دانشکده مهندسی کامپیوتر

پروژه میانترم درس سیستم های عامل نیم‌سال دوم ۱۴۰۲-۱۴۰۳

عنوان:

پیاده‌سازی یک System Call در XV6

استاد:

دکتر رضا انتظاری ملکی

دانشجویان:

آرین حاجی‌زاده - ۹۹۴۱۱۲۸۱

سیدفراز قریشی - ۴۰۱۳۰۰۰۲۸

اردیبهشت ماه ۱۴۰۳

استفاده از ابزار QEMU

ابزار QEMU که مخفف Quick Emulator می‌باشد، یک نرم‌افزار open-source است که برای اهداف تقلیدکننده ها (Emulator) کاربرد دارد. این ابزار این اجازه را به ما می‌دهد که سیستم عامل های مختلف و پلتفرم های سخت‌افزاری متفاوت را رو یک ماشین emulate کنیم. این ابزار عملکردی مشابه دیگر نرم‌افزار های مجازی‌سازی ماشین (Machine Virtualizer) مانند VirtualBox و VMWare دارد، با این تفاوت که می‌تواند کد های مهمان را مستقیما روی پردازنده میزبان اجرا کند و با این کار به اصطلاح می‌تواند به Near-Native Performance دست بیابد.

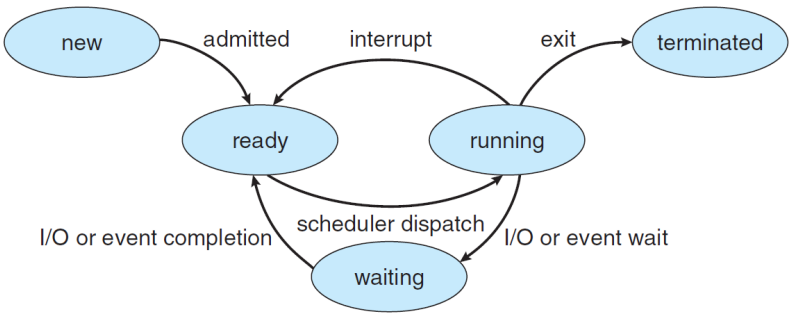
سیستم عامل XV6 یک سیستم عامل تحقیقاتی است که در فراهم کردن محیط مناسب برای اجرا و توسعه آن در این پروژه از QEMU استفاده می‌شود. این ابزار این امکان را فراهم می‌کند تا بتوان این سیستم عامل را روی یک ماشین مجازی اجرا نمود. همچنین با پشتیبانی از معماری های مختلف، با استفاده از این ابزار می‌توان XV6 را برای معماری های متفاوتی مانند x86 و RISC-V و ... توسعه داد.

پیاده سازی – بررسی ساختار فرآیند ها در سیستم عامل XV6

ساختار های struct proc و enum procstate در فایل *proc.h* به صورت زیر تعریف شده‌اند:

```
enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
```

در این ساختار تمامی حالاتی که یک فرآیند در سیستم عامل XV6 می‌تواند داشته باشد آورده شده‌است.



شکل ۱ – دیاگرام حالات یک فرآیند در یک سیستم عامل

در شکل۱ تمامی وضعیت های که یک فرآیند در یک سیستم عامل می‌تواند داشته باشد نشان داده شده‌است. می‌توان بین حالات نشان داده شده در این شکل و حالت های پیاده‌سازی شده در XV6 تطبیقی برقرار کرد. فرآیند

های Runnable در XV6 معادل فرآیند هایی هستند که در صف Ready قرار می‌گیرند و وقتی فرآیندی در حالت Running باشد و توسط درخواست های I/O یا وقفه، wait بخورد، در سیستم عامل XV6 فرآیند وارد حالت Sleeping می‌شود. در سیستم عامل XV6 اگر فرآیندی پس از اتمام اجرا به درستی خارج نشود، بطور مثال والدی یک فرآیند فرزند ایجاد کند و پس از اتمام اجرای آن با دستور wait منتظر تمام شدن آن نماند، فرآیند فرزند پس از مدتی وارد حالت Zombie می‌شود. به عبارتی یک فرآیند Zombie در XV6 اجرای خود را تمام کرده ولی exit status آن توسط فرآیند parent بررسی نشده و همچنان در سیستم باقی مانده است.

```
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
};
```

در ساختار struct proc تمامی اطلاعاتی که مربوط به یک فرآیند می‌شود جمع‌آوری شده است. در این ساختار نام و pid خود فرآیند به همراه struct فرآیند والد ذخیره می‌شود. همچنین حجم اشغالی آن در سیستم و page table که فرآیند در آن قرار دارد نیز وجود دارد. در ساختار struct proc یک ساختار struct context نیز ذخیره می‌شود که در آن مقادیر رجیستر های مربوطه برای context switching توسط Kernel ذخیره می‌شود. این ساختار به عبارتی همان PCB یا Process Control Block در سیستم عامل XV6 می‌باشد.

