به نام خدا

پروژه XV6

فاز یک

محمد جواد زندیه

شماره دانشجویی : 9831032

دانشکده مهندسی کامپیوتر دانشگاه صنعتی امیرکبیر

پروژه درس سیستم عامل 1400/7/28

getProcCount System Call

هدف : بدست آوردن تعداد process هایی را که در لحظه فراخوانی این فراخوانی های سیستمی وجود دارند (یعنی تمامی process هایی که در Unused state نباشند)

فایل هایی از کد سیستم عامل xv6 تغییر کرده اند تا این system call را به سیستم عامل اضافه کنیم :

1. syscall.h

```
24  #define SYS_getProcCount 23
```

در این فایل شماره متناظر با system call ای را که میخواهیم تعریف کنیم set می کنیم.

2. sysproc.c

```
99  int
100 sys_getProcCount(void)
101 {
102   return getProcCount();
103 }
```

در این فایل system call خود را تعریف میکنیم و پیاده سازی را به تابع getProcCount می سپاریم.

3. proc.c

```
543 int
544 getProcCount(void)
545 {
546   static char *states[] = {
547   [UNUSED]    "unused",
548   [EMBRYO]    "embryo",
549   [SLEEPING]  "sleep ",
550   [RUNNABLE]  "runble",
551   [RUNNING]   "run   ",
552   [ZOMBIE]    "zombie"
553   };
554   int procCount = 0;
555   struct proc *p;
556
557   cprintf("alive processes states:\n");
558   for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
559     if(p->state != UNUSED){
560       procCount ++;
561       cprintf("process %d : %s\n", procCount, states[p->state]);
562     }
563   }
564   cprintf("so number of alive processes is: %d\n", procCount);
565   return 0;
566 }
```

در این فایل همانطور که در مرحله قبل گفته شد به پیاده سازی انتظاری که از system call داریم می پردازیم. یک تعدادی اطلاعات اضافه هم به منظور تفهیم بهتر قرار داده شده است، از جمله اینکه هر یک از process ها در چه state ای قرار دارند. روش پیاده سازی از این قرار است که با پیمایش روی تمام process هایی که تاکنون در سیستم قرار گرفته اند آن process هایی را که در unused state هستند را لیست کرده و در نهایت تعداد آنها را بیان می کنیم.

4. syscall.c

```
107 extern int sys_getProcCount(void);
```

```
132 [SYS_getProcCount] sys_getProcCount,
```

با extern کردن تابع sys_getProcCount در این فایل، این system call را به function pointer array سیستم اضافه می کنیم.

5. defs.h

```
124 int getProcCount(void);
```

دسترسی به function های فایل proc.c از این فایل صورت می گیرد و باید تابع پیاده ساز این فراخوانی را در این فایل تعریف کنیم.

6. user.h

```
27 int getProcCount(void);
```

این فایل یک interface برای کاربر است و لیست توابع سیستمی در آن قرار می گیرد.

7. usys.S

```
33 SYSCALL(getProcCount)
```

یک فایل assembly است که می گوید عدد integer ای که برای هر system call در نظر گرفته شده است به ثبات eax منتقل شود.

8. getProcCountTest

```
1 #include "types.h"
2 #include "stat.h"
3 #include "user.h"
4
5 int main(void){
6     getProcCount();
7     printf(1, "successful to find number of processes wich are alive\n");
8     exit();
9 }
```

