

به نام خدا

گزارش نهایی پروژه درس سیستم های عامل

استاد درس: دکتر جلیلی

نگارنده: محمدحسن بیاتیانی

شماره دانشجویی: 401105691

بهار 1404

# گزارش فاز اول:

گزارش ارائه شده در فاز اول از طریق [این لینک](#) قابل دستیابی میباشد.

## نحوه پیاده سازی به همراه توضیحات کد منبع:

کد و موارد دیگر مربوط به پروژه در [این](#) ریپازیتوری پابلیک گیت‌هاب قرار داده شده اند. در ادامه به ترتیب توابع نابدیهی استفاده شده در کد شرح داده خواهند شد:

### analyze\_process\_status

با استفاده از اطلاعات موجود در فایل `proc/[pid]/status/` بررسی کند. با استخراج شناسه والد (PPid) و وضعیت پردازش (State)، اگر پردازش فرزند مستقیم پردازش والد داده شده باشد، آن را بسته به وضعیتش در یکی از دو لیست موجود در ساختار `ContainerProcessList` قرار می‌دهد: اگر پردازش در وضعیت `Zombie` باشد در لیست `Zombie`ها، و در غیر این صورت در لیست پردازش‌های فعال (`Running`).

### gather\_container\_processes

تمامی موارد موجود در مسیر `proc/` در این تابع خوانده میشوند و به تابع `analyze_process_status` پاس داده میشوند تا تعیین وضعیت شوند.

### allocate\_stack\_memory

این تابع برای هر پردازش حافظه مورد نیاز را `allocate` میکند و پوینتر بالای استک را برای محل آغاز حافظه در نظر گرفته شده برمیگرداند.

### Set\_{cpu/IO/memory}\_cgroup

این سه تابع وظیفه ذخیره سازی `limit`های تعریف شده برای منابع را دارند. به این صورت که در مسیر `sys/fs/cgroup/` مقدار مورد نظر را ذخیره میکنند. در ادامه در بخش نتایج آزمایش به صورت تصویری آپدیت شدن مسیر مذکور را نمایش خواهم داد.

### setup\_root\_directory

این تابع وظیفه کپی کردن لایبرری های موجود برای هر محفظه را دارد. در حقیقت از مسیرهای سیستم عامل مانند `lib/` و `lib64/` به درون فایل های مورد نظر در محفظه کپی میکند. همچنین در این قسمت بعد از کپی

کردن فایل های مذکور همچنان با یک ارور در dependency ها مواجه شدم که در قسمت چالش ها نحوه حل کردن آن را توضیح خواهم داد.

## CreateContainer

این تابع ابتدا با استفاده از تنظیمات موجود در ساختار ContainerConfig، کانتینر مقداردهی اولیه می شود و دایرکتوری ریشه ی جدید آن تنظیم می گردد. سپس با استفاده از chroot و chdir، فضای فایل سیستم برای کانتینر ایزوله شده و دایرکتوری /proc در آن ساخته و مانع می شود. نام میزبان کانتینر تنظیم می گردد و سپس یک پردازش فرزند با fork ایجاد شده که در آن برنامه ی اجرایی تعریف شده در کانتینر اجرا می شود. والد منتظر پایان یافتن فرزند می ماند و وضعیت خروج آن را چاپ می کند. در پایان، منابعی مانند /proc آزاد می شود. این تابع در واقع یک محیط ایزوله شبیه به کانتینر ایجاد و برنامه ای را در آن اجرا می کند.

## show\_container\_status

این تابع با استفاده از مقادیر موجود در ContainerProcessList وضعیت پردازش ها را در صورت اجرای دستور status نشان می دهد.

## terminate\_container

این تابع در صورت اجرای دستور terminate استفاده شده و پردازش مورد نظر را kill میکند.

## restart\_container

ابتدا پردازشی که شناسه ی آن به تابع داده شده با سیگنال SIGKILL متوقف می شود. سپس یک پوشه جدید برای کانتینر ساخته می شود (با استفاده از زمان فعلی برای تولید نام یکتا) و پیکربندی قبلی کانتینر (از آرایه configs) کپی شده و در موقعیت جدیدی ذخیره می شود. با استفاده از clone و پرچم های فضای نام (namespaces) مانند CLONE\_NEWPID و CLONE\_NEWUTS، یک پردازش جدید به عنوان کانتینر راه اندازی می شود که از تابع CreateContainer برای اجرا بهره می برد. در نهایت، شناسه پردازش جدید در pid\_map به روزرسانی می شود.

## parse\_cli\_args

این تابع وظیفه دارد آرگومان های دستور اصلی را برای پیکربندی کانتینر تجزیه کند و مقادیر مربوط به کانتینر را در آرایه ای از ContainerConfig ذخیره نماید. به ازای فلگ -x (اجرایی) یک کانتینر جدید تعریف شده و سایر فلگ ها (مانند محدودیت منابع) به همان کانتینر نسبت داده می شوند. اگر فلگی بدون تعریف کانتینر (یعنی بدون -x) داده شود، خطا نمایش داده می شود و برنامه خاتمه می یابد.

