بسم الله الرحمن الرحیم

پروژه دوم درس آزمایشگاه سیستم عامل

دکتر کارگهی

محمد امین توانایی - ۸۱۰۱۰۱۳۹۶

سید علی تهامی - ۸۱۰۱۰۱۳۹۷

مهدی وجهی - ۸۱۰۱۰۱۵۵۸

Question 1:

The ULIB variable contains files that implement the wrapper functions that are widely used in user programs and include such system calls, so that writing them as system calls in the user program is very complicated.

Some of these functions are like converting a string to a number or allocating dynamic memory or writing on the screen. Also, this variable contains a file that provides different system calls to the user so that he can use them while writing the program. Also this Programs can provide more understandable feedback to the user when there is an error.

If the implementation of the system calls are different in different systems, by providing a higher single layer such as wrapper functions, you can make yourself independent from the system and the libraries manage this difference internally.

Also, these libraries can have a significant impact on the performance of the program. Usually, overhead is imposed due to the addition of a new layer, but on the other hand, they can be implemented in a way that optimally uses different system calls and increases performance.

Question 2:

1. Virtual system call: Not every request really needs a context switch and for commonly used functions where shared data is accessed there are two machanisms. One is the legacy vsyscall memory mapping which contains simple pseudo-syscall functions like gettimeofday() and a shared memory region which holds the data to return.
2. Signals : A process can register a handler function to replace the default action to e.g. handle a Ctrl-C SIGABRT on the terminal. The handler is asynchronously invoked independently from the normal programm execution. A signal mark can block signals during handler execution
3. Sockets : Messages through sockets offer flexibility and portability compared to direct syscalls or shared memory.Concerning sockets there are various families available and it is to be distinguished between datagram and stream mode which gives a guaranteed ordering without the notion of packages.

سوال ۳

خیر، سیستم عامل برای محافظت و امنیت از اجرایی تمامی سیستم کال ها با سطح دسترسی یوزر مخالفت می کند و اکسپشن مناسبی را برمی گرداند زیرا در صورت عدم وجود این موضوع می تواند مشکلاتی برای سیستم عامل، خود برنامه و سایر برنامه ها ایجاد کند.

سوال ۴

وقتی می خواهیم سطح دسترسی را تغییر بدیم لازم است استک پوینتر را ذخیره کنیم که وقتی به برنامه بر می گردیم سر استک را بدانیم و همچنین لازم است سطح برنامه را هم ذخیره کنیم که در هنگام برگشت مجدد به همان مد برگردیم و واضح است وقتی این کار رو نمی کنیم لازم نیست این اطلاعات را ذخیره کنیم.

سوال ۵

تابع برای استخراج آرگومان ها به صورت زیر است:

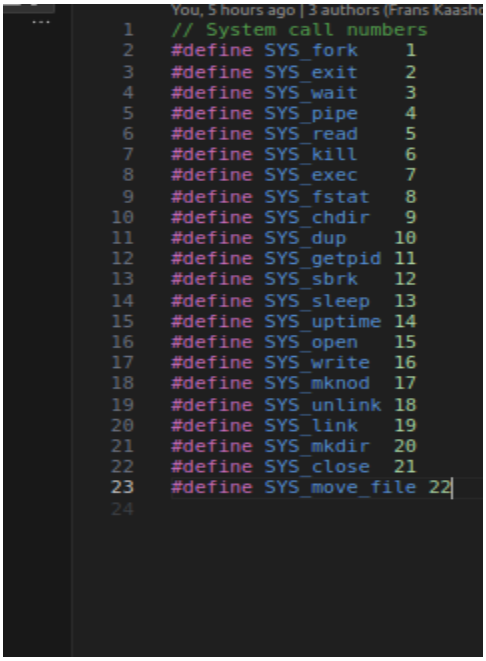
- argint(): برای استخراج یک عدد صحیح.
- argptr(): برای استخراج یک اشاره‌گر و بررسی بازه آدرس.
- argstr(): برای استخراج یک رشته و بررسی بازه آدرس.

هر کدام از این موارد بررسی می کنند که آدرس درخواستی در فضای حافظه باشد و دسترسی به آن مجاز باشد سپس مقادیر را در متغیر پاس داده شده قرار می دهد. برای argstr هم بازه درخواستی بررسی می شود که از محدوده برنامه خارج نشود. این موضوع از آن جهت می تواند خطرناک باشد که ما وقتی این تابع را صدا می کنیم در مد کرنل هستیم و اگر آدرس در محدوده برنامه هم نباشد اجازه دسترسی داریم و این موضوع می تواند مشکلات حفاظتی و امنیتی ایجاد کند. مثلا در تابع sys_read اگر این موضوع کنترل نشود کاربر می تواند با دادن مقدار بزرگی به عنوان طول رشته مقادیر موجود در باقی خانه های حافظه که مربوط به آن برنامه نیست را هم بخواند.