یک فایل تست است برای ارزیابی درستی اضافه شدن system call به این سیستم عامل

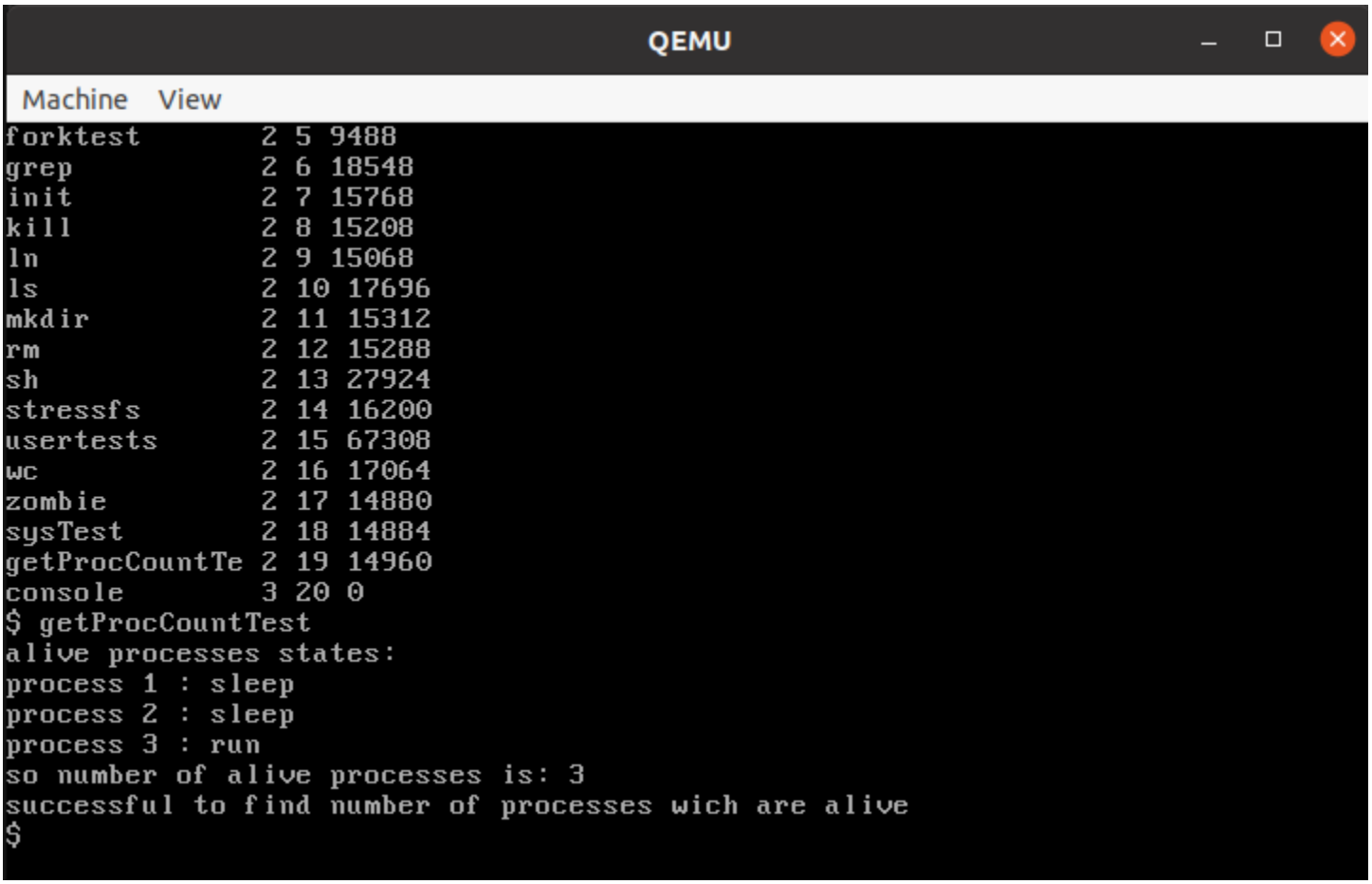
9. Makefile

```
185     _getProcCountTest\
```

```
259     getProcCountTest.c\
```

قسمت های EXTRA, PROGS از این فایل را به صورت عکس داده شده تغییر می دهیم تا در لیست ای که توسط دستور ls نشان داده می شود از system call ها، این فراخوانی هم اضافه شود.

و اما اگر بخواهیم یک نمونه از صدا کردن این فراخوانی را تست کنیم:



همانطور که مشاهده می شود getProcCount در لیست system call ها افزوده شده است و صدا کردن آن هم میتوان دید که 3 تا process در سیستم وجود داشته اند که دوتای آنها در sleep state بوده اند و یکی هم که خود این process بود در run state قرار دارد.

getReadCount System Call

هدف : بدست آوردن تعداد دفعاتی که فراخوانی سیستمی Read توسط هر پردازه دیگر کاربر فراخوانی شده است از زمانی که کرنل بوت شده است.

فایل هایی از کد سیستم عامل xv6 تغییر کرده اند تا این system call را به سیستم عامل اضافه کنیم :

توضیحات بسیاری از تغییرات همانند قسمت اول است پس فقط به توضیح قسمت هایی که متفاوت عمل شده می پردازیم.

1. proc.h

```
52    int readid;                    // keep acount for read calls
```

در این فایل یک struct به نام proc وجود دارد که مشخص می کند هر یک از process ها حاوی چه اطلاعاتی باشند. در این بخش می توان گفت که هر یک از process ها یک عدد که حاوی تعداد فراخوانی Read ای میباشد که تاکنون انجام داده را نگهداری کند.

