임베디드 시스템에서 권한 제한을 통한 가상 컨테이너 활용 방안

UST 석/박통합과정 인공지능학과 1학기
차주형, 권용인

UST
과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

IEMEK (사)대한임베디드공학회
Institute of Embedded Engineering of Korea

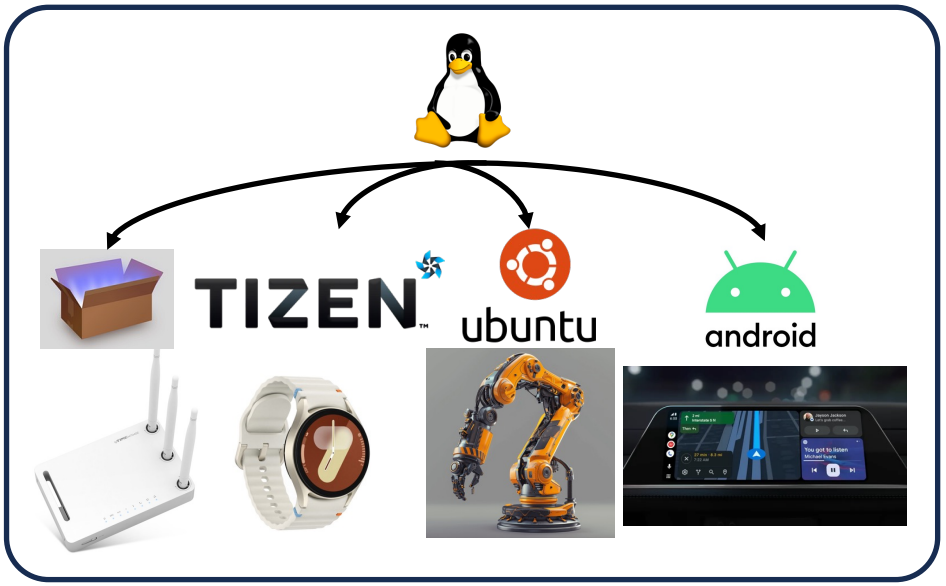
MOTIVATION

상용 임베디드에서 개발 환경 구성 및 실험의 어려움

문제 제기

1. 상용 임베디드 시스템에서 실험 및 연구를 위한 환경 세팅을 빠르게 할 수 없을까?
2. 가상 환경에서 실험한 결과를 실제 하드웨어에서 어떻게 테스트 해볼 수 있을까?
3. 임베디드에서 사용되고 있는 운영체제는 모두 동일한 구조를 지녔지만 어떤 차이가 있는가?

