Linux
システムプログラミング
入門

自己紹介

酒井　蓮耀（さかい　れんき）

https://twitter.com/harrowHell

東京大学の学部2年・理学部

セキュリティキャンプ全国参加は2016年

思ったことや質問など

- Slido

  匿名で質問やアンケートができます #051806

- Discordの実況チャンネルか講義チャンネル

  すぐに解決しなさそうな質問は講義チャンネルの方がいいかも？

- 困ったことがあれば可能な限り講師やチューターが助けます

  あなたが疑問に思ったことは他の人も疑問に思っている可能性が高いです.

slido

Join at slido.com #051806

ⓘ Start presenting to display the joining instructions on this slide.

雑談

好きな言語は？

slido

好きなプログラミング言語

ⓘ Start presenting to display the poll results on this slide.

雑談

好きなOSは？

slido

好きなOSは？

ⓘ Start presenting to display the poll results on this slide.

目標

- システムプログラミングに親しむ・楽しさを知ってもらう

  Linuxシステムがカーネルと連携して動作する仕組みについて理解することはシステム構築に役立つ

  どこを弄れば何ができるのか・どうやって動いているのか

- Linuxの名前空間について学ぶ

  名前空間を利用しているプログラム，コンテナなどを利用する時のトラブルシューティングに役立つ

- 環境の分離の必要性を知る

  あるプログラムが他のプログラムに与えうる影響について知る

今日やること

- 講義
  - 環境の分離の必要性
  - 何が問題か
  - 名前空間の概要
- 演習
  - プロジェクトの準備
  - プロセスを開始する関数
  - 名前空間を使用するように設定
  - ユーザ名前空間の準備：マッピング
  - ネットワーク名前空間の準備：NATやルーティング設定
  - PID名前空間の準備：procfsのマウント
  - 使ってみる
- まとめ

環境の分離の必要性

コンテナを使うと何が嬉しいか？

- 早く開発できる
  - 他のアプリケーションに影響をほぼ与えないで開発できる
- 早くデプロイできる
  - 簡単に数を増やしてスケールアウトできる
- ハードに依存しない
  - VM構成などに依存せずにスケールアップできる
- セキュリティ被害の最小化
  - 攻撃が成功した場合でも影響を減らす

環境の分離の必要性

つまり，ほかのアプリケーションに影響を与えない，ということが重要

太古(?)の分離機能chroot

仮想的に，あるディレクトリをルートディレクトリとすることで，そのディレクトリの外にアクセスできないようにする機能

とても古い(1979)

FTPサーバなどでセキュリティのために利用されてきた

chrootを使ってみる

chrootを試してみて，これでは何がいけないのか見てみる

演習環境にアクセスして，ターミナルを開く(SSHでもOK).

以下のコマンドを実行する

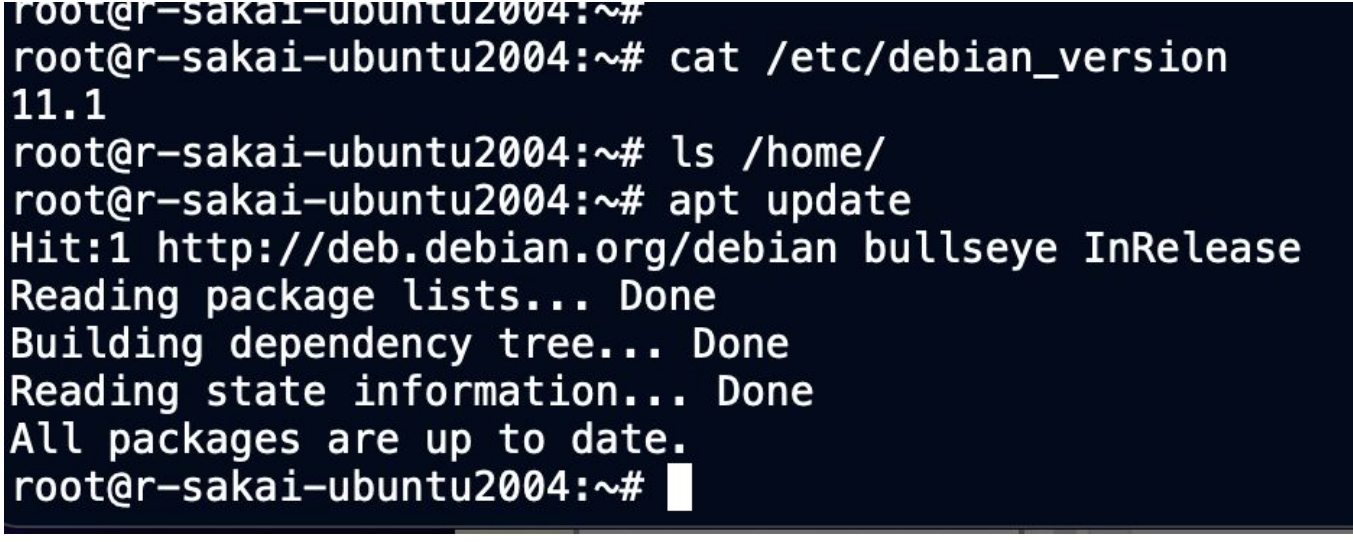
```
sudo tar xf sysroot-debian-bullseye.tar.xz
sudo cp /etc/resolv.conf ./sysroot-debian-bullseye/etc/
sudo chroot ./sysroot-debian-bullseye/
```

```
(base) minicamp@r-sakai-ubuntu2004:~$
(base) minicamp@r-sakai-ubuntu2004:~$ sudo tar xf sysroot-debian-bullseye.tar.xz
(base) minicamp@r-sakai-ubuntu2004:~$ sudo cp /etc/resolv.conf ./sysroot-debian-bullseye/etc/
(base) minicamp@r-sakai-ubuntu2004:~$ sudo chroot ./sysroot-debian-bullseye/
root@r-sakai-ubuntu2004:/#
```

