

«گزارش کار پروژه اول»

«آزمایشگاه سیستم عامل»

کسری کاشانی 810101490

البرز محمودیان 810101514

نرگس بابالار 810101557

• آشنایی با سیستم عامل xv6

1- معماری سیستم عامل xv6 چیست؟ چه دلایلی در دفاع از نظر خود دارید؟

درواقع سیستم عامل xv6 یک مدل از پیاده سازی مدرن اما ساده از سیستم عامل UNIX برای سیستم های چند پردازنده x86 و RISC_V است. معماری این سیستم عامل از معماری کلی UNIX تبعیت میکند و از ویژگی های آن میتوان به سادگی کد و ساختار، پشتیبانی از پردازش های چندگانه (multi_process) و مدیریت حافظه ساده اشاره کرد و همچنین معماری این سیستم عامل با وجود سادگی، بسیاری از مفاهیم پیشرفته سیستم عامل را نیز پوشش میدهد.

این سیستم در واقع یک شبه یونیکس (UNIX Like) است. از طرفی دسته بندی فایل های این سیستم عامل نیز مثل یونیکس calls system و program level user است.

این سیستم عامل با AMSIC و برای x86 multiprocessor و سیستم های RISC_V طراحی شده است؛ از دلایل این میتوان به وجود فایل asm.h که در آن استفاده از معماری x86 ذکر شده اشاره کرد.

2- یک پردازنده در سیستم عامل xv6 از چه بخشهایی تشکیل شده است؟ این سیستم عامل به طور کلی چگونه پردازنده را به پردازنده های مختلف اختصاص میدهد؟

در سیستم عامل XV6، پردازنده ها به وسیله ی زمان بندی (scheduler) به فرآیندهای مختلف اختصاص می یابند. هر پردازنده وظیفه دارد به نوبت فرآیندهایی که نیازمند اجرا هستند را اجرا کند. بخش هایی که یک پردازنده را در این سیستم عامل XV6 تشکیل می دهند

شامل فضای حافظه کاربر از جمله دستورات، داده ها و استک می باشد. همچنین شامل وضعیت هر پردازنده که به صورت خصوصی در اختیار kernel قرار دارد.

در واقع این سیستم عامل به شکل time_sharing عمل میکند و در هر زمان CPU را در اختیار یک فراند قرار میدهد تا اجرا شود.

زمانی که یک process اجرا نمیشود سیستم عامل محتوای آن را در رجیستر CPU ذخیره میکند.

و در واقع اگر برنامه ای در حال اجرا باشد اما سهم زمانی که به آن اختصاص داده شده تمام شود سیستم عامل محتوای مربوط به رجیستر های این فرایند را در حافظه ذخیره میکند و CPU را در اختیار فرایند بعدی قرار می دهد.

و زمانی که فرایند بعدی اجرا شد محتوا دوباره از مموری به رجیستر ها برگردانده میشود و CPU برای اجرای آن اختصاص داده میشود .

این روش به شکلی عمل میکند که همه فرایندها به شکل همروند باهم جلو می روند .
همچنین kernel سیستم عامل برای هر کدام از فرایندها یک pid اختصاص میدهد که بتواند آن را پیگیری کند.

3- مفهوم descriptor file در سیستم عاملهای مبتنی بر UNIX چیست؟ عملکرد pipe در سیستم عامل xv6 چگونه است و به طور معمول برای چه هدفی استفاده میشود؟

در این سیستم ها File descriptor یک عدد صحیح کوچک است که نشان دهنده یک object مدیریت شده توسط kernel است که ممکن است یک فرایند در آن بنویسد یا از آن بخواند.
در اصل ما ترجیح میدهیم یک توصیفگر فایل داشته باشیم تا اطلاعاتی درباره فایل باز مانند نوع آن ، دسترسی های مجاز مثل مثلا خواندن و نوشتن و غیره و مکان فعلی آن به ما بدهد.
سیستم با فراخوانی یک file توسط یک فرایند ، زمانی که فایلی باز بشود یک file descriptor به فرایند فراخوان باز میگرداند.

در سیستم عامل Xv6 ، kernel یک table از file descriptor ها دارد , درواقع هر فرایند یک فضای خصوصی رو برای file descriptor دارند که در ابتدا مقدار صفر میگیرند.
در اصل یک فرایند برای خواندن یا ورودی استاندارد مقدار 0 ، برای نوشتن یا خروجی استاندارد مقدار 1، و برای نوشتن پیام خطا یا ارور استاندارد مقدار 2 استفاده میشود.
عملکرد pipe به منظور ارتباط بین فرایندها استفاده میشود و امکان انتقال داده ها بین دو فرایند را فراهم می کند.

4- فراخوانی های سیستمی exec و fork چه عملی انجام میدهند؟ از نظر طراحی، ادغام نکردن این دو چه مزیتی دارد؟

فراخوانی سیستمی fork() :

فراخوانی سیستمی fork() فرآیندی را که فراخوانی را اجرا می کند، کپی کرده و یک فرایند جدید (فرزند) ایجاد می کند و یک نسخه کپی از پردازنده هایی میسازد که این فراخوانی صدا

زده است، در واقع این فرایند به واسطه فرایند اولی ایجاد میشود و در اصل با همان حافظه parent خود عمل میکند.

و همانطور که گفته شد kernel به هر فرایند یک کد pid اختصاص میدهد و زمانی که این فراخوان صورت میگیرد بعد از اجرا شدن فرایند parent زمانی که نوبت به اجرای فرایند child میرسد pid مربوط به آن را برمیگرداند.

پس از اجرای آن نیز برای نشان دادن خاتمه یافتن عملیات pid=0 برمیگرداند. یکی درگر از ویژگی های این فراخوانی این است که بعد از اتمام کار فرزند به فرایند parent برمیگردیم.

فراخوانی سیستمی exec():

فراخوانی سیستمی exec() یک برنامه با حافظه جدید که در آن فایل ELF بارگذاری شده را در فرآیند جاری لود کرده و جایگزین کد و داده های آن می کند.

