리눅스 커널 BPF 소개와 적용사례


리얼리눅스송태웅

http://reallinux.co.kr

What is BPF ?

What is BPF ?

“BPF in the kernel is similar with V8 in Chrome browser”

What is BPF ?

“Linux kernel **code execution** engine”

What is BPF ?

“Run **code** in the kernel”

What is BPF ?

“Run **code** in the kernel”

```
$ readelf -h bpf-prog.o  | grep Machine
  Machine:    Linux BPF
```

What is BPF ?

“Run **code** in the kernel”

```
$ readelf -h bpf-prog.o  | grep Machine
  Machine:     Linux BPF
  Machine:     Advanced Micro Devices X86-64
```

What is BPF ?

“Run code in the kernel”

↓

```
$ readelf -h bpf-prog.o  | grep Machine
  Machine:     Linux BPF
  Machine:     Advanced Micro Devices X86-64
```

writing C program

↓

clang / llc

↓

BPF instruction

What is BPF ?

“Run code in the kernel”

```
$ readelf -h bpf-prog.o  | grep Machine
  Machine:     Linux BPF
  Machine:     Advanced Micro Devices X86-64
```

but, restricted

writing C program

clang / llc

BPF instruction

What is BPF ?

“Run BPF code in the kernel”

```
$ readelf -h bpf-prog.o  | grep Machine
  Machine:     Linux BPF
  Machine:     Advanced Micro Devices X86-64
```

but, restricted

writing C program

clang / llc

BPF instruction

Is it **safe** ?

the image from http://www.tunisiaonline.com/travel-to-tunisia-in-2017/

Is it **safe** ?

“BPF verifier(in-kernel) guarantees”

Is it **safe** ?

“BPF verifier(in-kernel) guarantees”

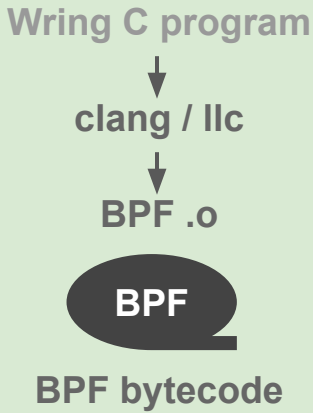
Check BPF program & **Prevent** problems
before the injection

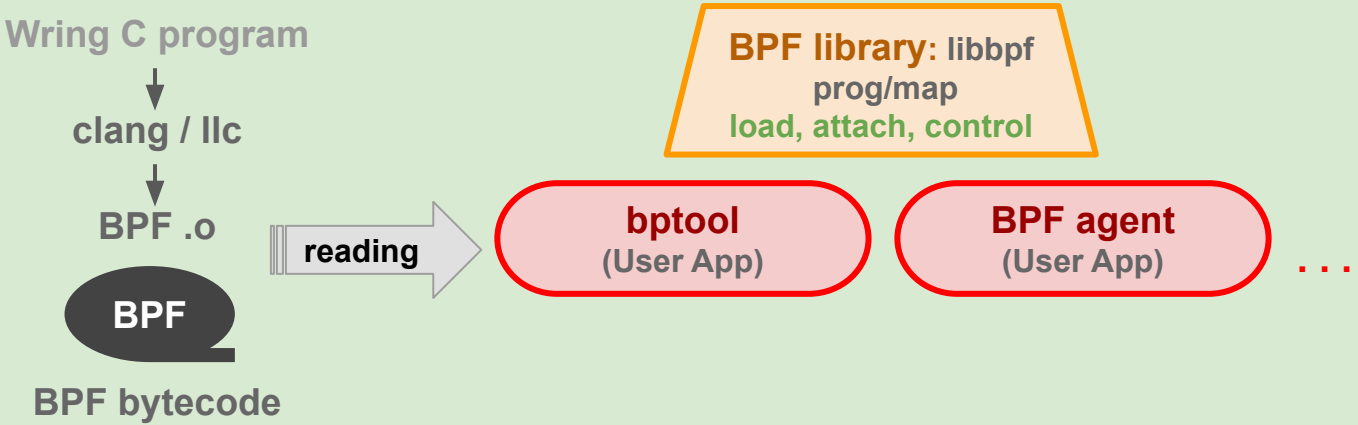
How to execute BPF program ?

