



دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

تمرین سری چهارم - عملی

پارسا ملکیان - ۰۹۱۷۱۰۷۵

فاطمه شفیعی - ۰۹۱۱۰۸۸۷

سیستم‌های عامل

دکتر اسدی

سوال ۱

در این تمرین نخست ساختار داده‌ها را همانطور که گفته شده بود پیاده سازی کردیم و تعریف ساختار هر ترد و کانتکست آن به شکل زیر در فایل uthread.h قرار گرفتند:

```
1 struct context {
2     uint64 ra;
3     uint64 sp;
4     uint64 s0; uint64 s1; ... uint64 s11;
5 };
6
7 struct thread {
8     char stack[STACK_SIZE];    //
9     int state;                // : FREE, RUNNING, RUNNABLE
10    struct context context;   //
11};
```

- حافظه‌ی اختصاصی ترد برای نگهداری داده‌ها و فراخوانی‌ها.

- وضعیت ترد.

- مقادیر ثبات‌ها که توسط uthread_switch استفاده می‌شود.

سپس به پیاده سازی توابع مورد نیاز پرداختیم که توضیح مختصی از عملکرد آنها در زیر آمده است:

thread_init

این تابع اولین ریسه (thread 0) را که ریسه‌ی main است، در حالت RUNNING قرار می‌دهد.

```
1 void
2 thread_init(void)
3 {
4     current_thread = &all_thread[0];
5     current_thread->state = RUNNING;
6 }
```

thread_create

با استفاده از این تابع، ریسه‌های جدید ایجاد شدند:

- بررسی می‌کند که ریسه‌ی آزاد موجود است یا خیر.
- کانتکست ریسه را صفر می‌کند و آدرس استک را در sp تنظیم می‌کند.
- آدرس تابع ریسه را در ra قرار می‌دهد.
- وضعیت ریسه را به RUNNABLE تغییر می‌دهد.

```

1 void
2 thread_create(void (*func)())
{
3     int tid;
4
5     for (tid = 0; tid < MAX_THREAD; tid++) {
6         if (all_thread[tid].state == FREE) break;
7     }
8
9     if (tid == MAX_THREAD) {
10        printf("create_thread: no free thread\n");
11        return;
12    }
13
14    struct thread *t = &all_thread[tid];
15
16    // allocate space to the context
17    int sz = sizeof(struct context);
18    memset(&t->context, 0, sz);
19
20    // allocate space to the stack
21    uint64 sp = (uint64)(t->stack + STACK_SIZE);
22    t->context.sp = sp;
23
24    // runnable
25    t->context.ra = (uint64)func;
26    t->state = RUNNABLE;
27}

```

thread_schedule

وظیفه زمانبندی ریسنهها بر عهده این تابع است:

- از ریسنهی جاری شروع کرده و در بین ریسنهای ۱ تا ۳ دنبال ریسنهی RUNNABLE می‌گردد.
- اگر ریسنهی قابل اجرا یافت شد، وضعیت ریسنه جاری را به RUNNABLE تغییر می‌دهد و ریسنهی بعدی را RUNNING می‌کند.
- سپس uthread_switch برای سوئیچ کانتکست فراخوانی می‌شود.

```

1 void
2 thread_schedule(void)
3 {
4     int i, next = -1;
5     int indx = current_thread - all_thread;
6     // find a runnable thread
7     for (i = indx; i < indx + MAX_THREAD ; i++) {
8         if ((all_thread[i % (MAX_THREAD - 1) + 1].state == RUNNABLE)) {
9             next = i % (MAX_THREAD - 1) + 1;
10            break;
11        }

```

```

12 }
13
14 // no runnable threads, nothing to schedule right now
15 if (next == -1) {
16     printf("thread_schedule: no runnable threads\n");
17     all_thread->state = RUNNING;
18     thread_switch(&current_thread->context, &all_thread->context);
19     current_thread = all_thread;
20     return;
21 }
22
23 struct thread *told = current_thread;
24 struct thread *tnext = &all_thread[next];
25
26 told->state = (told->state == RUNNING) ? RUNNABLE : told->state;
27 tnext->state = RUNNING;
28 current_thread = tnext;
29 thread_switch(&told->context, &tnext->context);

