دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

تمرین سری پنجم - عملی

پارسا ملکیان - ۴۰۲۱۷۱۰۷۵
فاطمه شفیعی - ۴۰۲۱۱۰۸۸۷

سیستم‌های عامل
دکتر اسدی

۱ گزارش پیاده‌سازی Lottery Scheduling در xv6

۱.۱ مقدمه

در این تمرین، الگوریتم زمان‌بندی Lottery Scheduling را برای سیستم‌عامل xv6 پیاده‌سازی کردیم. در این الگوریتم، به هر پردازه تعدادی بلیت (ticket) اختصاص داده می‌شود و پردازنده با یک قرعه‌کشی تصادفی، پردازه‌ای را برای اجرا انتخاب می‌کند.

۲.۱ مرحله ۱: اضافه کردن فیلد ticket به ساختار پردازه

۱.۲.۱ فایل: kernel/proc.h

فیلد ticket را به struct proc اضافه کردیم:

```
struct proc {
  // ...        ...
  char name[16];               // Process name (debugging)
  int ticket;                  // Lottery scheduling tickets
};
```

۳.۱ مرحله ۲: مقداردهی اولیه ticket و ارث‌بری در fork

۱.۳.۱ فایل: kernel/proc.c

۱.۲ مقداردهی اولیه در allocproc():

```
// Initialize lottery ticket to default value (10)
p->ticket = 10;
```

۲.۲ ارث‌بری در kfork():

```
// Inherit ticket count from parent
np->ticket = p->ticket;
```

۴.۱ مرحله ۳: تغییر تابع scheduler برای پیاده‌سازی Lottery Scheduling

۱.۴.۱ فایل: kernel/proc.c

الگوریتم جدید زمان‌بندی:

```
void scheduler(void)
{
  struct proc *p;
```

```
  struct cpu *c = mycpu();

  c->proc = 0;
  for(;;){
    intr_on();
    intr_off();

    // Lottery Scheduling: count total tickets of RUNNABLE processes
    int total_tickets = 0;
    for(p = proc; p < &proc[NPROC]; p++) {
      acquire(&p->lock);
      if(p->state == RUNNABLE) {
        total_tickets += p->ticket;
      }
      release(&p->lock);
    }

    if(total_tickets == 0) {
      asm volatile("wfi");
      continue;
    }

    // Generate random winning ticket (ensure positive with & 0x7FFFFFFF)
    int winner = (rand_int() & 0x7FFFFFFF) % total_tickets;
    int counter = 0;

    for(p = proc; p < &proc[NPROC]; p++) {
      acquire(&p->lock);
      if(p->state == RUNNABLE) {
        counter += p->ticket;
        if(counter > winner) {
          // This process wins the lottery
          p->state = RUNNING;
          c->proc = p;
          swtch(&c->context, &p->context);
          c->proc = 0;
          release(&p->lock);
          break;
        }
      }
      release(&p->lock);
    }
  }
}
```

۲.۴.۱ توضیح الگوریتم:

۱. ابتدا مجموع تمام بلیت‌های پردازه‌های RUNNABLE را محاسبه می‌کنیم

۲. یک عدد تصادفی بین ۰ تا total_tickets-1 تولید می‌کنیم

۳. با پیمایش پردازه‌ها و جمع کردن بلیت‌ها، پردازه برنده را پیدا می‌کنیم

۴. پردازه‌ای که بلیت بیشتری دارد، شانس بیشتری برای انتخاب دارد

۵.۱ مرحله ۴: اضافه کردن syscall جدید settickets

۱.۵.۱ فایل: kernel/syscall.h

```
#define SYS_settickets 22
```

۲.۵.۱ فایل: kernel/syscall.c

```
extern uint64 sys_settickets(void);
// ...
[SYS_settickets] sys_settickets,
```

۳.۵.۱ فایل: kernel/sysproc.c

```
uint64 sys_settickets(void)
{
  int pid, tickets;

  argint(0, &pid);
  argint(1, &tickets);

  if(tickets <= 0)
    return -1;

  return settickets(pid, tickets);
}
```

۴.۵.۱ فایل: kernel/proc.c

```
int settickets(int pid, int tickets)
{
  struct proc *p;

  for(p = proc; p < &proc[NPROC]; p++){
    acquire(&p->lock);
    if(p->pid == pid){
      p->ticket = tickets;
      release(&p->lock);
      return 0;
    }
    release(&p->lock);
  }
  return -1;
}
```

۵.۵.۱ فایل‌های user-space:

- user/usys.pl: **اضافه کردن** entry(”settickets”);
- user/user.h: **اضافه کردن** int settickets(int, int);