동일한 내부 구조



상용 임베디드 별 차이

운영체제 경량화로 인한 차이

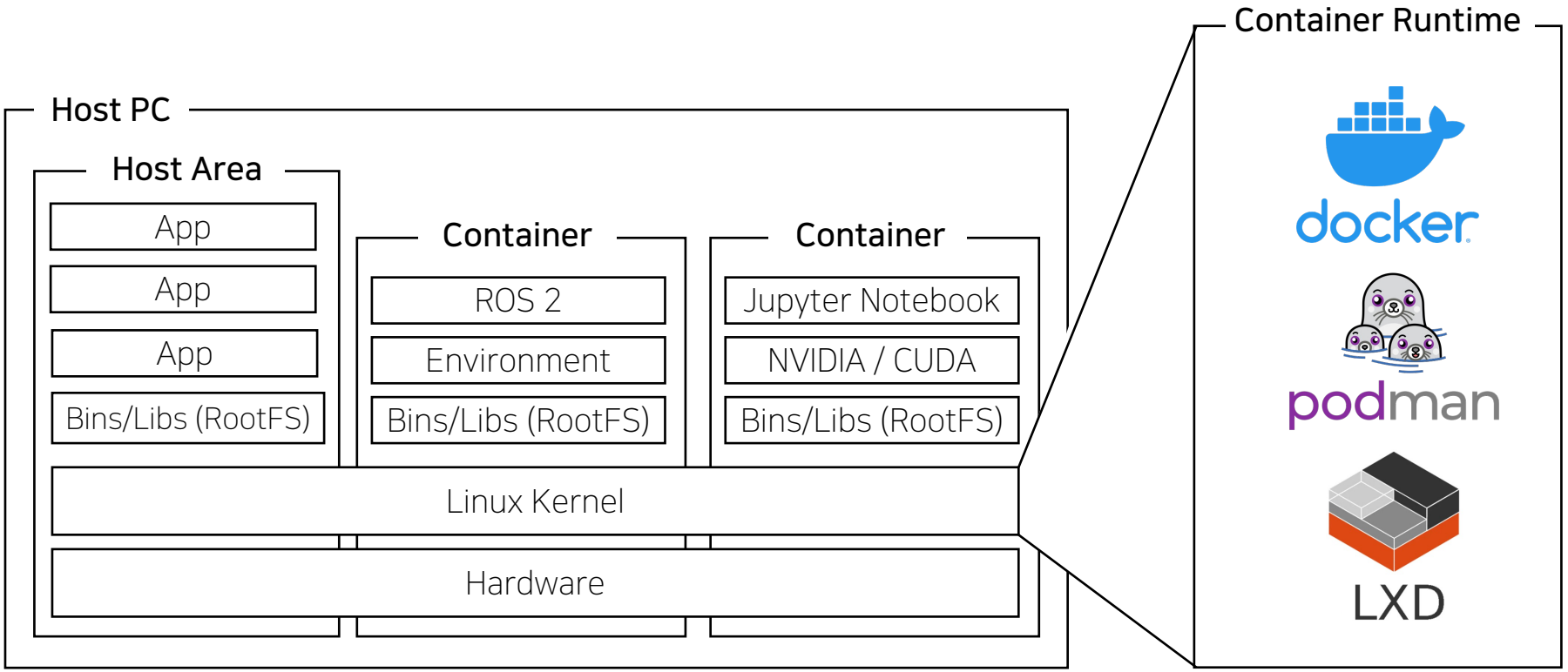
1. apt, yum 패키지 매니저
2. glibc : 표준 C++ 라이브러리
3. SSH, C++ : 개발 및 외부 통신

하드웨어로 인한 차이

1. 특정 하드웨어를 위한 전용 운영체제
2. 메모리와 스토리지 공간의 성능 차이

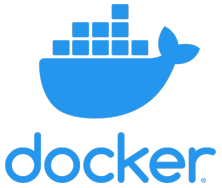
일관된 환경 구성을 위해 활용되는 컨테이너

1. 컨테이너는 어느 환경에서나 일관된 동작을 보장하여 널리 사용됨
2. HOST의 **하드웨어와 리눅스 커널은 공유**하지만, 운영체제(RootFS)는 별도로 사용함
3. 컨테이너 런타임들은 커널 수준에서 운영체제 가상화를 통해 컨테이너 관리함



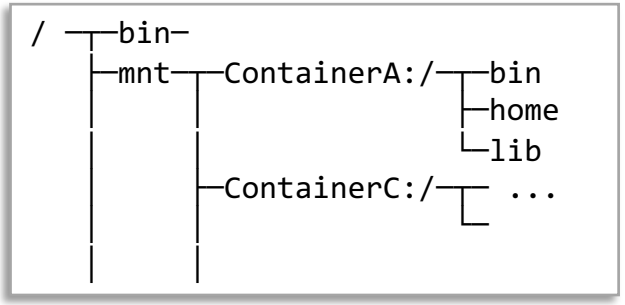
임베디드에 컨테이너 사용 방안 탐색

1. 대부분의 컨테이너 런타임은 Open Container Interface(OCI) 표준을 따름
2. 컨테이너 표준안(OCI) 에 Chroot, CGroups, Namespace가 모두 커널에서 제공 해야함
3. 즉, 모든 컨테이너는 표준에 따라 **커널 레벨**의 지원이 필요함.