در ادامه ساختار struct ptable در فایل *proc.c* بررسی می‌شود.

```
struct {
  struct spinlock lock;
  struct proc proc[NPROC];
} ptable;
```

این ساختار آرایه‌ای از struct proc ها می‌باشد که تعداد NPROC از فرآیند ها را در خود نگه می‌دارد. این مقدار در فایل *param.h* به مقدار 64 که حداکثر تعداد فرآیند ها در این سیستم عامل می‌باشد تعریف شده‌است. همچنین ساختار struct ptable دارای یک مکانیزم lock نیز می‌باشد که به هنگام خواندن اطلاعات از جدول فرآیند ها از این مکانیزم استفاده می‌شود تا در هنگام عملیات خواندن و بررسی اطلاعات جدول مقادیر داخل آن تغییر نکند.

پیاده سازی – ایجاد سیستم‌کال ساده

برای افزودن یک سیستم کال به این سیستم عامل، در فایل های مختلفی prototype ها، define ها و definition های مختلفی را اضافه نمود که در ادامه به هر کدام از آن ها پرداخته می‌شود. ابتدا به فایل *syscall.h* شماره سیستم‌کال جدید faps اضافه می‌شود.

```
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
#define SYS_mknod  17
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
#define SYS_faps   42
```

سپس prototype تابع این سیستم‌کال به فایل *defs.h* اضافه می‌شود تا در ادامه بتوان در فایل های *sysproc.c* و *proc.c* دسترسی داشت. همچنین prototype تابع به فایل *user.h* اضافه می‌شود تا در برنامه های سطح کاربر بتوان آن را فراخوانی کرد.

```
int faps(void);
```

سپس برای فراهم کردن interface سیستم‌کال ها به برنامه های سطح کاربر عبارت زیر به فایل *usys.S* اضافه می‌شود:

```
SYSCALL(faps)
```

در ادامه تابع sys_faps به فایل *sysproc.c* اضافه می‌شود. هنگامی که سیستم‌کال از داخل برنامه سطح کاربر صدا زده می‌شود در واقع این تابع فراخوانی می‌شود. همچنین prototype این تابع به فایل *syscall.c* اضافه می‌شود. داخل تابع sys_faps، تابع سیستم‌کال faps که تابع اصلی سیستم‌کال می‌باشد و در *proc.c* تعریف می‌شود صدا زده می‌شود. در ادامه که به این سیستم‌کال آرگومان اضافه می‌شود، handling این آرگومان ها در این تابع انجام می‌شود.

```
extern int sys_faps(void);
```

فراخوانی سیستم‌کال داخل تابع sys_faps:

```
int sys_faps(void)
{
  return faps();
}
```

تابع اصلی سیستم‌کال faps که در *proc.c* تعریف می‌شود به شرح زیر عمل می‌کند. ابتدا یک ساختار از نوع struct proc با نام process ایجاد می‌شود که از آن برای حرکت کردن روی جدول فرآیند ها استفاده می‌شود. سپس با دستور ()sti وقفه ها فعال می‌شوند. سپس قفل جدول فرآیند ها با توسط acquire انجام می‌شود. تابع acquire که در فایل *spinlock.c* تعریف شده است مسئولیت synchronize کردن این وقایع را دارد. بعد از آن تابع روی جدول فرآیند ها حرکت کرده و اطلاعات مربوط به فرآیند هایی که Unused نباشند را چاپ می‌کند و در پایان قفل مربوط به جدول فرآیند ها را release می‌کند.

```c
int faps (void)
{
  struct proc *process;

  sti();  // Enabling interrupts for locking mechanisms

  acquire(&ptable.lock);
  cprintf("\n\tNAME\t\t|\tPID\t|\tSTATE\t\t|\tSIZE\t|\tPARENT\t\n");
  for (process = ptable.proc; process < &ptable.proc[NPROC]; process++)
  {
    if (process->state != UNUSED)
    {
      cprintf("\t-------------------------------------------------------------------
---------------------------\t\n");

      if (process->state == EMBRYO)
        cprintf("\t%s\t\t|\t%d\t|\tEMBRYO\t\t|\t%d\t|\t%s\t\n",
        process->name, process->pid, process->sz, process->parent->name);
      else if (process->state == SLEEPING)
        cprintf("\t%s\t\t|\t%d\t|\tSLEEPING\t|\t%d\t|\t%s\t\n",
        process->name, process->pid, process->sz, process->parent->name);
      else if (process->state == RUNNABLE)
        cprintf("\t%s\t\t|\t%d\t|\tRUNNABLE\t\t|\t%d\t|\t%s\t\n",
        process->name, process->pid, process->sz, process->parent->name);
      else if (process->state == RUNNING)
        cprintf("\t%s\t\t|\t%d\t|\tRUNNING\t\t|\t%d\t|\t%s\t\n",
        process->name, process->pid, process->sz, process->parent->name);
      else if (process->state == ZOMBIE)
        cprintf("\t%s\t\t|\t%d\t|\tZOMBIE\t\t|\t%d\t|\t%s\t\n",
        process->name, process->pid, process->sz, process->parent->name);
    }
  }
  release(&ptable.lock);
  return 42;
}
```

