پروژه اول

افزودن سیستم کال به xv6

درس سیستم های عامل

نگین حقیقی – آنیتا تلخابی

99522284-99521226

استاد درس: دکتر رضا انتظاری ملکی

نیم سال اول 1402-1401

مقدمه:

در این پروژه قصد داریم یک سیستم کال به سیستم عامل xv6 اضافه کنیم، و دید عمیق تری نسبت به xv6 و سیستم کال ها بدست بیاوریم.

حال این سیستم کالی که قرار است طراحی کنیم و به xv6 اضافه کنیم، باید پراسس هایی که در RUNNING و RUNNABLE هستند رو در یک آرایه برگرداند و لازم به ذکر است که این آرایه بر حسب سایر پراسزها سورت شده و از کوچک به بزرگ مرتب میشود. اگر دو پراسس سایز memsize یکسانی داشتند، آنگاه بر اساس آیدی آنها (یعنی pid) قرار میگیرند.

روش انجام:

ابتدا دستور زیر را در ترمینال کپی کرده و سپس با دستور بعدی، فایل کدهای مربوط به xv6 را در سیستم خود کلون میکنیم.

```
sudo apt-get update && sudo apt-get install --yes build-essentials git qemu-system-x86

git clone https://github.com/mit-pdos/xv6-public
```

حال پوشه ایجاد شده را با ادیتور خود (مثلvscode) باز میکنیم تا یکسری تغییراتی ایجاد کرده و system call خود را ایجاد کنیم.

ابتدا در فایل syscall.h اسم سیستم کال را اضافه کرده و به این صورت به ته فایل، این خط را اضافه میکنیم:

```
19  #define SYS_unlink 18
20  #define SYS_link   19
21  #define SYS_mkdir  20
22  #define SYS_close  21
23  #define SYS_proc_dump  22
24
```

همانطور که میبینید، شماره سیستم کال ها تا 21 بود پس ما باید شماره 22 را به سیستم کال خود اختصاص دهیم.

سپس به defs.h رفته و header تابع خود را در بخش proc.c مطابق شکل زیر اضافه میکنیم.

```
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  int             proc_dump(void);
124
125  // swtch.S
```

در فایل user.h نیز مجدد هِدر تابعمان را به بخش system calls می افزاییم.

```
92
93  int
94  sys_proc_dump(void)
95  {
96    return proc_dump();
97  }
```

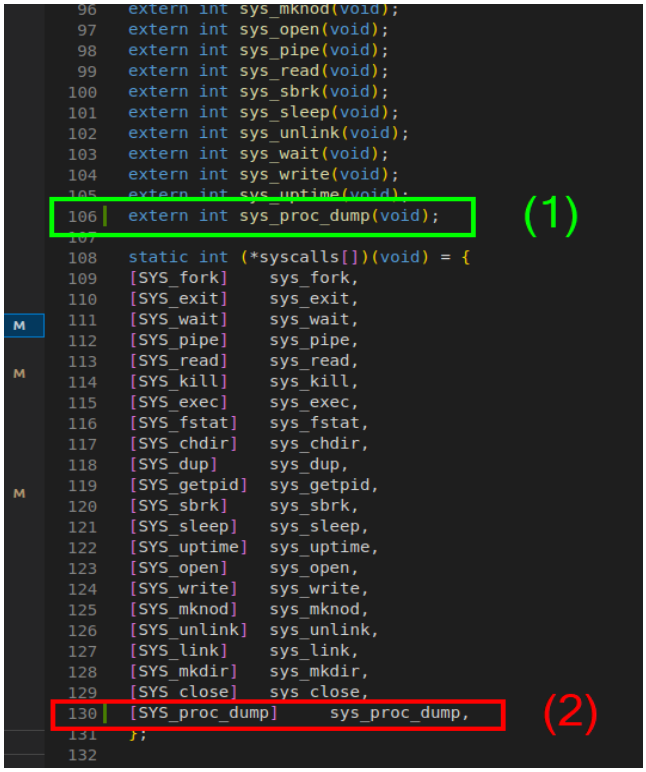
حالا به sysproc.c رفته و تابع sys_proc_dump را مطابق عکس، در سطح یوزر تعریف میکنیم.

(تابع proc_dump را در آینده ای نه چندان دور، در فایل proc.c تعریف خواهیم کرد.)

در مرحله بعد، به usys.S میرویم و این خط را به انتهای فایل اضافه میکنیم.

```
29  SYSCALL(sbrk)
30  SYSCALL(sleep)
31  SYSCALL(uptime)
32  SYSCALL(proc_dump)
33
34
```

حال به فایل syscall.c میرویم و در این فایل باید تو قسمت را به کد اضافه کنیم. که هر دو بخش در اسکرین شات زیر، با رنگ های سبز و قرمز مشخص شده اند.




```
extern int sys_mknod(void);
extern int sys_open(void);
extern int sys_pipe(void);
extern int sys_read(void);
extern int sys_sbrk(void);
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_proc_dump(void);

static int (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_kill]    sys_kill,
[SYS_exec]    sys_exec,
[SYS_fstat]   sys_fstat,
[SYS_chdir]   sys_chdir,
[SYS_dup]     sys_dup,
[SYS_getpid]  sys_getpid,
[SYS_sbrk]    sys_sbrk,
[SYS_sleep]   sys_sleep,
[SYS_uptime]  sys_uptime,
[SYS_open]    sys_open,
[SYS_write]   sys_write,
[SYS_mknod]   sys_mknod,
[SYS_unlink]  sys_unlink,
[SYS_link]    sys_link,
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
[SYS_proc_dump]     sys_proc_dump,
};
```

حال باید به فایل proc.c رفته و تابع proc_dump را همانطور که قبلا اشاره کردیم، آنجا تعریف کنیم. این تابع بخش نسبتا اساسی تر تغییرات هست.