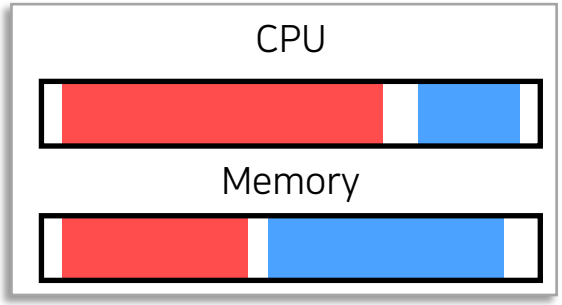
컨테이너

Chroot (저장소의 격리)

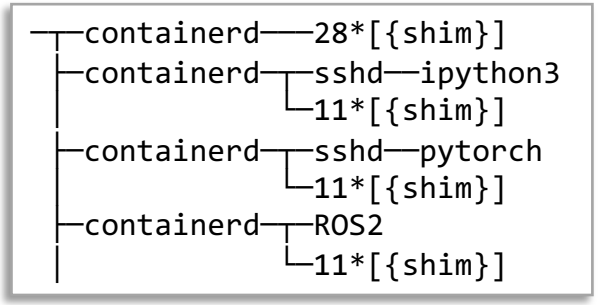


정보 보안 및 격리

CGroups (하드웨어 자원 격리)



Namespace (프로세스간 통신 격리)



현대 컨테이너의 표준 Open Container Interface(OCI)

1. 모든 컨테이너는 표준에 따라 구현이 되면서 **커널 레벨**의 지원이 필요함.

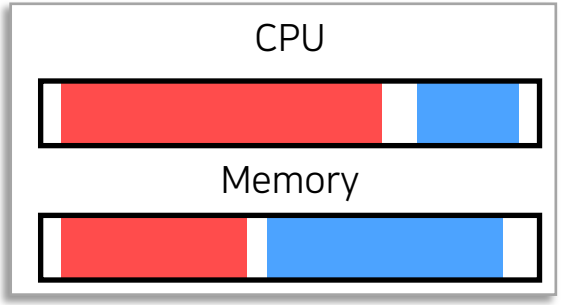
컨테이너

Chroot (저장소의 격리)

```
/ ─┬─bin─
   ├─mnt─┬─ContainerA:/─┬─bin
   │     │              ├─home
   │     │              └─lib
   │     ├─ContainerC:/─┬─ ...
   │     │              └─
   │     │
   │     │
```

정보 보안 및 격리

CGroups (하드웨어 자원 격리)



Namespace (프로세스간 통신 격리)

```
┬─containerd──28*[{shim}]
├─containerd─┬─sshd──ipython3
│            └─11*[{shim}]
├─containerd─┬─sshd──pytorch
│            └─11*[{shim}]
├─containerd─┬─ROS2
│            └─11*[{shim}]
```

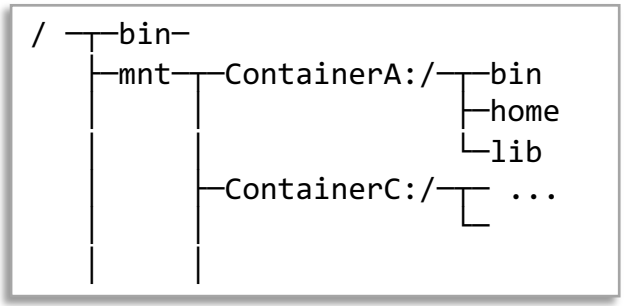
! 상용 임베디드 시스템은 경량화로 인해 OCI 표준을 모두 지원하지 않음 !

임베디드 시스템을 위한 원시 컨테이너 기반

1. 새로운 운영체제의 컨테이너 생성하여 환경 구성에 대한 시간을 단축하는 것이 중요함
2. 이는, 실험과 연구 목적에서는 **보안**과 **격리**는 컨테이너 구성에 실질적으로 불 필요함
3. 루트 디렉토리를 변경하여 컨테이너를 생성하는 1982년에 UNIX v7를 기반으로 컨테이너 생성이 가능함

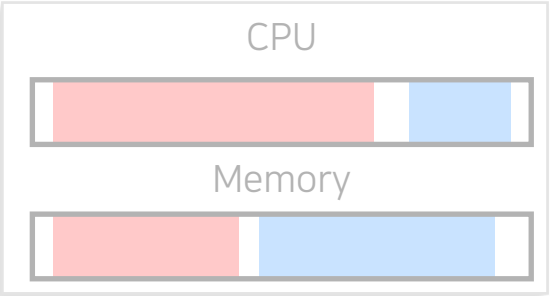
컨테이너

Chroot (저장소의 격리)