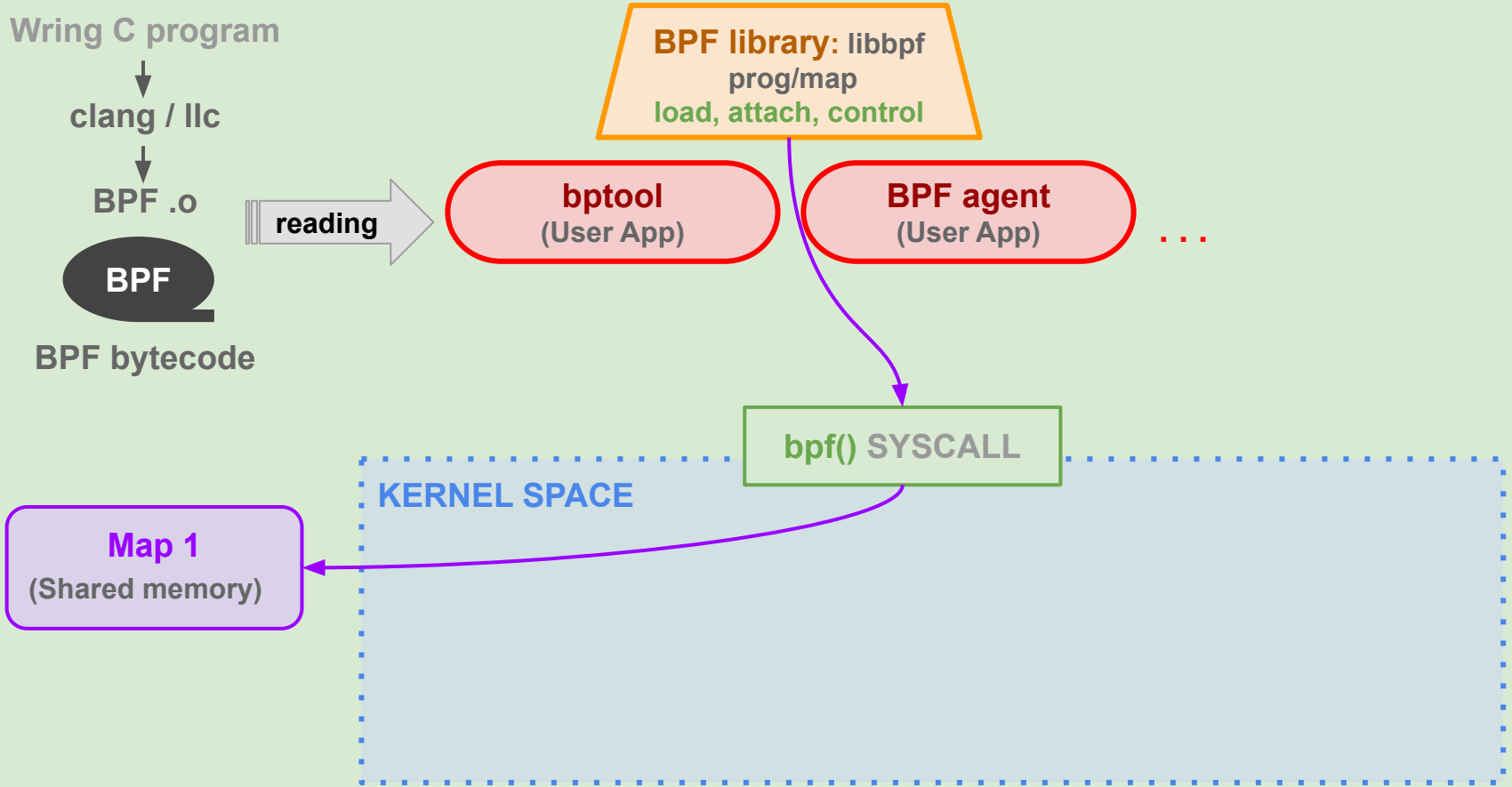
LOAD

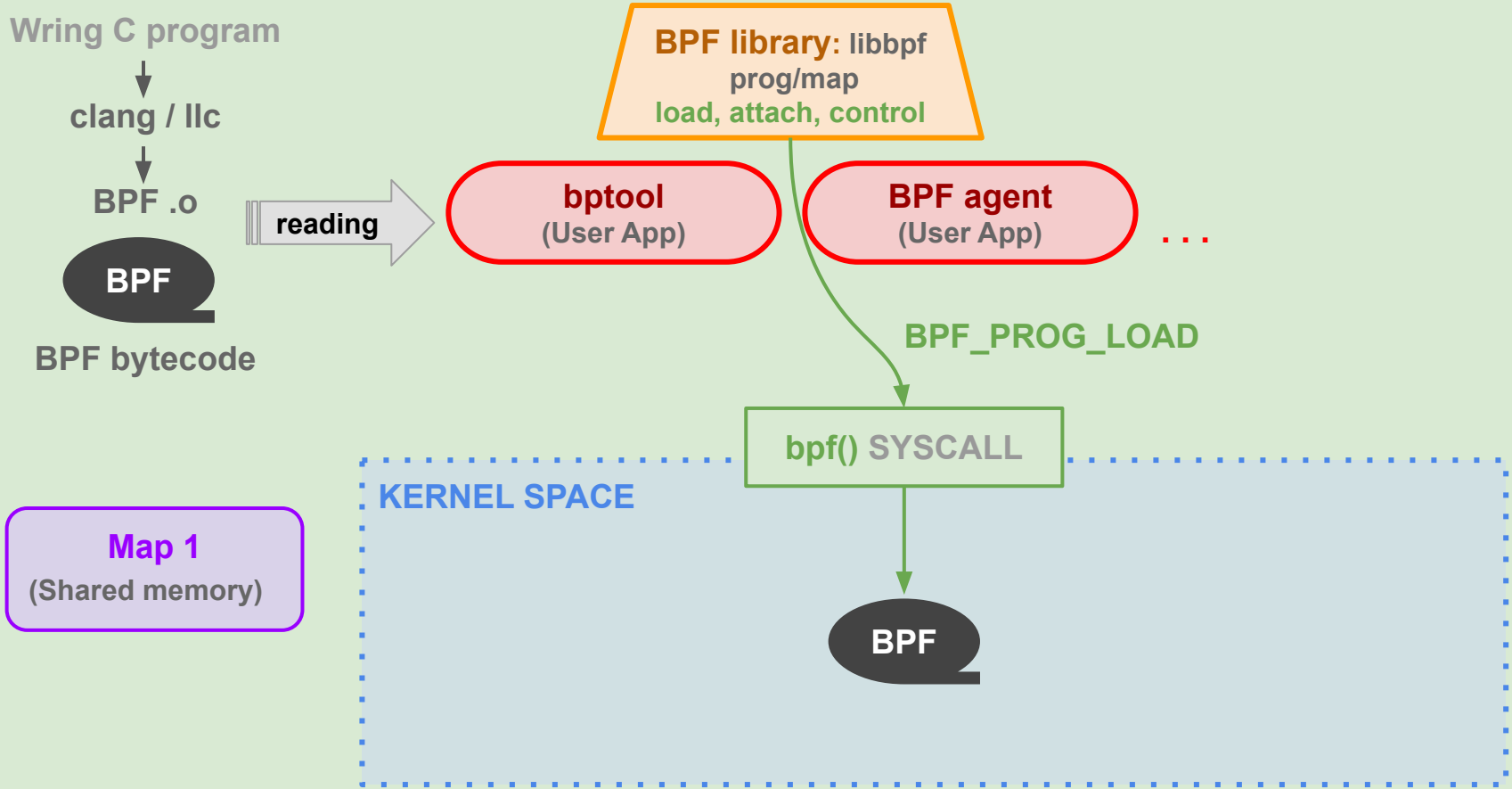