```

thread_yield

این تابع باعث می‌شود ریسنه‌ی جاری خود را به حالت آماده اجرا درآورد و تابع زمان‌بندی را صدا بزند.

```

1 void
2 thread_yield(void)
3 {
4     if (current_thread->state == RUNNING) current_thread->state = RUNNABLE;
5     thread_schedule();
6 }

```

thread_c , thread_b , thread_a

- هر ریسنه ابتدا با متغیرهای c_started b_started a_started شروع خود را علامت‌گذاری می‌کند و تا وقتی همه ریسنه‌ها شروع نشده‌اند، صبر می‌کند.

- سپس حلقه‌ای از ۹۹ تا ۰ اجرا می‌شود و در هر مرحله، خروجی شمارنده ریسنه چاپ می‌شود.

- بعد از اتمام حلقه، ریسنه به حالت FREE تغییر می‌کند و پیام اتمام چاپ می‌شود.

```

1 void
2 thread_a(void)
3 {
4     a_started = 1;
5     printf("thread_a started\n");
6     while (!(a_started && b_started && c_started)) {
7         thread_yield();
8     }
9
10    for ( ; a_n < 100; a_n++) {
11        printf("thread_a %d\n", a_n);

```

```

12     // let others run
13     thread_yield();
14 }
15 current_thread->state = FREE;
16 printf("thread_a: exit after 100\n");
17 thread_schedule();
18 }
```

در نهایت نتیجه اجرای دستور `uthread` در سیستم عامل داده شده به شکل زیر شد که مطابق خواسته سوال است:

```
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ uthread
thread_a started
thread_b started
thread_c started
thread_c 0
thread_a 0
thread_b 0
thread_c 1
thread_a 1
thread_b 1
thread_c 2
thread_a 2
thread_b 2
thread_c 3
```

```

thread_c 95
thread_a 95
thread_b 95
thread_c 96
thread_a 96
thread_b 96
thread_c 97
thread_a 97
thread_b 97
thread_c 98
thread_a 98
thread_b 98
thread_c 99
thread_a 99
thread_b 99
thread_c: exit after 100
thread_a: exit after 100
thread_b: exit after 100
thread_schedule: no runnable threads
```