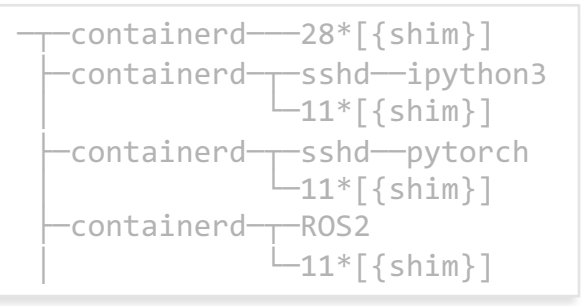


정보 보안 및 격리

CGroups (하드웨어 자원 격리)



Namespace (프로세스간 통신 격리)

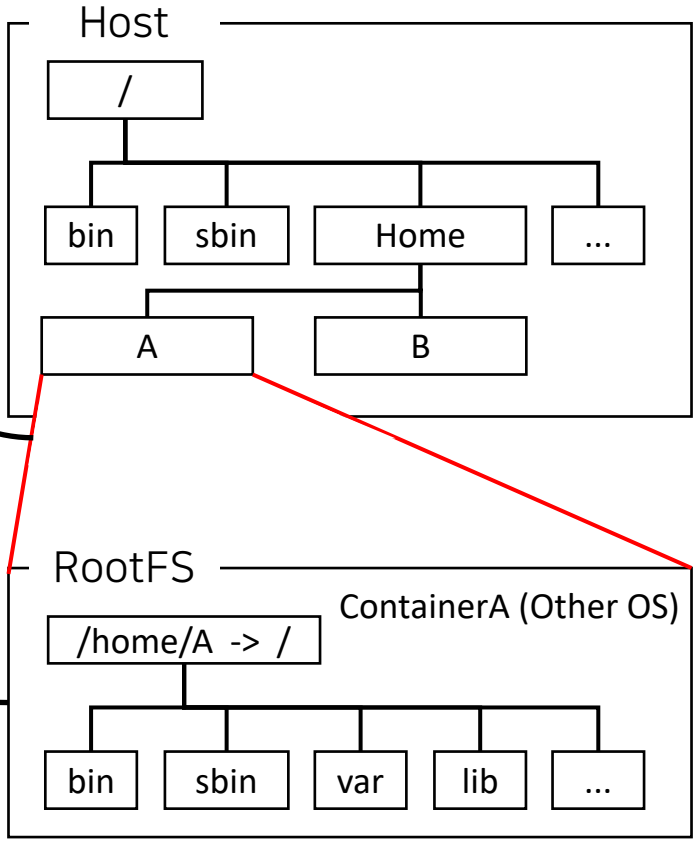


초기 버전의 컨테이너

1982년에 UNIX v7 에 적용된 초기 컨테이너 구조

1. Change Root Directory(Chroot)
   - 매핑 Host:/home/A ~ ContainerA:/
   - 호스트에서 컨테이너의 파일 접근 가능
   - 컨테이너에서 호스트로 파일 접근 불가능

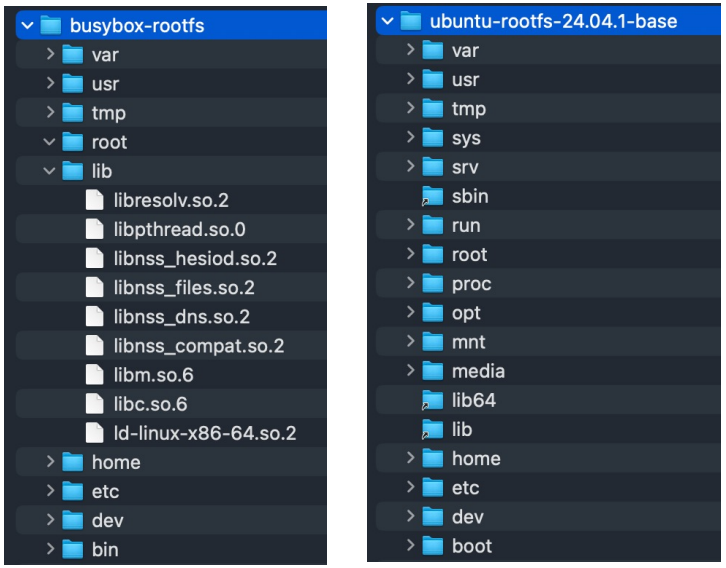
2. Root File System
   - 운영체제 구성하는 기본 파일의 집합
   - 패키지 매니저, 드라이버
   - 표준 C 라이브러리



chroot와 RootFS의 예시와 내부 데이터 형태

1. chroot는 루트 디렉토리의 위치를 옮기는 것
2. RootFS는 운영체제의 필수적인 바이너리가 포함된 데이터
3. 컨테이너의 기반 이미지가 모두 RootFS 기반으로 이뤄져 있음.

