الناز رضایی 98411387 و حوریه سبزواری 98412004

در این پروژه باید یک system call به os سبک آموزشی xv6 اضافه میکردیم و یک برنامه تست مینوشتیم و از system call ای که نوشتیم استفاده میکردیم.

ابتدا هدر proc_info را به فایل proc.h اضافه می کنیم (چون هدر های مربوط به process در آنجا تعریف شده اند)

```
struct proc_info
{
  int pid;
  int memsize; // in bytes
};
```

در فایل syscall.h میایم system call جدید مون رو تعریف میکنیم از آنجایی که system call های از قبل تعریف شده به ترتیب شماره گذاری شده بود الگو را به هم نمیزنیم و شماره اش را 22 میذاریم

```
#define SYS_proc_dump 22
```

و در فایل user.h هم امضای تابع آن را اضافه میکنیم

```
int proc_dump(int,void*);
```

و در فایل uSys.h هم آن را اضافه میکنیم

```
SYSCALL(proc_dump)
```

در فایل syscall.c هم مثل سایر system call ها سیگنیچر تابع رو extern میکنیم

```
extern int sys_proc_dump(void);
```

و به آرایه فانکشن پوینتر های syscalls اضافش میکنیم.

```
[SYS_proc_dump] sys_proc_dump,
```

و در proc.c لاجیک اصلی systemcall رو پیاده سازی میکنیم.

در اینجا چون ptable را systemcall های دیگر هم ازش استفاده میکنند باید قبل استفاده لاک شود که مطمئن شویم تنها یک systemcall به آن دسترسی دارد سپس اطلاعات memsize و pid پروسس های RUNNING یا RUNNABLE را در آرایه proc_info میریزیم سپس با کمک bubble_sort آن را بر اساس memsize مرتب میکنیم و برش میگردونیم.

```
void bubble_sort(struct proc_info table[NPROC], int size)
{

  for (int i = 0; i < size - 1; i++)
    for (int j = 0; j < size - i - 1; j++)
      if (table[j].memsize > table[j + 1].memsize)
      {
        int temp_memsize = table[j].memsize;
        int temp_pid = table[j].pid;
        table[j].memsize = table[j + 1].memsize;
        table[j].pid = table[j + 1].pid;
        table[j] = table[j + 1];
        table[j + 1].memsize = temp_memsize;
        table[j + 1].pid = temp_pid;
      }
}
```

```
struct proc_info *
get_table_pinfo(void)
{
  struct proc *p;
  acquire(&ptable.lock);
  //initialize pinfotable
  int idx = 0;
  for (p = ptable.proc; p < &ptable.proc[NPROC]; p++)
    if (p->state == RUNNING || p->state == RUNNABLE)
    {
      pinfotable.procInfo[idx].pid = p->pid;
      pinfotable.procInfo[idx].memsize = p->sz;
      idx++;
    }
  release(&ptable.lock);
  bubble_sort(pinfotable.procInfo, idx);
  return pinfotable.procInfo;}
```

در sysproc.c هم تابع اصلی systemcall یعنی sys_proc_dump را پیاده سازی میکنیم

به این صورت که تعداد آرگومان ها و همچنین پوینتر به آرگومان را به ترتیب در size و buf میریزیم

سپس تابعی که لاجیک این system call را در آن پیاده سازی کرده بودیم را فراخوانی میکنیم

سپس پوینتر آرگومان رو به سمت اطلاعات جدیدی که از get_table_pinfo گرفتیم تغییر میدهیم.

```
extern struct proc_info * get_table_pinfo(void);
int sys_proc_dump(void)
{
  int size;
  char *buf;
  char *s;
  if (argint(0, &size) < 0 || argptr(1, &buf, size) < 0)
    return -1;

  s = buf;
  struct proc_info *p = get_table_pinfo();

  while (buf + size > s)
  {
    *(int *)s = p->pid;
    s += 4;
    *(int *)s = p->memsize;
    s += 4;
    p++;
  }
  return 0;
}
```

حال نوبت این رسیده که user program خودمون رو بنویسیم و از system call جدیدی که تعریف کردیم استفاده کنیم.

برای اینکار فایل جدید myprogram.c رو میسازیم و آن را به UPROGS و EXTRA فایل Makefile مون اضافه میکنیم.

```c
#include "types.h"
#include "stat.h"
#include "user.h"
#include "param.h"
struct proc_info
{
  int pid;
  int memsize;
};
int main(int argc, char *argv[])
{
  struct proc_info ptable[NPROC];
  struct proc_info *p;
  int p1 = fork();
  int p2 = fork();

  if (p1 > 0 && p2 > 0)
  {
    malloc(1200);
    proc_dump(NPROC * sizeof(struct proc_info), &ptable);
    int idx = 0;
    p = &ptable[idx];
    while (idx < NPROC)
    {
      if (p->memsize > 0)
      {
        wait();
        printf(1, "SIZE: %d | PID: %d\n", p->memsize, p->pid);
      }
      p = &ptable[idx++];
    }
  }
  else if (p1 == 0 && p2 > 0)
    malloc(200);
  else if (p1 > 0 && p2 == 0)
  {
    malloc(400);
  }
  else
    malloc(800);
  exit();}
```

در اینجا ما ۲ بار پروسس رو فورک کردیم که تعداد پروسس ها بیشتر شود و با توجه به pid هر کدام جدا به اندازه های مختلف فضای الکی اشغال کردیم و چون خداروشکر gc هم نداریم کسی مزاحم شون نمیشه و در پروسس parent سیستم کال proc_dump را صدا زدیم و اطلاعات آن را چاپ کردیم.