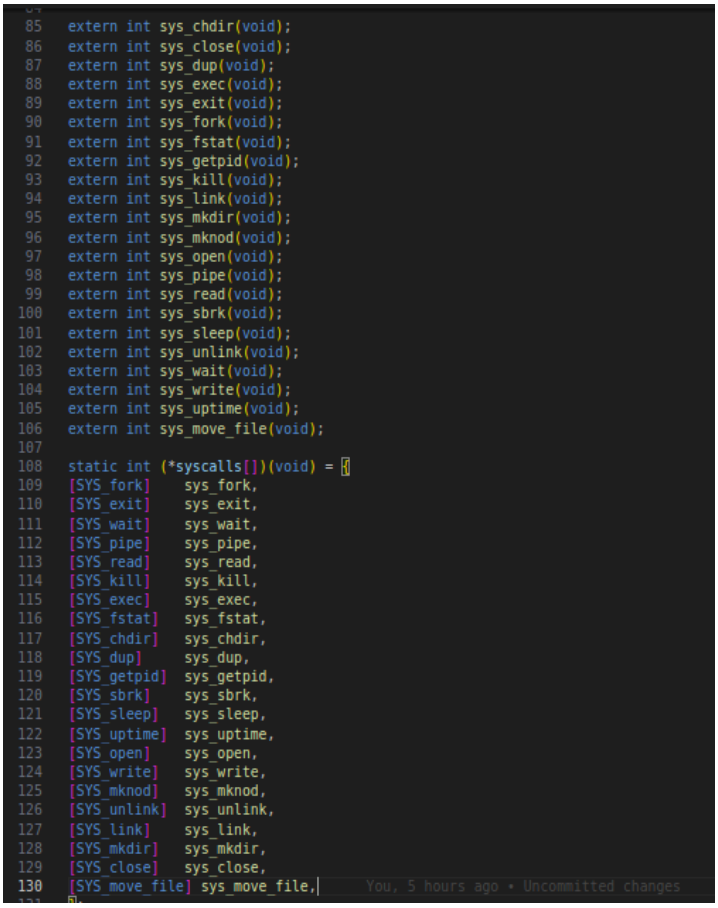
اضافه کردن سیستم کال ها

1. Move_file

a. اضافه کردن عدد مربوط به سیستم کال در فایل syscall.h

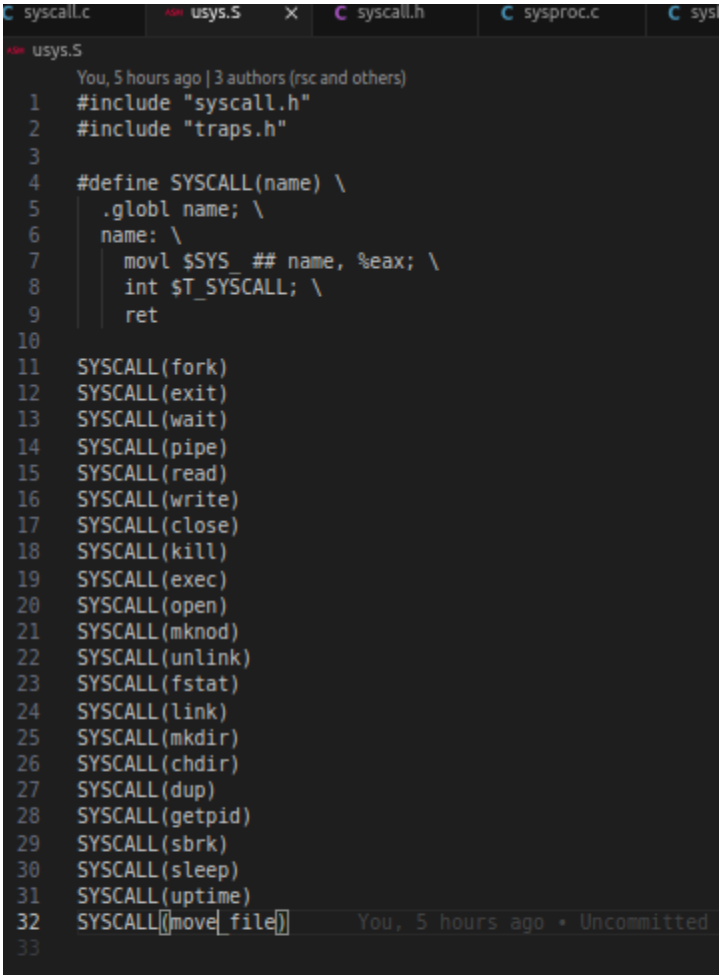


b. متصل کردن سیستم کال به تابع سیستم کال مدنظر در syscall.c

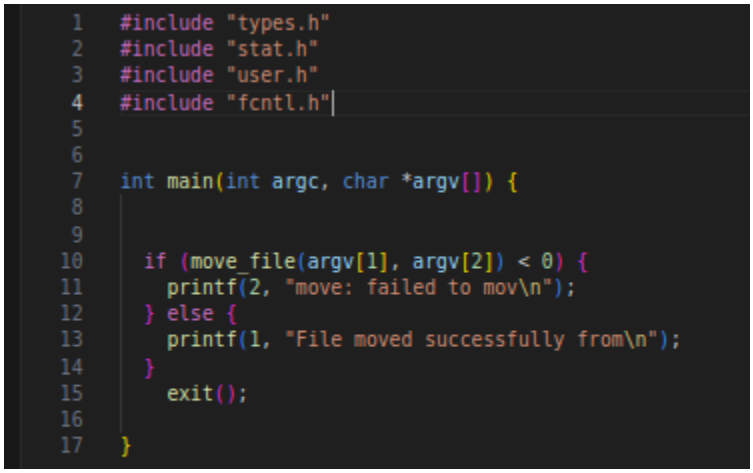


c. اضافه کردن تابع مربوط استفاده یوزر در user.h

d. تنظیم usys.S برای هماهنگی بین سیستم کال بدون آرگومان سطح سیستم و تابع سطح یوزر