RootFS 의 데이터 구조



Chroot 의 결과

Host

```
$ pwd
/home

$ ls
ubuntu-rootfs-24.04.1-base
busybox-rootfs

$ chroot ./busybox-rootfs
```

Container

```
/ # ls
var usr tmp root lib home etc dev bin
```

실험 환경

IPTime A1004NS

OS : BusyBox v1.8.2

CPU : MT7620A

Arch : MIPSEL (RISC-V)

Experiments : QEMU

ASUS RT-AX53U

OS : OpenWRT 23.05.5

CPU : MT7621AT

Arch : MIPSEL (RISC-V)

Experiments : SSH

Q. SD 865 HDK (Galaxy S20)

OS : Android 10

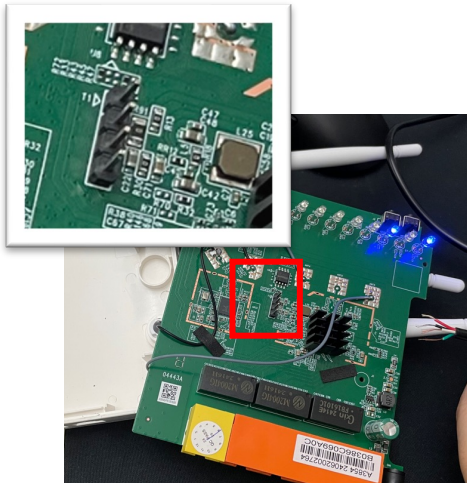
CPU : Q. SD 865

Arch : ARM

Experiments : ADB

실험 환경

IPTime A1004NS



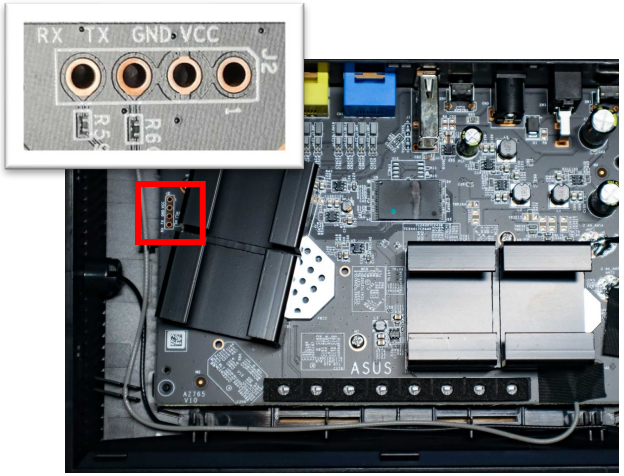
OS : BusyBox v1.8.2

CPU : MT7620A

Arch : MIPSEL (RISC-V)

Experiments : QEMU

ASUS RT-AX53U



OS : OpenWRT 23.05.5

CPU : MT7621AT

Arch : MIPSEL (RISC-V)

Experiments : SSH

Q. SD 865 HDK (Galaxy S20)

OS : Android 10

CPU : Q. SD 865

Arch : ARM

Experiments : ADB

실험 과정

실험 시나리오 (공통)

(1) 상용 임베디드에서 데비안 컨테이너로 운영체제 전환

(1) 패키지 매니저의 정상 동작 유무 확인

(2) Python의 NumPy로 LeNet 모델 추론 연산

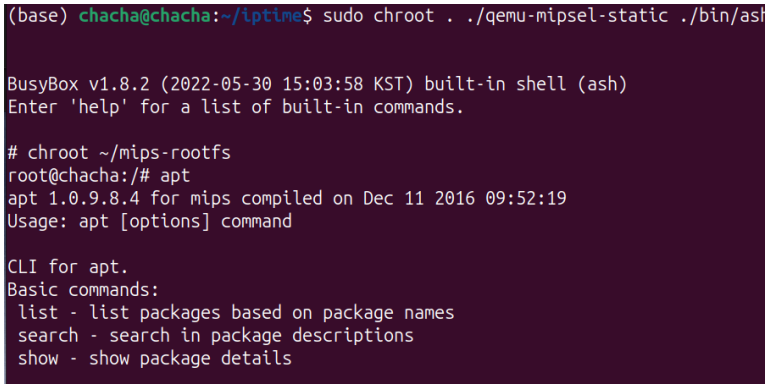
실험 시나리오 (ARM 환경)