در ادامه برای اینکه بتوان سیستم‌کال را به صورت یک برنامه سیستمی از ترمینال اجرا کرد فایل *faps.c* را ایجاد کرده و Makefile به نحوی تغییر داده می‌شود که در کنار کد های دیگر سیستم عامل برنامه faps نیز کامپایل شود. فایل Makefile شامل target های مختلفی می‌باشد که در کامپایل کردن سورس کد ها کمک می‌کنند. هر target طبق rule که برای آن تعریف شده است در صورت وجود تغییرات در سورس کد عمل می‌کند. برای کامپایل کد های اضافه شده، به بخش برنامه های کاربر UPROGS و برنامه های اضافی EXTRA فایل های مربوط به برنامه faps اضافه می‌شود.

در هنگام کامپایل سورس کد های کرنل در Makefile مشخص شده است که از فایل *linker.ld* به عنوان script لینکر استفاده شود. در فایل linker بخش های مختلف کد و Memory Layout برای این سیستم عامل مشخص شده است. همچنین آدرس ها و aligment های بخش های مختلف heap و stack در این script آورده شده

است. یکی از مهمترین پارامتر هایی که در linker تنظیم شده است، قسمت entry point می‌باشد. این قسمت نقطه شروع اجرا image کرنل را هنگام load شدن داخل حافظه مشخص می‌کند.

یک برنامه سطح کاربر *test.c* نوشته شده و در آن یک آرایه به چندین بخش مختلف تقسیم شده و هر بخش آن در یک process جداگانه sort می‌شود. در پایان اجرای برنامه از سیستم کال faps استفاده می‌شود و جدول فرآیند های سیستم چاپ می‌شود.

```
$ test
Operation finished in process 4.

        NAME            |       PID     |       STATE           |       SIZE    |       PARENT
        ------------------------------------------------------------------------------------------
        init            |       1       |       SLEEPING        |       12288   |       S
        ------------------------------------------------------------------------------------------
        sh              |       2       |       SLEEPING        |       16384   |       init
        ------------------------------------------------------------------------------------------
        test            |       3       |       SLEEPING        |       12288   |       sh
        ------------------------------------------------------------------------------------------
        test            |       4       |       RUNNING         |       12288   |       test
Operation finished in process 5.

        NAME            |       PID     |       STATE           |       SIZE    |       PARENT
        ------------------------------------------------------------------------------------------
        init            |       1       |       SLEEPING        |       12288   |       S
        ------------------------------------------------------------------------------------------
        sh              |       2       |       SLEEPING        |       16384   |       init
        ------------------------------------------------------------------------------------------
        test            |       3       |       SLEEPING        |       12288   |       sh
        ------------------------------------------------------------------------------------------
        test            |       5       |       RUNNING         |       12288   |       test
Operation finished in process 6.

        NAME            |       PID     |       STATE           |       SIZE    |       PARENT
        ------------------------------------------------------------------------------------------
        init            |       1       |       SLEEPING        |       12288   |       S
        ------------------------------------------------------------------------------------------
        sh              |       2       |       SLEEPING        |       16384   |       init
        ------------------------------------------------------------------------------------------
        test            |       3       |       SLEEPING        |       12288   |       sh
        ------------------------------------------------------------------------------------------
        test            |       6       |       RUNNING         |       12288   |       test
Operation finished in process 7.

        NAME            |       PID     |       STATE           |       SIZE    |       PARENT
        ------------------------------------------------------------------------------------------
        init            |       1       |       SLEEPING        |       12288   |       S
        ------------------------------------------------------------------------------------------
        sh              |       2       |       SLEEPING        |       16384   |       init
        ------------------------------------------------------------------------------------------
        test            |       3       |       SLEEPING        |       12288   |       sh
        ------------------------------------------------------------------------------------------
        test            |       7       |       RUNNING         |       12288   |       test
Operation finished in process 8.

        NAME            |       PID     |       STATE           |       SIZE    |       PARENT
        ------------------------------------------------------------------------------------------
        init            |       1       |       SLEEPING        |       12288   |       S
        ------------------------------------------------------------------------------------------
        sh              |       2       |       SLEEPING        |       16384   |       init
        ------------------------------------------------------------------------------------------
        test            |       3       |       SLEEPING        |       12288   |       sh
        ------------------------------------------------------------------------------------------
        test            |       8       |       RUNNING         |       12288   |       test

Sorting Finished.

        NAME            |       PID     |       STATE           |       SIZE    |       PARENT
        ------------------------------------------------------------------------------------------
        init            |       1       |       SLEEPING        |       12288   |       S
        ------------------------------------------------------------------------------------------
        sh              |       2       |       SLEEPING        |       16384   |       init
        ------------------------------------------------------------------------------------------
        test            |       3       |       RUNNING         |       12288   |       sh
```