سوال ۲

بخش الف و ب: شروع کار و کشف مشکل اول اول کد رو از git گرفتم و به tag t2 رفتم. بعد کانتینر رو با دستور `sh -containertest -name= run ./zocker` اجرا کردم. وقتی داخل کانتینر دستور `ps` رو زدم، دیدم که تمام پردازهای سیستم host رو نشان میده. مشکل: متوجه شدم که دستور `ps` اطلاعاتش رو از /proc می خونه و چون /proc از سیستم اصلی mount شده بود، همه پردازهها رو نشون میداد.

```
malekian-shafiee@Malekian-Shafiee: ~/zocker
-- More --
malekian-shafiee@Malekian-Shafiee:~/home/malekian-shafiee# cd zocker/
malekian-shafiee@Malekian-Shafiee:~/home/malekian-shafiee/zocker# ls
main.c Makefile zocker
malekian-shafiee@Malekian-Shafiee:~/home/malekian-shafiee/zocker# ./zocker run --name test-container 'sh'
Running child with pid: 1
# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START  TIME COMMAND
root         1  2.6  0.1 168068 12416 ?        Ss  04:42  0:10 /sbin/init
root         2  0.0  0.0     0     0 ?        S  04:42  0:00 [kthread]
root         3  0.0  0.0     0     0 ?        I< 04:42  0:00 [rcu_gp]
root         4  0.0  0.0     0     0 ?        I< 04:42  0:00 [rcu_par_gp]
root         5  0.0  0.0     0     0 ?        I< 04:42  0:00 [slub_flushwq]
root         6  0.0  0.0     0     0 ?        I< 04:42  0:00 [netns]
root         8  0.0  0.0     0     0 ?        I< 04:42  0:00 [kworker/0:0H-events_highpri]
root        10  0.0  0.0     0     0 ?        I< 04:42  0:00 [mm_percpu_wq]
root        11  0.0  0.0     0     0 ?        I  04:42  0:00 [rcu_tasks_kthread]
root        12  0.0  0.0     0     0 ?        I  04:42  0:00 [rcu_tasks_rude_kthread]
root        13  0.0  0.0     0     0 ?        I  04:42  0:00 [rcu_tasks_trace_kthread]
root        14  0.0  0.0     0     0 ?        S  04:42  0:00 [ksoftirqd/0]
root        15  0.3  0.0     0     0 ?        I  04:42  0:01 [rcu_prempt]
root        16  0.0  0.0     0     0 ?        S  04:42  0:00 [migration/0]
root        17  0.0  0.0     0     0 ?        I  04:42  0:00 [kworker/0:1-events_power_efficient]
root        18  0.0  0.0     0     0 ?        S  04:42  0:00 [cpuhp/0]
root        19  0.0  0.0     0     0 ?        S  04:42  0:00 [cpuhp/1]
root        20  0.2  0.0     0     0 ?        S  04:42  0:00 [migration/1]
root        21  0.0  0.0     0     0 ?        S  04:42  0:00 [ksoftirqd/1]
root        23  0.0  0.0     0     0 ?        I< 04:42  0:00 [kworker/1:0H-events_highpri]
root        24  0.0  0.0     0     0 ?        S  04:42  0:00 [cpuhp/2]
root        25  0.2  0.0     0     0 ?        S  04:42  0:01 [migration/2]
root        26  0.0  0.0     0     0 ?        S  04:42  0:00 [ksoftirqd/2]
root        28  0.0  0.0     0     0 ?        I< 04:42  0:00 [kworker/2:0H-events_highpri]
root        29  0.0  0.0     0     0 ?        S  04:42  0:00 [cpuhp/3]
root        30  0.2  0.0     0     0 ?        S  04:42  0:01 [migration/3]
root        31  0.0  0.0     0     0 ?        S  04:42  0:00 [ksoftirqd/3]
root        32  0.0  0.0     0     0 ?        I  04:42  0:00 [kworker/3:0-rcu_gp]
root        33  0.0  0.0     0     0 ?        I< 04:42  0:00 [kworker/3:0H-events_highpri]
root        34  0.0  0.0     0     0 ?        S  04:42  0:00 [cpuhp/4]
root        35  0.3  0.0     0     0 ?        S  04:42  0:01 [migration/4]
root        36  0.0  0.0     0     0 ?        S  04:42  0:00 [ksoftirqd/4]
root        38  0.0  0.0     0     0 ?        I< 04:42  0:00 [kworker/4:0H-events_highpri]
root        39  0.0  0.0     0     0 ?        S  04:42  0:00 [cpuhp/5]
root        40  0.3  0.0     0     0 ?        S  04:42  0:01 [migration/5]
root        41  0.0  0.0     0     0 ?        S  04:42  0:00 [ksoftirqd/5]
root        43  0.0  0.0     0     0 ?        I< 04:42  0:00 [kworker/5:0H-events_highpri]
root        44  0.0  0.0     0     0 ?        S  04:42  0:00 [cpuhp/6]
root        45  0.3  0.0     0     0 ?        S  04:42  0:01 [migration/6]
root        46  0.0  0.0     0     0 ?        S  04:42  0:00 [ksoftirqd/6]
root        48  0.0  0.0     0     0 ?        I< 04:42  0:00 [kworker/6:0H-events_highpri]
root        49  0.0  0.0     0     0 ?        S  04:42  0:00 [cpuhp/7]
root        50  0.3  0.0     0     0 ?        S  04:42  0:01 [migration/7]
```