chrootを使ってみる

続けて以下のようにコマンドを入力してみます．ぱっと見Debianに見えるし，これまで利用していたファイルは見えない

```
cat /etc/debian_version
ls /home/
apt update
```



chrootを使ってみる

しかし，以下のようにコマンドを入力してみると，元の環境の様々なものが見えてしまっていることがわかる

```
mount -t proc proc /proc/
ps -ax
```

```
root@r-sakai-ubuntu2004:~# mount -t proc proc /proc/
root@r-sakai-ubuntu2004:~# ps -ax
    PID TTY      STAT   TIME COMMAND
      1 ?        Ss     0:03 /sbin/init
      2 ?        S      0:00 [kthreadd]
      3 ?        I<     0:00 [rcu_gp]
      4 ?        I<     0:00 [rcu_par_gp]
```

chrootを使ってみる

しかし，以下のようにコマンドを入力してみると，元の環境の様々なものが見えてしまっていることがわかる

```
uname -a
ip addr
```

```
root@r-sakai-ubuntu2004:~# uname -a
Linux r-sakai-ubuntu2004 5.11.0-1022-gcp #24~20.04.1-Ubuntu SMP Thu Oct 21 16:03:19 UTC 2021 x
root@r-sakai-ubuntu2004:~# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc mq state UP group default qlen 1000
    link/ether 42:01:c0:a8:01:0d brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.13/32 scope global dynamic ens4
       valid_lft 2452sec preferred_lft 2452sec
    inet6 fe80::4001:c0ff:fea8:10d/64 scope link
```

chrootを使ってみる

chrootの中でこれらの資源にアクセスすると，それは別のchrootや親となる環境にも影響を与える

これを防ぐために，コンテナでは名前空間が活用されている

寄り道：仮想化技術

環境の分離の手段として，名前空間を利用したコンテナの他に，ハードウェアの仮想化技術を利用した仮想マシンがある

より強い分離が得られるが，柔軟性はコンテナの方が上

仮想化技術を利用したコンテナもある

名前空間の概要

名前空間はリソースを分離する機能

1つの名前空間に属するプロセスたちがリソースを共用する

プロセスごと，リソース種別ごとに設定できる

Dockerで使われているruncはこれを利用している

名前空間の種類

種類	分離するもの
Cgroup	Cgroup root directory
IPC	System V IPC, POSIX message queues
Network	Network devices, stacks, ports, etc.
Mount	Mount points
PID	Process IDs
Time	Boot and monotonic clocks
User	User and group IDs
UTS	Hostname and NIS domain name