۶.۱ مرحله ۵: برنامه تست

۱.۶.۱ فایل: user/lotterytest.c

```c
#include "kernel/types.h"
#include "kernel/stat.h"
#include "user/user.h"

#define LOOP_COUNT 100000000

int main(void)
{
  int pid1, pid2, pid3, pid4;
  int counter1 = 0, counter2 = 0, counter3 = 0, counter4 = 0;

  printf("Lottery Scheduling Test\n");
  printf("Creating 4 child processes with tickets: 10, 20, 30, 40\n\n");

  pid1 = fork();
  if(pid1 == 0) {
    settickets(getpid(), 10);
    for(int i = 0; i < LOOP_COUNT; i++) counter1++;
    printf("Child 1 (10 tickets): counter = %d\n", counter1);
    exit(0);
  }

  pid2 = fork();
  if(pid2 == 0) {
    settickets(getpid(), 20);
    for(int i = 0; i < LOOP_COUNT; i++) counter2++;
    printf("Child 2 (20 tickets): counter = %d\n", counter2);
    exit(0);
  }

  pid3 = fork();
  if(pid3 == 0) {
    settickets(getpid(), 30);
    for(int i = 0; i < LOOP_COUNT; i++) counter3++;
    printf("Child 3 (30 tickets): counter = %d\n", counter3);
    exit(0);
  }

  pid4 = fork();
  if(pid4 == 0) {
    settickets(getpid(), 40);
    for(int i = 0; i < LOOP_COUNT; i++) counter4++;
    printf("Child 4 (40 tickets): counter = %d\n", counter4);
```

```
    exit(0);
  }

  wait(0); wait(0); wait(0); wait(0);

  printf("\nTest completed!\n");
  printf("Expected ratio: 10:20:30:40 = 1:2:3:4\n");

  exit(0);
}
```

۷.۱ نتیجه اجرا

۱.۷.۱ دستورات اجرا:

```
make clean
make CPUS=1 qemu
```

در xv6 shell:

```
lotterytest
```

۲.۷.۱ خروجی نمونه:

```
\$ lotterytest
Lottery Scheduling Test
Tickets: 10, 20, 30, 40 (ratio 1:2:3:4)

Child 4 (40 tickets): 466837 iterations
Child 3 (30 tickets): 444985 iterations
Child 2 (20 tickets): 323828 iterations
Child 1 (10 tickets): 139893 iterations

Expected ratio: 1:2:3:4 (10%:20%:30%:40%)
```

۳.۷.۱ تحلیل نتایج:

پردازه	بلیت	Iterations	درصد واقعی	درصد مورد انتظار
Child 1	۱۰	۱۳۹،۸۹۳	۲%.۱۰	۱۰%
Child 2	۲۰	۳۲۳،۸۲۸	۵%.۲۳	۲۰%
Child 3	۳۰	۴۴۴،۹۸۵	۴%.۳۲	۳۰%
Child 4	۴۰	۴۶۶،۸۳۷	۹%.۳۳	۴۰%

نتیجه: الگوریتم Lottery Scheduling به درستی کار می‌کند:

- پردازه با بلیت بیشتر (Child 4) بیشترین زمان CPU را دریافت کرده
- پردازه با بلیت کمتر (Child 1) کمترین زمان CPU را دریافت کرده
- نسبت تقریباً ۱:۲:۳:۴ رعایت شده است

توجه: از آنجا که Lottery Scheduling یک الگوریتم احتمالی است، نتایج دقیقاً مطابق نسبت بلیت‌ها نیست، اما با افزایش زمان اجرا، نتایج به نسبت مورد انتظار نزدیک‌تر می‌شوند.

```
xv6 kernel is booting

init: starting sh
$ lotterytest
Lottery Scheduling Test
Tickets: 10, 20, 30, 40 (ratio 1:2:3:4)

Child 2 (20 tickets): 366822 iterations
Child 4 (40 tickets): 472318 iterations
Child 3 (30 tickets): 373160 iterations
Child 1 (10 tickets): 265177 iterations

Expected ratio: ~1:2:3:4 (10%:20%:30%:40%)
$ lotterytest
exec lotterytest failed
$ lotterytest
Lottery Scheduling Test
Tickets: 10, 20, 30, 40 (ratio 1:2:3:4)

Child 4 (40 tickets): 466837 iterations
Child 3 (30 tickets): 444985 iterations
Child 2 (20 tickets): 323828 iterations
Child 1 (10 tickets): 139893 iterations

Expected ratio: ~1:2:3:4 (10%:20%:30%:40%)
$ lotterytest
exec lotterytest failed
$ lotterytest
Lottery Scheduling Test
Tickets: 10, 20, 30, 40 (ratio 1:2:3:4)

Child 4 (40 tickets): 536996 iterations
Child 3 (30 tickets): 431824 iterations
Child 2 (20 tickets): 238237 iterations
Child 1 (10 tickets): 225830 iterations

Expected ratio: ~1:2:3:4 (10%:20%:30%:40%)
$
```