e. اضافه کردن تابع چک کننده عملکرد سیستم کال



f. اضافه کردن تعریق تابع sys_move_file در sysfile.c

برای این کار لازم است تا آیپی و نود فایل مبدا را پیدا و دایرکتوری پدر آن را معلوم کنیم سپس لینک بین این فایل و پدرش را قطع کرده و آن را به پدر جدیدش لینک کنیم.

2. Sort_syscalls

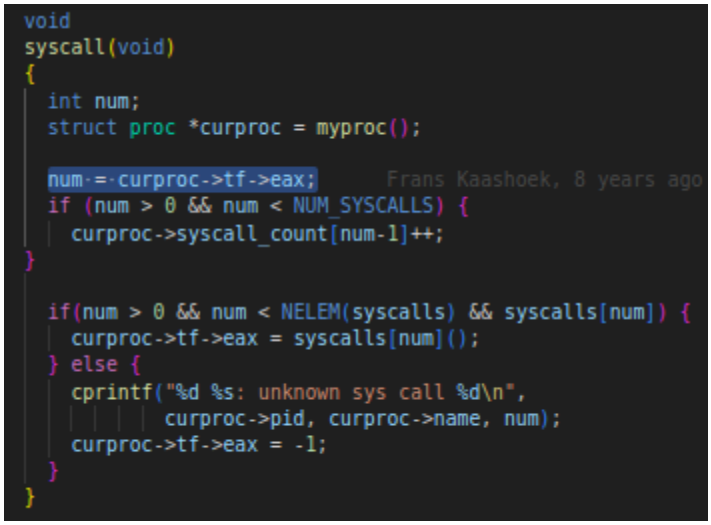
a. مراحل a ,b ,c ,d ,e را همانند move_file انجام میدهیم.