بخش ج: حل مشکل ps برای حل این مشکل، باید یک proc namespace خودمون می کردیم. کدی که نوشتیم:

```
1 if (mount("proc", "/proc", "proc", 0, NULL) == -1) {
2     perror("mount /proc failed");
3     exit(1);
4 }
```

```

GNU nano 7.2                                malekian-shafiee@Malekian-Shafiee: ~/zocker
if (cfg.subcommand == NONE) {
    fprintf(stderr, "[ERR] Missing subcommand (run|exec)\n");
    return 1;
}

if (strcmp(cfg.name, "") == 0) {
    strncpy(cfg.name, "bib", sizeof(cfg.name)-1);
}

if (strcmp(cfg.command, "") == 0) {
    fprintf(stderr, "[ERR] Missing command (e.g. 'sleep 1000')\n");
    return 1;
}
return 0;
}

int run_container(struct config cfg) {
    pid_t pid;

    if (unshare(CLONE_NEWPID) != 0) {
        fprintf(stderr, "[ERR] Failed to unshare(2).");
        return 1;
    }

    pid = fork();
    if (pid < 0) {
        return 1;
    }
    if (pid == 0) {
        if (mount("proc", "/proc", "proc", 0, NULL) == -1) {
            perror("mount /proc failed");
            return -1;
        }

        printf("Running child with pid: %d\n", getpid());
        execl("/bin/sh", "sh", "-c", cfg.command, NULL);
    } else {
        sleep(2);
        waitpid(pid, NULL, 0);
        printf("[Parent] Stopping...\n");
    }
    return 0;
}

int main(int argc, char **argv) {
    struct config cfg = {
^C Help          ^O Write Out     ^W Where Is      ^K Cut           ^T Execute       ^C Location      M-U Undo       M-A Set Mark   M-[ To Bracket M-Q Where Was M-Q Previous
^X Exit         ^R Read File     ^Y Replace      ^U Paste         ^J Justify      ^P Go To Line    M-E Redo       M-G Copy      ^Q Where Was   M-W Next