۴.۷.۱ فایل‌های تغییر یافته:

۱. kernel/proc.h – **اضافه کردن فیلد** ticket
۲. kernel/proc.c – allocproc، kfork، scheduler، settickets
۳. kernel/syscall.h – **شماره** syscall **جدید**
۴. kernel/syscall.c – **ثبت** syscall
۵. kernel/sysproc.c – **پیاده‌سازی** sys_settickets
۶. kernel/defs.h – declaration **تابع** settickets
۷. user/usys.pl – stub **برای** user-space
۸. user/user.h – declaration **برای** user-space
۹. user/lotterytest.c – **برنامه تست**
۱۰. Makefile – **اضافه کردن** lotterytest

همه این فایل‌ها در پوشه ارسالی قرار داده شده است.

۲ گزارش تمرین ۲ - ایزوله‌سازی فایل‌سیستم کانتینرها

۱.۲ مقدمه

در این تمرین، هدف پیاده‌سازی ایزوله‌سازی فایل‌سیستم برای کانتینرها با استفاده از فراخوان سیستمی chroot(2) است. این کار باعث می‌شود کانتینر نتواند به فایل‌های میزبان دسترسی داشته باشد.

۲.۲ بخش اول - بررسی داکر

۱.۲.۲ گام آ: اجرای کانتینر busybox

```
docker run -d --name busybox-test busybox sleep 1000
```

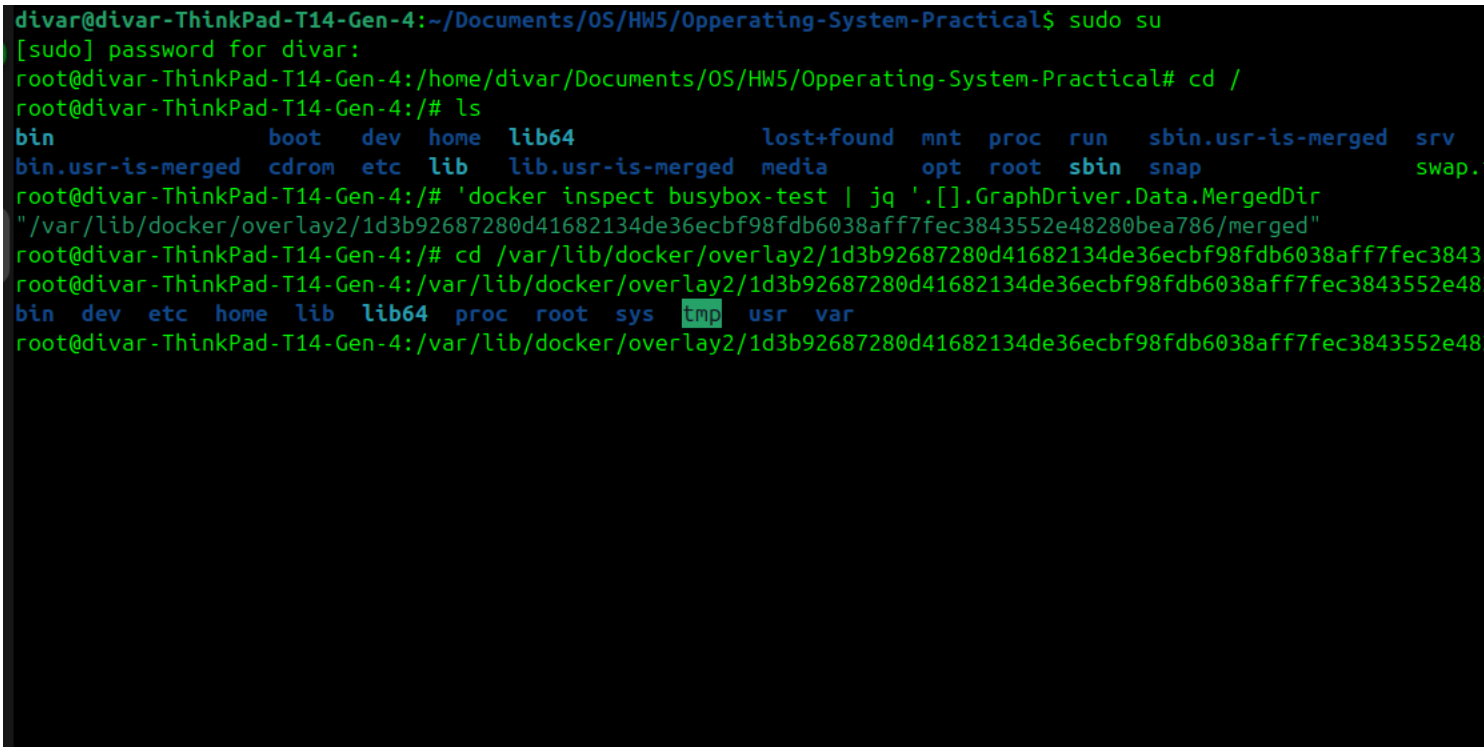
۲.۲.۲ گام ب: اجرای شل درکانتینر

```
docker exec -it busybox-test sh
cd /
ls
```

```
divar@divar-ThinkPad-T14-Gen-4:~/Documents/OS/HW5/Opperating-System-Practical$ docker run -d --name busybox-te
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
e59838ecfec5: Pull complete
Digest: sha256:e3652a00a2fabd16ce889f0aa32c38eec347b997e73bd09e69c962ec7f8732ee
Status: Downloaded newer image for busybox:latest
8e325ebd7ef96c13aa84a176a229c8034445a90f7ee1bbd25c4cb4f2b8fb096f
divar@divar-ThinkPad-T14-Gen-4:~/Documents/OS/HW5/Opperating-System-Practical$ docker run -d --name busybox-te
docker: Error response from daemon: Conflict. The container name "/busybox-test" is already in use by containe
have to remove (or rename) that container to be able to reuse that name.

Run 'docker run --help' for more information
divar@divar-ThinkPad-T14-Gen-4:~/Documents/OS/HW5/Opperating-System-Practical$ docker ps
CONTAINER ID   IMAGE         COMMAND                  CREATED         STATUS                  PORTS
8e325ebd7ef9   busybox       "sleep 1000"             17 seconds ago  Up 16 seconds
49c117463a46   postgres:15   "docker-entrypoint.s…"   13 days ago     Up 11 days (healthy)    0.0.0.0:5433->54
divar@divar-ThinkPad-T14-Gen-4:~/Documents/OS/HW5/Opperating-System-Practical$ docker exec -it busybox-test sh
/ # cd /
/ # ls
bin    dev    etc    home   lib    lib64  proc   root   sys    tmp    usr    var
/ # \q
sh: q: not found
/ # exit
divar@divar-ThinkPad-T14-Gen-4:~/Documents/OS/HW5/Opperating-System-Practical$ docker inspect busybox-test | j
"/var/lib/docker/overlay2/1d3b92687280d41682134de36ecbf98fdb6038aff7fec3843552e48280bea786/merged"
```

