_write]   sys_write,
128   [SYS_mknod]   sys_mknod,
129   [SYS_unlink]  sys_unlink,
130   [SYS_link]    sys_link,
131   [SYS_mkdir]   sys_mkdir,
132   [SYS_close]   sys_close,
133   [SYS_calculate_sum_of_digits] sys_calculate_sum_of_digits,
134   [SYS_get_file_sectors] sys_get_file_sectors,
135   [SYS_get_parent_pid] sys_get_parent_pid,
136   [SYS_set_process_parent] sys_set_process_parent,
137   };
```

اضافه کردن یک پوینتر به آرایه function pointers در ایندکس تخصیص داده شده

3.

sysproc.c

```
int
sys_get_parent_pid(void)
{
    if (myproc()->debugging_mode)
        return myproc()->main_parent->pid;
    return myproc()->parent->pid;
}

void
sys_set_process_parent(void)
{
    int pid;
    argint(0, &pid);

    struct proc* proc_;
    proc_ = set_process_parent_(pid);
    proc_->debugging_mode = 1;
    proc_->main_parent = proc_->parent;
    proc_->parent = myproc();
    cprintf("Parent of pid %d has changed: ", pid);
    cprintf("\nMain parent: %d\nTemperory parent: %d\n", proc_->main_parent->pid, proc_->parent->pid);
}

```

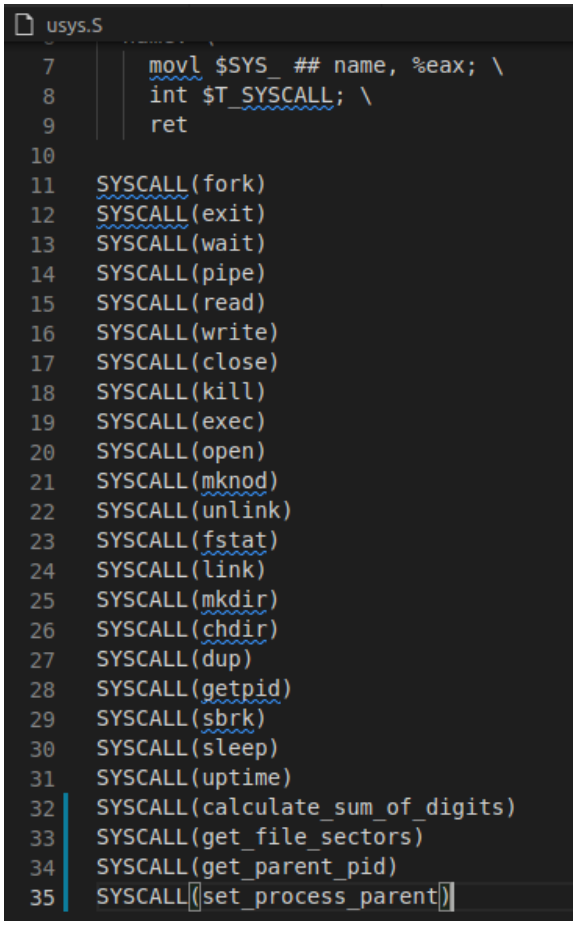
Proc.c

```
proc.c > ...
531         }
532         cprintf("\n");
533       }
534     }
535
536     struct proc*
537     set_process_parent_(int pid)
538     {
539         struct proc* proc_;
540         proc_ = ptable.proc;
541         while (proc_->pid != pid)
542         {
543             if(proc_ >= &ptable.proc[NPROC])
544                 break;
545             proc_++;
546         }
547         return proc_;
548     }
549
```

Function implementation

4.

usys.S



اینترفیس لازم برای دسترسی کاربر به فراخوانی سیستمی

(user interface)

5.

user.h

```
h user.h > ...
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  int calculate_sum_of_digits(int);
27  int get_file_sectors(int, uint*, int);
28  int get_parent_pid(void);
29  void set_process_parent(int);
30
31  // ulib.c
32  int stat(const char*, struct stat*);
33  char* strcpy(char*, const char*);
34  void *memmove(void*, const void*, int);
35  char* strchr(const char*, char c);
36  int strcmp(const char*, const char*);
```

تابعی که قرار است کاربر آن را صدا بزند را اضافه می کنیم. (user interface)

کد تست اول به صورت زیر ست:

Test_SPP1.c > main(int, char * [])

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

int
main(int argc, char *argv[])
{
    printf(1, "Current Pid: %d\nParent Pid: %d\n", getpid(), get_parent_pid());
    sleep(1500);
    printf(1, "After debugging: \nCurrent Pid: %d\nParent Pid: %d\n", getpid(), get_parent_pid());
    exit();
}

```

کد تست دوم(دیباگ) به صورت زیر ست:

Test_SPP2.c > main(int, char * [])

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

int my_stoi(char *inp){…

int
main(int argc, char *argv[])
{
    printf(1, "Debugging mode has started with pid %d\n", getpid());
    int pid = my_stoi(argv[1]);
    set_process_parent(pid);
    wait();
    printf(1, "Debugging mode has stopped\n");
    exit();
}

```

تغییرات زیر در Makefile صورت گرفته است:

Makefile

```
167
168  UPROGS=\
169      _cat\
170      _echo\
171      _forktest\
172      _grep\
173      _init\
174      _kill\
175      _ln\
176      _ls\
177      _mkdir\
178      _rm\
179      _sh\
180      _stressfs\
181      _usertests\
182      _wc\
183      _zombie\
184      _Test_CSOD\
185      _Test_GFS\
186      _Test_GPP\
187      _Test_SPP1\
188      _Test_SPP2\
189
```

اضافه کردن _Test_SPP1\ , _Test_SPP2

```
EXTRA=\
    mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
    ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c Test_CSOD.c Test_GFS.c Test_GPP.c Test_SPP1.c Test_SPP2.c\
    printf.c umalloc.c\
    README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
    .gdbinit.tmpl gdbutil\
```

اضافه کردن \Test_SPP1 Test_SPP2.c

در نهایت خروجی فراخوانی سیستمی ما به صورت زیر است:

QEMU

Machine View

```
Booting from Hard Disk...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group #15:
1. Ali Pakdel
2. Ali Keramati
3. Mohammad Honarjo
$ Test_SPP1 &
$ Current Pid: 4
Parent Pid: 1
Test_SPP2 4
Debugging mode has started with pid 5
Parent of pid 4 has changed:
Main parent: 1
Temperory parent: 5
After debugging:
Current Pid: 4
Parent Pid: 5
Debugging mode has stopped
$ _
```