
# Adding a system call and test it

تهیه کنندگان: هلیا وفایی – ستاره باباجانی

استاد: دکتر انتظاری

نیم سال اول سال

هدف پروژه:

هدف ما در این پروژه اضافه کردن یک system call به xv6 است به طوری که پراسس های در حالت runnable و running هستند برگردانده شود.

توضیح مربوط به کارهای انجام شده:

1) در فایل system.h یک system call به نام SYS_proc_dump با استفاده از define تعریف کرده و مقدار اولیه آن را 22 می گذاریم.

```
#define SYS_proc_dump  22
```

2) در قسمت proc.c موجود در فایل defs.h و قسمت system call موجود در فایل user.h ، function prototype مربوط به تابع را اضافه می کنیم.

```
int             proc_dump(void);
```

```
int proc_dump(void);
```

3) در فایل sysproc.c تابع sys_proc_dump را تعریف می کنیم.

```
int
sys_proc_dump(void)
{
  return proc_dump();
}
```

4) در فایل usys.S ، syscall مربوط به تابع را اضافه می کنیم.

```
SYSCALL(proc_dump)
```

5) در فایل syscall.c یک global variable مربوط به تابع تعریف می کنیم. در مرحله ی بعد در خانه ی آخر آرایه یعنی خانه ی 22 یک پوینتر به تابع نوشته شده در فایل sysproc.c اضافه می کنیم.

```
[SYS_proc_dump]     sys_proc_dump,
```

6) در فایل proc.c تابع proc_dump را اضافه می کنیم. در تابع همه ی پراسس ها را پیمایش کرده و پراسس های RUNNABLE و RUNNING را پیدا می کنیم. پراسس های مربوطه را به محض پیدا شدن در آرایه می ریزیم و در آخر تابع bubble sort را تعریف می کنیم. هدف ما در این تابع مرتب کردن پراسس ها بر اساس size آنهاست بدین صورت که در نیاز جا به جایی size ، state ، id ، name پراسس ها جا به جا می شود.

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

```
//Bubble sort
for (int i=0; i<pcount-1; i++)
{
  for (int j = 0; j<pcount-1; j++)
  {
    if (my_listProcs[j]->sz > my_listProcs[j+1]->sz)
    {
      int size = my_listProcs[j]->sz;
      int state = my_listProcs[j]->state;
      int pid = my_listProcs[j]->pid;
      char name[16];
      int k=0;
      for(k=0; k<16; k++)
        name[k] = ptable.proc[j].name[k];
      my_listProcs[j]->sz = my_listProcs[j+1]->sz;
      my_listProcs[j]->pid = my_listProcs[j+1]->pid;
      my_listProcs[j]->state = my_listProcs[j+1]->state;
      for(k=0; k<16; k++)
        my_listProcs[j]->name[k] = my_listProcs[j+1]->name[k];
      my_listProcs[j+1]->sz = size;
      my_listProcs[j+1]->pid = pid;
      my_listProcs[j+1]->state = state;
      for(k=0; k<16; k++)
        my_listProcs[j+1]->name[k] = name[k];
    }
  }
}
```

```
cprintf("-------------------------------------------------\n");
cprintf(" Sorted array of process\n");
for (int i=0; i<pcount; i++)
{
  cprintf(" %d \t %s \t %d \t \n ", my_listProcs[i]->pid, my_listProcs[i]->name, my_listProcs[i]->sz);
}
return 22;
}
```

7) یک فایل جدید به نام ps.c می سازیم که در آن با استفاده از 2 تابع fork و malloc تست های مربوطه به اضافه کردن system call را می نویسیم.

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


int main(int argc, char* argv[])
{
    char* tmp = 0;
    int x = 1;
    int m = 1;
    for(int i=0; i<m; i++)
    {
        int id = fork();
        if(id == 0)
        {
            tmp = malloc((x*1000) * sizeof(char));
            while(1){}//for zombie mode
            *tmp = 'c';
        }
    }

    proc_dump();
    exit();
}
```

8) در قسمت UPROGS موجود در فایل Makefile دستور ps و در قسمت EXTRA نام فایل ps.c را اضافه می کنیم.

```
_ps\
```

```
ps.c\
```

## مشکلات موجود در پروژه و چگونگی رفع آن ها:

یکی از مشکلات پیش آمده هنگام اجرای کد برخوردن به حالت zombie بود که برای رفع آن در فایل ps.c یک while(1){} گذاشتیم.

```
while(1){}//for zombie mode
```

اشکال دیگری که به آن برخوردیم مشخص کردن type ، my_listProcs در فایل proc.c بود که متوجه شدیم type آن struck است که می خواستیم برای این کار struck جدیدی تعریف کنیم که در آخر متوجه شدیم نیازی نیست و از proc استفاده کردیم.

```
struct proc* my_listProcs[NPROC];
```