```

حالا ps فقط پردازه‌های داخل کانتینر رونشون میداد

```

malekian-shafiee@Malekian-Shafiee: ~/zocker

Debian++ 1151 0.2 0.1 463468 12592 tty1 Sl+ 04:43 0:01 /usr/libexec/gsd-sharing
Debian++ 1153 0.6 0.2 338912 26484 tty1 Sl+ 04:43 0:03 /usr/libexec/gsd-wacom
Debian++ 1156 0.6 0.2 340024 24876 tty1 Sl+ 04:43 0:03 /usr/libexec/gsd-color
Debian++ 1158 0.5 0.1 338352 24084 tty1 Sl+ 04:43 0:03 /usr/libexec/gsd-keyboard
Debian++ 1160 0.2 0.1 247224 13140 tty1 Sl+ 04:43 0:01 /usr/libexec/gsd-print-notifications
Debian++ 1164 0.2 0.0 454772 8348 tty1 Sl+ 04:43 0:01 /usr/libexec/gsd-rfkill
Debian++ 1166 0.2 0.0 312052 11648 tty1 Sl+ 04:43 0:01 /usr/libexec/gsd-smartcard
Debian++ 1168 0.2 0.1 355456 13412 tty1 Sl+ 04:43 0:01 /usr/libexec/gsd-datetime
Debian++ 1169 0.7 0.2 520856 29032 tty1 Sl+ 04:43 0:04 /usr/libexec/gsd-media-keys
Debian++ 1171 0.1 0.0 233188 8140 tty1 Sl+ 04:43 0:08 /usr/libexec/gsd-screensaver-proxy
Debian++ 1172 0.1 0.0 319576 11248 tty1 Sl+ 04:43 0:01 /usr/libexec/gsd-sound
Debian++ 1175 0.1 0.0 307560 10536 tty1 Sl+ 04:43 0:00 /usr/libexec/gsd-a11y-settings
Debian++ 1176 0.2 0.0 309128 8980 tty1 Sl+ 04:43 0:01 /usr/libexec/gsd-housekeeping
Debian++ 1179 0.7 0.2 450316 27776 tty1 Sl+ 04:43 0:04 /usr/libexec/gsd-power
Debian++ 1255 0.1 0.1 341920 15864 tty1 Sl+ 04:43 0:01 /usr/libexec/gsd-printer
Debian++ 1343 0.4 0.2 2923776 26964 tty1 Sl+ 04:43 0:02 /usr/bin/gjs /usr/share/gnome-shell/org.gnome.ScreenSaver
root 1357 0.2 0.0 17888 11004 ?
Ss 04:43 0:01 sshd: malekian-shafiee [priv]
Debian++ 1365 0.2 0.1 312452 16168 tty1 Sl 04:43 0:01 ibus-daemon --panel disable -r --xim
Debian++ 1375 0.0 0.0 234200 11124 tty1 Sl 04:43 0:00 /usr/libexec/ibus-dconf
Debian++ 1378 1.6 0.2 344288 32116 tty1 Sl 04:43 0:09 /usr/libexec/ibus-extension-gtk3
Debian++ 1380 0.2 0.6 396908 74116 tty1 Sl 04:43 0:01 /usr/libexec/ibus-x11 --kill-daemon
Debian++ 1383 0.0 0.0 234168 9212 tty1 Sl+ 04:43 0:00 /usr/libexec/ibus-portal
root 1384 0.0 0.0 0 0 ?
I 04:43 0:00 [kworker/10:3-mm_percpu_wq]
malekia+ 1412 0.2 0.0 18148 6868 ?
S 04:43 0:01 sshd: malekian-shafiee@pts/1
Debian++ 1414 0.0 0.0 160368 8996 tty1 Sl 04:43 0:00 /usr/libexec/ibus-engine-simple
malekia+ 1416 0.1 0.0 8376 5108 pts/1 Ss 04:43 0:00 -bash
root 1509 0.0 0.0 0 0 ?
I 04:47 0:00 [kworker/11:0-ata_sff]
root 1515 0.1 0.0 9076 3776 pts/1 S 04:47 0:00 su -
root 1519 0.2 0.0 9652 5864 pts/1 S 04:47 0:00 -bash
root 1528 0.0 0.0 0 0 ?
I 04:48 0:00 [kworker/7:0]
root 1562 0.0 0.0 0 0 ?
I 04:50 0:00 [kworker/u24:0-events_unbound]
root 1581 0.4 0.0 2336 884 pts/1 S 04:52 0:00 ./zocker run --name test-container sh
root 1582 1.0 0.0 2584 872 pts/1 S 04:52 0:00 sh -c sh
root 1583 0.7 0.0 2584 896 pts/1 S 04:52 0:00 sh
root 1584 283 0.0 11224 4648 pts/1 R+ 04:52 0:00 ps aux
#
# exit
sh: 2: Cannot set tty process group (No such process)
[Parent] Stopping...
root@Malekian-Shafiee:/home/malekian-shafiee/zocker# nano main.c
root@Malekian-Shafiee:/home/malekian-shafiee/zocker# make
gcc -Wall -Wextra -std=c99 -o zocker main.c
sudo setcap cap_sys_admin+ep zocker
root@Malekian-Shafiee:/home/malekian-shafiee/zocker# ./zocker run --name test-container 'sh'
Running child with pid: 1
# ps aux
USER      PID %CPU %MEM    VSSZ   RSS TTY      STAT START   TIME COMMAND
root        1  1.5  0.0  2584  888 pts/1    S  04:54  0:00 sh -c sh
root        2  0.9  0.0  2584  896 pts/1    S  04:54  0:00 sh
root        3 100  0.0 11092 4380 pts/1   R+ 04:54  0:00 ps aux
#

```

بخش ۴: مشکل جدید با df و قطی mount رو درست کردم، دیدم که دستورات دیگه مثل df خطای میدن. حتی بعد از خروج از کانتینر، سیستم host مشکل پیدا کرده بود.

فهمیدم که به خاطر mount propagation فرض، تغییرات به parent namespace mount بود. به طور پیشفرض، وقتی ما /proc/unmount را روی host کردیم، روی MS_SHARED (MS_SHARED) منتشر میشه.