(1) PyTorch기반의 LeNet 모델 학습

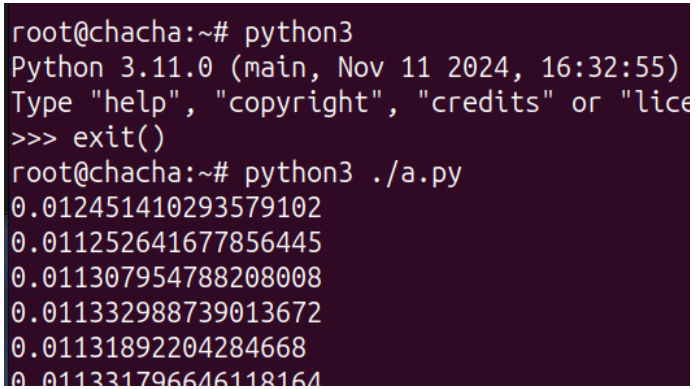
IPTime A1004NS

1. FLASH ROM의 용량이 16MB, 운영체제 용량이 12.6MB로 추가적인 소프트웨어 설치 자체가 불가능 하였음
2. 가상 머신 기반으로 IPTime A1004NS 펌웨어 1.2.152 버전으로 실험을 수행함
3. 펌웨어에 chroot가 기본 내장되어 있었기 때문에 데비안 컨테이너를 생성이 가능하였음
4. 외장 디스크를 통해 공간의 확충이 가능하다면 실제 하드웨어에서 컨테이너 생성의 가능성을 시사함

데비안 컨테이너 생성 결과



컨테이너와 Python에서 LeNet 연산



ASUS RT-AX53U

1. 저장 공간의 부족하였지만, 외장 디스크를 통해 실제 하드웨어에서 실험을 진행함
2. 펌웨어에 chroot가 기본 내장되어 있었기 때문에 데비안 컨테이너를 생성이 가능하였음
3. MIPSEL(RISC-V)에서 Python 환경에서 LeNet 모델 연산과 패키지 매니저가 동작함을 확인하였음.
4. LeNet 모델에 대해 1회 추론에 **37.3ms** 로 얻을 수 있었음.

데비안 컨테이너 생성 결과

```
(base) chacha@chacha:/$ ssh etri@129.254.196.83 -p 3333
etri@129.254.196.83's password:
etri@RT-AX53U-3600:/tmp/home/root# cd /mnt/sdb1/
etri@RT-AX53U-3600:/tmp/mnt/sdb1# chroot ./mips-rootfs
root@RT-AX53U-3600:/# apt

apt 1.0.9.8.4 for mips compiled on Dec 11 2016 09:52:19
Usage: apt [options] command

CLI for apt.
Basic commands:
 list - list packages based on package names
 search - search in package descriptions
 show - show package details

 update - update list of available packages
```

컨테이너와 Python에서 MNIST 연산

```
etri@RT-AX53U-3600:/tmp/mnt/sdb1# python3
Python 3.11.0 (main, Nov 11 2024, 16:32:55) [GCC 10.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
etri@RT-AX53U-3600:/tmp/mnt/sdb1# python3 ./lenet.py
0.4360527992248535
0.37165212631225586
0.37251734733581543
0.37381982803344727
0.3729710578918457
0.37374138832092285
0.3733081817626953
0.3731210231781006
0.3738255500793457
```

Qualcomm SD 865 HDK

1. 데비안 계열 컨테이너를 생성하여 NumPy와 PyTorch를 통한 딥 러닝 연산이 가능하였음
2. **자바 없이,** OpenCL / GPU 연산, 리눅스 GUI 개발이 **파이썬과 C++**로 가능하였음
3. 컨테이너 상에서 하드웨어 제어와 가속이 가능하였으며, 하나의 시스템으로 확장이 가능하였음