در این تابع، دو کار مهم انجام میشود، یکی اینکه تمام پراسس ها بررسی شده و آنهایی که شرط تعیین شده توسط ما را دارند، یعنی stateشان RUNNING یا RUNNABLE است، چاپ میشوند، و کار دوم این است که این پراسس ها را برحسب سایزشان مرتب و سورت میکنیم.

```
int
proc_dump()
{
  struct proc* AllProcess[NPROC];
  struct proc *pro;
  sti();
  acquire(&ptable.lock);
  cprintf("_____\n");
  int pcount = 0;
  cprintf("name \t pid \t state \t size \n");
  for (pro = ptable.proc; pro < &ptable.proc[NPROC]; pro++)
  {
    if (pro->state ==   RUNNING)
    {
      AllProcess[pcount++] = pro;
      cprintf("%s \t %d \t RUNNING \t %d \t\n ", pro->name, pro->pid, pro->sz);
    }
    else if (pro->state ==   RUNNABLE)
    {
      AllProcess[pcount++] = pro;
      cprintf("%s \t %d \t RUNNABLE \t %d \t\n ", pro->name, pro->pid, pro->sz);
    }

  }
  release(&ptable.lock);
```

بخش اول، در خط 536 تا 560 انجام شده و همانطور که میبینید، در خط 539 یک لیست از پراسس ها با طول اولیه NPROC ساخته شده و قرار است تمام پراسسهایی که شرط تعیین شده را دارند، در آن بریزیم. پس یک for میزنیم و هردفعه stateِ آن پراسس(pro) را چک میکنیم و اگر RUNNINGو RUNNABLE بود، به لیست اد کرده و تعداد processها را که با pcount نمایش میدهیم را یکی زیاد میکنیم. (خط550) و در نهایت مشخصات آن پراسس را مانند خط 551، چاپ میکنیم.

```
561
562     int i = 0;
563     while (i < pcount-1)
564     {
565       for (int j = 0; j<pcount-1; j++)
566       {
567         if (AllProcess[j]->sz > AllProcess[j+1]->sz)
568         {
569           int pid = AllProcess[j]->pid;
570           int state = AllProcess[j]->state;
571           int size = AllProcess[j]->sz;
572           char name[16];
573           int k=0;
574           while (k < 16)
575           {
576             name[k] = ptable.proc[j].name[k];
577           }
578           AllProcess[j]->pid = AllProcess[j+1]->pid;
579           AllProcess[j]->state = AllProcess[j+1]->state;
580           AllProcess[j]->sz = AllProcess[j+1]->sz;
581           for(k=0; k<16; k++)
582           {
583             AllProcess[j]->name[k] = AllProcess[j+1]->name[k];
584           }
585           AllProcess[j+1]->pid = pid;
586           AllProcess[j+1]->state = state;
587           AllProcess[j+1]->sz = size;
588           k=0;
589           while (k < 16)
590           {
591             AllProcess[j+1]->name[k] = name[k];
592           }
593         }
594       }
595       i++;
596     }
597     cprintf("_____(( this is sorted array of process ))_____\n");
598     cprintf("name \t pid  \t size \t\n");
599     for (int j=0; j<pcount; j++)
600     {
601       cprintf("%s \t %d \t %d \t \n ", AllProcess[j]->name, AllProcess[j]->pid, AllProcess[j]->sz);
602     }
603     cprintf("_____\n");
604     return 22;
605   }
```

در بخش دوم تابع، قصد داریم آرایه مان را سورت کنیم تا processها به ترتیب memsizeشان قرار گیرند. برای اینکار ما از bubble sort استفاده کردیم و همانطور که در تصویر میبینید، هردفعه خانه j را با j+1 مقایسه کرده، و در صورت بزرگتر بودن، آنها را جابه جا میکنیم. و درنهایت مجدد پراسس های مرتب شده را پرینت میکنیم.

در گام بعدی، یک فایل به نام ps.c ساخته و در تابع main آن، قصد داریم سیستم کال خود را تست کنیم. از fork وmalloc مطابق کد زیر استفاده کرده ایم. و سپس تابع proc_dump را صدا میزنیم.

```c

int main(int argc, char* argv[])
{
    int* arr = 0;

    int size = 1;
    int count = 1;
    for (int j = 0; j<count; j++)
    {
        int id = fork();
        if(id == 0)
        {
            arr = malloc(size*1000 * sizeof(int));
            while(true)
            {
                continue;
            }
            *arr = 2;
        }
    }
    proc_dump();
    exit();
}
```

در مرحله آخر، در Makefile باید ps.c را به بخش UPROGS و EXTRA اضافه کنیم.

حال با اجرای دستورات make clean و سپس make qemu و وارد کردن دستور سیستم کال ps ، میتوان خروجی را مشاهده کرد.

```
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ ps
_____________________________________________
name     pid     state   size
ps       3       RUNNING         12288
 ps      4       RUNNING         45056
 ______(( this is sorted array of process ))______
name     pid     size
ps       3       12288
 ps      4       45056
 _____________________________________________
$
```

مشکلاتی که در حین انجام پروژه با آن برخوردیم:

1) استفاده از دستور make qemu-nox به جای make qemu برای برطرف کردن ارور مربوطه در wsl
2) ایجاد پراسس زامبی پس از اجرای دستور ps -> با افزودن یک وایل بینهایت در پراسس child در تابعmain فایل ps.c (مطابق خط20)

```
While(true)
{
  Continue;
}
```

3) استفاده از دستور exit در انتهای تابع main : اگر این کار را نکنید، پس از اجرای ps ارور مربوطه را در کنسول مشاهده میکنید.