ATTACH

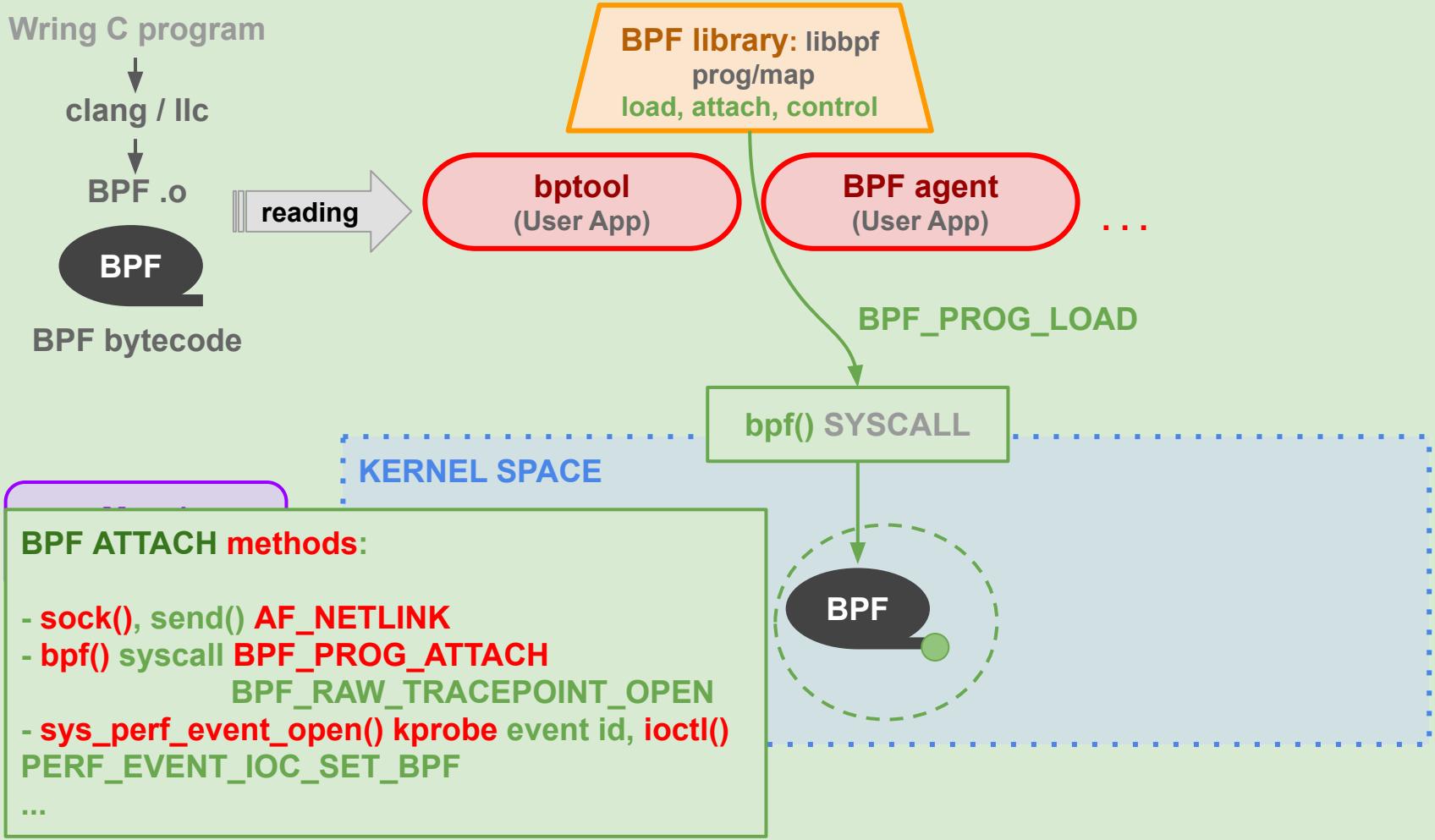
CALLBACK

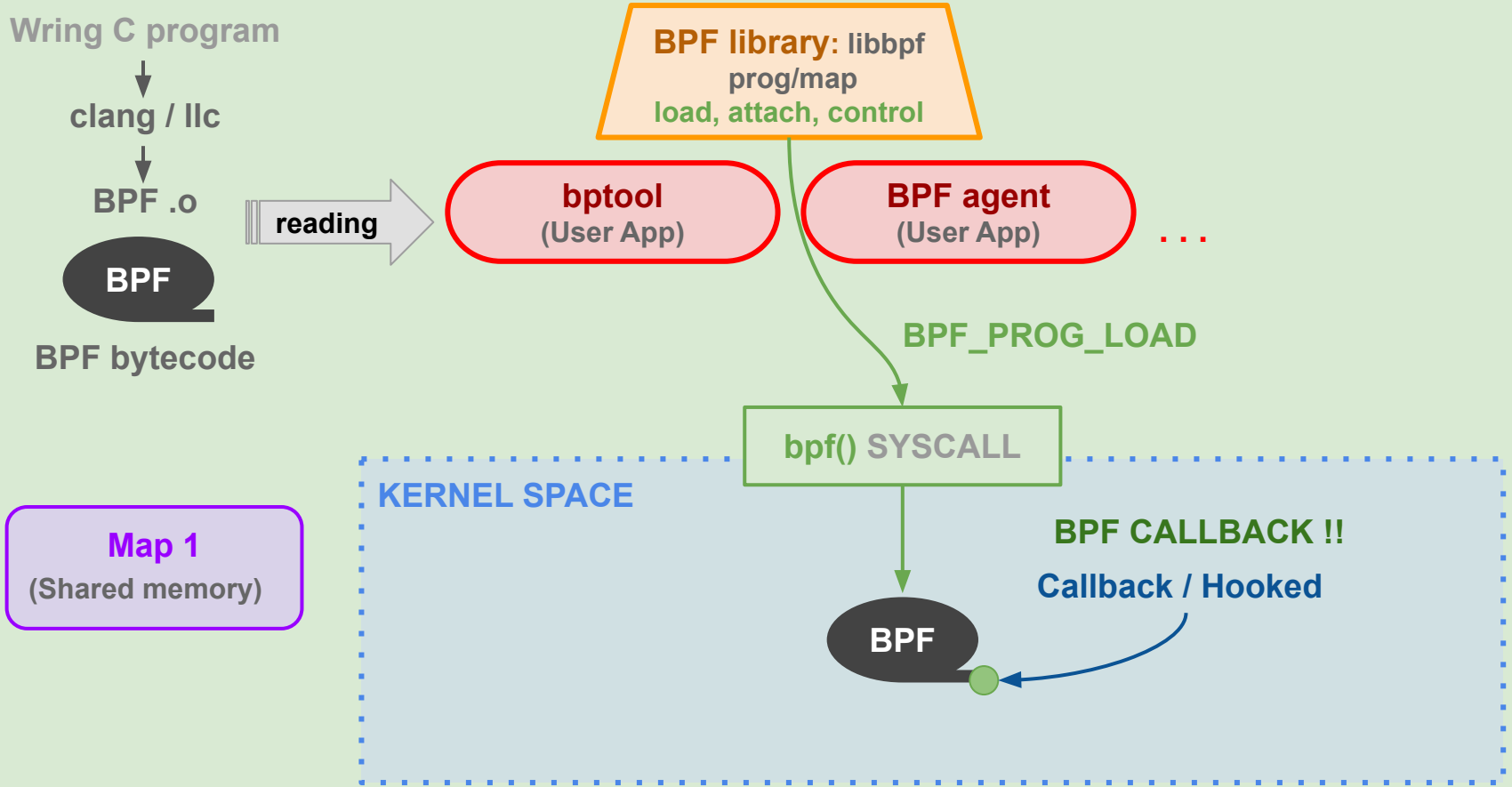