در واقع برای این فراخوانی ما مجاز به اجرای برنامه جدید در فرایند فعلی هستیم. و زمانی که بخواهیم یک برنامه هنگام اجرای یک برنامه دیگر اجرا شود از این فراخوانی استفاده میکنیم.

در این فراخوانی برنامه به فراخوان برنمیگردد و برنامه دیگر اجرا میشود. مگر زمانی که یک خطا رخ بدهد و برنامه در حال اجرا، اجرای پردازنده را خاتمه می دهد.

مزیت جدا نگه داشتن این دو فراخوانی در این است که امکان اجرای کدهای سفارشی بین ایجاد فرآیند جدید و لود برنامه جدید فراهم می شود که انعطاف پذیری بیشتری به توسعه دهندگان می دهد.

• اضافه کردن یک متن به boot message

برای اضافه کردن یک پیام هنگام boot شدن سیستم، در فایل init.c از تابع استاندارد printf() استفاده می کنیم تا در کنسول یعنی fd = 1 پیام مورد نظر را وارد کند.

```

home > Kasra > Desktop > Lab1 > xv6-public-master > C init.c
1 // init: The initial user-level program
2
3 #include "types.h"
4 #include "stat.h"
5 #include "user.h"
6 #include "fcntl.h"
7
8 char *argv[] = { "sh", 0 };
9
10 int
11 main(void)
12 {
13     int pid, wpid;
14
15     if(open("console", O_RDWR) < 0){
16         mknod("console", 1, 1);
17         open("console", O_RDWR);
18     }
19     dup(0); // stdout
20     dup(0); // stderr
21
22     for(;;){
23         printf(1, "init: starting sh\n");
24         printf(1, "Group members:\n Kasra Kashani\n Albroz Mahmoudian\n Narges Babalar\n");
25         pid = fork();
26         if(pid < 0){
27             printf(1, "init: fork failed\n");
28             exit();
29         }
30         if(pid == 0){
31             exec("sh", argv);
32             printf(1, "init: exec sh failed\n");
33             exit();
34         }
35         while((wpid=wait()) >= 0 && wpid != pid)
36             printf(1, "zombie!\n");
37     }
38 }

```

```

QEMU

Machine View

SeaBIOS (version 1.16.3-debian-1.16.3-2)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1EFCB050+1EF0B050 CA00

Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group members:
Kasra Kashani
Albroz Mahmoudian
Narges Babalar
$

```

• شرح پروژه

1. برای تغییر cursor به چپ و راست، باید در هر مرتبه فشردن یکی از کلید های چپ یا راست، ابتدا موقعیت فعلی cursor را بگیریم و سپس آن را آپدیت کنیم. در واقع هنگام چپ رفتن، مقدار pos را یکی کم و هنگام راست رفتن نیز مقدار pos را یکی زیاد کنیم و این pos جدید را به عنوان cursor جدید اعلام کنیم. همچنین باید توجه داشت که cursor از سر خط نمی تواند چپ تر برود و از راست ترین کاراکتر موجود در خط نیز نمی تواند راست تر برود.

```
71 static void update_cursor(int move)
72 {
73     int pos;
74
75     // Get current cursor position
76     outb(CRTPORT, 14);
77     pos = inb(CRTPORT + 1) << 8;
78     outb(CRTPORT, 15);
79     pos |= inb(CRTPORT + 1);
80
81     switch(move){
82     case 0:
83         pos --;
84         break;
85     case 1:
86         pos ++;
87         break;
88     default:
89         break;
90     }
91
92     // Reset cursor position
93     outb(CRTPORT, 14);
94     outb(CRTPORT + 1, pos >> 8);
95     outb(CRTPORT, 15);
96     outb(CRTPORT + 1, pos);
97 }
```

```

602     case LEFTARROW: // Left
603         if((input.e - lefts) > input.w){
604             update_cursor(0);
605             lefts ++;
606         }
607         break;
608     case RIGHTARROW: // Right
609         if(lefts > 0){
610             update_cursor(1);
611             lefts --;
612         }
613         break;

```

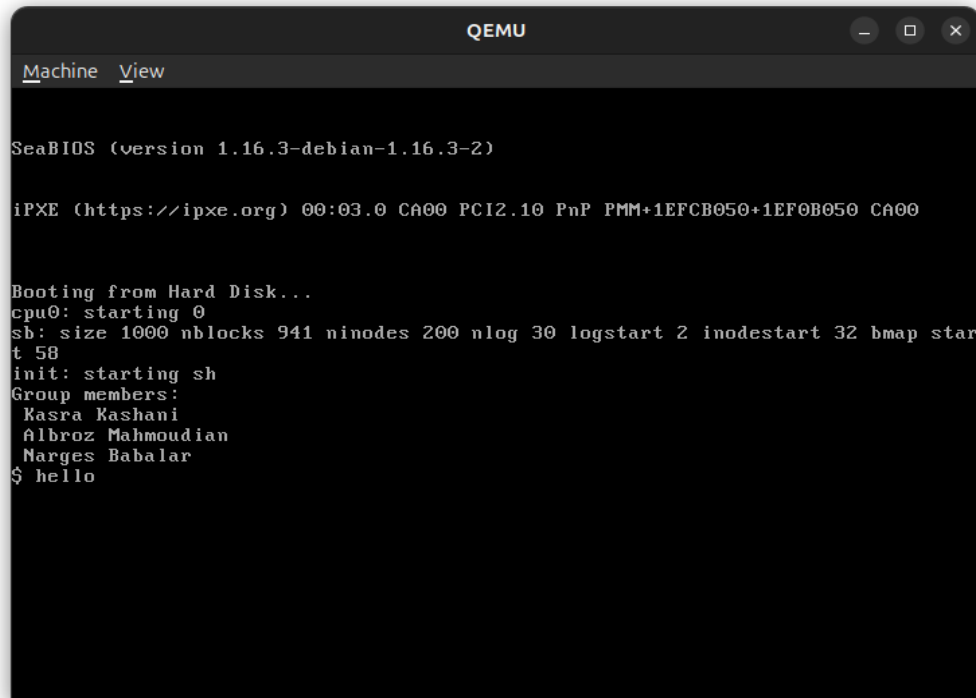
متغیر سراسری lefts را نیز تعریف می کنیم تا تعداد چپ هایی که رفتیم را ذخیره کند و هنگام وارد کردن ورودی یا backspace زدن در موقعیت cursor، بتوانیم به کمک این متغیر عملیات shift دادن کاراکتر های سمت راست cursor را به درستی انجام دهیم.

```

99     static void shift_left_next_chars(int pos)
100     {
101         for (int i = pos - 1; i < pos + lefts; i++)
102             crt[i] = crt[i + 1];
103
104         for (int i = input.e - lefts; i < input.e; i++){
105             input.buf[i] = input.buf[i + 1];
106             input.buf_copy[i] = input.buf_copy[i + 1];
107         }
108     }
109
110     static void shift_right_next_chars(int pos)
111     {
112         for (int i = pos + lefts; i > pos ; i--)
113             crt[i] = crt[i - 1];
114
115         for (int i = input.e + 1; i > input.e - lefts; i--){
116             input.buf[i] = input.buf[i - 1];
117             input.buf_copy[i] = input.buf_copy[i - 1];
118         }
119     }