۳.۲.۲ گام ج: بررسی MergedDir

```
docker inspect busybox-test | jq '.[].GraphDriver.Data.MergedDir'
sudo su
cd <path>
ls
```



محتوای این مسیر با ریشه کانتینر یکسان است.

۳.۲ سوال ۱: چگونه ممکن است مسیر ریشه کانتینر داکر از مسیر ریشه میزبان متفاوت باشد؟

۱.۳.۲ پاسخ:

داکر از فراخوان سیستمی chroot() یا pivot_root() استفاده می‌کند تا مسیر ریشه را برای پروسه کانتینر تغییر دهد.

وقتی یک کانتینر شروع به کار می‌کند:

۱. داکر ایمیج را در یک دایرکتوری استخراج می‌کند (مثلاً /var/lib/docker/overlay2/.../merged)

۲. با استفاده از chroot() یا pivot_root()، ریشه کانتینر به آن دایرکتوری تغییر می‌کند

۳. از دید کانتینر، / همان فایل‌سیستم ایمیج است، نه ریشه واقعی میزبان

به همین دلیل است که با دستور docker inspect می‌توان مسیر واقعی میزبان (MergedDir) را که معادل / کانتینر است مشاهده کرد.

۴.۲ بخش دوم – پیاده‌سازی در Zocker

۱.۴.۲ پیش‌نیازها

```
mkdir -p /tmp/zocker
git checkout t3
```

۲.۴.۲ پاکسازی قبل از هر اجرا

```
rm -rf /tmp/zocker/test-container
```

۳.۴.۲ گام د: اضافه کردن chroot به src/run.c

تغییرات انجام شده در فایل src/run.c:

```
if (chroot(container_dir) != 0) {
  fprintf(stderr, "[ERR] Failed to chroot to %s: %s\n", container_dir,
          strerror(errno));
  return 1;
}

if (chdir("/") != 0) {
  fprintf(stderr, "[ERR] Failed to chdir to /: %s\n", strerror(errno));
  return 1;
}
```

این کد پس از setup_container_dir() و قبل از mount کردن /proc اضافه شد.

