سیستم مینی کانتینر - مستندات API

نمای کلی

این مستندات API پیاده‌سازی سی‌پلاس‌پلاس سیستم مینی کانتینر را شامل می‌شود. جهت مطالعه فایل README را ابتدا مطالعه بفرمایید.

مدیریت کانتینتر ها (container manager)

مقداردهی اولیه و پاکسازی

```
int container_manager_init(container_manager_t *cm, int max_containers);
void container_manager_cleanup(container_manager_t *cm);
```

مدیریت‌کننده کانتینر را با حداکثر تعداد کانتینرها مقداردهی کنید.

چرخه حیات کانتینر

```
int container_manager_create(container_manager_t *cm, const container_config_t
*config);
int container_manager_run(container_manager_t *cm, container_config_t *config);
int container_manager_start(container_manager_t *cm, const char *container_id);
int container_manager_stop(container_manager_t *cm, const char *container_id);
int container_manager_destroy(container_manager_t *cm, const char *container_id);
```

عملیات اساسی چرخه حیات برای کانتینرها.

توضیحات:

- `container_manager_create` : کانتینر را ایجاد می‌کند اما اجرا نمی‌کند (state: CREATED)
- `container_manager_run` : کانتینر را ایجاد و بلافاصله اجرا می‌کند (state: RUNNING)
- `container_manager_start` : کانتینر متوقف شده را دوباره اجرا می‌کند
- `container_manager_stop` : کانتینر در حال اجرا را متوقف می‌کند (state: STOPPED)
- `container_manager_destroy` : کانتینر را حذف می‌کند (state: DESTROYED)

عملیات کانتینر

```
int container_manager_exec(container_manager_t *cm, const char *container_id, char
**command, int argc);
container_info_t **container_manager_list(container_manager_t *cm, int *count);
container_info_t *container_manager_get_info(container_manager_t *cm, const char
*container_id);
```

دستورات را در کانتینرها اجرا کرده و اطلاعات کانتینر را دریافت کنید.

مدیریت namespace ها

پیکربندی

```
void namespace_config_init(namespace_config_t *config);
```

تنظیمات فضای نام را پیکربندی کنید.

ایجاد فرایند

```
pid_t namespace_clone_process(int flags, void *child_stack, int stack_size,
                              void (*child_func)(void *), void *arg);
int namespace_setup_isolation(const namespace_config_t *config);
```

فرایندهای ایزوله ایجاد کرده و ایزولاسیون فضای نام را تنظیم کنید.

عملیات های namespace

```
int namespace_join(pid_t target_pid, int ns_type);
pid_t namespace_create_container(const namespace_config_t *config,
                                 char **command, int argc);
```

به فضای نام‌های موجود بپیوندید و فرایندهای کانتینر ایجاد کنید.

مدیریت منابع (resource manager)

مقداردهی اولیه

```
int resource_manager_init(resource_manager_t *rm, const char *base_path);
void resource_limits_init(resource_limits_t *limits);
```

مدیریت منابع را مقداردهی کرده و محدودیت‌های پیش‌فرض را تنظیم کنید.

عملیات Cgroup

```
int resource_manager_create_cgroup(resource_manager_t *rm, const char *container_id,
                                   const resource_limits_t *limits);
int resource_manager_add_process(resource_manager_t *rm, const char *container_id,
pid_t pid);
int resource_manager_remove_process(resource_manager_t *rm, const char *container_id,
pid_t pid);
int resource_manager_destroy_cgroup(resource_manager_t *rm, const char
*container_id);
```

گروه‌های کنترل را برای کنترل منابع مدیریت کنید.

آمار

```
int resource_manager_get_stats(resource_manager_t *rm, const char *container_id,
                               unsigned long *cpu_usage, unsigned long *memory_usage);
```

آمار استفاده از منابع را دریافت کنید.

مدیریت فایل‌سیستم (file system manager)

پیکربندی

```
void fs_config_init(fs_config_t *config);
```

پیکربندی فایل‌سیستم را مقداردهی کنید.

عملیات فایل‌سیستم ریشه

```
int fs_create_minimal_root(const char *root_path);
int fs_populate_container_root(const char *root_path, const char *host_root);
int fs_cleanup_container_root(const char *root_path);
```

فایل‌سیستم‌های ریشه کانتینر را ایجاد و مدیریت کنید.

