**بسمه تعالی**

گزارش تمرین عملی دوم – سوال اول – درس سیستم‌های عامل دکتر اسدی – نیمسال اول 405-404

نویسندگان:

1. سیداحمد موسوی‌اول – 402106648
2. عرفان تیموری – 402105813

---

گزارشکار.

آ) هدف از این سوال بررسی وضعیت هسته (Kernel) و پروسه‌ی کاربر در لحظه‌ی ورود به یک فراخوانی سیستمی (syscall) با استفاده از GDB است.

ابتدا هسته xv6 را تحت qemu-gdb اجرا می‌کنیم. یک نقطه‌ی توقف (Breakpoint) در تابع syscall واقع در (kernel/syscall.c) تنظیم می‌کنیم و پس از اجرای برنامه (با دستور c)، اجرای برنامه را در این نقطه متوقف می‌کنیم.

```
kernel/proc.c
426 {
427   struct proc *p;
428   struct cpu *c = mycpu();
429
430   c->proc = 0;
431   for(;;){
432     // The most recent process to run may have had interrupts
433     // turned off; enable them to avoid a deadlock if all
434     // processes are waiting. Then turn them back off
435     // to avoid a possible race between an interrupt
436     // and wfi.
>   437     intr_on();
438     intr_off();
439
440     int found = 0;
441     for(p = proc; p < &proc[NPROC]; p++) {
442       acquire(&p->lock);
443       if(p->state == RUNNABLE) {
444         // Switch to chosen process.  It is the process's job
445         // to release its lock and then reacquire it
446         // before jumping back to us.
447         p->state = RUNNING;
448         c->proc = p;
449         swtch(&c->context, &p->context);
450
remote Thread 1.3 (src) In: scheduler                                  L437  PC: 0x80001dce
(gdb)
```

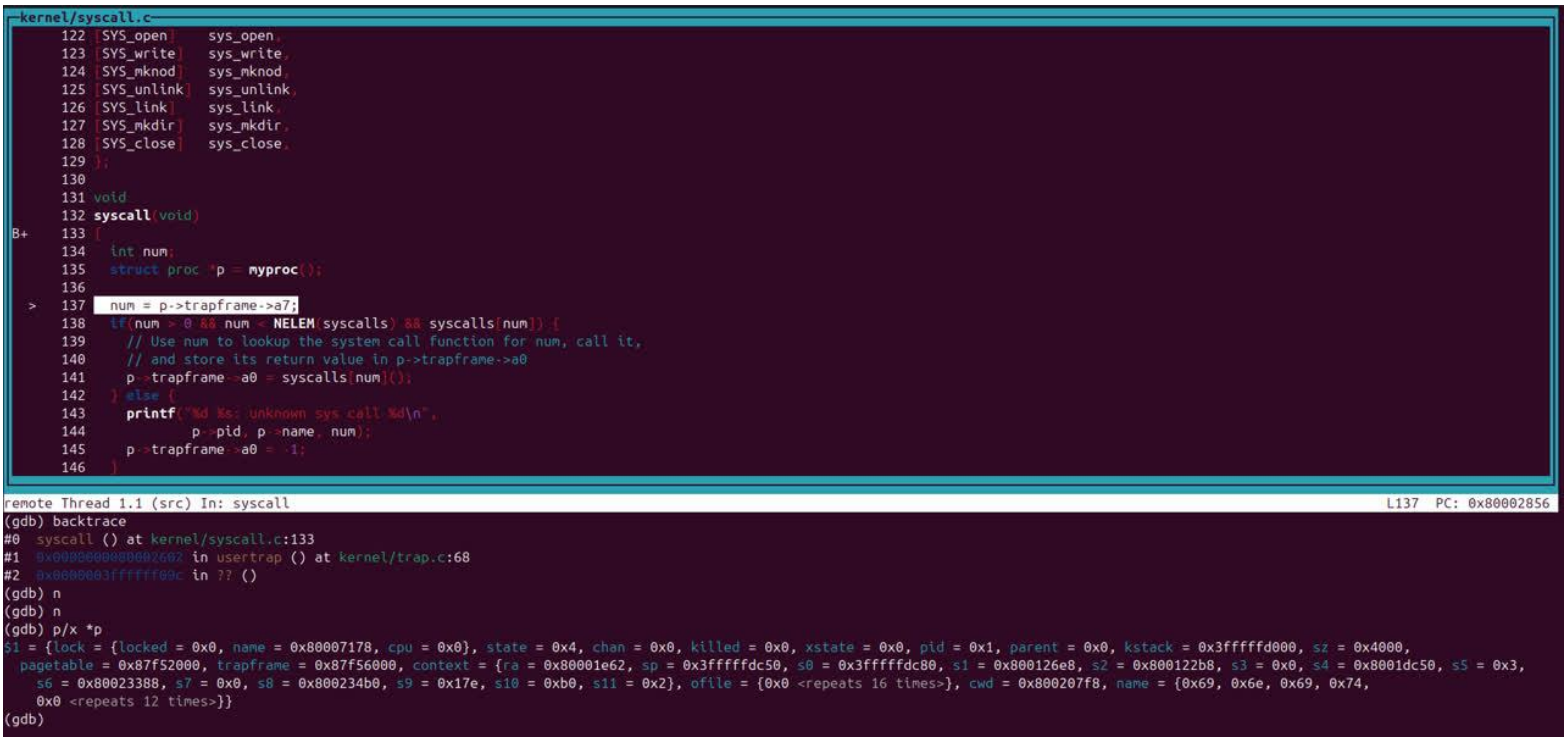
برای پاسخ به سوال اول، گام‌های زیر برداشته می‌شود:

1. بررسی استک: ابتدا از دستور backtrace استفاده می‌شود. این دستور پشته‌ی فراخوانی را نشان می‌دهد و تایید می‌کند که ما از یک تله (trap) در مد کاربر به تابع usertrap و سپس به syscall رسیده‌ایم.

2. شناسایی پروسه جاری: در xv6، تابع ()myproc پوینتری به ساختار struct proc پروسه‌ای که در حال حاضر روی CPU اجرا می‌شود (حتی اگر در مد هسته باشد) برمی‌گرداند.

پس به کمک دستور n (next) آنقدری جلو می‌رویم که به struct p دسترسی داشته باشیم.

3. چاپ ساختار پروسه: با استفاده از دستور p/x *myproc() (gdb)، تمام محتویات struct proc چاپ می‌شود.




نتیجه: این خروجی شامل اطلاعات حیاتی پروسه مانند pid (شناسه پروسه)، state، sz (اندازه حافظه) و مهم‌تر از همه، پوینتر trapframe است.

برای پاسخ به سوال دوم، وضعیت ذخیره‌شده‌ی پروسه در trapframe بررسی می‌شود:

1. دسترسی به رجیستر a7: طبق قرارداد فراخوانی در RISC-V، پروسه‌ی کاربر قبل از اجرای دستور ecall (که منجر به تله و ورود به هسته می‌شود)، شماره‌ی فراخوانی سیستمی مورد نظر خود را در رجیستر a7 قرار می‌دهد، هسته این رجیسترها را در trapframe پروسه ذخیره می‌کند. همچنین دستور p/x myproc()->trapframe->a7 (gdb) مستقیما این مقدار را از trapframe می‌خواند و نمایش می‌دهد. اگر این دستور را اجرا کنیم خروجی 0xf حاصل می‌شود که یعنی اولین فراخوانی سیستمی xv6 دارای آیدی 15 است. حال اگر دستور

```
1. P syscall[p → trapframe → a7]
```

را اجرا کنیم syscall موردنظر به ما خروجی داده می‌شود که همان sys-open است؛ بنابراین اولین syscall اجرا شده در xv6 دستور sys-open است.