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode

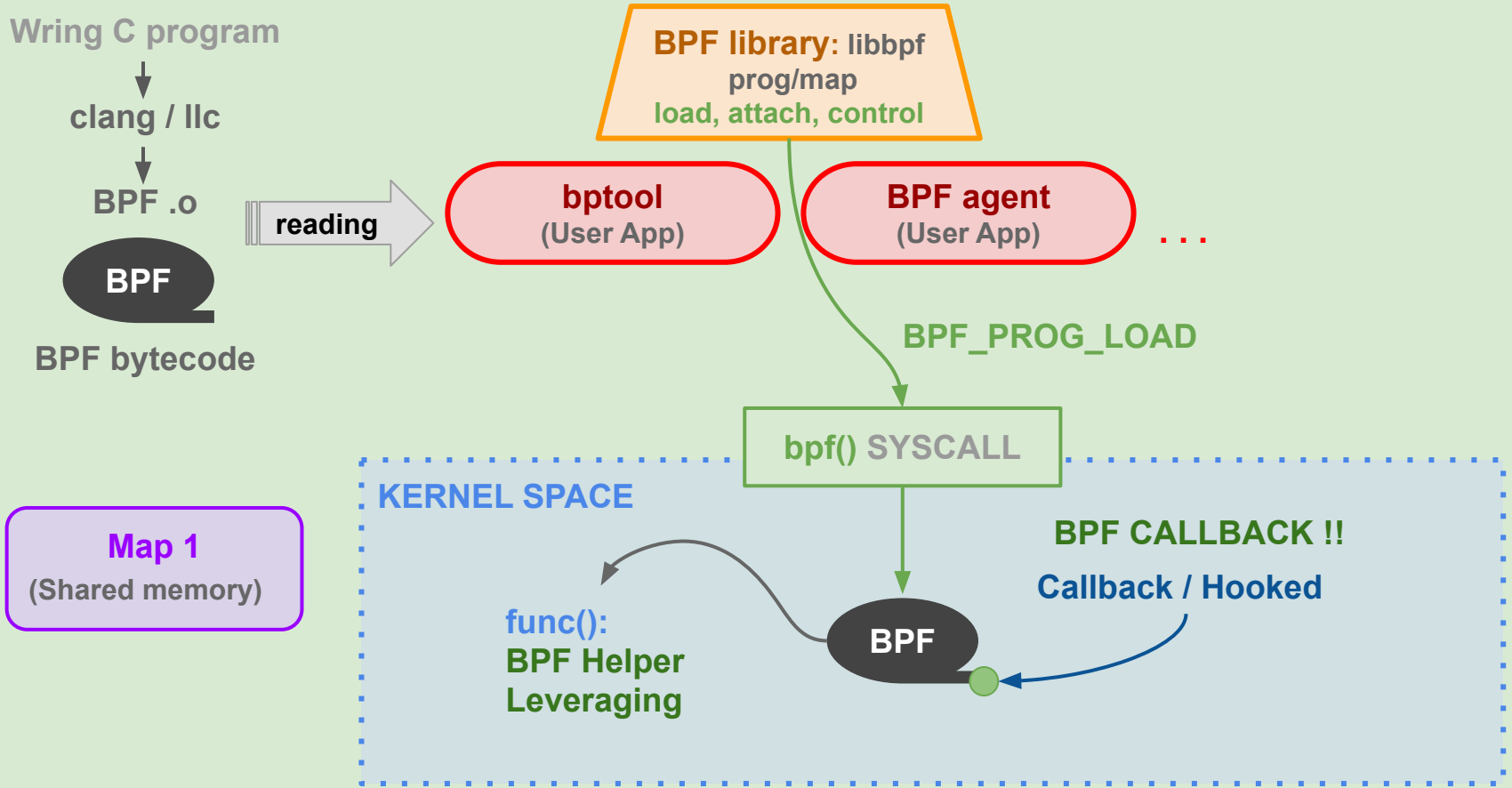

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
BPF library: libbpf
prog/map
load, attach, control
bptool
(User App)
BPF agent
(User App)
. . .