روش‌های ایزولاسیون

```
int fs_setup_pivot_root(const char *new_root, const char *put_old);
int fs_mount_container_filesystems(const char *root_path);
```

ایزولاسیون فایل‌سیستم را با استفاده از pivot_root تنظیم کنید. `pivot_root` روش مدرن‌تر و امن‌تر برای ایزوله کردن فایل‌سیستم است که امکان unmount کردن root قدیمی را فراهم می‌کند.

ساختارهای داده

پیکربندی کانتینر

```
typedef struct {
    char *id;
    char *root_path;
    namespace_config_t ns_config;
    resource_limits_t res_limits;
    fs_config_t fs_config;
    char **command;
    int command_argc;
} container_config_t;
```

پیکربندی کانتینر شامل تمام تنظیمات ایزولاسیون.

محدودیت‌های منابع

```
typedef struct {
    struct {
        int shares;
        int quota_us;
        int period_us;
    } cpu;
    struct {
        unsigned long limit_bytes;
        unsigned long swap_limit_bytes;
    } memory;
    int enabled;
} resource_limits_t;
```

محدودیت‌های منابع CPU و حافظه.

اطلاعات کانتینر

```
typedef struct {
    char *id;
    pid_t pid;
    container_state_t state;
    time_t created_at;
    time_t started_at;
    time_t stopped_at;
    container_config_t *saved_config;  // ذخیره config برای restart
} container_info_t;
```

اطلاعات زمان اجرا درباره کانتینرها.

فیلدها:

- `id` : شناسه یکتا کانتینر
- `pid` : Process ID کانتینر (۰ اگر اجرا نشده باشد)
- `state` : وضعیت فعلی کانتینر (CREATED, RUNNING, STOPPED, DESTROYED)
- `created_at` : زمان ایجاد کانتینر
- `started_at` : زمان شروع اجرای کانتینر (۰ اگر هرگز اجرا نشده باشد)
- `stopped_at` : زمان توقف کانتینر (۰ اگر متوقف نشده باشد)
- `saved_config` : پیکربندی ذخیره شده برای امکان restart کانتینر

مقدار دهی ها و کلاس بندی ها

مدیریت Namespace

```
enum {
    NS_PID = CLONE_NEWPID,
    NS_MNT = CLONE_NEWNS,
    NS_NET = CLONE_NEWNET,
    NS_USER = CLONE_NEWUSER
};
```

حالت کانتینرها

```
typedef enum {
    CONTAINER_CREATED,
    CONTAINER_RUNNING,
    CONTAINER_STOPPED,
    CONTAINER_DESTROYED
} container_state_t;
```

متدهای فایل سیستم

```
typedef enum {
    FS_PIVOT_ROOT
} fs_isolation_method_t;
```

نکته: `FS_PIVOT_ROOT` روش پیشنهادی و استاندارد برای ایزولاسیون فایل‌سیستم است که امنیت و انعطاف‌پذیری بیشتری نسبت به روش‌های قدیمی‌تر دارد.

API وب سرور (Web Server API)

سیستم شامل یک وب سرور ساده برای مانیتورینگ کانتینرها از طریق رابط وب است.

کلاس SimpleWebServer

```
class SimpleWebServer {
public:
    SimpleWebServer(container_manager_t* cm, int port = 808);
    ~SimpleWebServer();
    void start();
    void stop();
};
```

Endpoints HTTP

GET `/` یا `/index.html`

صفحه اصلی مانیتورینگ کانتینرها را برمی‌گرداند. این صفحه شامل:

- لیست تمام کانتینرهای فعال
- نمودارهای CPU و Memory برای هر کانتینر
- به‌روزرسانی خودکار هر ۱۰ ثانیه

Response: HTML page

GET `/api/containers`

لیست تمام کانتینرهای فعال را به صورت JSON برمی‌گرداند.

Response:

```
{
  "containers": [
    {
      "id": "container_id",
      "pid": 12345,
      "state": "RUNNING",
      "cpu_usage": 1000000000,
      "cpu_limit": 25000,
      "cpu_percent": 25.5,
      "memory_usage": 67108864,
      "memory_limit": 134217728,
      "memory_percent": 50.0
    }
  ]
}
```

فیلدها:

- `id` : شناسه کانتینر
- `pid` : Process ID

- `state` : وضعیت کانتینر (CREATED, RUNNING, STOPPED, DESTROYED)
- `cpu_usage` : استفاده CPU به نانوثانیه (فقط برای RUNNING)
- `cpu_limit` : محدودیت CPU quota_us (فقط برای RUNNING)
- `cpu_percent` : درصد استفاده CPU (فقط برای RUNNING)
- `memory_usage` : استفاده حافظه به بایت (فقط برای RUNNING)
- `memory_limit` : محدودیت حافظه به بایت (فقط برای RUNNING)
- `memory_percent` : درصد استفاده حافظه (فقط برای RUNNING)

GET `/api/system`

اطلاعات سیستم را به صورت JSON برمی‌گرداند.