```

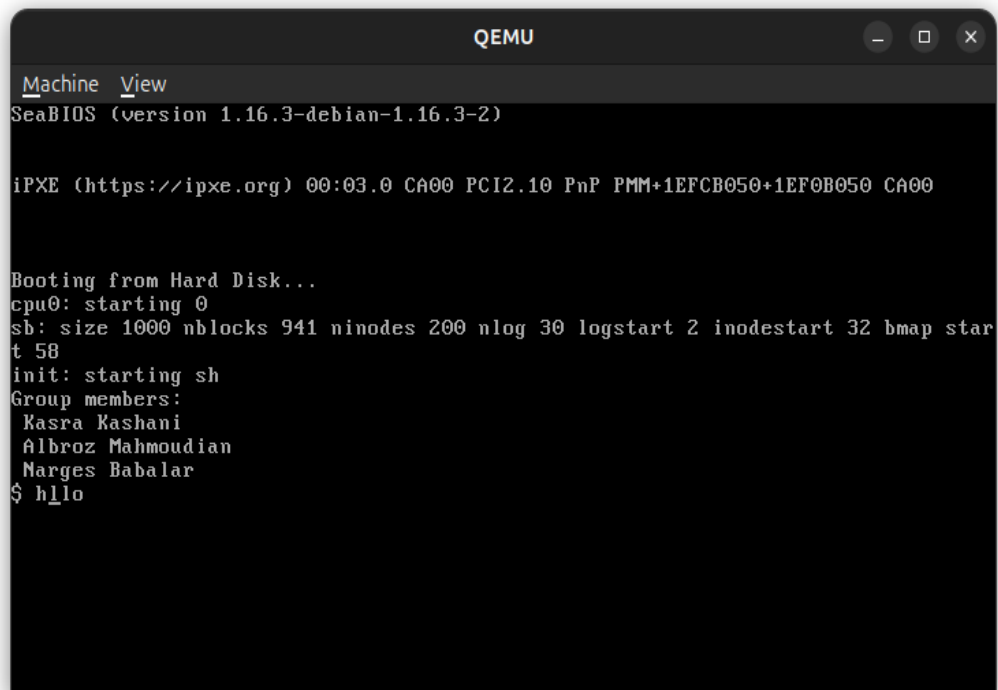
به عنوان نمونه، ابتدا hello را وارد می کنیم. سپس cursor را به چپ می بریم و حرف e را پاک می کنیم و دوباره به راست می آییم و به جای یکی از حرف های a نیز حرف f را وارد می کنیم و سپس Enter می زنیم.



```
QEMU
Machine View
SeaBIOS (version 1.16.3-debian-1.16.3-2)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1EFCB050+1EF0B050 CA00

Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
Group members:
  Kasra Kashani
  Albroz Mahmoudian
  Marges Babalar
$ hello
```



```
QEMU
Machine View
SeaBIOS (version 1.16.3-debian-1.16.3-2)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1EFCB050+1EF0B050 CA00

Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
Group members:
  Kasra Kashani
  Albroz Mahmoudian
  Marges Babalar
$ h_llo
```



```
QEMU
Machine View

SeaBIOS (version 1.16.3-debian-1.16.3-2)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1EFCB050+1EF0B050 CA00

Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group members:
  Kasra Kashani
  Albroz Mahmoudian
  Marges Babalar
$ hlfo
exec: fail
exec hlfo failed
$
```

2. برای ذخیره ی دستورات استفاده شده ی اخیر، از یک structure جدید به نام History استفاده می کنیم تا 10 دستور اخیر، ایندکس دستوری که الان روی آن هستیم، ایندکس آخرین دستور اضافه شده، و تعداد دستورات ذخیره شده (حداکثر 10) را در خود نگه دارد.

```
60 struct History
61 {
62     struct Input hist[10];
63     int index ;
64     int count ;
65     int last ;
66 };
```

برای جا به جایی میان حداکثر 10 دستورت قبلی استفاده شده به کمک کلید های بالا و پایین، باید با توجه به index متوجه شویم که الان روی کدام دستور یا همان hist هستیم و اگر کلید بالا زده شده بود، index را یکی کم کنیم و اگر کلید پایین زده شده بود، index را یکی زیاد

