سیستم مینی کانتینر - گزارش فنی

۱. مقدمه

این گزارش فنی پیاده‌سازی سیستم مینی کانتینر مشابه داکر را توصیف می‌کند که به عنوان پروژه درس سیستم‌عامل طراحی شده است. سیستم از C به C++ تبدیل شده و مفاهیم اصلی سیستم‌عامل شامل ایزولاسیون فرایند، مدیریت منابع هسته و فراخوانی‌های سیستمی را نشان می‌دهد، در حالی که روش‌های برنامه‌نویسی مدرن C++ را به نمایش می‌گذارد.

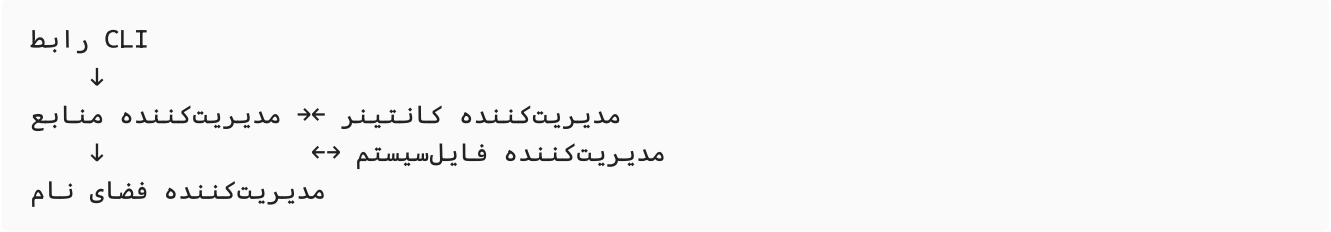
۲. معماری سیستم

سیستم به کامپوننت‌های مدولار سازماندهی شده است که هر کدام جنبه خاصی از کانتینری‌سازی را مدیریت می‌کنند. پیاده‌سازی C++ API سازگار با C را برای عملیات سطح سیستم حفظ می‌کند در حالی که ویژگی‌های مدرن C++ را برای بهبود مدیریت حافظه و ایمنی کد معرفی می‌کند.

۲.۱ کامپوننت‌های اصلی

- **مدیریت‌کننده کانتینر**: عملیات چرخه حیات کانتینر را هماهنگ می‌کند
- **مدیریت‌کننده فضای نام**: فضای نام لینوکس را برای ایزولاسیون فرایند مدیریت می‌کند
- **مدیریت‌کننده منابع**: محدودیت‌های CPU و حافظه را با استفاده از cgroups کنترل می‌کند
- **مدیریت‌کننده فایل‌سیستم**: ایزولاسیون فایل‌سیستم را از طریق pivot_root فراهم می‌کند
- **رابط CLI**: رابط خط فرمان برای تعامل کاربر
- **وب سرور**: رابط وب برای مانیتورینگ کانتینرها

۲.۲ تعامل کامپوننت‌ها



۳.ایزولاسیون فرایند

۳.۱ مفاهیم فضای نام

- **فضای نام PID (namespace)**: شناسه فرایندها را ایزوله می‌کند و به هر کانتینر درخت فرایند خود را از PID ۱ می‌دهد
- **فضای نام Mount (namespace)**: نقاط mount فایل‌سیستم را ایزوله می‌کند و امکان جداول mount خصوصی را می‌دهد
- **فضای نام Network (namespace)**: رابط‌های شبکه، جداول مسیریابی و قوانین فایروال را ایزوله می‌کند
- **فضای نام User (namespace)**: نگاشت‌های شناسه کاربر و گروه را ایزوله می‌کند

۳.۲ جزئیات پیاده‌سازی

مدیریت‌کننده فضای نام از فراخوانی سیستمی `()clone` با پرچم‌های فضای نام استفاده می‌کند:

```
pid_t pid = clone(child_func, stack, CLONE_NEWPID | CLONE_NEWNS, args);
```

جنبه‌های کلیدی پیاده‌سازی:

- **تخصیص پشته**: فرایندهای فرزند نیاز به فضای پشته خود دارند
- **تنظیم فضای نام**: تابع فرزند ایزولاسیون را پس از ایجاد فضای نام پیکربندی می‌کند
- **فضای نام PID**: ایزولاسیون شناسه فرایند از PID ۱ فراهم می‌کند
- **فضای نام Mount**: امکان نمایش‌های خصوصی فایل‌سیستم را می‌دهد

۳.۳ پیوستن به فضای نام

برای اجرای دستورات در کانتینرهای موجود، سیستم به فضای نام موجود می‌پیوندد:

```
int namespace_join(pid_t target_pid, int ns_type) {
}
```

۴. گروه‌های کنترل (cgroups) - مدیریت منابع

۴.۱ مفاهیم Cgroup

گروه‌های کنترل محدودیت‌های سلسله مراتبی منابع و حسابداری فراهم می‌کنند:

- **کنترل CPU**: استفاده از CPU را از طریق shares و quotas محدود می‌کند
- **کنترل حافظه**: استفاده از حافظه را با محدودیت‌های سخت و کنترل swap محدود می‌کند
- **سلسله مراتب**: ساختار درختی امکان کنترل منابع تو در تو را می‌دهد
- **حسابداری**: استفاده از منابع را برای هر گروه ردیابی می‌کند

۴.۲ جزئیات پیاده‌سازی

مدیریت‌کننده منابع با فایل‌سیستم `sys/fs/cgroup/` تعامل می‌کند:

```
// create cgroup
mkdir("/sys/fs/cgroup/cpu/mini_container/container_id", 0755);

// setup CPU shares
FILE *fp = fopen("/sys/fs/cgroup/cpu/mini_container/container_id/cpu.shares", "w");
fprintf(fp, "%d\n", shares);

// add process to cgroup
fp = fopen("/sys/fs/cgroup/cpu/mini_container/container_id/tasks", "w");
fprintf(fp, "%d\n", pid);
```

۴.۳ محدودیت‌های منابع

- **CPU Shares:** وزن نسبی برای زمان‌بندی CPU (پیش‌فرض: ۱۰۲۴)
- **CPU Quota:** محدودیت‌های مطلق زمان CPU (میکروثانیه در هر دوره)
- **محدودیت‌های حافظه**: محدودیت‌های سخت حافظه به بایت
- **محدودیت‌های Swap**: کنترل فضای swap

۵. ایزولاسیون فایل‌سیستم

۵.۱ Pivot Root

سیستم از `pivot_root()` برای ایزولاسیون فایل‌سیستم استفاده می‌کند که روش مدرن‌تر و امن‌تر است:

- **pivot_root()**: امکان unmount کردن فایل‌سیستم ریشه قدیمی را می‌دهد
  - **امنیت بیشتر**: نسبت به chroot امن‌تر است
  - **کنترل بهتر**: امکان مدیریت کامل فایل‌سیستم ریشه

۵.۲ پیاده‌سازی

```
int fs_setup_pivot_root(const char *new_root, const char *put_old) {
    // mount new root
    mount(new_root, new_root, "bind", MS_BIND | MS_REC, NULL);

    // create put_old directory
    mkdir(put_old, 0755);

    // pivot root
    syscall(SYS_pivot_root, new_root, put_old);

    // change to new root
    chdir("/");

    // unmount old root
    umount2(put_old, MNT_DETACH);
    rmdir(put_old);
}
```

۵.۳ ایجاد فایل‌سیستم ریشه

سیستم فایل‌سیستم‌های ریشه کانتینر حداقل ایجاد می‌کند:

- دایرکتوری‌های ضروری: `/dev` ، `/sys` ، `/proc` ، `/lib` ، `bin`
- گره‌های دستگاه: `/dev/random` ، `/dev/zero` ، `dev/null`
- فایل‌های پیکربندی پایه: `/etc/group` ، `etc/passwd`

۶. فراخوانی‌های سیستمی و تعامل با هسته

۶.۱ فراخوانی‌های سیستمی اصلی

- **clone()**: فرایندهای جدید را در فضای نام ایجاد می‌کند
- **unshare()**: فرایندهای موجود را به فضای نام جدید منتقل می‌کند
- **pivot_root()**: تعویض ریشه پیشرفته و امن
- **mount()**: عملیات mount فایل‌سیستم

۶.۲ رابط مستقیم هسته

بر خلاف داکر، این سیستم به طور مستقیم با موارد زیر تعامل می‌کند:

- برای مدیریت منابع `/sys/fs/cgroup`
- برای عملیات فضای نام `/proc/<pid>/ns/`

- فراخوانی‌های سیستمی بدون تجرید سطح بالا

۷. مدیریت چرخه حیات کانتینر

۷.۱ وضعیت‌های کانتینر

- **CREATED**: کانتینر تعریف شده اما در حال اجرا نیست
- **RUNNING**: فرایند کانتینر فعال است
- **STOPPED**: فرایند کانتینر خاتمه یافته
- **DESTROYED**: منابع کانتینر پاکسازی شده

۷.۲ عملیات چرخه حیات

- **Create**: تنظیم cgroups، فضای نام، فایل‌سیستم
- **Run**: ایجاد و راه‌اندازی بلافاصله کانتینر (Create + Start)
- **Start**: راه‌اندازی فرایند کانتینر با ایزولاسیون (برای کانتینرهای متوقف شده)
- **Stop**: خاتمه فرایند کانتینر به طور graceful
- **Destroy**: پاکسازی همه منابع

۷.۳ ذخیره‌سازی پیکربندی

سیستم پیکربندی کانتینرها را ذخیره می‌کند تا امکان restart کانتینرهای متوقف شده فراهم شود:

- **saved_config**: ذخیره می‌شود `container_info_t` پیکربندی کامل کانتینر در
- **Restart**: کانتینرهای متوقف شده می‌توانند با همان پیکربندی قبلی دوباره راه‌اندازی شوند