شکل ۲ – نتایج تست سیستم‌کال *faps* در برنامه سطح کاربر

پیاده سازی – اضافه کردن آرگومان به سیستم‌کال

در این قسمت به سیستم‌کال ساده قسمت قبلی یک آرگومان از نوع int اضافه می‌شود که نشان دهنده pid یک فرآیند می‌باشد. در سیستم‌کال جدید، در جدول فرآیند ها این pid جست و جو شده و در صورت وجود اطلاعات آن چاپ می‌شود. این سیستم‌کال به همراه آرگومان عددی آن از یک برنامه سطح کاربر صدا زده می‌شود. در این حالت عدد وارد شده در فضای stack برنامه سطح کاربر قرار دارد. فضای پشته در برنامه کاربر توسط esp آدرس‌دهی می‌شود. اجرای خود سیستم‌کال در فضای آدرس‌دهی کرنل رخ می‌دهد. برای دریافت پوینتر مربوطه به آرگومان سیستم‌کال که در user-stack قرار دارد از تابع argint استفاده می‌شود. تابع argint که در فایل *syscall.c* تعریف شده است، تابع fetchint را صدا می‌زند که این تابع از آدرس‌دهی صحیح پوینتر آرگومان در فضای آدرس‌دهی کرنل و کاربر اطمینان حاصل می‌کند. تابع faps که اجرای سیستم‌کال را به عهده دارد از تابع sys_faps صدا زده می‌شود، بنابراین کنترل (handling) آرگومان های سیستم‌کال در این تابع انجام می‌شود. این تابع به صورت زیر تغییر داده می‌شود:

```
int sys_faps(void)
{
  int pid;

  if(argint(0, &pid) < 0)
    return -1;

  return faps(pid);
}
```

همچنین فایل *faps.c* به صورت زیر تغییر داده می‌شود تا بتوان این سیستم‌کال را با آرگومان از ترمینال اجرا کرد:

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

int main (int argc, char *argv[])
{
    if (argc != 2)
    {
        printf(2, "Wrong arguments...\n");
        exit();
    }