کنیم و hist مربوط به آن index را در کنسول نمایش دهیم. همچنین باید توجه داشت که از اولین یا حداکثر 10 امین دستور قبلی نمی توانیم بالا تر برویم و از آخرین یا جدیدترین دستور نیز نمی توانیم پایین تر بیاییم.

```
128 static void move_through_history(int move)
129 {
130     lefts = 0;
131     for ( int i = input.e ; i > input.w ; i-- ){
132         if (input.buf[i - 1] != '\n'){
133             consputc(BACKSPACE);
134         }
135     }
136     if (move == 0) //Up
137     {
138         history.index --;
139
140         input = history.hist[history.index];
141         input.e -- ;
142         input.buf[input.e] = '\0';
143     }
144     if (move == 1) //Down
145     {
146         if(history.index == -1)
147             history.index ++;
148         history.index ++;
149
150         input = history.hist[history.index];
151         input.e -- ;
152         input.buf[input.e] = '\0';
153     }
154
155     for(int i = input.r; i < input.e; i++){
156         consputc(input.buf[i]);
157     }
158 }
159 }
```

```
614 case UPARROW: // Up
615     if ((history.count > 0) && (history.index > 0))
616         move_through_history(0);
617     break;
618 case DOWNARROW: // Down
619     if ((history.count > 0) && (history.index < history.count - 1))
620         move_through_history(1);
621     break;
```

در ادامه، پس از وارد کردن دستور history باید حداکثر 10 دستور قبلی استفاده شده را روی کنسول نمایش دهیم. بدین منظور ابتدا باید دستور وارد شده بررسی شود که آیا history هست یا نه و اگر بود، به تعداد count، دستور های قبلی را پرینت کنیم. همچنین پس از وارد کردن هر دستوری، باید آن دستور را در محل last در hist ها درج کنیم و در صورتی که count برابر با 10 شده بود، بقیه دستورات داخل hist را یکی به چپ شیفت دهیم و دستور جدید را به hist اضافه کنیم و مقادیر index و last و count را نیز آپدیت کنیم.

```
168 static int is_command_history(){
169     int j = 0;
170     char hist[7] = {'h', 'i', 's', 't', 'o', 'r', 'y'};
171     for(int i = input.r; i < input.e - 1; i++){
172         if(input.buf[i] != hist[j])
173             return 0;
174         j++;
175     }
176     return 1;
177 }
178
179 static void print_history(){
180
181     for(int i = 0; i < history.count; i++){
182         release(&cons.lock);
183         cprintf(&history.hist[i].buf[history.hist[i].r]);
184         acquire(&cons.lock);
185     }
186 }
```

```
121 static void shift_left_previous_histories(){
122     for (int i = 0; i < 9; i++) {
123         history.hist[i] = history.hist[i+1];
124     }
125 }
```

```

664         if(c == '\n' || c == C('D') || input.e == input.r+INPUT_BUF){
665             input.flag_copy = 0;
666
667             if(is_command_history())
668                 print_history();
669
670             if (history.count == 10)
671                 shift_left_previous_histories();
672
673             if(history.count < 10){
674                 history.count ++;
675                 history.last ++;
676                 history.index = history.last + 1;
677             }
678             else
679                 history.index = 10;
680
681             history.hist[history.last] = input;
682
683             input.w = input.e;
684             wakeup(&input.r);
685         }
686     }
687     break;
688 }
689 }

```

به عنوان نمونه، دستورات 1 و 2 و 3 و 4 را وارد میکنیم، سپس با زدن 2 بار کلید بالا دستور 3 را می آوریم، و در انتها نیز با وارد کردن دستور history، این دستور ها را روی کنسول نمایش می دهیم.

The screenshot shows a QEMU window with a terminal interface. The terminal output is as follows:

```

Machine  View
Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group members:
  Kasra Kashani
  Albroz Mahmoudian
  Narges Babalar
$ 1
exec: fail
exec 1 failed
$ 2
exec: fail
exec 2 failed
$ 3
exec: fail
exec 3 failed
$ 4
exec: fail
exec 4 failed
$

```

```
QEMU
Machine View
Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group members:
  Kasra Kashani
  Albroz Mahmoudian
  Marges Babalar
$ 1
exec: fail
exec 1 failed
$ 2
exec: fail
exec 2 failed
$ 3
exec: fail
exec 3 failed
$ 4
exec: fail
exec 4 failed
$ 3
```

```
QEMU
Machine View
Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
Group members:
  Kasra Kashani
  Albroz Mahmoudian
  Marges Babalar
$ 1
exec: fail
exec 1 failed
$ 2
exec: fail
exec 2 failed
$ 3
exec: fail
exec 3 failed
$ 4
exec: fail
exec 4 failed
$
```

3. پس از فشردن کلید Ctrl+S و وارد کردن هر تعداد کاراکتری در هر جایی از کنسول، هنگام فشردن کلید Ctrl+F، باید تمام آن تغییرات جدید اعمال شده روی کنسول و در جایگاه cursor چاپ و paste شوند. بدین منظور، ابتدا در structure Input یکسری تغییرات انجام می دهیم. یک بافر کپی یا buf_copy به عنوان نگه داری 1 ها، یک فلگ flag_copy برای فهمیدن اینکه Ctrl+S زده شده است، و یک فلگ flag_paste برای فهمیدن اینکه تا قبل از اینکه Ctrl+F زده شود باید روی بافر کپی تغییرات یا شیفت انجام دهیم.

```
49 struct Input {
50     char buf[INPUT_BUF];
51     char buf_copy[INPUT_BUF];
52     uint r; // Read index
53     uint w; // Write index
54     uint e; // Edit index
55     int flag_copy;
56     int flag_paste;
57
58 } input = {.flag_copy = 0 , .flag_paste = 0};
```