b. برای اینکه بتوانیم تعداد فراخوانی های سیستم کال ها را برای یک پردازه بدانیم لازم است یک آرایه داشته باشیم که ایندکس آن شماره سیستم کال و مقدار آن تعداد استفاده از آن پردازه است.این آرایه را به استراکچر پردازه در فایل proc.h اضافه میکینم.

```
proc.h > ...
28  struct context {
33    uint eip;
34  };
35
36  enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
37
38  // Per-process state
    You, 2 hours ago | 6 authors (kolya and others)
39  struct proc {
40    uint sz;                     // Size of process memory (bytes)
41    pde_t* pgdir;                // Page table
42    char *kstack;                // Bottom of kernel stack for this process
43    enum procstate state;        // Process state
44    int pid;                     // Process ID
45    struct proc *parent;         // Parent process
46    struct trapframe *tf;        // Trap frame for current syscall
47    struct context *context;     // swtch() here to run process
48    void *chan;                  // If non-zero, sleeping on chan
49    int killed;                  // If non-zero, have been killed
50    struct file *ofile[NOFILE];  // Open files
51    struct inode *cwd;           // Current directory
52    char name[16];               // Process name (debugging)
53    int syscall_count[NUM_SYSCALLS];
54  };
55  int sort_syscalls(int pid);
56
57  // Process memory is laid out contiguously, low addresses first:
58  //   text
59  //   original data and bss
60  //   fixed-size stack
61  //   expandable heap
62
```

c. همچنین هر زمان که پردازه ای بوجود آمد باید این آرایه مقدار دهی اولیه شود این و هر بار که سیستم کالی صدا زده شد مقدار مربوط به خانه آن یکی اضافه شود. مقدار دهی اولیه در فایل

proc.c و آپدیت آن در فایل syscall.c انجام شده است.

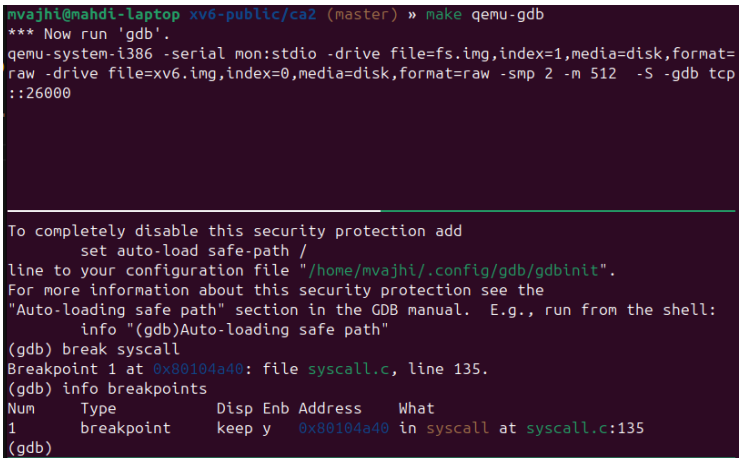


```
void
syscall(void)
{
  int num;
  struct proc *curproc = myproc();

  num = curproc->tf->eax;      Frans Kaashoek, 8 years ago
  if (num > 0 && num < NUM_SYSCALLS) {
    curproc->syscall_count[num-1]++;
}

  if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
    curproc->tf->eax = syscalls[num]();
  } else {
    cprintf("%d %s: unknown sys call %d\n",
            curproc->pid, curproc->name, num);
    curproc->tf->eax = -1;
  }
}
```

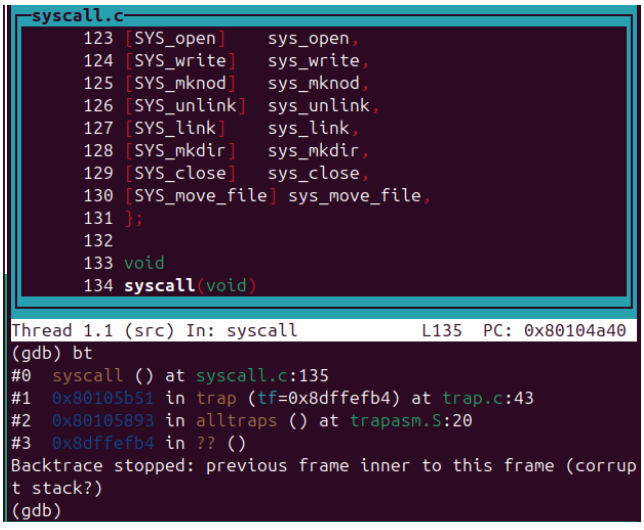
d. خود کد سیستم کال نیز به این صورت عمل میکند که صرفا این آرایه را پرینت میکند.