    faps(atoi(argv[1]));
    exit();
}
```

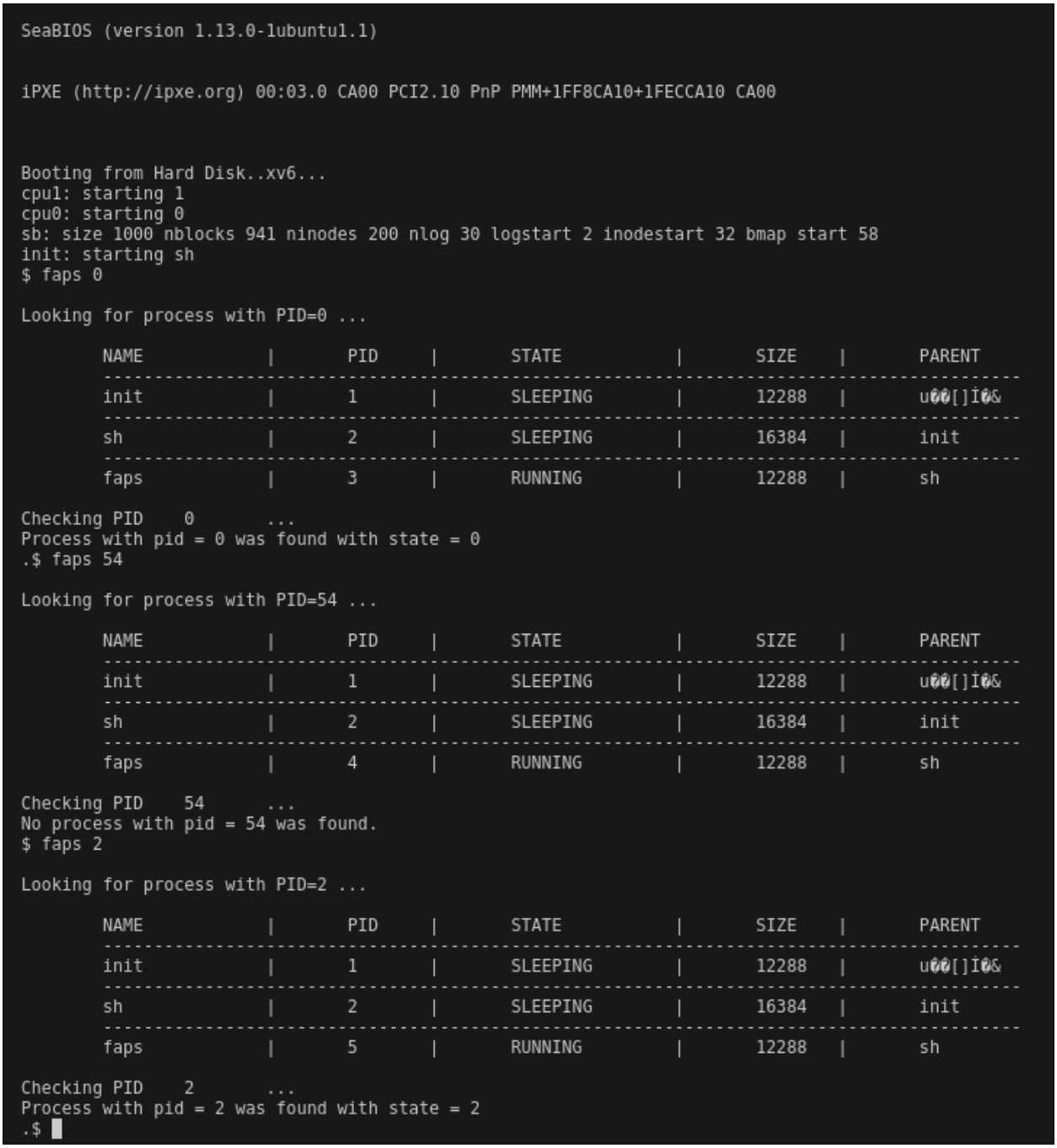
همچنین تمامی prototype هایی که از تابع faps در فایل های *defs.h* و *user.h* و *syscall.c* وجود دارد برای داشتن یک آرگومان از نوع int تغییر داده می‌شوند. بخش اجرایی تابع نیز به صورت زیر نوشته می‌شود:

```
int faps (int pid)
{
  struct proc *process;

  sti();  // Enabling interrupts for locking mechanisms

  acquire(&ptable.lock);
  cprintf("\nLooking for process with PID=%d ...\n", pid);
  for (process = ptable.proc; process < &ptable.proc[NPROC]; process++)
  {
    if (process->pid == pid)
    {
      cprintf("Process with pid = %d was found with state = %d\n\r.",
              pid, (int)process->state);
      release(&ptable.lock);
      return 42;
    }
  }