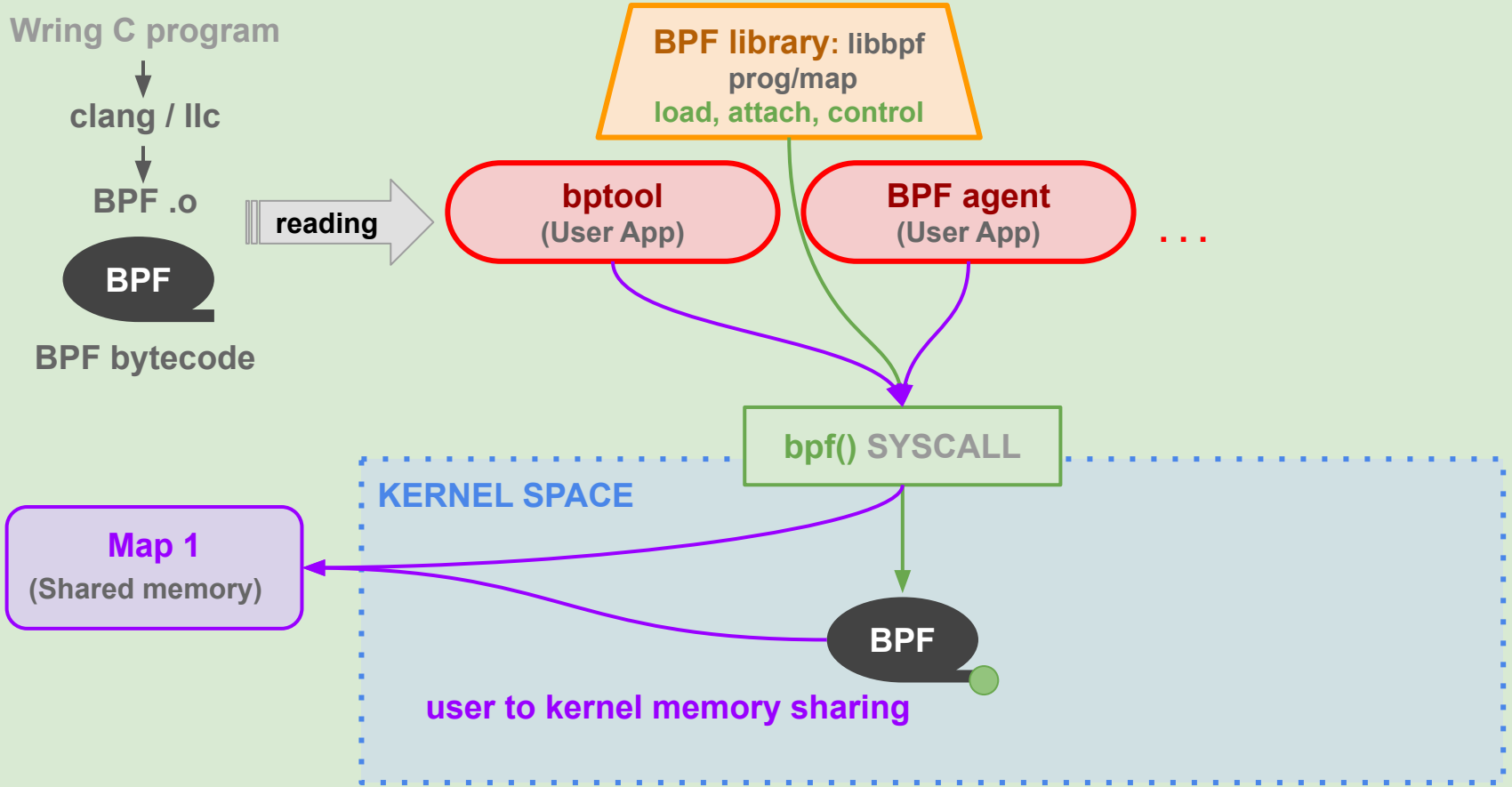

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
BPF library: libbpf
prog/map
load, attach, control
bptool
(User App)
BPF agent
(User App)
. . .
bpf() SYSCALL
KERNEL SPACE
Map 1
(Shared memory)