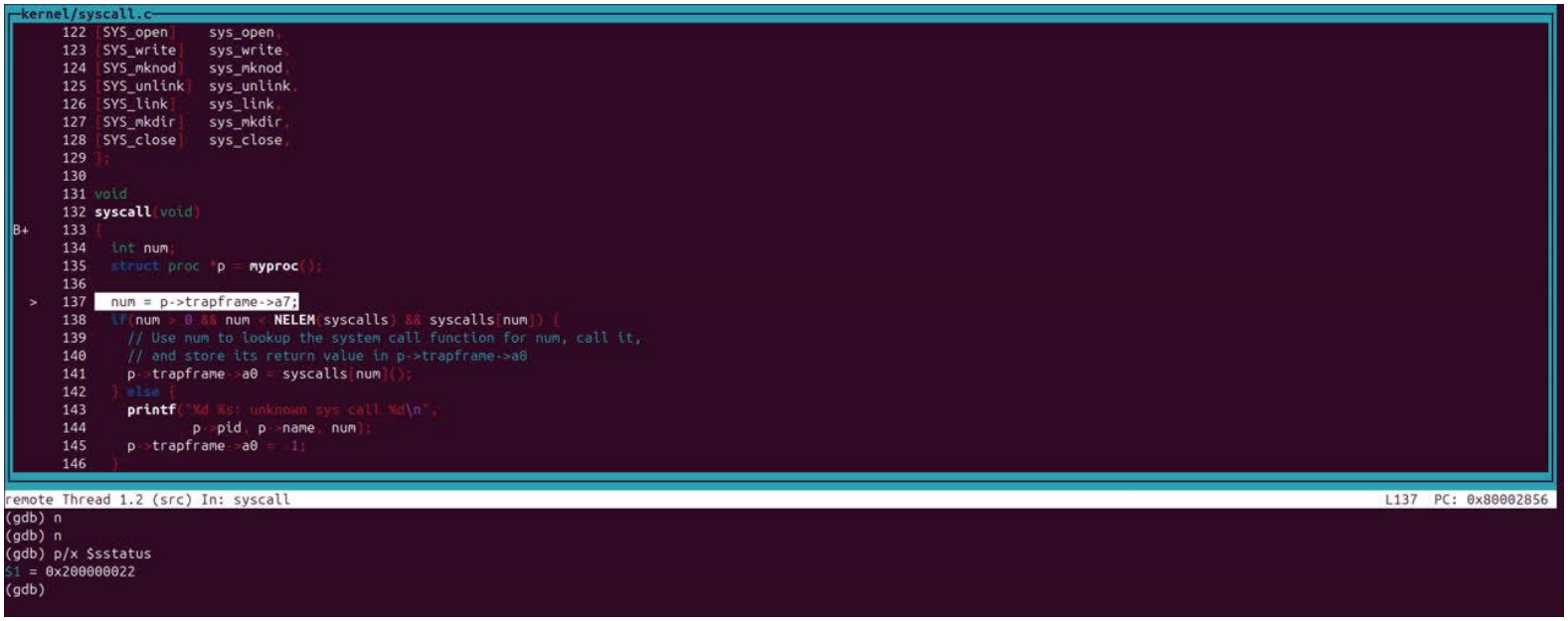
2. تفسیر مقدار a7: مقدار نمایش داده شده (مثلا 0x7 برای exec یا 0x1 برای fork) مشخص می‌کند که پروسه‌ی کاربر کدام سرویس را از هسته درخواست کرده است (این همان مقداری است که در کدهایی مانند user/initcode.S قبل از ecall در a7 بارگذاری می‌شود).

3. بررسی sstatus: رجیستر sstatus (Status Register) وضعیت پردازنده مانند فعال بودن وقفه‌ها یا مد قبلی (User/Supervisor) را قبل از وقوع تله نشان می‌دهد. این مقدار نیز در trapframe ذخیره می‌شود و با دستور

```
1.  (gdb) p/x myproc()->trapframe->sstatus
```

قابل مشاهده است (همچنین می‌توان مستقیما با (gdb) p/x $sstatus به رجیستر CSR دسترسی پیدا کرد).