**فلگ های پشتیبانی شده:**

- -x <executable>: مسیر برنامه‌ای که باید در کانتینر اجرا شود
- -n <hostname>: نام میزبان کانتینر
- -m <mem>: محدودیت حافظه
- -u <cpu>: محدودیت CPU
- -i <read-iops>: IOPS محدودیت عملیات خواندن
- -o <write-iops>: محدودیت عملیات نوشتن

## نتایج آزمایش ها:

در ادامه به ترتیب موارد مختلف تست شده بر روی برنامه نوشته شده را به همراه تصویر نمایش میدهیم.

### آزمایش استفاده از دستورات

- اجرای برنامه به همراه ورودی باینری کد test و host=mmd و memory limit = 32Mb  
کد test به صورت زیر میباشد:

```
#include <unistd.h>
#include <stdio>

int main()
{
    printf("test starting ...\n");
    fflush(stdout);
    sleep(10);
    printf("test ends ...\n");

    return 0;
}
```

- دستور status اجرا شده و وضعیت این پردازش در حال اجرا می باشد.

```
mohammadhossein@mohammadhossein-ThinkBook-15-G2-ITL:~/Desktop/OS$ sudo ./container -x ./test -n mmd -m 33554432
[Manager] PID 18966 started.
[Manager] Container started (PID: 18967, Hostname: mmd)
[Command] Enter [status, terminate <PID>, restart <PID>, exit]: Initializing container...
Setting up root directory...
Changing root to: ./containers/folder_1750964142
Attempting to run: /test
test starting ...
status
[Status] Active Containers:
PID: 18967      Hostname: mmd

[Status] Zombie Containers:

[Command] Enter [status, terminate <PID>, restart <PID>, exit]: test ends ...

[parent] Child exited with code 0
```

- اجرای دوباره دستور status پس از گذشت 10 ثانیه و تبدیل پردازش به zombie

```
status
[Status] Active Containers:

[Status] Zombie Containers:
PID: 18967      Hostname: mmd
```

- تست دستور restart. ابتدا پردازش قبلی را restart کرده و مشاهده می کنیم که در حال اجراست و پس از گذشت 10 ثانیه با استفاده از status مشاهده می کنیم که به zombie تبدیل شده است.

```
[Command] Enter [status, terminate <PID>, restart <PID>, exit]: restart 18967
[Restart] Replaced PID 18967 with PID 19772 (Hostname: mmd)
[Command] Enter [status, terminate <PID>, restart <PID>, exit]: Initializing container...
Setting up root directory...
Changing root to: ./containers/folder_1750964163
Attempting to run: /test
test starting ...
status
[Status] Active Containers:
PID: 19772      Hostname: mmd

[Status] Zombie Containers:
PID: 18967      Hostname: mmd

[Command] Enter [status, terminate <PID>, restart <PID>, exit]: test ends ...

[parent] Child exited with code 0
status
[Status] Active Containers:

[Status] Zombie Containers:
PID: 18967      Hostname: mmd
PID: 19772      Hostname: mmd
```

- تست دستور terminate. ابتدا پردازش قبلی را restart کرده و مشاهده می کنیم که عبارت آغاز تست چاپ شده است. اما قبل از تمام شدن کار پردازش (پیش از گذشت 10 ثانیه) آن را terminate می کنیم و

مشاهده میشود که عبارت انتهایی تست چاپ نمیشود و همچنین در دستور status نیز مشاهده میشود که جز پرده های zombie قرار گرفته است.

```
[Command] Enter [status, terminate <PID>, restart <PID>, exit]: restart 18967
[Restart] Replaced PID 18967 with PID 20575 (Hostname: mmd)
[Command] Enter [status, terminate <PID>, restart <PID>, exit]: Initializing container...
Setting up root directory...
Changing root to: ./containers/folder_1750964186
Attempting to run: /test
test starting ...
terminate 20575
[Terminate] Process 20575 killed.
[Command] Enter [status, terminate <PID>, restart <PID>, exit]: status
[Status] Active Containers:

[Status] Zombie Containers:
PID: 18967      Hostname: mmd
PID: 19772      Hostname: mmd
PID: 20575      Hostname: mmd
```

- به ازای هر پرده یک فایل جداگانه در مسیر containers ساخته میشود که در آن اطلاعات مورد نیاز و همچنین library ها و دیگر مواردی همچون فایل باینری قرار میگیرند که هر محفظه را از دیگر بخش های سیستم جدا میکند.

```
[Manager] Shutdown complete.
mohammadhossein@mohammadhossein-ThinkBook-15-G2-ITL:~/Desktop/OS$ sudo ls ./containers/folder_1750964186
bin dev etc home lib lib64 media mnt proc root run sbin srv sys test tmp usr var
mohammadhossein@mohammadhossein-ThinkBook-15-G2-ITL:~/Desktop/OS$ ls containers
folder_1750963445 folder_1750963515 folder_1750963962 folder_1750964018 folder_1750964078 folder_1750964116 folder_1750964163
folder_1750963468 folder_1750963532 folder_1750964002 folder_1750964046 folder_1750964095 folder_1750964142 folder_1750964186
mohammadhossein@mohammadhossein-ThinkBook-15-G2-ITL:~/Desktop/OS$
```

