بسمه تعالی

گزارش تمرین عملی چهارم – سوال دوم – درس سیستم‌های عامل دکتر اسدی – نیمسال اول۴۰۴-۴۰۵

نویسندگان:

۱. سیداحمد موسوی‌اول – ۴۰۲۱۰۶۶۴۸

۲. عرفان تیموری – ۴۰۲۱۰۵۸۱۳

در این تمرین که ادامه تمرین قبلی است، باید تغییراتی برای دستور ps که در کانتینر اجرا می‌شود داشته باشیم تا درست باشد و دیگر ارتباطی با سیستم اصلی نداشته باشد.

ابتدا سوال‌ها را پاسخ می‌دهیم:

در مورد سوال اول: چرا دستور ps همچنان خوب کار نمی‌کند و تمامی فرایندها را نشان می‌دهد؟

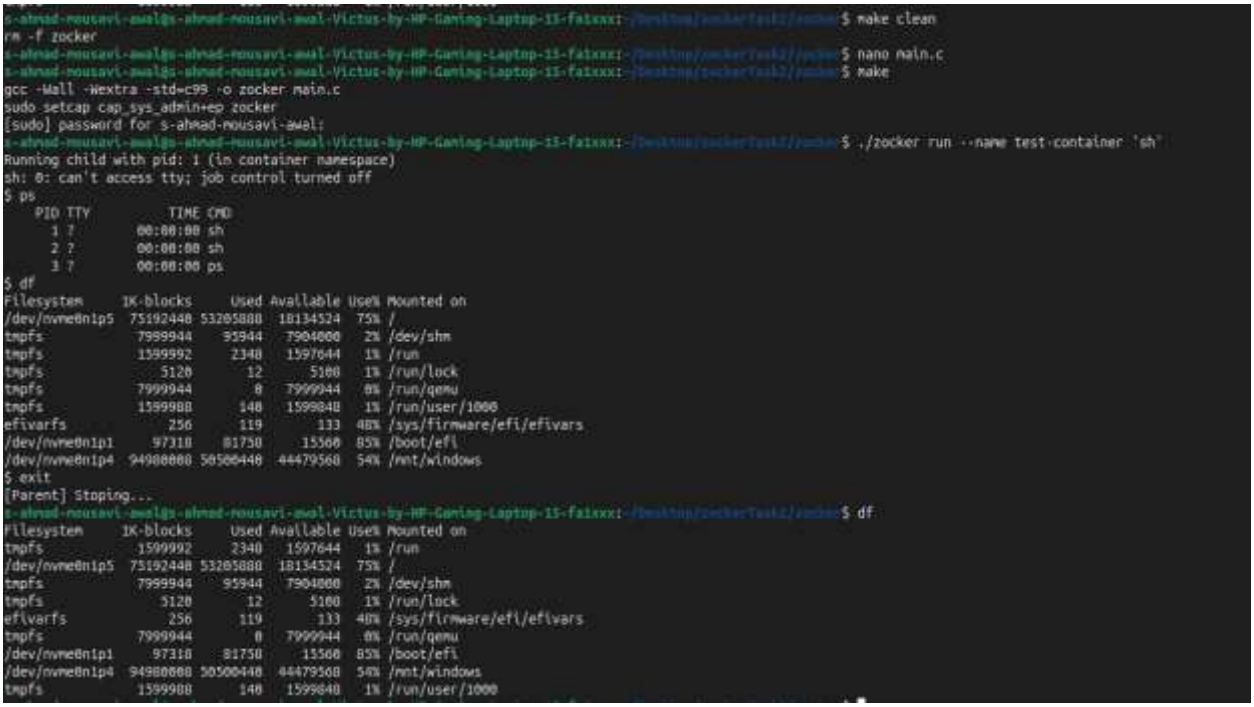
دستور ps برای مشاهده لیست فرآیندها، اطلاعات خود را از سیستم فایل /proc می‌خواند. در مرحله اولیه، فضای نام mount جدیدی ایجاد نشده است. بنابراین، کانتینر همچنان از /proc مربوط به سیستم اصلی (host) استفاده می‌کند.

حال به سراغ بخش ج رفتیم و مورد گفته شده را با انجام مواردی که در لینک اشاره شده بود و با استفاده از mount انجام دادیم که نتایج در زیر مشهودند.