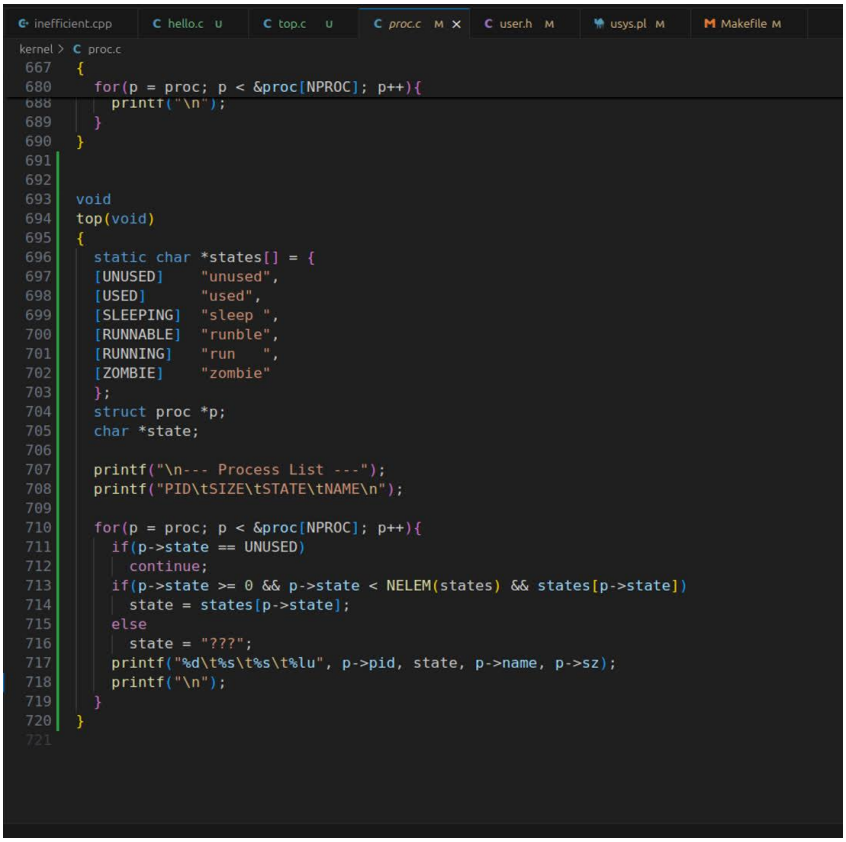
برای پاسخ به سوال سوم نیز کافیست دستور (gdb) p/x $sstatus را اجرا کنیم. خروجی این دستور یک عدد 32 بیتی است که بیت 8 ام آن نشان‌دهنده مد قبلی سیستم قبل از وقوع تله است.




اگر این بیت 0 باشد یعنی در حالت user بوده‌ایم و اگر 1 باشد یعنی در حالت kernel بوده‌ایم. حال با توجه به تصویر بالا این بیت 0 است؛ بنابراین قبل از وقوع وقفه در حالت user بوده‌ایم.