## آزمایش محدودیت منابع

ابتدا یک محفظه جدید ساخته و محدودیت ها را با استفاده از فلگ های مذکور اعمال میکنیم. سپس از مسیر sys/fs/cgroup/folder\_number موارد مورد نظر را مشاهده میکنیم که به درستی نوشته شده باشند.

```
mohammadhossein@mohammadhossein-ThinkBook-15-G2-ITL:~/Desktop/OS$ sudo ./container -x ./test -n mmd -m 33554432 -i 50 -o 25 -u 25000
[Manager] PID 60221 started.
[Manager] Container started (PID: 60222, Hostname: mmd)
[Command] Enter [status, terminate <PID>, restart <PID>, exit]: Initializing container...
Setting up root directory...
Changing root to: ./containers/folder_1750965561
Attempting to run: /test
test starting ...
test ends ...

[parent] Child exited with code 0
exit
[Wait] Waiting for containers to finish...
[Manager] Shutdown complete.
mohammadhossein@mohammadhossein-ThinkBook-15-G2-ITL:~/Desktop/OS$ cd /sys/fs/cgroup/folder_1750965561
mohammadhossein@mohammadhossein-ThinkBook-15-G2-ITL:/sys/fs/cgroup/folder_1750965561$ cat cpu.max
250000 1000000
mohammadhossein@mohammadhossein-ThinkBook-15-G2-ITL:/sys/fs/cgroup/folder_1750965561$ cat memory.m
memory.max memory.min
33554432
mohammadhossein@mohammadhossein-ThinkBook-15-G2-ITL:/sys/fs/cgroup/folder_1750965561$ cat memory.max
33554432
mohammadhossein@mohammadhossein-ThinkBook-15-G2-ITL:/sys/fs/cgroup/folder_1750965561$ cat io.max
8:0 rbps=max wbps=max riops=50 wiops=25
mohammadhossein@mohammadhossein-ThinkBook-15-G2-ITL:/sys/fs/cgroup/folder_1750965561$
```

## نقد و بررسی سیستم:

چند مورد از مشکلاتی که در پیاده سازی پروژه با آنها مواجه شدم در ادامه ذکر میشوند.

### مشکلات

#### مشکل cgroup version

در ابتدا کدی که پیاده سازی کردم برای اعمال محدودیت ها از V1 استفاده میکرد درحالیکه سیستم عامل انتظار دستورات V2 را داشت. به همین دلیل ارور های عدم یافتن فایل یا permission denied دریافت میکردم تا در نهایت در دیباگ برنامه متوجه مشکل ورژن دستورات شدم و آن ها را مطابق به V2 آپدیت کردم.

#### افزودن تمامی dependency ها

همانطور که در قسمت setup root directory توضیح داده شد پس از کپی کردن کتابخانه های مورد نیاز همچنان با ارور lib/x86\_64-linux-gnu: No such file or directory/ مواجه میشدم که در نهایت با جست و جو در اینترنت یک فایل با نام alpine-minirootfs-3.7.0-x86\_64.tar یافتم که کتابخانه های مورد نظر در آن وجود داشت که پس از استفاده از آن مشکل این قسمت نیز رفع گردید.

#### نصب eBPF

برای انجام این نیازمندی پروژه تلاش های بسیاری انجام دادم و کامندهای مختلفی را برای نصب آن استفاده کردم که از apt , pip استفاده میکردند. همچنین تلاش کردم تا مستقیماً از زبان cpp برای استفاده از این تکنولوژی استفاده کنم که موفق نشدم. بعد از آن حتی تلاش برای استفاده از این تکنولوژی در زبان python نیز موفقیت آمیز نبود. در نهایت حتی تلاش کردم تا از ریپازیتوری های پابلیک موجود استفاده کنم و clone کردن آنها من را در نصب و راه اندازی این تکنولوژی یاری کند که باز هم موفق نبودم و در نهایت قادر به انجام این بخش نشدم.

### پیشنهادهات

#### اجرای چند برنامه

میتوان برنامه را به صورتی بهبود داد که به صورت همزمان چند برنامه را دریافت کرده و در محفظه های جداگانه اجرا کند.

## تبدیل کردن برنامه به حالت stateful

یکی از کارهایی که در جهت بهبود میتوان انجام داد این است که استتیت اجرای برنامه در هر محفظه را ذخیره سازی کرد تا در صورت خروج از برنامه بتوان با اجرای مجدد آن برنامه را از سرگیری کرد. برای مثال مقادیر متغیرها و وضعیت اجرایی برنامه ذخیره شود.

## افزودن دستورات بیشتر

دستوراتی همچون pause , resume نیز میتوان به برنامه اضافه کرد که اجرای یک برنامه را برای لحظاتی متوقف کنند و دوباره از سرگیری شوند. همچنین میتوان دستوراتی همچون sleep زماندار را برای برنامه ها در نظر گرفت که برای یک بازه زمانی یک pid مورد نظر به خواب برود.