نتیجه: پس از اجرا، با خطای زیر مواجه می‌شویم:

```
[ERR] Failed to remount /proc: No such file or directory
```

```
divar@divar-ThinkPad-T14-Gen-4:~/Documents/OS/HW5/Opperating-System-Practical$ make
gcc -Wall -Wextra -std=c99 -MMD -MP -c src/config.c -o src/config.o
gcc -Wall -Wextra -std=c99 -MMD -MP -c src/main.c -o src/main.o
gcc -Wall -Wextra -std=c99 -MMD -MP -c src/run.c -o src/run.o
gcc -Wall -Wextra -std=c99 -MMD -MP -c src/setup.c -o src/setup.o
gcc -Wall -Wextra -std=c99 -MMD -MP -o zocker src/config.o src/main.o src/run.o src/setup.o
sudo setcap cap_sys_admin+ep zocker
divar@divar-ThinkPad-T14-Gen-4:~/Documents/OS/HW5/Opperating-System-Practical$ sudo setcap cap_sys_admin,cap_sy
divar@divar-ThinkPad-T14-Gen-4:~/Documents/OS/HW5/Opperating-System-Practical$ rm -rf /tmp/zocker/test-containe
divar@divar-ThinkPad-T14-Gen-4:~/Documents/OS/HW5/Opperating-System-Practical$ ./zocker run --name test-contair
[ERR] Failed to remount /proc: No such file or directory
[ERR] Running container failed due to some internal errors.
[Parent] Stoping...
divar@divar-ThinkPad-T14-Gen-4:~/Documents/OS/HW5/Opperating-System-Practical$
```

۵.۲ سوال ۲: علت بروز خطای /proc چیست؟

۱.۵.۲ پاسخ:

پس از اجرای chroot()، ریشه جدید فایل‌سیستم به مسیر /tmp/zocker/test-container تغییر می‌کند. این دایرکتوری خالی است و پوشه /proc در آن وجود ندارد.

وقتی می‌خواهیم procfs را در /proc مانت کنیم، چون این پوشه وجود ندارد، عملیات mount با خطای "No such file or directory" شکست می‌خورد.

۲.۵.۲ راه‌حل:

ایجاد پوشه /proc پس از chroot و قبل از mount:

```
if (mkdir("/proc", 0755) == -1 && errno != EEXIST) {
  fprintf(stderr, "[ERR] Failed to create /proc: %s\n", strerror(errno));
  return 1;
}
```

۳.۵.۲ گام ه و و: خطای بعدی

پس از رفع خطای قبلی، با خطای زیر مواجه می‌شویم:

```
[ERR] Failed to call create container process: No such file or directory
```

```
divar@divar-ThinkPad-T14-Gen-4:~/Documents/OS/HW5/Opperating-System-Practical$ make
gcc -Wall -Wextra -std=c99 -MMD -MP -c src/run.c -o src/run.o
gcc -Wall -Wextra -std=c99 -MMD -MP -o zocker src/config.o src/main.o src/run.o src/setup.o
sudo setcap cap_sys_admin+ep zocker
divar@divar-ThinkPad-T14-Gen-4:~/Documents/OS/HW5/Opperating-System-Practical$ sudo setcap cap_sys_admin,cap_sy
divar@divar-ThinkPad-T14-Gen-4:~/Documents/OS/HW5/Opperating-System-Practical$ rm -rf /tmp/zocker/test-containe
divar@divar-ThinkPad-T14-Gen-4:~/Documents/OS/HW5/Opperating-System-Practical$ ./zocker run --name test-contair
Running child with pid: 1
[ERR] Failed to call create container process: No such file or directory
[ERR] Running container failed due to some internal errors.
[Parent] Stoping...
divar@divar-ThinkPad-T14-Gen-4:~/Documents/OS/HW5/Opperating-System-Practical$
```

۶.۲ سوال ۳: چرا این خطا رخ می‌دهد؟

۱.۶.۲ پاسخ:

پس از chroot()، پروسه سعی می‌کند /bin/sh را اجرا کند. اما در ریشه جدید (یعنی /tmp/zocker/test-container)، فایل /bin/sh وجود ندارد.

توابع setup_bin_dir و setup_lib_dir فقط پوشه‌های خالی ایجاد می‌کنند و هیچ فایل باینری یا کتابخانه‌ای در آن‌ها کپی نمی‌شود.