  cprintf("No process with pid = %d was found.\n\r", pid);
  release(&ptable.lock);
  return 42;
}
```

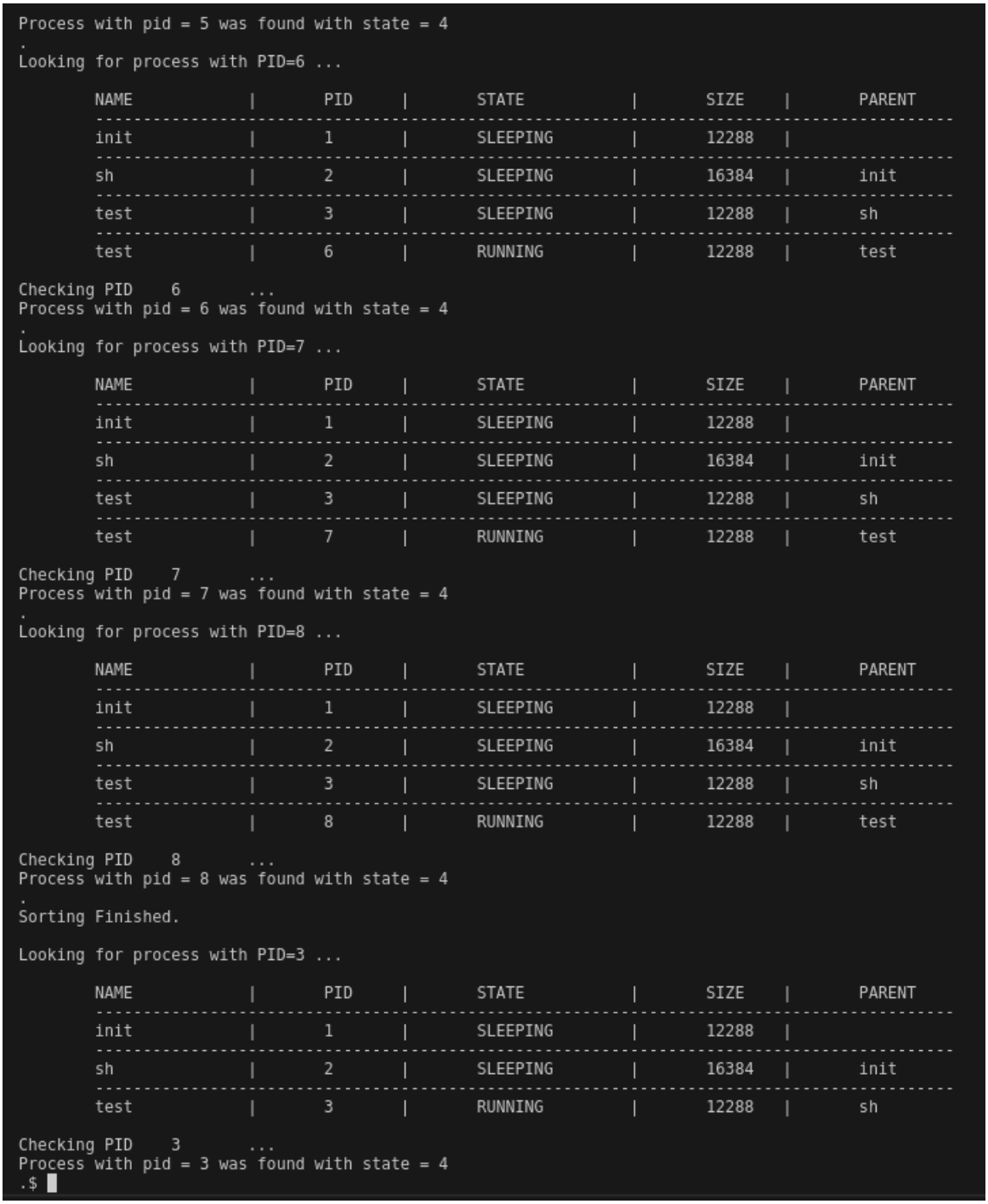
برنامه سطح کاربر نیز برای سنجیدن تغییرات جدید سیستم‌کال نیز تغییر داده می‌شود. در این برنامه فراخوانی های سیستم‌کال faps به همراه تابع getpid() که از سیستم‌کال های از پیش تعریف شده XV6 می‌باشد، انجام می‌شود.

```
faps(getpid());
```

در ادامه نتایج تست سیستم‌کال با آرگومان pid آورده شده است. ابتدا فراخوانی سیستم‌کال در ترمینال و سپس توسط برنامه سطح کاربر انجام می‌شود.



شکل ۳ – فراخوانی سیستم‌کال $faps$ با آرگومان pid از ترمینال



شکل ۴ – فراخوانی سیستم‌کال $faps$ با آرگومان pid از داخل برنامه سطح کاربر

پیاده سازی – آرگومان ها و تغییر منطق سیستم‌کال

در ادامه یک سیستم‌کال جدید با نام ps به سیستم عامل اضافه می‌شود. مطابق مراحل ذکر شده در بالا به این سیستم عامل یک سیستم‌کال جدید اضافه می‌شود که ۳ آرگومان دریافت می‌کند. آرگومان اول از نوع int می‌باشد و نشان دهنده حالت یک فرآیند خواهد بود. مقداردهی این آرگومان مطابق با عدد های ساختار enum procstate انجام خواهد شد. آرگومان دوم از نوع int بوده و نشان دهنده pid فرآیند مورد نظر است. آرگومان سوم از نوع پوینتر به ساختار struct Process_Info می‌باشد. در یک برنامه سطح کاربر با استفاده از malloc بخشی از حافظه به این ساختار اختصاص داده می‌شود و پوینتر مربوطه به آن که توسط malloc برگردانده می‌شود به عنوان آرگومان ورودی سوم به سیستم‌کال ps داده می‌شود. این سیستم‌کال با توجه به ورودی های state و pid در جدول فرآیند ها حرکت کرده و نزدیکترین فرآیند را می‌یابد. سپس ساختار struct Process_Info که پوینتر آن را گرفته است را با اطلاعات فرآیند پیدا شده پر کرده و بر می‌گرداند. در برنامه سطح کاربر اطلاعات پر شده توسط سیستم‌کال داخل struct چاپ می‌شود. عملیات handling آرگومان ها توسط تابع sys_ps داخل فایل *sysproc.c* و بخش اجرایی سیستم‌کال توسط تابع ps داخل فایل *proc.c* انجام می‌شود. برای دسترسی به ساختار struct مورد نظر در این فایل ها، ساختار struct Process_Info در فایل *defs.h* تعریف می‌شود. برای دسترسی آن در برنامه سطح کاربر این تعریف در فایل *user.h* نیز انجام می‌شود.