هدف از استفاده از این بافر کپی، این است که ترتیب کاراکتر های وارد شده روی کنسول حین فشردن این دو کلید حفظ شود و ایندکس آن کاراکتر ها طبق ایندکس بافر اصلی، داخل بافر کپی برابر با 1 شود و در صورت لزوم نیز شیفت بخورند. سپس پس از زدن کلید Ctrl+F، کاراکتر های روی بافر اصلی که ایندکس آن ها داخل بافر کپی برابر 1 است به ترتیب چاپ می شوند.

```
596 case C('S'): // Copy
597     clear_buf_copy();
598     input.flag_copy = 1;
599     input.flag_paste = 1;
600     break;
```

```

621     default:
622         if(c != 0 && input.e-input.r < INPUT_BUF){
623             c = (c == '\r') ? '\n' : c;
624
625             if(c == C('F')){ //Paste
626                 if(input.flag_copy){
627                     temp = input.e;
628                     for(int i = input.r; i < temp; i++){
629                         if(input.buf_copy[i] == '1'){
630                             temp_char = input.buf[i];
631                             consputc(input.buf[i]);
632                             input.buf[(input.e - lefts) % INPUT_BUF] = temp_char;
633                             input.buf_copy[(input.e - lefts) % INPUT_BUF] = '\0';
634                             if((input.e - lefts) - 1 < i){
635                                 //XZ();
636                                 i++;
637                             }
638                             if((input.e - lefts) - 1 < get_last_index_copy()){
639                                 //XZ();
640                                 temp++;
641                             }
642                             input.e ++;
643                         }
644                     }
645                     input.flag_paste = 0;
646                 }
647             }

```

4. در هر زمانی که روی کنسول یک ورودی به الگوی $NON=?$ وارد شود، کل این عبارت از روی کنسول پاک شده و به جایش پاسخ این عملیات چاپ می شود. بدین منظور، یک structure به نام NON تعریف می کنیم که شامل $sign_n1$ و $sign_n2$ برای علامت هر دو عدد، S_n1 و S_n2 برای ایندکس شروع هر دو عدد روی بافر، E_n1 و E_n2 برای ایندکس پایان هر دو عدد روی بافر، و op برای مشخص کردن نوع عملیات (+ یا - یا * یا /) می باشد.

```

39     struct NON{
40         int sign_n1;
41         int sign_n2;
42         uint S_n1;
43         uint E_n1;
44         uint S_n2;
45         uint E_n2;
46         uint op;
47     } non;

```

همچنین باید در هر مرحله ی بررسی صحت این الگو، عدد بودن و نیز اپراتور بودن یک کاراکتر را به ترتیب الگو بررسی کنیم.

```
201 int is_c_in_numbers(char c){
202     char numbers[10] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};
203     for(int i = 0; i < 10; i++){
204         if(c == numbers[i])
205             return 1;
206     }
207     return 0;
208 }
209
210 int is_c_in_operations(char c){
211     char operations[4] = {'+', '-', '*', '/'};
212     for(int i = 0; i < 4; i++){
213         if(c == operations[i])
214             return 1;
215     }
216     return 0;
217 }
```

حال هنگام مشاهده ی کاراکتر '?' باید وجود الگوی NON را بررسی کنیم و آنقدر روی بافر به سمت چپ برویم تا درستی این الگو ثابت شود و سپس عملیات های پاک کردن این الگوی ورودی، محاسبه ی عملیات وارد شده، و جایگزینی پاسخ این عملیات روی کنسول انجام شوند.


```

416 static int check_NON_pattern(){
417     int flag1 = 0, flag2 = 0;
418
419     if(input.buf[input.e - lefts - 1] != '='){
420         return 0;
421     }
422
423     if(!is_c_in_numbers(input.buf[input.e - lefts - 2]))
424         return 0;
425
426     for (int i = input.e - lefts - 3; i >= input.r; i--){
427         if(!is_c_in_numbers(input.buf[i])){
428             if(is_c_in_operations(input.buf[i])){
429                 if(is_c_in_operations(input.buf[i - 1])){
430                     if(input.buf[i] == '-'){
431                         non.op = i - 1;
432                         non.S_n2 = i + 1;
433                         non.E_n2 = input.e - lefts - 2;
434                         flag1 = 1;
435                         non.sign_n2 = 1;
436                         break;
437                     }
438                     else{
439                         return 0;
440                     }
441                 }
442                 else{
443                     non.op = i;
444                     non.S_n2 = i + 1;
445                     non.E_n2 = input.e - lefts - 2;
446                     flag1 = 1;
447                     non.sign_n2 = 0;
448                     break;
449                 }
450             }
451             else
452                 return 0;
453         }
454     }

```

```
378 static void getting_NON_values(){
379     int num1 = 0;
380     int num2 = 0;
381     int digit;
382     float result;
383
384     for(int i = non.S_n1; i <= non.E_n1; i++){
385         digit = ((int)input.buf[i]) - 48;
386         num1 = (num1 * 10) + digit;
387     }
388
389     for(int i = non.S_n2; i <= non.E_n2; i++){
390         digit = ((int)input.buf[i]) - 48;
391         num2 = (num2 * 10) + digit;
392     }
393
394     if(non.sign_n1 == 1)
395         num1 = -num1;
396
397     if(non.sign_n2 == 1)
398         num2 = -num2;
399
400     char operator = input.buf[non.op];
401
402     if(operator == '+')
403         result = (float)num1 + (float)num2;
404     else if(operator == '-')
405         result = (float)num1 - (float)num2;
406     else if(operator == '*')
407         result = (float)num1 * (float)num2;
408     else if(operator == '/')
409         result = (float)num1 / (float)num2;
410
411     replacing_NON(result);
412 }
```

```

329
330 static void replacing_NON(float result){
331     int flag_negative = 0;
332     int i = non.S_n1;
333
334     if(result < 0){
335         flag_negative = 1;
336         result = -result;
337     }
338
339     if(non.sign_n1 == 1)
340         i--;
341
342     lefts--;
343     input.flag_x = 1;
344     update_cursor(1);
345
346     for(int j = non.E_n2 + 2; j >= i; j--)
347         consputc(BACKSPACE);
348
349     for(i ; i <= non.E_n2 + 3; i++)
350         consputc(BACKSPACE);
351     consputc(BACKSPACE);
352
353     if(flag_negative){
354         consputc('-');
355         input.e++;
356         input.buf[(input.e - lefts) % INPUT_BUF] = '-';
357         if(input.flag_paste){
358             input.buf_copy[(input.e - lefts) % INPUT_BUF] = '1';
359         }
360     }
361
362     int integer = (int)result;
363     int fraction = (int)((result - (float)integer) * 10.0);
364
365     print_NON_result(integer);
366
367     if(fraction != 0){
368         consputc('.');
369         input.e++;
370         input.buf[(input.e - lefts) % INPUT_BUF] = '.';
371         if(input.flag_paste){
372             input.buf_copy[(input.e - lefts) % INPUT_BUF] = '1';
373         }
374         print_NON_result(fraction);
375     }
376 }