۲.۶.۲ راه‌حل:

تغییر توابع در src/setup.c برای:

۱. کپی کردن باینری‌های لازم (/bin/sh، /bin/ls، /bin/cat، /bin/pwd) به دایرکتوری کانتینر
۲. کپی کردن کتابخانه‌های مشترک مورد نیاز (با استفاده از ldd)

۷.۲ تغییرات کد

۱.۷.۲ فایل src/run.c

```
//           mount private
if (chroot(container_dir) != 0) {
  fprintf(stderr, "[ERR] Failed to chroot to %s: %s\n", container_dir,
          strerror(errno));
  return 1;
}

if (chdir("/") != 0) {
  fprintf(stderr, "[ERR] Failed to chdir to /: %s\n", strerror(errno));
  return 1;
}

if (mkdir("/proc", 0755) == -1 && errno != EEXIST) {
  fprintf(stderr, "[ERR] Failed to create /proc: %s\n", strerror(errno));
  return 1;
}
```

۲.۷.۲ فایل src/setup.c

تابع کمکی برای کپی فایل‌ها و کتابخانه‌ها:

```
static int copy_file(const char *src, const char *dst) {
  char cmd[512];
  snprintf(cmd, sizeof(cmd), "cp %s %s", src, dst);
  return system(cmd);
}

static int copy_libs_for_binary(const char *binary, const char *container_dir) {
  //        ldd
  //
{
```

تغییر تابع setup_bin_dir:

```
const char *binaries[] = {"/bin/sh", "/bin/ls", "/bin/cat", "/bin/pwd", NULL};

for (int i = 0; binaries[i] != NULL; i++) {
  //
  copy_file(binaries[i], dest);
  //
  copy_libs_for_binary(binaries[i], container_dir);
}
```

۸.۲ گام ز: تأیید عملکرد

۱.۸.۲ دستورات اجرا:

```
make
sudo setcap cap_sys_admin,cap_sys_chroot+ep zocker
rm -rf /tmp/zocker/test-container
./zocker run --name test-container 'sh'
```

۲.۸.۲ تست داخل کانتینر:

```
$ pwd
/

$ ls /
bin  lib  lib32  lib64  proc

$ cd /..
$ pwd
/

$ ls
bin  lib  lib32  lib64  proc

$ cat /etc/passwd
cat: /etc/passwd: No such file or directory
```

```
divar@divar-ThinkPad-T14-Gen-4:~/Documents/OS/HW5/Opperating-System-Practical$ make
gcc -Wall -Wextra -std=c99 -MMD -MP -c src/setup.c -o src/setup.o
src/setup.c: In function ‘setup_bin_dir’:
src/setup.c:144:56: warning: ‘%s’ directive output may be truncated writing up to 511 bytes into a region of s
  144 |         snprintf(chmod_cmd, sizeof(chmod_cmd), "chmod +x %s", dest);
      |                                                        ^~   ~~~~
src/setup.c:144:7: note: ‘snprintf’ output between 10 and 521 bytes into a destination of size 512
  144 |         snprintf(chmod_cmd, sizeof(chmod_cmd), "chmod +x %s", dest);
      |         ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
gcc -Wall -Wextra -std=c99 -MMD -MP -o zocker src/config.o src/main.o src/run.o src/setup.o
sudo setcap cap_sys_admin+ep zocker
divar@divar-ThinkPad-T14-Gen-4:~/Documents/OS/HW5/Opperating-System-Practical$ sudo setcap cap_sys_admin,cap_s
divar@divar-ThinkPad-T14-Gen-4:~/Documents/OS/HW5/Opperating-System-Practical$ rm -rf /tmp/zocker/test-contain
divar@divar-ThinkPad-T14-Gen-4:~/Documents/OS/HW5/Opperating-System-Practical$ ./zocker run --name test-contai
Running child with pid: 1
$ ls
bin  lib  lib32  lib64  proc
$ pwd
/
$ cd proc
$ ls
1       bootconfig  consoles   dma           filesystems  irq        kmsg          loadavg  modules  pagetyp
78      buddyinfo   cpuinfo    driver        fs           kallsyms   kpagecgroup   locks    mounts   partiti
80      bus         crypto     dynamic_debug interrupts   kcore      kpagecount    mdstat   mtd      pressur
acpi    cgroups     devices    execdomains   iomem        key-users  kpageflags    meminfo  mtrr     schedst
asound  cmdline     diskstats  fb            ioports      keys       latency_stats misc     net      scsi
$ cd ..
$ ls
bin  lib  lib32  lib64  proc
$ cat /etc/passwd  # Should fail - file doesn't exist
cat: /etc/passwd: No such file or directory
$
```

۳.۸.۲ مقایسه با میزبان:

در میزبان:

```
\$ ls /
bin   boot   dev   etc   home   lib   lib64   media   mnt   opt   proc   root   run   sbin   srv
    sys   tmp   usr   var
```

نتیجه: کانتینر فقط فایل‌های موجود در دایرکتوری ایزوله خود را می‌بیند و دسترسی به فایل‌های میزبان ندارد.