```
struct Process_Info
{
  char parent_name[16];             // Parent process name
  int pid;                          // Process ID
  int state;                        // Process state
  char name[16];                    // Process name
};
```

برای چک کردن معتبر بودن آرگومان های int که از فضای کاربر مقدار دهی شده‌اند در فضای کرنل مانند قسمت قبل از تابع argint استفاده می‌شود. برای بررسی آرگومان سوم که از نوع پوینتر struct Process_Info* می‌باشد از تابع argptr استفاده می‌شود. این تابع در عمل شبیه به argint عمل می‌کند. این تابع nامین آرگومان ۳۲ بیتی سیستم‌کال را بررسی می‌کند. تابع argptr، تابع argint را فراخوانی می‌کند تا ابتدا این آرگومان را به عنوان آرگومان عددی fetch کند. سپس چک می‌کند که آیا این integer به عنوان پوینتر در بخش کاربر فضای آدرس‌دهی قرار دارد یا خیر. در طی اجرا شدن تابع argptr ۲ مرتبه چک کردن انجام می‌شود. ابتدا پوینتر stack کاربر در مرحله fetch کردن آرگومان چک می‌شود و سپس خود آرگومان به عنوان پوینتر کاربر چک می‌شود.

برای بررسی گرفتن صحیح آرگومان ها در تابع sys_ps به صورت زیر آرگومان های ورودی بررسی می‌شوند. در این تابع برای هر ۳ آرگومان مورد نظر به صورت local متغیر های مربوطه تعریف می‌شوند. توابع argint و argptr این متغیر های محلی که در فضای کرنل قرار دارند با مقادیر صحیح از فضای کاربر مقداردهی می‌کنند. آدرس محلی ساختار struct Process_Info در تابع sys_ps به عنوان یک پوینتر void به تابع argptr پاس داده می‌شود و این تابع آدرس local را با پوینتر گرفته شده از فضای کاربر تطبیق می‌دهد. سپس تابع ps بر روی این متغیر های local فراخوانی می‌شود.

```
int sys_ps(void)
{
  int pid;
  int state;
  struct Process_Info *process_info;

  if (argint(0, &pid) < 0   ||
      argint(1, &state) < 0 ||
      argptr(2, (void *)&process_info, sizeof(*process_info)) < 0)
    return -1;

  return ps(pid, state, process_info);
}
```

در ادامه تابع ps در فایل *proc.c* آورده شده‌است. در این تابع برای یافتن نزدیکترین فرآیند موجود در سیستم با pid داده شده نیاز است تا اختلاف pid ورودی و pid فرآیند های جدول را در هر قدم داشته باشیم و در صورتی که فرآیندی پیدا شد که اختلاف pid آن نسبت به فرآیند قبلی کمتر باشد آن را جایگزین کنیم. به همین منظور در ابتدا یک متغیر به اسم pid_diff تعریف شده و با حداکثر مقدار ممکن initialize می‌شود. با حرکت روی جدول فرآیند ها نیاز است تا abs اختلاف pid ها حساب شود. از آنجایی که این تابع در کتابخانه *math.h* وجود دارد ولی نمی‌توان از آن در XV6 استفاده کرد، شرط کمتر بودن اختلاف pid ها برای هر دو حالت مثبت و منفی بودن نوشته می‌شود. با حرکت روی جدول فرآیند ها ابتدا فرآیند ها بر حسب تطبیق حالتشان با state ورودی فیلتر شده و سپس نزدیکترین pid بدست می‌آید. در این حالت، سیستم‌کال ساختار struct Process_Info که پوینتر به آن داده شده است را با اطلاعات فرآیند پیدا شده پر می‌کند. در نهایت مانند سیستم‌کال ساده اولیه قفل جدول فرآیند ها release شده و اجرا به برنامه سطح کاربر بر می‌گردد.

