Университет ИТМО

Факультет ФПИ и КТ

### Отчёт

### по лабораторной работе 2

# «Операционные системы»

Вариант

procfs: task\_struct, page

Студент:

Ляо Ихун

Гр.P33131

Преподаватель:  
Осипов Святослав Владимирович

# Задача:

Разработать комплекс программ на пользовательском уровне и уровне ярда, который собирает информацию на стороне ядра и передает информацию на уровень пользователя, и выводит ее в удобном для чтения человеком виде. Программа на уровне пользователя получает на вход аргумент(ы) командной строки (не адрес!), позволяющие идентифицировать из системных таблиц необходимый путь до целевой структуры, осуществляет передачу на уровень ядра, получает информацию из данной структуры и распечатывает структуру в стандартный вывод. Загружаемый модуль ядра принимает запрос через указанный в задании интерфейс, определяет путь до целевой структуры по переданному запросу и возвращает результат на уровень пользователя.

Интерфейс передачи между программой пользователя и ядром и целевая структура задается преподавателем. Интерфейс передачи может быть один из следующих:

1. syscall - интерфейс системных вызовов.
2. ioctl - передача параметров через управляющий вызов к файлу/устройству.
3. procfs - файловая система /proc, передача параметров через запись в файл.
4. debugfs - отладочная файловая система /sys/kernel/debug, передача параметров через запись в файл.

Целевая структура может быть задана двумя способами:

1. Именем структуры в заголовочных файлах Linux
2. Файлом в каталоге /proc. В этом случае необходимо определить целевую структуру по пути файла в /proc и выводимым данным.

# Выполнение:

## 1.Программа на уровне ядра:

### 1.Необходимые структуры

#### 1.init\_task

Для получения 0-ого процесса. И при его помощи получаем информации всех процесс

init\_task.h в /include/linux/

init\_task.c в /init/

#### 2.task\_struct

/include/linux/sched.h

图示, 工程绘图

描述已自动生成

#### 3.mm\_struct

/include/linux/mm\_types.h

#### 4.vm\_area\_struct

/include/linux/mm\_types.h

#### 5.page

/include/linux/mm\_types.h

图示, 工程绘图

描述已自动生成

### 2.Код

task\_struct определёна в /include/linux/sched.h

//Where defined struct task\_struct

#include <linux/sched.h>

//where defined initialization and exit

#include <linux/init.h>

//where diefined basical functions and macros

#include <linux/kernel.h>

//where definded function, fieds about kenal module

#include <linux/module.h>

//where definded init\_task, which is used to get 0 proccess.

#include <linux/init\_task.h>

//where struct page defined

#include <linux/mm\_types.h>

//pgd\_list and pgd\_lock

#include <asm/pgtable.h>

#include <linux/list.h>

//spinlock

//pg\_data\_t and macro

#include <linux/mmzone.h>

//#include <asm/numa.h>

//#include

#define WRITEBUF 1024

#define READBUF 5120

//proc

struct proc\_dir\_entry \*proc\_dir = NULL;

struct proc\_dir\_entry \*proc\_file = NULL;

char write\_buf[WRITEBUF];

char read\_buf[READBUF];

int pid = 1;

int length = 0;

#include <linux/proc\_fs.h>

MODULE\_DESCRIPTION("procfs:task\_struct, page");

MODULE\_VERSION("1");

MODULE\_LICENSE("GPL");

LIST\_HEAD(pgd\_list);

DEFINE\_SPINLOCK(pgd\_lock);

//count how many proccesses

int count\_proccess(struct task\_struct \* zero\_T){

struct list\_head \*pos;

int count = 0;

list\_for\_each(pos,&zero\_T->tasks)

{

count++;

}

return count;

}

void get\_state(long state, char \*\* result){

if(state==0){

\*result = "RUNNING";

}else if(state==1){

\*result = "INTERRUPTIBLE SLEEPING";

}else if(state==2){

\*result = "UNITERRUPTIBLE SLEEPING";

}else if(state==64){

\*result = "DEAD";

}else if(state==128){

\*result = "WAKEKILL";

}else if(state==256){

\*result = "WAKING";

}else if(state==512){

\*result = "PARKED";

}else if(state=1024){

\*result = "NOLOAD";

}else if(state==2048){

\*result = "NEW";

}else if(state==4096){

\*result = "STATE\_MAX";

}else if(state==1026){

\*result = "ZOMBIE";

}else{

\*result = "UNKNOWN";

}

}

//print numbers of proccess in different states

void print\_numbers\_state(struct task\_struct \*t){

struct list\_head \*pos;

struct task\_struct \*p;

int zombie = 0;

int running = 0;

int interruptible = 0;

int uninterruptible = 0;

int dead = 0;

int wakekill = 0;

int waking = 0;

int new = 0;

int parked = 0;

int noload = 0;

int unknown = 0;

int state\_max = 0;

list\_for\_each(pos, &t->tasks){

p = list\_entry(pos,struct task\_struct,tasks);

long state = p->\_\_state;

if(state==0){

running++;

}else if(state==1){

interruptible++;

}else if(state==2){

uninterruptible++;

}else if(state==64){

dead++;

}else if(state==128){

wakekill++;

}else if(state==256){

waking++;

}else if(state==512){

parked++;

}else if(state=1024){

noload++;

}else if(state==2048){

new++;

}else if(state==4096){

state\_max++;

}else if(state==1026){

zombie++;

}else{

unknown++;