Response:

```
{
  "used_memory": 2147483648,
  "total_memory": 8589934592,
  "cpu_percent": 45.3
}
```

فیلدها:

- `used_memory` : حافظه استفاده شده سیستم (بایت)
- `total_memory` : کل حافظه سیستم (بایت)
- `cpu_percent` : درصد استفاده CPU سیستم

مدیریت Signal و Graceful Shutdown

سیستم از signal handler برای graceful shutdown استفاده می‌کند.

Signal Handler

```
void signal_handler(int signum);
```

پشتیبانی از Signals:

- `SIGINT` (Ctrl+C): توقف همه کانتینرها و خروج از برنامه
- `SIGTERM` : توقف همه کانتینرها و خروج از برنامه

رفتار:

۱. همه کانتینرهای RUNNING را با SIGTERM متوقف می‌کند
۲. صبر می‌کند تا processes به صورت graceful terminate شوند
۳. کانتینرهای RUNNING را به حالت STOPPED تغییر می‌دهد
۴. کانتینرهای CREATED را به حالت STOPPED تغییر می‌دهد
۵. اگر process هایی باقی مانده باشند، آنها را با SIGKILL kill می‌کند
۶. state نهایی را ذخیره می‌کند
۷. وب سرور را متوقف می‌کند
۸. منابع را آزاد می‌کند و برنامه را می‌بندد

توابع کمکی برای تست

تست مموری و CPU

```
void run_memory_cpu_test();
```

این تابع برای تست سیستم استفاده می‌شود و:

- ۳ کانتینر برای تست مموری ایجاد می‌کند (C۱MEM, C۲MEM, C۳MEM)
- C۱MEM: ۱/۱۶ کل مموری سیستم
- C۲MEM: ۱/۸ کل مموری سیستم
- C۳MEM: ۱/۴ کل مموری سیستم
- ۳ کانتینر برای تست CPU ایجاد می‌کند (C۱CPU, C۲CPU, C۳CPU)
- C۱CPU: ۱/۱۶ CPU (quota: ۶۲۵۰, period: ۱۰۰۰۰۰)
- C۲CPU: ۱/۸ CPU (quota: ۱۲۵۰۰, period: ۱۰۰۰۰۰)
- C۳CPU: ۱/۴ CPU (quota: ۲۵۰۰۰, period: ۱۰۰۰۰۰)

توابع کمکی سیستم

```
static unsigned long get_total_system_memory();
static int get_cpu_count();
```

- `get_total_system_memory()` : کل مموری سیستم را به بایت برمی‌گرداند
- `get_cpu_count()` : تعداد CPU cores را برمی‌گرداند

مثال‌های استفاده

ایجاد و اجرای کانتینر

```
container_config_t config;
namespace_config_init(&config.ns_config);
resource_limits_init(&config.res_limits);
fs_config_init(&config.fs_config);

config.id = strdup("my_container");
config.res_limits.memory.limit_bytes = 128 * 1024 * 1024; // 128 MB
config.res_limits.cpu.shares = 1024;
config.fs_config.root_path = strdup("/");

char *args[] = {"/bin/sh", "-c", "echo Hello World", nullptr};
config.command = args;
config.command_argc = 3;

if (container_manager_run(&cm, &config) == 0) {
    printf("Container created and started\n");
}
```

دریافت اطلاعات کانتینر

```
container_info_t *info = container_manager_get_info(&cm, "my_container");
if (info) {
    printf("Container ID: %s\n", info->id);
    printf("State: %d\n", info->state);
    printf("PID: %d\n", info->pid);

    if (info->state == CONTAINER_RUNNING) {
        unsigned long cpu_usage = 0, memory_usage = 0;
        resource_manager_get_stats(cm.rm, info->id, &cpu_usage, &memory_usage);
        printf("CPU Usage: %lu ns\n", cpu_usage);
        printf("Memory Usage: %lu bytes\n", memory_usage);
    }
}
```

توقف و راه‌اندازی مجدد کانتینر

```
if (container_manager_stop(&cm, "my_container") == 0) {
    printf("Container stopped\n");
}

if (container_manager_start(&cm, "my_container") == 0) {
    printf("Container restarted\n");
}
```