2. proc.c

```
93    p->readid = 0;
```

در این فایل مقدار تعداد فراخوانی را برابر صفر قرار میدهیم(مقدار دهی اولیه میکنیم).

این مقدار دهی در قسمت allocproc از برنامه صورت می گیرد یعنی جایی که هر process شروع به گرفتن حافظه از ram می کند.

```
571  int
572  getReadCount(void)
573  {
574    int numread = 0;
575    struct proc *p;
576    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
577      numread += p->readid;
578    }
579    cprintf("num: %d\n", numread);
580    return 0;
581  }
```

در این فایل همچنین به پیاده سازی تابع getReadCount می پردازیم و با iterate کردن روی تمام پردازه ها می توان تعداد فراخوانی های Read ای که تاکنون انجام داده اند را بدست آورد و جمع زد.

3. sysproc.c

```
105 int
106 sys_getReadCount(void)
107 {
108   return getReadCount();
109 }
```

در این فایل system call خود را تعریف میکنیم و پیاده سازی را به تابع getReadCount می سپاریم.

4. syscall.c

```
149   if(num == SYS_read){
150     readcount++;
151   }
152   if(num == SYS_getReadCount){
153     curproc->readid = readcount;
154   }
```

در این فایل، تابع syscall هر زمانی که یک system call صورت می گیرد اجرا می شود، پس می توان هر وقت که Read فراخوانی شد، به مقدار readid از پردازه کنونی یک واحد اضافه کرد.

```
138   [SYS_getReadCount] sys_getReadCount,
```

```
111   extern int sys_getReadCount(void);
```

با extern کردن تابع sys_getReadCount در این فایل، این system call را به function pointer array سیستم اضافه می کنیم.

5. syscall.h

```
25   #define SYS_getReadCount 24
```

در این فایل شماره متناظر با system call ای را که میخواهیم تعریف کنیم set می کنیم.

6. defs.h

```
125   int           getReadCount(void);
```

دسترسی به function های فایل proc.c از این فایل صورت می گیرد و باید تابع پیاده ساز این فراخوانی را در این فایل تعریف کنیم.

7. user.h

```
28   int getReadCount(void);
```

این فایل یک interface برای کاربر است و لیست توابع سیستمی در آن قرار می گیرد.

8. usys.S

```
34  SYSCALL(getReadCount)
```

یک فایل assembly است که می گوید عدد integer ای که برای هر system call در نظر گرفته شده است به ثبات eax منتقل شود.

9. getReadCountTest

```
1  #include "types.h"
2  #include "stat.h"
3  #include "user.h"
4
5  int main(void){
6      getReadCount();
7      printf(1, "successful\n");
8      exit();
9  }
```

یک فایل تست است برای ارزیابی درستی اضافه شدن system call به این سیستم عامل

10. Make file

```
261     getReadCountTest.c\
```

```
186     _getReadCountTest\
```

قسمت های EXTRA, PROGS از این فایل را به صورت عکس داده شده تغییر می دهیم تا در لیست ای که توسط دستور ls نشان داده می شود از system call ها، این فراخوانی هم اضافه شود.

و اما اگر بخواهیم یک نمونه از صدا کردن این فراخوانی را تست کنیم:

با فراخوانی ls میتوان دید که هر دو system call مورد نظر ما قرار دارند.

با فراخوانی getReadCountTest قابل مشاهده است که نتیجه بیان می کند که 70 فراخوانی سیستمی Read توسط پردازه های ما از زمان بوت شدن انجام شده است.

```
$ ls
.              1 1 512
..             1 1 512
README         2 2 2286
cat            2 3 16356
echo           2 4 15212
forktest       2 5 9528
grep           2 6 18576
init           2 7 15796
kill           2 8 15244
ln             2 9 15096
ls             2 10 17724
mkdir          2 11 15340
rm             2 12 15316
sh             2 13 27952
stressfs       2 14 16232
usertests      2 15 67336
wc             2 16 17096
zombie         2 17 14908
sysTest        2 18 14916
getProcCountTe 2 19 14944
getReadCountTe 2 20 14944
console        3 21 0
```

```
$ getProcCountTest
alive processes states:
process 1 : sleep
process 2 : sleep
process 3 : run
so number of alive processes is: 3
successful
$ getReadCountTest
number of read calls: 70
successful
$ _
```