---

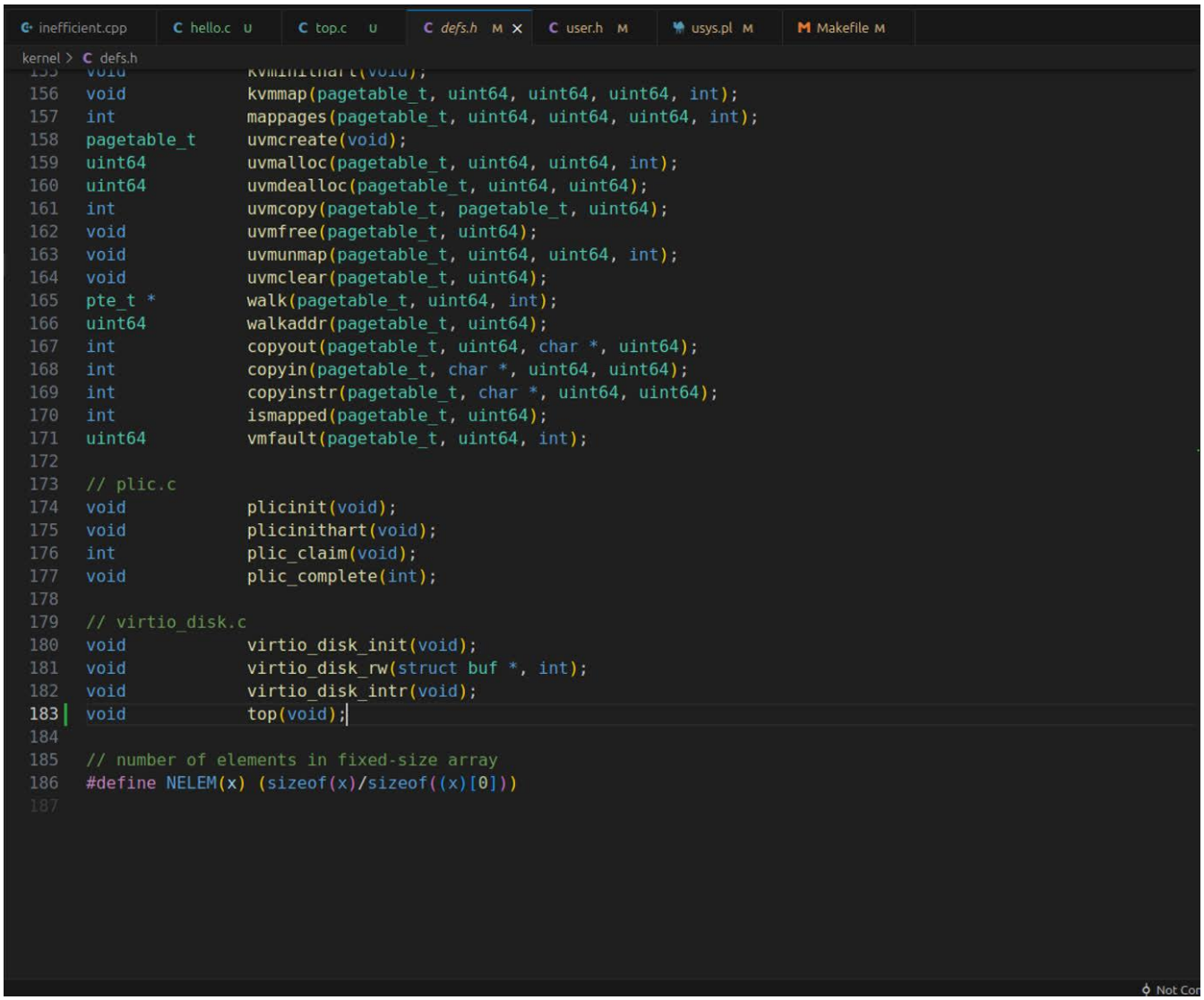
ب) در این قسمت از ما خواسته شده است دستور سیستمی top را به سیستم‌عامل اضافه کنیم. برای اینکار ابتدا قطعه کد زیر را به کد proc.c اضافه می‌کنیم:




این کد به ترتیب PID، وضعیت process، نام آن و سایز آن را نشان می‌دهد.

در ادامه به فایل sysfile.c فراخوانی سیستمی که ساختیم را اضافه می‌کنیم:


مشابه تمرین 1 به بعضی از فایل‌ها باید این دستور جدید را اضافه کنیم؛ برای مثال به فایل‌های syscall.h دستور top را به همراه id آن اضافه می‌کنیم، یا در syscall.c فراخوانی سیستمی موردنظر را اضافه می‌کنیم و ...



در نهایت با اضافه کردن دستور top به تمام بخش‌هایی که لازم است خروجی دستور به شکل زیر می‌شود:


مشاهده می‌کنیم که PID پردازه‌ها به همراه STATE، SIZE و دستوری که باعث اجرای آن شده است بعد از زدن دستور top که خودمان آن را تعریف کردیم به نمایش در می‌آید.