```
int ps(int pid, int state, struct Process_Info *process_info)
{
  struct proc *process;
  int pid_diff = __INT_MAX__;

  sti();

  acquire(&ptable.lock);

  for (process = ptable.proc; process < &ptable.proc[NPROC]; process++)
  {
    if ((int)process->state == state)
    {
      if (((process->pid - pid) < 0 && (pid - process->pid) < pid_diff) ||
          ((process->pid - pid) > 0 && (pid - process->pid) < pid_diff))
      {
        pid_diff = pid - process->pid;

        process_info->pid = process->pid;
        process_info->state = (int)process->state;
        strncpy(process_info->name, process->name, 16);
        strncpy(process_info->parent_name, process->parent->name, 16);
      }
    }
  }

  release(&ptable.lock);
  return 22;
}
```

در برنامه سطح کاربر ساختار struct Process_Info همانند گذشته تعریف شده و با استفاده از malloc بخشی از حافظه به این ساختار اختصاص داده می‌شود. دستور malloc پوینتری از نوع void* بر می‌گرداند. برای استفاده از آن در آدرس دهی ساختار مورد نظر، پوینتر برگردانده شده به نوع پوینتر struct Process_Info* تبدیل (cast) می‌شود.

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

struct Process_Info
{
  char parent_name[16];             // Parent process ID
  int pid;                          // Process ID
  int state;                        // Process state
  char name[16];                    // Process name
};

int main()
{
    struct Process_Info* process_info = (struct Process_Info*)
                                        malloc(sizeof(struct Process_Info));
```

```
    ps(6, 4, process_info);
    printf(1, "Process found with the following details:\n\r");
    printf(1, "Name\t%s\n", process_info->name);
    printf(1, "PID\t%d\n", process_info->pid);
    printf(1, "State\t%d\n", process_info->state);
    printf(1, "Parent Name\t%s\n", process_info->parent_name);
    free(process_info);
    faps(getpid());
    exit();
}
```

در ادامه در برنامه سطح کاربر حالت های مختلفی که ممکن است در اجرای این سیستم‌کال پیش بیاید بررسی می‌شود.

```
Checking PID    6       ...
Process with pid = 6 was found with state = 4
.
Looking for process with PID=7 ...

        NAME            |       PID     |       STATE           |       SIZE    |       PARENT
        ------------------------------------------------------------------------------------------
        init            |       1       |       SLEEPING        |       12288   |
        ------------------------------------------------------------------------------------------
        sh              |       2       |       SLEEPING        |       16384   |       init
        ------------------------------------------------------------------------------------------
        test            |       3       |       SLEEPING        |       12288   |       sh
        ------------------------------------------------------------------------------------------
        test            |       7       |       RUNNING         |       12288   |       test

Checking PID    7       ...
Process with pid = 7 was found with state = 4
.
Looking for process with PID=8 ...

        NAME            |       PID     |       STATE           |       SIZE    |       PARENT
        ------------------------------------------------------------------------------------------
        init            |       1       |       SLEEPING        |       12288   |
        ------------------------------------------------------------------------------------------
        sh              |       2       |       SLEEPING        |       16384   |       init
        ------------------------------------------------------------------------------------------
        test            |       3       |       SLEEPING        |       12288   |       sh
        ------------------------------------------------------------------------------------------
        test            |       8       |       RUNNING         |       12288   |       test

Checking PID    8       ...
Process with pid = 8 was found with state = 4
.
Sorting Finished.

Looking for process with PID=3 ...

        NAME            |       PID     |       STATE           |       SIZE    |       PARENT
        ------------------------------------------------------------------------------------------
        init            |       1       |       SLEEPING        |       12288   |
        ------------------------------------------------------------------------------------------
        sh              |       2       |       SLEEPING        |       16384   |       init
        ------------------------------------------------------------------------------------------
        test            |       3       |       RUNNING         |       12288   |       sh

Checking PID    3       ...
Process with pid = 3 was found with state = 4
.$ ps_test


parent id:      sh

Process found with the following details:
Name    ps_test
PID     9
State   4
Parent Name     sh

Looking for process with PID=9 ...

        NAME            |       PID     |       STATE           |       SIZE    |       PARENT
        ------------------------------------------------------------------------------------------
        init            |       1       |       SLEEPING        |       12288   |       ??
        ------------------------------------------------------------------------------------------
        sh              |       2       |       SLEEPING        |       16384   |       init
        ------------------------------------------------------------------------------------------
        ps_test         |       9       |       RUNNING         |       45056   |       sh

Checking PID    9       ...
Process with pid = 9 was found with state = 4
.$ ||
```

شکل ۵ – نتیجه فراخوانی سیستم‌کال ps از داخل برنامه سطح کاربر