۹.۲ گام ح: دسترسی به فایل‌های میزبان از داخل کانتینر

می‌توان با استفاده از **bind mount** یک دایرکتوری میزبان را قبل از chroot به داخل دایرکتوری کانتینر مانت کرد:

```
char host_mount[512];
snprintf(host_mount, sizeof(host_mount), "%s/host", container_dir);
mkdir(host_mount, 0755);
mount("/home", host_mount, NULL, MS_BIND, NULL);
```

پس از این تغییر، داخل کانتینر می‌توان با ls /host به فایل‌های home/ میزبان دسترسی داشت.

گزارش سوال ۳

در این پروژه، یک سیستم‌فایل ساده مبتنی بر بلاک پیاده‌سازی شده است. هدف اصلی این بخش، بهبود مدیریت فضای آزاد با استفاده از ساختار داده لیست پیوندی است که هر گره شامل شروع و پایان ناحیه آزاد می‌باشد.

۳ ساختار داده جدید

۱.۳ تغییر ساختار FreeBlockNode

ساختار قبلی freelist شامل start_block و block_count بود. این ساختار به صورت زیر تغییر کرد:

```
typedef struct FreeBlockNode {
    int32_t start_block;
    int32_t end_block;
    struct FreeBlockNode* next;
} FreeBlockNode;
```

مزایای این تغییر:

- امکان محاسبه سریع تعداد بلاک‌ها: count = end_block - start_block + 1
- سادگی در ادغام بلاک‌های مجاور
- نمایش بهتر محدوده‌های آزاد

۴ پیاده‌سازی توابع

۱.۴ تابع fs_alloc(size)

این تابع یک ناحیه با اندازه مشخص را از فضای آزاد تخصیص می‌دهد:

```
int fs_alloc(int size) {
    int needed_blocks = (size + BLOCK_SIZE - 1) / BLOCK_SIZE;
    if (needed_blocks == 0) needed_blocks = 1;

    FreeBlockNode* curr = free_list;
    FreeBlockNode* prev = NULL;

    while (curr) {
        int available = curr->end_block - curr->start_block + 1;

        if (available >= needed_blocks) {
            int allocated = curr->start_block;

```

```
            if (available == needed_blocks) {
                if (prev) prev->next = curr->next;
                else free_list = curr->next;
                free(curr);
            } else {
                curr->start_block += needed_blocks;
            }

            super_block.used_blocks += needed_blocks;
            return allocated;
        }
        prev = curr;
        curr = curr->next;
    }

    return -1;
}
```

۲.۴ تابع fs_free(start, size)

این تابع یک ناحیه را آزاد کرده و در صورت مجاورت با نواحی آزاد دیگر، آن‌ها را ادغام می‌کند:

```
void fs_free(int start, int size) {
    int block_count = (size + BLOCK_SIZE - 1) / BLOCK_SIZE;
    int end = start + block_count - 1;

    FreeBlockNode* curr = free_list;
    FreeBlockNode* prev = NULL;

    while (curr && curr->start_block < start) {
        prev = curr;
        curr = curr->next;
    }

    int merged_with_prev = 0;
    if (prev && prev->end_block + 1 == start) {
        prev->end_block = end;
        merged_with_prev = 1;
    }

    if (curr && end + 1 == curr->start_block) {
        if (merged_with_prev) {
            prev->end_block = curr->end_block;
            prev->next = curr->next;
            free(curr);
        } else {
            curr->start_block = start;
        }
    } else if (!merged_with_prev) {
        FreeBlockNode* new_node = malloc(sizeof(FreeBlockNode));
        new_node->start_block = start;
        new_node->end_block = end;
        new_node->next = curr;
```

```
        if (prev) prev->next = new_node;
        else free_list = new_node;
    }

    super_block.used_blocks -= block_count;
}
```

۳.۴ قانون ادغام

نکته مهم: در این پیاده‌سازی، دو بلاک آزاد مجاور نمی‌توانند کنار هم وجود داشته باشند. هنگام آزادسازی:

۱. اگر بلاک جدید دقیقاً قبل از یک ناحیه آزاد باشد ← ادغام با بلاک بعدی

۲. اگر بلاک جدید دقیقاً بعد از یک ناحیه آزاد باشد ← ادغام با بلاک قبلی

۳. اگر بلاک جدید بین دو ناحیه آزاد باشد ← ادغام سه‌گانه

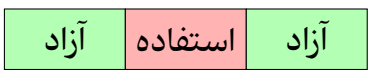
۵ دستور viz

دستور viz به رابط کاربری اضافه شد تا فضاهای آزاد را به صورت جدول نمایش دهد:

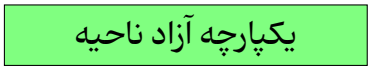
```
void fs_visualize_free_list() {
    printf("Index    Start      End        Blocks       Size (KB)\n");

    FreeBlockNode* curr = free_list;
    int index = 0;

    while (curr) {
        int block_count = curr->end_block - curr->start_block + 1;
        printf("%-8d %-10d %-10d %-12d %-12.2f\n",
               index, curr->start_block, curr->end_block,
               block_count, (block_count * BLOCK_SIZE) / 1024.0);
        index++;
        curr = curr->next;
    }
}
```

۶ نمودار ادغام بلاک‌ها

آزادسازی: از قبل



آزادسازی: از بعد



۷ تست عملکرد

۱.۷ اسکریپت تست

برای تست صحت عملکرد ادغام بلاک‌ها، دستورات زیر را اجرا کنید:

```
# compile and run
make clean && make
rm -f filesys.db
./myfs
```

سپس دستورات زیر را در شل وارد کنید:

```
viz

create file1.txt
open file1.txt rw
write 0 AAAAAAAAAA
close

create file2.txt
open file2.txt rw
write 0 BBBBBBBBBB
close

create file3.txt
open file3.txt rw
write 0 CCCCCCCCCC
close

viz

rm file2.txt
viz

rm file1.txt
viz

rm file3.txt
viz

exit
```

۲.۷ نتیجه مورد انتظار

```


    Welcome to MyFileSystem

Type 'help' for commands
myfs> viz
```

```
  Free Space Visualization:
Index    Start      End        Blocks       Size (KB)

0        15         20479      20465        10232.50

 Summary: 1 regions, 20465 free blocks (10232.50 KB / 9.99 MB)



myfs> create file1.txt
 File file1.txt created
 File file1.txt created successfully

myfs> open file1.txt rw
 File file1.txt opened (fd=0)
 File file1.txt opened (fd=0)

myfs [file1.txt]> write 0 AAAAAAAAAA
  10 bytes written at position 0
 10 bytes written

myfs [file1.txt]> close
 File closed (fd=0)
 File file1.txt closed

myfs> create file2.txt
 File file2.txt created
 File file2.txt created successfully

myfs> open file2.txt rw
 File file2.txt opened (fd=0)
 File file2.txt opened (fd=0)

myfs [file2.txt]> write 0 BBBBBBBBBB
  10 bytes written at position 0
 10 bytes written

myfs [file2.txt]> close
 File closed (fd=0)
 File file2.txt closed

myfs> create file3.txt
 File file3.txt created
 File file3.txt created successfully

myfs> open file3.txt rw
 File file3.txt opened (fd=0)
 File file3.txt opened (fd=0)

myfs [file3.txt]> write 0 CCCCCCCCCC
  10 bytes written at position 0
 10 bytes written
```

```
myfs [file3.txt]> close
 File closed (fd=0)
 File file3.txt closed

myfs> viz

  Free Space Visualization:

Index    Start      End        Blocks       Size (KB)

0        18         20479      20462        10231.00

 Summary: 1 regions, 20462 free blocks (10231.00 KB / 9.99 MB)


myfs> rm file2.txt
  File file2.txt deleted
 File file2.txt deleted

myfs> viz

  Free Space Visualization:

Index    Start      End        Blocks       Size (KB)

0        16         16         1            0.50
1        18         20479      20462        10231.00

 Summary: 2 regions, 20463 free blocks (10231.50 KB / 9.99 MB)


myfs> rm file1.txt
  File file1.txt deleted
 File file1.txt deleted

myfs> viz

  Free Space Visualization:

Index    Start      End        Blocks       Size (KB)

0        15         16         2            1.00
1        18         20479      20462        10231.00

 Summary: 2 regions, 20464 free blocks (10232.00 KB / 9.99 MB)

```

```
107
108 myfs> rm file3.txt
109   File file3.txt deleted
110  File file3.txt deleted
111
112 myfs> viz
113
114   Free Space Visualization:
115
116 Index    Start      End        Blocks       Size (KB)
117
118 0        15         20479      20465        10232.50
119
120  Summary: 1 regions, 20465 free blocks (10232.50 KB / 9.99 MB)
121
122
123 myfs> exit
124  Goodbye!
125
126  Filesystem saved and closed
```

تصاویر این خروجی همراه تمام فایل‌های پروژه در پوشه ارسالی قرار دارد