```

```

298 static void print_NON_result(int result){
299     int num_of_digits = 0;
300     float temp = result;
301
302     if(result == 0)
303         num_of_digits = 1;
304
305     while(temp >= 1){
306         num_of_digits++;
307         temp = temp / 10.0;
308     }
309
310     int digit;
311     char dig;
312     char consoled_result[INPUT_BUF];
313
314     for(int i = num_of_digits - 1; i >= 0; i--){
315         digit = result % 10;
316         dig = (char)(digit + 48);
317         consoled_result[i] = dig;
318         result = result / 10;
319     }
320     for(int i = 0; i < num_of_digits; i++){
321         consputc(consoled_result[i]);
322         input.e++;
323         input.buf[(input.e - lefts) % INPUT_BUF] = consoled_result[i];
324         if(input.flag_paste){
325             input.buf_copy[(input.e - lefts) % INPUT_BUF] = '1';
326         }
327     }
328 }

```

```

674     if(c == '\n' || c == C('D') || input.e == input.r+INPUT_BUF){
675         input.flag_copy = 0;
676
677         if(is_command_history())
678             print_history();

```

• برنامه سطح کاربر

دو فایل encode.c و decode.c را ایجاد کرده و همچنین در Makefile تغییرات لازم را انجام می دهیم. با توجه به محاسبات، این نگاشت با $key = 5$ صورت می گیرد.

```
1  #include "types.h"
2  #include "stat.h"
3  #include "user.h"
4  #include "fcntl.h"
5
6  int main(int argc, char *argv[])
7  {
8      unlink("result.txt");
9      int fd = open("result.txt", O_CREATE | O_RDWR);
10     int key = (90 + 14 + 57) % 26;
11
12     if(fd < 0){
13         printf(2, "strdiff: cannot open/create result.txt\n");
14         exit();
15     }
16
17     for (int i = 1; i < argc; i++){
18         for (int j = 0; j < strlen(argv[i]); j++){
19             if(((int)argv[i][j]) > 64 && (int)argv[i][j] < 91)
20                 argv[i][j] = (char)((((int)argv[i][j] - 65 + key) % 26) + 65;
21             else if(((int)argv[i][j]) > 96 && (int)argv[i][j] < 123)
22                 argv[i][j] = (char)((((int)argv[i][j] - 97 + key) % 26) + 97;
23         }
24         write(fd, argv[i], strlen(argv[i]));
25         write(fd, " ", 1);
26     }
27
28     write(fd, "\n", 1);
29     close(fd);
30     exit();
31 }
```

```
1  #include "types.h"
2  #include "stat.h"
3  #include "user.h"
4  #include "fcntl.h"
5
6  int main(int argc, char *argv[])
7  {
8      unlink("result.txt");
9      int fd = open("result.txt", O_CREATE | O_RDWR);
10     int key = (90 + 14 + 57) % 26;
11
12     if(fd < 0){
13         printf(2, "strdiff: cannot open/create result.txt\n");
14         exit();
15     }
16
17     for (int i = 1; i < argc; i++){
18         for (int j = 0; j < strlen(argv[i]); j++){
19             if(((int)argv[i][j]) > 64 && (int)argv[i][j] < 91)
20                 argv[i][j] = (char)((((int)argv[i][j] - 65 + (26 - key)) % 26) + 65;
21             else if(((int)argv[i][j]) > 96 && (int)argv[i][j] < 123)
22                 argv[i][j] = (char)((((int)argv[i][j] - 97 + (26 - key)) % 26) + 97;
23         }
24         write(fd, argv[i], strlen(argv[i]));
25         write(fd, " ", 1);
26     }
27
28     write(fd, "\n", 1);
29     close(fd);
30     exit();
31 }
```

برای مثال، ورودی زیر را encode می کنیم.

```
QEMU
Machine View

SeaBIOS (version 1.16.3-debian-1.16.3-2)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1EF0B050+1EF0B050 CA00

Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
Group members:
Kasra Kashani
Albroz Mahmoudian
Narges Babalar
$ encode aBcD 1212 EFg H
fGhI 1212 JKl M
$ _
```

● مقدمه ای درباره سیستم عامل و xv6

1. سه وظیفه اصلی سیستم عامل را نام ببرید.

1. یک واسط میان پایین ترین لایه سیستم (سخت افزار) و لایه بالایی (برنامه ها و کاربران)

2. مدیریت منابع

3. مدیریت کاربران و برنامه های کاربردی

2. فایل های اصلی سیستم عامل xv6 در صفحه یک کتاب xv6 لیست شده اند.

به طور مختصر هر گروه را توضیح دهید. نام پوشه اصلی فایل های هسته سیستم عامل، فایل های سرایند و فایل سیستم در سیستم عامل لینوکس چیست؟ در مورد محتویات آن مختصرا توضیح دهید.