بررسی فراخوانی سیستمی با gdb

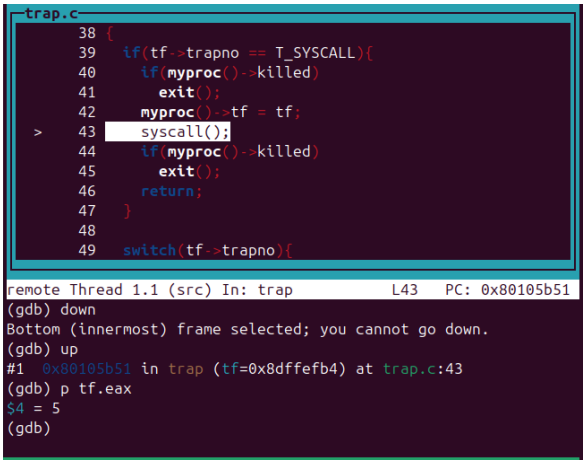
```c
//get.c
#include "types.h"
#include "user.h"
int main(int argc, char* argv[]) {
int pid = getpid();
printf(1, "Process ID: %d\n", pid);
exit();
}
```



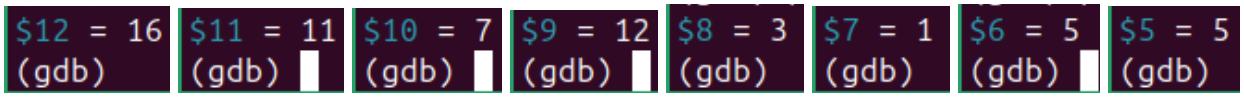
همانطور که مشاهده می کنید با دستور مربوط به دیباگ برنامه را اجرا کردیم سپس در جی دی بی به آن وصل شدیم برنامه را ادامه دادیم تا تنظیمات اولیه انجام شود و سپس برنامه را متوقف کردیم و یک نقطه توقف در syscall تنظیم کردیم. سپس برنامه ی get را اجرا کردیم و برنامه در syscall متوقف شد.

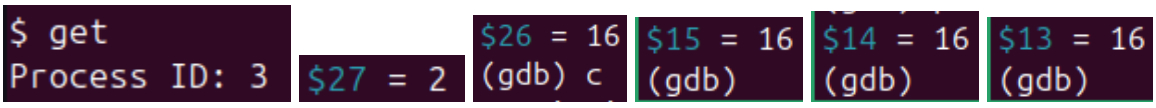


همانطور که مشاهده می کنید پشته صدا شدن تابع را نشان می دهد که تابع فعلی در بالای استک و برنامه سطح کاربر get در آخر پشته قرار دارد. همچنین می بینیم که آدرس دستوری که از آن پرش شده و تابع جدیدی صدا شده نمایش داده می شود همچنین می دانیم که در این استک متغیر های محلی، آرگومان ها و موارد لازم برای اجرا تابع در زمان برگشت ذخیره می شود. همانطور هم که در توضیحات فایل پروژه داشتم ابتدا trapasm اجرا شده و بعد به وسیله آن ابتدا trap frame ساخته می شود و سپس آدرس برگشت روی استک قرار می گیرد و تابع trap صدا می شود. همچنین در این عملیات سطح دسترسی هم به حالت هسته می رود. سپس در این تابع فریم trap بررسی می شود و سپس syscall فراخوانی می شود. سپس در این تابع بردار فراخوانی های سیستم موجود است و با پیدا کردن فراخوانی مربوطه آن را صدا می کند.



طبق خواسته سوال دستور down را اجرا می کنیم و همانطور که مشاهده می کنید اجرا نمی شود. این دستور درواقع برای پیمایش استک تابع هست و مثلا اگر در حال مشاهده تابع خانه ۱ استک باشیم با اجرای این دستور به تابع خانه ۰ استک می رویم همچنین دستور دیگری هم هست به نام up که دقیقا برعکس آن کار می کند و در اینجا باید از آن استفاده کنیم و وقتی به تابع trap رفتیم با استفاده از دستور چاپ مقدار tf.eax را نمایش می دهیم که برابر با ۵ است.






```
// System call numbers (syscall.h)
#define SYS_fork  1
#define SYS_exit  2
#define SYS_wait  3
#define SYS_read  5
#define SYS_exec  7
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_write  16
```

شماره فراخوانی و مقدار آن در هر تصویر مشخص هست و مقادیر تعریف شده هم قرار داده شده. دلیل این موضوع که در ابتدا ۱۱ نیست واضح است بعد از دریافت فرمان ما سیستم باید ابتدا ورودی را بخواند و سپس برنامه را اجرا کند و منتظر خروجی آن بماند و سپس برنامه حافظه مورد نیاز رو میگیرد و فایل آن اجرا می شود. در اینجاست که برنامه id خود را می گیرد و در قسمت های بعد هم خروجی را چاپ می کند و در آخر خارج می شود.