ここまでのまとめ

コンテナの恩恵は分離されていることによるもの

OSの提供する環境はファイルだけではなくネットワークなど多い

これらを分離することが必要

名前空間を使うとこれらを分離できる

演習パート

ここからは，名前空間を実際に利用してコンテナもどきを作っていく

chrootに加えて名前空間を適用することで環境を分離する

今回の講義で(たぶん)一番楽しいところです！

演習パートの進め方

- 時間が書いていない所は，様子をみつつ大半が終わってそうであれば次にすすむ
- 時間が書いてある所は，時間が残り少なく(目安1/3)なってきたら講師がコーディングを進めるので，それを参考に仕上げてもらう

- 事前課題で書いたものと類似の部分が多いので，なるべく自力で書けるようトライしてください．
- 各段階が次のステップに必要なので，自力で書くのが難しそうであれば，広義のペースについていくことを優先してください．

基本的な構成

子プロセスを名前空間内に作る

親プロセスで名前空間の設定をする

設定が終わったら，子プロセスでchrootしてinitを実行する

子プロセスが終了したら親プロセスで後片付けをする

プロジェクトの準備(5分)

~/camp_container/にプロジェクトを準備する

netbeansでも良いしターミナルで作業してもOK

initを作る(5分)

```
sudo nano ~/sysroot-debian-bullseye/bin/init
```

以下の内容を書く

```
#! /bin/bash
source /etc/profile
mount -t proc proc /proc
mount -t sysfs sysfs /sys
set -m
/bin/bash --login
umount /proc
umount /sys
```

initを作る(5分)

保存する(Ctrl-O Ctrl-X)

実行できるように権限を与える

```
sudo chmod +x ~/sysroot-debian-bullseye/bin/init
```

chrootしてディレクトリを変更する関数を書く(5分)

```
int chroot_dir(const char*const path){
    if(chroot(path) != 0) return -1;
    if(chdir(“/”) != 0) return -1;
    return 0;
}
```

子プロセス作成とinit実行(30分)

親プロセスはまず子プロセスを作成(clone)する

子プロセスでは，chroot_dir(“/home/minicamp/sysroot-debian-bullseye”)を呼んだあと，先ほど作成したinitを実行(execve)する.

chrootされると，先ほど作成したinitのパスは/bin/initに見える

子プロセスが終了するまで親プロセスはwaitpidで待つ.

子プロセスを作成する関数はint start_child()

成功したら0，失敗したら-1を返す

実行してみよう

main関数でstart_child()を呼ぶ.

ビルドしてsudoをつけて実行しよう.

CUIの場合

```
cd build

cmake ..

make

sudo ./camp_container
```

シェルが実行されましたか？

実行してみよう

この状態で，以下のコマンドを実行する

```
ps

id

ip link
```

終わったら，Ctrl-Dを押すと終了する

PID名前空間を使ってみる

cloneするときのフラグにCLONE_NEWPIDをつける

ビルドして，sudo ./camp_container

その後以下のコマンドを実行する

```
ps -ax
```

終わったら，Ctrl-Dを押すと終了する

PID名前空間を使ってみる

プロセスIDが小さい番号になるはずです.

また，名前空間の外のプロセスは見えなくなっています.

killすることもできません.

ユーザ名前空間を使ってみる

cloneするときのフラグにCLONE_NEWUSERとCLONE_NEWNSを追加する

ビルドして，sudo ./camp_container

その後以下のコマンドを実行する

```
id
```

終わったら，Ctrl-Dを押すと終了する

ユーザ名前空間を使ってみる

ユーザIDやグループIDが65534(nobody)になっているのは，マッピングが作成されていないからです.

事前課題で扱ったように，/proc/pid/uid_mapなどに書き込んでマッピングを作成できます.

ファイルに何か書き込む関数を書く(10分)

指定したファイルに指定した内容を書き込む関数を書く

openやwriteを使う

成功したら0，失敗したら-1を返す

int write_file(const char*const path,const char* const str);

標準入出力にログなど書きたければ自由にどうぞ

マッピングを書き込む関数を書く(5分)

```
int write_ug_map(pid_t child) {
    char path[STR_BUF_SIZE], data[STR_BUF_SIZE];
    snprintf(path, STR_BUF_SIZE, "/proc/%d/setgroups", child);
    if (write_file(path, "deny") == -1) return -1;
    snprintf(path, STR_BUF_SIZE, "/proc/%d/gid_map", child);
    snprintf(data, STR_BUF_SIZE, "0 %d 1\n", getgid());
    if (write_file(path, data) == -1) return -1;
    snprintf(path, STR_BUF_SIZE, "/proc/%d/uid_map", child);
    snprintf(data, STR_BUF_SIZE, "0 %d 1\n", getuid());
    if (write_file(path, data) == -1) return -1;
    return 0;
}
```

親プロセスが準備している間待つようにする(15分)

start_childでパイプを作成する.

