(یا مرتجی)

گزارش پروژه ی میانترم سیستم عامل

پیاده سازی یک system call در XV6

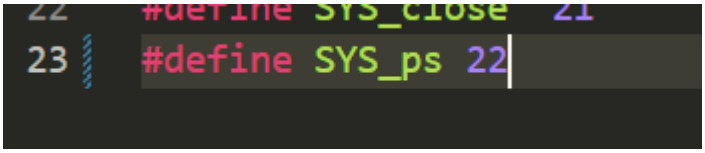
زهرا عباسقلی ۴۰۰۵۲۲۰۲۲

معصومه غفاری ۴۰۰۵۲۲۰۸۵

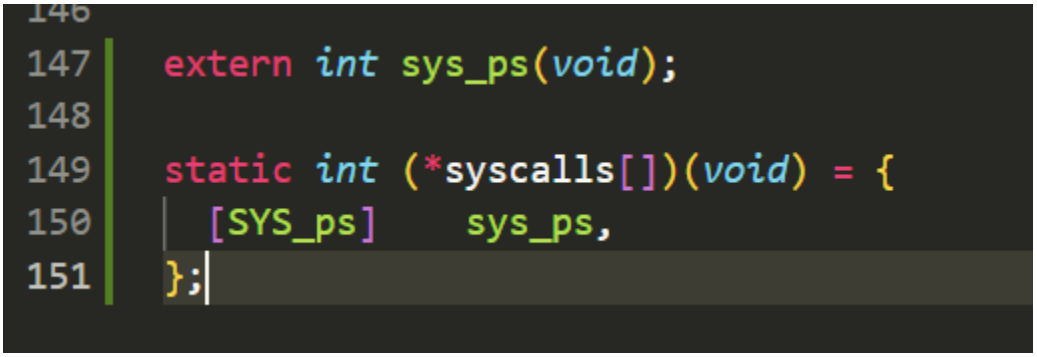
۱. ایجاد system call ساده:

برای پیاده‌سازی یک سیستم کال ساده در xv6 که روی جدول فرآیندها حرکت کند و حالت فعلی فرآیندهای موجود را روی کنسول چاپ کند، باید چندین مرحله را انجام دهیم. این مراحل شامل اضافه کردن سیستم کال به هسته xv6 ، تعریف آن، و نوشتن یک برنامه کاربر برای تست آن است.

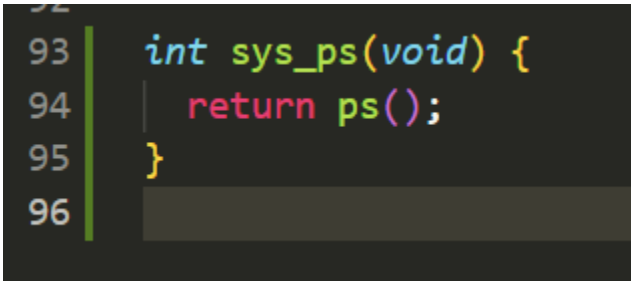
در فایل های syscall.h یک شماره برای سیستم کال جدید تعریف میکنیم.



در فایل syscall.c سیستم کال جدید را به آرایه syscalls اضافه میکنیم.



در فایل sysproc.c فانکشن sys_ps را اضافه میکنیم.



فانکشن ps را در proc.c تعریف میکنیم.

```
535
536 int ps(void) {
537   static char *states[] = {
538   [UNUSED]    "unused",
539   [EMBRYO]    "embryo",
540   [SLEEPING]  "sleep ",
541   [RUNNABLE]  "runnable",
542   [RUNNING]   "running",
543   [ZOMBIE]    "zombie"
544   };
545   struct proc *p;
546   sti();  // Enable interrupts
547   acquire(&ptable.lock);
548   cprintf("PID\tState\tName\n");
549   for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
550     if(p->state == UNUSED)
551       continue;
552     cprintf("%d\t%s\t%s\n", p->pid, states[p->state], p->name);
553   }
554   release(&ptable.lock);
555   return 0;
556 }
```

در فایل proc.h پروتوتایپ فانکشن ps را اضافه میکنیم.

```
59
60  int ps(void);
```

حال برای **اضافه کردن سیستم کال جدید به جدول سیستم کال‌ها**، فایل usys.S را باز کرده و سیستم کال جدید را به آن اضافه کرده.

```
31  SYSCALL(uptime)
32  SYSCALL(ps)
33
```

و سپس فایل syscall.h را باز کرده و پروتوتایپ سیستم کال جدید را به آن اضافه میکنیم.

```
#define SYS_close 21
#define SYS_ps 22
int ps(void);
```

برای **نوشتن برنامه کاربر برای تست سیستم کال**، یک فایل جدید به نام pstest.c ایجاد میکنیم و در آن برنامه تست سیستم کال را مینویسیم.

pstest.c > ...

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

int main(void) {
  ps();
  exit();
}

```

۲. آرگومان ها و تغییر منطق system call:

در فایل های syscall.h یک شماره برای سیستم کال جدید تعریف میکنیم.

```
#define SYS_close 21
#define SYS_ps 22
```

در فایل sysproc.c فانکشن sys_ps را اضافه میکنیم.

```
93  int sys_ps(void) {
94    return ps();
95  }
96
```

فانکشن ps را در proc.c تعریف میکنیم.

```
535
536 int ps(void) {
537   static char *states[] = {
538   [UNUSED]    "unused",
539   [EMBRYO]    "embryo",
540   [SLEEPING]  "sleep ",
541   [RUNNABLE]  "runnable",
542   [RUNNING]   "running",
543   [ZOMBIE]    "zombie"
544   };
545   struct proc *p;
546   sti();  // Enable interrupts
547   acquire(&ptable.lock);
548   cprintf("PID\tState\tName\n");
549   for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
550     if(p->state == UNUSED)
551       continue;
552     cprintf("%d\t%s\t%s\n", p->pid, states[p->state], p->name);
553   }
554   release(&ptable.lock);
555   return 0;
556 }
```

در فایل proc.h پروتوتایپ فانکشن ps را اضافه میکنیم.

```
59
60  int ps(void);
```

حال برای **اضافه کردن سیستم کال جدید به جدول سیستم کال‌ها**، فایل usys.S را باز کرده و سیستم کال جدید را به آن اضافه کرده و سپس فایل syscall.h را باز کرده و پروتوتایپ سیستم کال جدید را به آن اضافه میکنیم.

برای **نوشتن برنامه کاربر برای تست سیستم کال**، یک فایل جدید به نام pstestArg.c ایجاد میکنیم و در آن برنامه تست سیستم کال را مینویسیم.

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

int main(int argc, char *argv[])
{
    int state = -1; // Default: show all states
    int pid = -1;   // Default: show all PIDs
    if (argc > 1)
        state = atoi(argv[1]);
    if (argc > 2)
        pid = atoi(argv[2]);
    ps(state, pid);
    exit();
}
```

با اضافه کردن آرگومان ها میتوان فیلتر های بیشتری اعمال کرد تا اطلاعات نمایش داده شده، دقیقتر باشند.

نحوه اجرا کردن آرگومان ها:

Pstest 2 3