}

}

length += sprintf(read\_buf + length,"running:%d zombie:%d interruptible:%d uninterruptible:%d dead:%d new:%d parked:%d state\_max:%d wakekill:%d noload:%d waking:%d unknown %d \n", running, zombie, interruptible, uninterruptible, dead, new, parked, state\_max, wakekill, noload, waking, unknown);

}

struct page\* get\_one\_page(struct mm\_struct \*mm, long va){

pgd\_t\* pgd;

p4d\_t\* p4d;

pud\_t \*pud;

pmd\_t \*pmd;

pte\_t \*pte;

struct page\* page;

pgd = pgd\_offset(mm,va);

if(!pgd\_present(\*pgd)){

return NULL;

}

p4d = p4d\_offset(pgd, va);

if (!p4d\_present(\*p4d)){

return NULL;

}

pud = pud\_offset(p4d, va);

if(!pud\_present(\*pud)){

return NULL;

}

pmd = pmd\_offset(pud,va);

if(!pmd\_present(\*pmd)){

return NULL;

}

pte = pte\_offset\_kernel(pmd, va);

if(!pte\_present(\*pte)){

return NULL;

}

page = pte\_page(\*pte);

return page;

}

//print all informations of all proccesses(task\_struct)

void print\_task(struct task\_struct\* zero\_T){

struct list\_head \*pos;

struct task\_struct \*p;

list\_for\_each(pos, &zero\_T->tasks){

char\* state\_string;

p = list\_entry(pos,struct task\_struct, tasks);

if(p->pid == pid){

long state = p->\_\_state;

get\_state(state,&state\_string);

length += sprintf(read\_buf + length,"pid:%d state:%s\ prio:%d static prio:%d recent used cpu:%d\n",p->pid, state\_string, p->prio,p->static\_prio, p->recent\_used\_cpu);

length += sprintf(read\_buf + length,"Struct pages in this task:\n");

struct mm\_struct \*mm = p->mm;

if(mm){

struct vm\_area\_struct\* vas = mm->mmap;

long va;

int page\_count = 0;

for(va = vas->vm\_start; va <= vas->vm\_end; va = va + PAGE\_SIZE){

page\_count++;

struct page\* p = get\_one\_page(mm,va);

if(p){

length += sprintf(read\_buf + length,"%d page:\n",page\_count);

length += sprintf(read\_buf + length," Flags:%lu\n",p->flags);

length += sprintf(read\_buf + length," Start:%lx\n",va);

length += sprintf(read\_buf + length," End:%lx\n",va+PAGE\_SIZE);

}

}

}else{

length += sprintf(read\_buf + length,"Failed to read pages for this task\n");

}

return;

}

}

length += sprintf(read\_buf + length, "No task with such pid\n");

}

ssize\_t write\_proc(struct file\* filp, const char \_\_user \* buffer, size\_t len, loff\_t\* offset){

printk("Call write\n");

int pid\_read;

int numbers;

if(copy\_from\_user(write\_buf,buffer,len)){

return -ENOMEM;

}

numbers = sscanf(write\_buf,"%d",&pid\_read);

if(numbers!=1){

return -EFAULT;

}

pid = pid\_read;

int new\_len = strlen(write\_buf);

\*offset = new\_len;

printk("Chosen pid is %d\n", pid);

return new\_len;

}

ssize\_t read\_proc(struct file\* file, char \_\_user\* data,size\_t count, loff\_t \*off){

struct task\_struct \*task;

task = &init\_task;

if (\*off >0 || count < READBUF)

return 0;

printk("Now read begins\n");

int number = count\_proccess(task);

length += sprintf(read\_buf + length,"Numbers of proccesses:%d\n",number);

print\_numbers\_state(task);

print\_task(task);

if (copy\_to\_user(data,read\_buf,length)){

return -EFAULT;

}

\*off = length;

printk("Read finished\n");

return length;

}

struct proc\_ops operations = {

.proc\_read = read\_proc,

.proc\_write = write\_proc,

};

int \_\_init task\_init (void){

proc\_dir = proc\_mkdir("lab2", NULL);

proc\_file = proc\_create("lab\_info",0666,proc\_dir,&operations);

return 0;

}

void \_\_exit task\_exit(void){

if(proc\_file){

proc\_remove(proc\_file);

}

if(proc\_dir){

proc\_remove(proc\_dir);

}

printk("Finished\n");

}

module\_init(task\_init);

module\_exit(task\_exit);

## 2.Программа на уровне пользователи:

### 1.Код

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <string.h>

bool is\_int(char argv[]){

}

int main(int argc, char \*argv[]){

if(argc != 2){

fprintf(stderr,"This progromnma only accept one parameter\n");

return 1;

}

char\* in = argv[1];

char\* error;

long int pid = strtol(in,&error,10);

if(\*error != '\0')

{

fprintf(stderr, "Please only input number cause pid can only be a number\n");

return 1;

}

if(pid < 0){

fprintf(stderr,"Pid should be bigger than 0\n");

}

char input\_buf[1024];

char result\_buf[5120];

int fd = open("/proc/lab2/lab\_info",O\_RDWR);

//int fd = open("./test.txt",O\_RDWR);

write(fd,argv[1],strlen(argv[1]));

close(fd);

int new\_fd = open("/proc/lab2/lab\_info", O\_RDWR);

read(new\_fd,result\_buf,5120);

puts(result\_buf);

return 0;

}ы

# Результат:



# Вывод:

В ходе прочитал часть исходного кода Linux, которая мне нужна. Научился как создать интерфейс передачи между kernel и программой пользователей при помощи profcs.