ولی به مشکلی که گفته شده خوردیم و حال به سوال گفته شده پاسخ می‌دهیم:

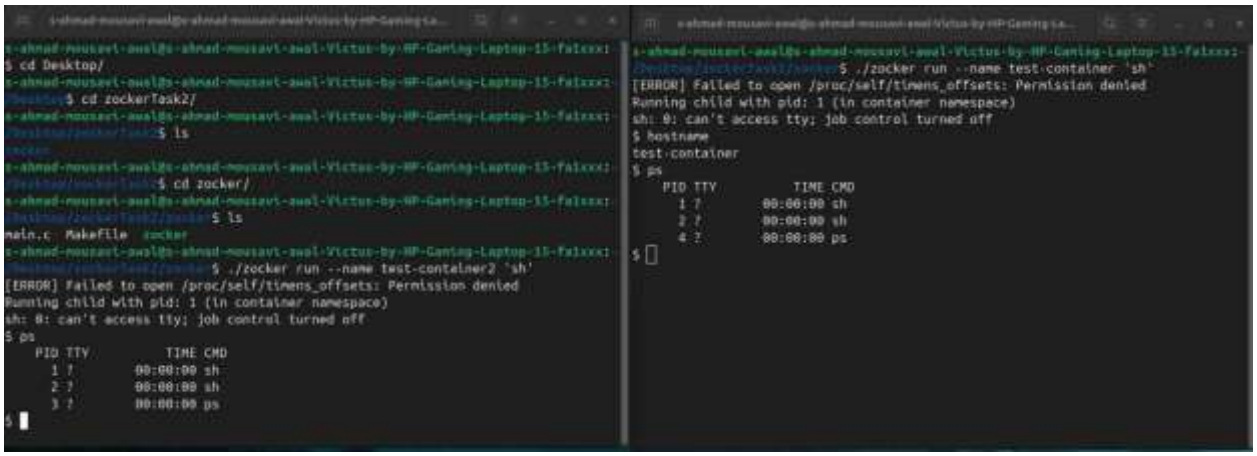
دلیل این مشکل این است که به صورت پیش‌فرض در بسیاری از سیستم‌های لینوکس، نقاط اتصال (Mount Points) در حالت MS_SHARED قرار دارند. این بدان معناست که هرگونه عملیات Mount یا Unmount در داخل فضای نام جدید، به Host نیز منتشر (Propagate) می‌شود و می‌تواند سیستم Host را دچار مشکل کند. که برای این مشکل از MS_PRIVATE استفاده می‌کنیم.

در نتیجه باید از فلگ MS_PRIVATE استفاده میکردیم که در قسمت ه از آن استفاده کرده و نتایج به شرح زیر مشهودند:



که دستور گفته شده درست کار می‌کند.

در ادامه برای بخش و باید در قسمت Unshare این را هم می‌زدیم: CLONE_NEWUTS که این را هم اضافه کرده و بنابراین برای هرکانتینر یک فضای نام به صورت جداگانه اختصاص پیدا می‌کند که این در عکس زیر مشهود است:



که در آن فضای نام ها جدا اند.

در ادامه برای بخش ز نیاز بود از دستور setHostname گفته شده استفاده کنیم که با استفاده از آن توانستیم اسم را به کانتینر اختصاص دهیم مانند زیر:

همانطور که مشهود است دستور فوق در کانتینر و شل با یکدیگر فرق دارد.

برای بخش ح هم یک تابع جدید اضافه کردیم تا زمان را درست کند و عکس آن در زیر مشهود است که زمان از **0** شروع شده که مشخصا زمان دستگاه آن نیست:

کدهای زده شده به شرح زیر است که فایل main.c هم در کنار همین فایل قرار دارد و می‌توانید آن را ببینید

```c
void setup_time_offsets() {
    struct timespec now;

    if (clock_gettime(CLOCK_MONOTONIC, &now) != 0) {
        perror("clock_gettime failed");
        return;
    }

    long sec = -(now.tv_sec + 1);
    long nsec = 1000000000L - now.tv_nsec;

    FILE *f = fopen("/proc/self/timens_offsets", "w");
    if (f == NULL) {
        perror("[ERROR] Failed to open /proc/self/timens_offsets");
        return;
    }

    fprintf(f, "monotonic %ld %ld\n", sec, nsec);
    fprintf(f, "boottime %ld %ld\n", sec, nsec);

    fflush(f);
    fclose(f);
}

int run_container(struct config cfg) {
    pid_t pid;

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

    pid = fork();
    if (pid < 0) {
        return 1;
    }
    if (pid == 0) {
    if (sethostname(cfg.name, strlen(cfg.name)) != 0) {
            perror("Failed to set hostname");
            exit(1);
        }
```

```
43.
44.         if (mount(NULL, "/", NULL, MS_REC | MS_PRIVATE, NULL) != 0) {
45.             perror("Failed to make root mount private");
46.             exit(1);
47.         }
48.
49.         if (mount("proc", "/proc", "proc", 0, NULL) != 0) {
50.             perror("Failed to mount /proc");
51.             exit(1);
52.         }
53.
54.     setsid();
55.
56.         printf("Running child with pid: %d (in container namespace)\n",
    getpid());
57.
58.         // Execute the command
59.         char *args[] = {"/bin/sh", "-c", cfg.command, NULL};
60.         execvp(args[0], args);
61.         perror("Execvp failed");
62.         exit(1);
63.     } else {
64.         waitpid(pid, NULL, 0);
65.         printf("[Parent] Stoping...\n");
66.     }
67.     return 0;
68. }
```