بخش ۵: جلوگیری از انتشار mount برای حل این مشکل، باید private mount propagation رو به تبدیل میکردیم: همچنین باید هم NEWNS و هم NEWPID رو unshare نماییم.

```

1
2 if (unshare(CLONE_NEWPID | CLONE_NEWNS) != 0) {
3     fprintf(stderr, "[ERR] Failed to unshare(2).");
4     return 1;
5 }
6
7 if (mount(NULL, "/", NULL, MS_PRIVATE | MS_REC, NULL) == -1) {
8     perror("make-rprivate failed");
9     return -1;

```

بخش دوم بلا فاصله بعد از آنشیر نوشته شده
پرچم‌ها:

روی همه زیرشاخه‌ها اعمال بشه MS_REC -
تغییرات فقط تو namespace خودمون بمونه MS_PRIVATE -

حالا سیستم host دیگه تحت تاثیر قرار نمی‌گرفت.

```
malekian-shafiee@Malekian-Shafiee: ~/zocker
}
    strcpy(cfg.command, argv[i], sizeof(cfg.command) - 1);
    i++;
}
if (validate_config(cfg) != 0) {
    return 1;
}

switch (cfg.subcommand) {
    case RUN:
        if (run_container(cfg) != 0) {
            fprintf(stderr, "[ERR] Running container failed due to some internal errors.\n");
            return 1;
        }
        break;
    case EXEC:
        printf("EXEC subcommand have not implemented yet...\n");
        break;
    case NONE:
    default:
        break;
}
return 0;
}
root@Malekian-Shafiee:/home/malekian-shafiee/zocker# nano main.c
root@Malekian-Shafiee:/home/malekian-shafiee/zocker# make
gcc -Wall -Wextra -std=c99 -o zocker main.c
sudo setcap cap_sys_admin+ep zocker
root@Malekian-Shafiee:/home/malekian-shafiee/zocker# ./zocker run --name test-container 'sh'
Running child with pid: 1
# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START  TIME COMMAND
root       1  0.3  0.0  2584   892 pts/1    S   05:10  0:00 sh -c sh
root       2  0.6  0.0  2584   908 pts/1    S   05:10  0:00 sh
root       3  116  0.0 11092  4372 pts/1   R+  05:10  0:00 ps aux
# exit
sh: 2: Cannot set tty process group (No such process)
[Parent] Stopping...
root@Malekian-Shafiee:/home/malekian-shafiee/zocker# df
Filesystem 1K-blocks Used Available Use% Mounted on
udev       6094312   0  6094312   0% /dev
tmpfs      1224580 1084 1223496  1% /run
/dev/sda1  29801344 11089496 17172676 40% /
tmpfs      6122880   0  6122880   0% /dev/shm
tmpfs       5120     0    5120   0% /run/lock
tmpfs      1224576   60 1224516  1% /run/user/111
tmpfs      1224576    52 1224524  1% /run/user/1000
root@Malekian-Shafiee:/home/malekian-shafiee/zocker#
```

بخش و: اضافه کردن UTS Namespace برای اینکه hostname هر کانتینر جدا باشه، رو به unshare اضافه کردم:

```
1 if (unshare(CLONE_NEWPID | CLONE_NEWNS | CLONE_NEWUTS) == -1) {
2     perror("unshare failed");
3     return -1;
```

4

بخش ز: تنظیم hostname با استفاده از sethostname اسما کانتینر را به عنوان hostname تنظیم کرد:

```

1 if (sethostname(container_name, strlen(container_name)) == -1) {
2     perror("sethostname failed");
3     exit(1);
4 }
```