데비안 컨테이너 생성 결과

```
❯ adb shell
kona:/ # cd /data/local/tmp
kona:/data/local/tmp # ls
demo start-ubuntu20.sh ubuntu20-binds ubuntu20-fs
kona:/data/local/tmp # sh ./start-ubuntu20.sh
root@localhost:~# apt
apt 2.0.9 (arm64)
Usage: apt [options] command

apt is a commandline package manager and provides commands for
searching and managing as well as querying information about packages.
It provides the same functionality as the specialized APT tools,
like apt-get and apt-cache, but enables options more suitable for
interactive use by default.

Most used commands:
  list - list packages based on package names
  search - search in package descriptions
  show - show package details
  install - install packages
```

컨테이너와 Python에서 MNIST 연산

```
Python 3.8.10 (default, May 26 2023, 14:05:08)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
root@localhost:~# python3 ./lenet.py
10.58 ms
4.14 ms
2.88 ms
2.93 ms
2.87 ms
2.82 ms
2.91 ms
2.75 ms
2.17 ms
2.27 ms
```

Qualcomm SD 865 HDK에서 PyTorch를 통한 LeNet 모델 학습

```
  GNU nano 4.8                                    train.py
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
import argparse

# Define transformation functions
def normalize_img():
    """Normalizes images: `uint8` -> `float32` scaled to [0,1]."""
    return transforms.Normalize((0.5,), (0.5,))

def reshape_image():
    return transforms.Resize((7, 7))

# Model definition
class SimpleModel(nn.Module):
    def __init__(self):
        super(SimpleModel, self).__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(7 * 7, 30)
        self.fc2 = nn.Linear(30, 10)

    def forward(self, x):
        x = self.flatten(x)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```
root@localhost: ~

6
7
8
9
root@localhost:~#
root@localhost:~# ls
data  lenet  lenet.py  main.cpp  sample.txt  test.txt  train.py
root@localhost:~# nano train.py
root@localhost:~# python3 ./train.py --save_weight ./model_weights.h5 --epoch 10 --seed 142
Epoch [1/10], Loss: 1.3596
Validation Accuracy: 82.44%
Epoch [2/10], Loss: 0.5727
Validation Accuracy: 86.26%
Epoch [3/10], Loss: 0.4674
Validation Accuracy: 87.66%
Epoch [4/10], Loss: 0.4297
Validation Accuracy: 88.64%
Epoch [5/10], Loss: 0.4073
Validation Accuracy: 89.09%
Epoch [6/10], Loss: 0.3908
Validation Accuracy: 89.12%
Epoch [7/10], Loss: 0.3764
Validation Accuracy: 89.80%
Epoch [8/10], Loss: 0.3627
Validation Accuracy: 90.20%
Epoch [9/10], Loss: 0.3506
Validation Accuracy: 90.47%
Epoch [10/10], Loss: 0.3399
Validation Accuracy: 90.95%
root@localhost:~#
root@localhost:~# ls
data  lenet  lenet.py  main.cpp  model_weights.h5  sample.txt  test.txt  train.py
root@localhost:~#
```

DISCUSSION

장점 :

- 리눅스 커널이 있다면 어느 환경에서나 원시 컨테이너 생성하여 새로운 운영체제로 전환 가능
- 원시 컨테이너를 활용하여 운영체제의 라이브러리와 소프트웨어의 활용 가능
- 커널의 수정과 높은 사전 지식 없이 임베디드 시스템에 복잡한 라이브러리 사용이 가능
- 가상 머신 기반이 아닌, 리눅스 커널과 RootFS 기반이므로 성능 저하가 거의 없음

단점 :

- 파이썬이 RISC-V CPU를 지원하지 않아 PyTorch와 같은 라이브러리 설치가 불가능하였음.
- 프로세스와 하드웨어 자원의 격리가 이뤄지지 않아 보안에 취약

UST
과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

IEMEK (사)대한임베디드공학회
Institute of Embedded Engineering of Korea

THANK YOU

UST 석/박통합과정 인공지능학과
발표자 차주형
jh.cha@etri.re.kr