Basic header:

این فایل شامل ثبات ها و قرارداد هایی است که این سیستم عامل به آن ها نیاز دارد.

system calls:

کدهایی که ارتباط بین کاربر و kernel سیستم را فراهم میکند و مربوط به سیستم کال است.

string operations:

این فایل شامل عملیاتی روی string ها میباشد.

entering xv6:

این فایل شامل ضروریاتی برای اجرا شدن سیستم عامل xv6 میباشد. در واقع برای شروع کار این فایل نیاز است.

file system:

این فایل شامل ساختار های می باشد برای مدیریت فایل ها فولدر ها و حافظه دیسک که نیاز است.

low-level hardware:

این فایل های سخت افزاری سطح پایین برای اطمینان ارتباط موثر سیستم عامل و اجزای سخت افزاری هستند. در واقع این فایل ها شامل کد هایی هستند که مدیریت device ها، جداسازی رفتار های سخت افزاری و مدیریت وقفه ها را انجام میدهند.

Locks:

این فایل شامل قسمت هایی است که دسترسی به منابع مشترک را هماهنگ و فراهم میکند. در محیط های چند پردازشی بسیار ضروری می باشد.

user-level:

این فایل ها شامل کدهای init می باشد که همواره در حال اجرا میباشد.

processes:

این فایل شامل کدهایی است که برای اجرای فرایندهای کاربر، برنامه‌های چندگانه و... به کار می‌رود.

bootloader:

کدهایی به زبان سی اسمبلی که kernel را از دیسک به حافظه منتقل می‌کند.

pipes:

برای ارتباط بین پردازنده‌ها به کار می‌روند.

link:

این فایل‌ها برای اشتراک‌گذاری نام‌های متعدد برای یک فایل واحد هستند.

• کامپایل سیستم عامل xv6

3. دستور `n -make` را اجرا نمایید. کدام دستور، فایل نهایی هسته را می‌سازد؟

4. در `Make file` متغیرهایی به نامهای `UPROGS` و `ULIB` تعریف شده است. کاربرد آنها چیست؟

در واقع `UPROGS` مخفف کلمه `User program` می‌باشد که همان برنامه‌های کاربر است.

و `ULIB` مخفف `User libraries` می‌باشد که همان کتابخانه کاربر محسوب می‌شود. در واقع شامل تعدادی از کتابخانه‌های زبان C است.

در بخش `UPROGS` ما برنامه‌های کاربر را مشاهده می‌کنیم و یا برنامه‌های خود را اضافه می‌کنیم.

و در بخش `ULIB` کتابخانه‌هایی وجود دارد که سیستم عامل `xv6` برای اجرا از توابع این کتابخانه‌ها بهره گرفته است.

• اجرا بر روی شبیه ساز QEMU

5. دستور `qemu make -n` را اجرا نمایید. دو دیسک به عنوان ورودی به شبیه ساز داده شده است. محتوای آنها چیست؟ (راهنمایی: این دیسک ها حاوی سه خروجی اصلی فرایند بیلد هستند.)

دیسک اول: (Kernel Disk)

این دیسک شامل فایل باینری `kernel` و `xv6` است که حاصل اصلی از فرایند بیلد می باشد، در واقع در این فایل کد مربوط به هسته سیستم عامل قرار دارد که مدیریت حافظه منابع سیستم و سایر بخش های سیستم عامل است.

دیسک دوم: (File System Disk)

این دیسک حاوی فایلی می باشد که در آن برنامه های کاربری و فایل های مورد نیاز برای اجرای سیستم عامل `xv6` قرار دارد.

• مراحل بوت سیستم عامل xv6

8. علت استفاده از دستور `objcopy` در حین اجرای عملیات `make` چیست؟

از این دستور برای تبدیل کردن فایل های اجرایی استفاده میشود. برای مثال با حذف کردن اطلاعات اضافی که برخی از فایل ها دارند و یا تبدیل فایل به فرمت مناسب یعنی فایل خام باینری آن را برای بوت لودر قابل استفاده میکند.

13. کد `bootmain.c` هسته را با شروع از سکتور بعد از سکتور بوت خوانده و در آدرس `0x100000` قرار میدهد. علت انتخاب این آدرس چیست؟

انتخاب این آدرس در واقع به معماری `X86` برمیگردد، برخی دلایل از جمله محفوظ بودن آدرس های پایین تر از 1 مگ برای دستگاه های دیگر یا بایوس یا مثلاً رعایت یک استاندارد قدیمی که سیستم های `X86` از آن پیروی میکنند.

• اجرای هسته xv6

18. علاوه بر صفحه بندی در حد ابتدایی از قطعه بندی به منظور حفاظت هسته استفاده خواهد شد. این عملیات توسط (`seginit`) انجام میگردد. همانطور که ذکر شد، ترجمه قطعه تأثیری بر ترجمه آدرس منطقی نمیگذارد. زیرا تمامی قطعه ها اعم از کد و داده روی یکدیگر میافتند. با این حال برای کد و دادههای سطح کاربر پرچم `USER_SEG` تنظیم شده است. چرا؟ راهنمایی: علت مربوط به ماهیت دستورالعملها و نه آدرس است.

فلگ `SEG USER` به این منظور استفاده می شود که پردازنده های سطح `kernel` و پردازنده های سطح `user` قابل تفکیک باشند.

• اجرای نخستین برنامه سطح کاربر

19. جهت نگهداری اطلاعات مدیریتی برنامه های سطح کاربر ساختاری تحت عنوان (`proc struct` خط ۲۳۳۶) ارائه شده است. اجزای آن را توضیح داده و ساختار معادل آن در سیستمعامل لینوکس را بیابید.

اجزای ساختار `proc struct` :

Pgdir : پوینتر متعلق به `page table`. از این بخش برای مدیریت حافظه مجازی و تبدیل آدرس های مجازی به فیزیکی استفاده می شود.

Name : نام پردازنده مورد استفاده

Parent : سازنده پردازنده. این بخش به سیستم عامل کمک می کند تا فرایندها را در قالب سلسله مراتبی سازمان دهی کند.

Pid : عدد اختصاص داده شده به این پردازنده. برای تمایز دادن فرایندها استفاده میشود.