子プロセスに読み取り側のパイプのファイルディスクリプタを渡す.

子プロセスは開始されるとすぐにそのパイプをreadする. 何も書き込まれていないので子プロセスの処理はいったん止まる.

親プロセスは子プロセスを開始した後に`write_ug_map`を呼ぶ(準備).

終わったらパイプに何かを`write`する(ヌル文字など).

`write`されると，子プロセス側で`read`されるので，子プロセスの処理が再開

実行してみよう

先ほどと同じようにビルドする.

cd build

make

そのあと, sudoをつけずに実行してみる

./camp_container

様子を見よう

実行してみよう

続けて以下のコマンドを入力する

id

なぜかrootになっている？

終わったら，`Ctrl-D`を押すと終了する

ネットワーク名前空間を使ってみる

cloneするときのフラグにCLONE_NEWNETを追加する

ビルドして，sudo ./camp_container

その後以下のコマンドを実行する

```
ip link show
```

終わったら，Ctrl-Dを押すと終了する

ネットワーク名前空間を使ってみる

以下のようになるはず

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN mode DEFAULT group default qlen 1000

    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

ネットワーク名前空間に入ってネットワークデバイスが見えなくなった

コマンドを実行する関数を書く(25分)

指定したコマンドを実行する関数を書く

cloneとexecveなどを利用する

実行するファイルは”/bin/bash”で，"-c"と与えられたコマンド文字列を引数として与える

成功したら0を返して，失敗したら-1

execve(“/bin/bash”,{“/bin/bash”,”-c”,cmd,NULL},{NULL});　(イメージ)

int exec_command(const char*const cmd);

標準入出力はパイプで受け取っても良いしそのまま標準出力に出してもOK

ネットワークの設定(親プロセス)(10分)

以下のコマンドを実行するような関数int parent_install_network(pid_t child)を書く.
PID_CHILDは子プロセスのプロセスIDに置き換える.

ip link add vethA type veth peer vethB

ip link set dev vethA up

ip address add 10.0.0.1/24 dev vethA

ip link set dev vethB netns PID_CHILD

ヒント：snprintf(cmd, STR_BUF_SIZE, "ip link set dev vethB netns %d",child);

ネットワークの設定(子プロセス)(5分)

以下のコマンドを実行するような関数int child_install_network()を書く.

ip link set dev vethB up

ip address add 10.0.0.2/24 dev vethB

ip route add default via 10.0.0.1

ネットワークの片付け(親プロセス)(5分)

以下のコマンドを実行するような関数int parent_uninstall_network()を書く.

ip link delete vethA

ネットワーク関連の処理を実行するようにする(10分)

親プロセスで，parent_write_ug_mapを実行した後にparent_install_networkも呼ぶ.

子プロセスはchroot_dirを実行した後にchild_install_networkも呼ぶ.

子プロセスが終了したらparent_uninstall_networkを呼ぶ

iptablesの設定

以下のコマンドを実行する(事前課題と同じ)

sudo iptables --table nat --append POSTROUTING --source 10.0.0.0/24 --out-interface ens4 --jump MASQUERADE

sudo iptables --table filter --append FORWARD --source 10.0.0.0/24 --jump ACCEPT

sudo iptables --table filter --append FORWARD --destination 10.0.0.0/24 --match conntrack --ctstate ESTABLISHED,RELATED --jump ACCEPT

実行してみよう

先ほどと同じようにビルドする.

cd build

make

そのあと実行してみる

sudo ./camp_container

実行してみよう

続けて以下のコマンドを入力する

ping 8.8.8.8 -c 4

成功すればインターネットに接続できていることを示す

終わったら，Ctrl-Dを押すと終了する

さらに発展

ネットワークの設定にはroot権限が必要なので，ここを工夫すればroot権限なしでコンテナを実行できるようになる(rootlessコンテナ).

まとめ

- 名前空間を利用するとOSの提供する資源を分離できる
- これを利用して簡易的なコンテナを作った

一番言いたいこと

- Linuxのプロセスやシステムコールについて知り，プロセスのメモリ構造やシステムソフトウェアが動作するしくみを理解してほしい
- Linuxカーネルは楽しい

オレオレシステムコールも生やせます！

終わり

質問とかあればどうぞ