وقتی داخل کانتینر رو میزدم، اسم کانتینر رو نشون میداد نه hostname سیستم اصلی.

```

malekian-shafiee@Malekian-Shafiee: ~/zocker

case NONE:
default:
    break;
}
return 0;
}
root@Malekian-Shafiee:/home/malekian-shafiee/zocker# nano main.c
root@Malekian-Shafiee:/home/malekian-shafiee/zocker# make
gcc -Wall -Wextra -std=c99 -o zocker main.c
sudo setcap cap_sys_admin+ep zocker
root@Malekian-Shafiee:/home/malekian-shafiee/zocker# ./zocker run --name test-container 'sh'
Running child with pid: 1
# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START  TIME COMMAND
root         1  0.3  0.0  2584   892 pts/1    S  05:10  0:00 sh -c sh
root         2  0.6  0.0  2584   908 pts/1    S  05:10  0:00 sh
root         3 116  0.0 11092  4372 pts/1    R+  05:10  0:00 ps aux
# exit
sh: 2: Cannot set tty process group (No such process)
[Parent] Stopping...
root@Malekian-Shafiee:/home/malekian-shafiee/zocker# df
Filesystem 1K-blocks Used Available Use% Mounted on
udev       6094312   0  6094312  0% /dev
tmpfs      1224580 1084 1223496  1% /run
/dev/sda1  29801344 11089496 17172676 40% /
tmpfs      6122880   0  6122880  0% /dev/shm
tmpfs       5120     0   5120  0% /run/lock
tmpfs      1224576   60  1224516  1% /run/user/111
tmpfs      1224576   52  1224524  1% /run/user/1000
root@Malekian-Shafiee:/home/malekian-shafiee/zocker# sudo umount /proc
root@Malekian-Shafiee:/home/malekian-shafiee/zocker# nano main.c
root@Malekian-Shafiee:/home/malekian-shafiee/zocker# make
gcc -Wall -Wextra -std=c99 -o zocker main.c
main.c: In function 'run_container':
main.c:58:26: error: 'container_name' undeclared (first use in this function)
  58 |         if (sethostname(container_name, strlen(container_name)) == -1) {
      |             ^
main.c:58:26: note: each undeclared identifier is reported only once for each function it appears in
make: *** [Makefile:7: zocker] Error 1
root@Malekian-Shafiee:/home/malekian-shafiee/zocker# nano main.c
root@Malekian-Shafiee:/home/malekian-shafiee/zocker# make
gcc -Wall -Wextra -std=c99 -o zocker main.c
sudo setcap cap_sys_admin+ep zocker
root@Malekian-Shafiee:/home/malekian-shafiee/zocker# ./zocker run --name malekian-shafiee 'sh'
Running child with pid: 1
# whoami
root
# hostname
malekian-shafiee
# 
```

بخش ح: Namespace Time و User در آخر namespace های کاربر و زمان رو هم اضافه کرد:

```

1 if (unshare(CLONE_NEWPID | CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWUSER) == -1) {
2     perror("unshare failed");
3     return -1;
```

کد کامل بعد از تغییرات داخل فایل های ارسالی در پوشه `q2` قرار دارد.

سوال ۳

در این تمرین یک سیستم مدیریت فایل ساده پیاده سازی کردم که روی یک فایل معمولی کارمی کند و اندازه ثابت دارد.

ساختار کلی

سیستم از یک فایل به نام `filesys.db` استفاده می‌کند که اندازه آن ثابت است و در واقع نقش یک دیسک مجازی را دارد. این فایل به بلاک‌های ۵۱۲ بایتی تقسیم شده است.

در ابتدای فایل یک `SuperBlock` قرار دارد که حاوی اطلاعات کلی سیستم است شامل `magic number` برای اعتبارسنجی، نسخه، تعداد فایل‌ها، تعداد بلاک‌های کل و استفاده شده. بعد از آن یک آرایه ثابت از `FileEntry`‌ها وجود دارد که هر کدام اطلاعات یک فایل رانگه می‌دارند.

هر `FileEntry` شامل نام فایل، سایز، بلاک شروع، تعداد بلاک‌ها، دسترسی‌ها و زمان ساخت و ویرایش است. همچنین یک فیلد `is_used` دارد که مشخص می‌کند این ورودی استفاده شده یا خالی است.

مدیریت فضای خالی

برای مدیریت فضای خالی از یک لیست پیوندی استفاده کردم که هر نод آن نشان دهنده یک بازه پیوسته از بلاک‌های خالی است. وقتی فایل جدیدی ساخته می‌شود از ابتدای این لیست بلاک‌های لازم اختصاص داده می‌شود و وقتی فایلی پاک می‌شود بلاک‌های آن به لیست اضافه می‌شوند.

تابع `allocate_blocks` روی لیست پیوندی حرکت می‌کند تا اولین بلاک خالی با فضای کافی را پیدا کند. اگر پیدا کرد آن قسمت را از لیست حذف یا کوچک می‌کند و آدرس بلاک شروع را برمی‌گرداند.

تابع `free_blocks` یک نود جدید با آدرس و تعداد بلاک‌های آزاد شده می‌سازد و آن را به ابتدای لیست اضافه می‌کند.