State : وضعیت پردازنده. نشان میدهد فرایند در چه مرحله ای قرار دارد.

SZ : سائز حافظه پردازنده. چه مقدار حافظه به فرایند اختصاص داده شده.

Kstack : اشاره گری به استک kernel که در پردازنده وجود دارد.

Killed: اگر صفر نباشد به معنای kill شدن پردازنده های پردازنده است.

Chan: در صورت صفر بودن به این معناست که پردازنده موقتاً غیر فعال است.

Context : برای switching context نگهداری شده است.

Ofile : آرایه ای از اشاره گر ها به فایل های باز شده

Cwd : نمایانگر پوشه کنونی

Tf : چارچوب interrupt trap برای فراخوانی سیستمی فعلی

در سیستم عامل لینوکس، معادل ساختار struct proc در xv6، ساختار task_struct است:

اجزای اصلی task_struct :

1. **state**: وضعیت فرآیند.

2. **parent**: اشاره گر به فرآیند والد.

3. **comm**: نام فرآیند

4. **stack**: اشاره گر به پشته کرنل فرآیند.

5. **files**: اشاره گر به ساختار فایل های باز شده توسط فرآیند.

6. **mm**: اشاره گر به فضای آدرس حافظه مجازی فرآیند.

7. **Pid**: شناسه فرآیند.

23. کدام بخش از آماده سازی سیستم، بین تمامی هسته های پردازنده مشترک و کدام بخش اختصاصی است؟

در سیستم عامل های چند هسته ای چند هسته ای برخی بخش ها و منابع سیستم به شکل مشترک بین تمام هسته های پردازنده مورد استفاده قرار میگیرد اما در برخی مواقع برای یک هسته خاص مورد استفاده قرار میگیرد.

برای بخش اختصاصی: در واقع هر هسته پردازنده یک task state segment دارد که مجزا از یکدیگرند.

انها مسئول ذخیره وضعیت فعلی وظیفه را بر عهده دارند. به همین دلیل هر هسته task state segment به خصوص خود را دارند تا بتوانند به طور اختصاصی فرایند هارا مدیریت کنند.

برای بخش مشترک: در واقع page table ها یک ابزار برای به اشتراک گذاشتن منابع و حافظه های مشترک میباشند و این برای دسترسی همه هسته ها به یک فضای ادرس یکسان می باشد .

• روند اجرای GDB

در ابتدا یک breakpoint در ابتدای تابع consoleintr قرار می دهیم.

1- برای مشاهده breakpoint ها از دستور info breakpoints استفاده می کنیم.

2- برای حذف یک breakpoint از دستور del a استفاده می کنیم به طوری که a همان شماره ی breakpoint ای است که می خواهیم حذفش کنیم و در info قابل مشاهده بود.

```

Kasra@Kasra:~$ cd Desktop/Lab1/xv6-public-master/
Kasra@Kasra:~/Desktop/Lab1/xv6-public-master$ gdb kernel
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel...
warning: File "/home/Kasra/Desktop/Lab1/xv6-public-master/.gdbinit" auto-loading has been declined by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
    add-auto-load-safe-path /home/Kasra/Desktop/Lab1/xv6-public-master/.gdbinit
line to your configuration file "/home/Kasra/.config/gdb/gdbinit".
To completely disable this security protection add
    set auto-load safe-path /
line to your configuration file "/home/Kasra/.config/gdb/gdbinit".
--Type <RET> for more, q to quit, c to continue without paging--c
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
    info "(gdb)Auto-loading safe path"
(gdb) b consoleintr
Breakpoint 1 at 0x80100c20: file console.c, line 561.
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x80100c20 in consoleintr at console.c:561
(gdb) del 1
(gdb) info breakpoints
No breakpoints, watchpoints, tracepoints, or catchpoints.

```

• کنترل روند اجرا و دسترسی به حالت سیستم

3- دستور bt یا همان backtrace، به ازای هر frame stack یک خط را چاپ می کند.

4- دستور x برای چک کردن خانه های حافظه و دستور print برای نمایش یک عبارت در gdb استفاده می شود. همچنین این عبارت برای متغیر های local استفاده می شوند اما در دستور x با خانه های حافظه کار داریم. برای چاپ کردن محتوای یک ثبات خاص، پشت نام آن ثبات علامت \$ را گذاشته و آن را print می کنیم.

5- برای نمایش وضعیت ثبات ها از دستور info registers و برای نمایش متغیر های محلی از دستور info locals استفاده می کنیم.


```

47
48 struct Input {
49     char buf[INPUT_BUF];
50     char buf_copy[INPUT_BUF];
51     uint r; // Read index
52     uint w; // Write index
53     uint e; // Edit index
54     int flag_copy;
55     int flag_paste;
56

```

Input.e دارای ایندکس آخرین کاراکتر در بافر است که با backspace زدن باید -- و با وارد شدن هر کاراکتری باید ++ شود.

Input.w دارای ایندکسی از بافر است که بافر تا آن ایندکس را تا الان در کنسول نوشته است.

Input.r دارای ایندکس اولین خانه از بافر است که بافر از آن ایندکس به بعد هنوز در کنسول نوشته نشده است که با Enter زدن باید بافر از ایندکس r تا e در کنسول نوشته شود پس ایندکس w نیز به ایندکس e اپدیت می شود که یعنی بافر تا ایندکس e در کنسول نوشته شده است.

buf نیز ارایه ای از کاراکتر ها است که ورودی کاربر در آن ذخیره می شود و پس از هر بار enter زدن در کنسول برای کاربر نوشته می شود.

• اشکال زدایی در سطح کد اسمبلی

-7 Layout asm یک کد اسمبلی است که مربوط به نقطه ی اجرای فعلی است. اما layout src سورس کد است که مربوط به زبان برنامه نویسی فایل می باشد.

8- برای جا به جایی میان توابع زنجیره فراخوانی جاری از دستورات
backtrace و up و down و ... می توان استفاده کرد.