پایداری داده‌ها

برای اینکه داده‌ها بعد از بستن برنامه حفظ شوند دو تابع `load_metadata` و `save_metadata` نوشتتم. تابع `save` در ابتدای فایل دیسک `SuperBlock` و سپس کل آرایه `file_table` را می‌نویسد.

تابع `load` در شروع برنامه این اطلاعات را از دیسک می‌خواند و سپس لیست فضای خالی را بازسازی می‌کند. برای بازسازی ابتداء فرض می‌کند همه فضا خالی است و سپس با چک کردن فایل‌های موجود بلاک‌های استفاده شده را از لیست خالی حذف می‌کند.

عملیات فایل

تابع `fs_create` یک ورودی خالی در `file_table` پیدا می‌کند و اطلاعات فایل جدید را در آن می‌نویسد. در این مرحله فقط `metadata` ساخته می‌شود و هنوز بلاکی اختصاص داده نشده است.

تابع `fs_open` نام فایل را جستجو می‌کند و در صورت پیدا شدن یک `descriptor file` برمی‌گرداند. از یک آرایه `open_files` برای نگهداری ارتباط بین `fd` و `index` فایل در `file_table` استفاده کردم.

تابع `fs_write` ابتدا محاسبه می‌کند چند بلاک برای نوشتن نیاز است. اگر فایل بلاک نداشت از `allocate_blocks` بلاک‌های لازم را می‌گیرد و اگر بلاک‌های فعلی نبود بلاک اضافی درخواست می‌کند. سپس با `fseek` به موقعیت مناسب در فایل دیسک می‌رود و داده را می‌نویسد.

تابع `fs_read` مشابه `write` عمل می‌کند با این تفاوت که فقط می‌خواند و اگر درخواست خواندن بیشتر از سایز فایل بود آن را محدود می‌کند.

تابع `fs_shrink` فایل را به سایز کوچکتری تبدیل می‌کند. بلاک‌های اضافی را محاسبه کرده و با `free_blocks` آنها را آزاد می‌کند و سایز فایل را به روز می‌کند.

تابع `fs_delete` پس از پیدا کردن فایل بلاک‌های آن را آزاد می‌کند و فیلد `is_used` را صفر می‌کند. همچنین `descriptor file` های مربوط به آن فایل را می‌بندد.

رابط کاربری

یک CLI ساده پیاده سازی کردم که در یک حلقه دستورات کاربر را می‌خواند و اجرا می‌کند. دستوراتی مثل `create` برای ساخت فایل، `open` برای باز کردن، `read` و `write` برای خواندن و نوشتن، `ls` برای لیست فایل‌ها و `stat` برای نمایش وضعیت فایل سیستم.

برای پردازش دستورات از `strtok` استفاده کردم که رشته ورودی را به کلمات جداگانه تقسیم می‌کند. اولین کلمه نام دستور و بقیه آرگومان‌ها هستند.

در `prompt` نشان داده می‌شود که آیا فایلی باز است یا نه و اگر باز باشد نام آن نمایش داده می‌شود.
نمونه استفاده:

```
Type 'help' for commands
myfs> ls
█ Files (1):
-----  
1. a.txt  
-----  
  
myfs> format
△ Are you sure? All data will be erased! (yes/no): yes
█ Disk formatted
✓ Disk formatted  
  
myfs> create test.txt
█ File test.txt created
✓ File test.txt created successfully  
  
myfs> open test.txt c
█ File test.txt opened (fd=0)
✓ File test.txt opened (fd=0)  
  
myfs [test.txt]> write 0 hello_world
✗ 11 bytes written at position 0
✓ 11 bytes written  
  
myfs [test.txt]> read 0 11
█ 11 bytes read from position 0  
  
█ Content (11 bytes):
-----  
hello_world  
-----  
  
myfs [test.txt]> close
█ File closed (fd=0)
✓ File test.txt closed  
  
myfs> ls
█ Files (1):
-----  
1. test.txt  
-----  
  
myfs> █
```

کد کامل، همراه با فایل `readme` و دستور `help` برای استفاده راحت از `cli` کد، در پوشش `q3` قرار دارد.