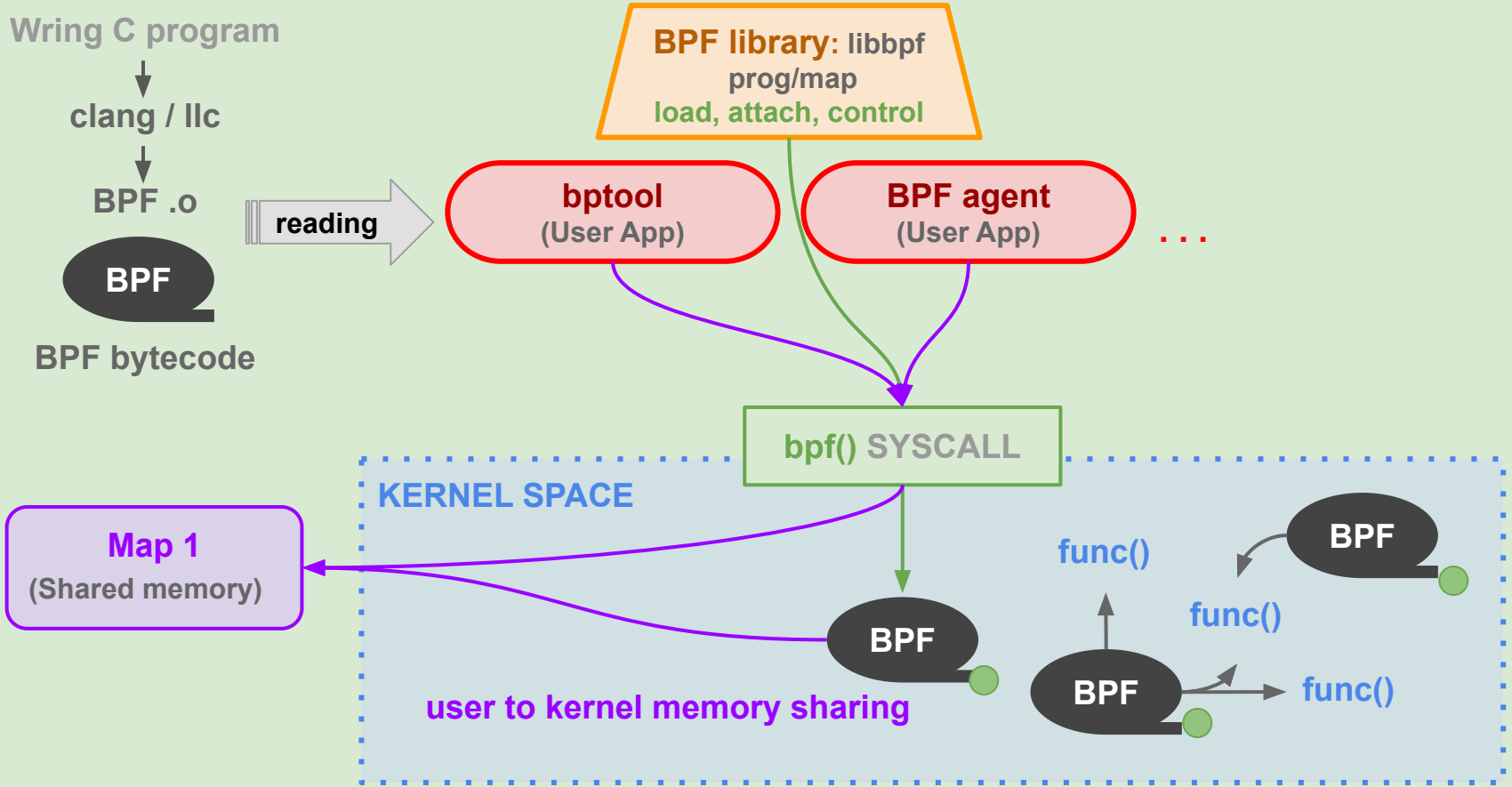

Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
BPF library: libbpf
prog/map
load, attach, control
bptool
(User App)
BPF agent
(User App)
. . .
BPF_PROG_LOAD
bpf() SYSCALL
KERNEL SPACE
BPF
Map 1
(Shared memory)


Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
BPF library: libbpf
prog/map
load, attach, control
bptool
(User App)
BPF agent
(User App)
. . .
BPF_PROG_LOAD
bpf() SYSCALL
KERNEL SPACE
BPF
BPF ATTACH methods:
- sock(), send() AF_NETLINK
- bpf() syscall BPF_PROG_ATTACH
BPF_RAW_TRACEPOINT_OPEN
- sys_perf_event_open() kprobe event id, ioctl()
PERF_EVENT_IOC_SET_BPF
...


Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
BPF library: libbpf
prog/map
load, attach, control
bptool
(User App)
BPF agent
(User App)
. . .
BPF_PROG_LOAD
bpf() SYSCALL
KERNEL SPACE
BPF CALLBACK !!
Callback / Hooked
BPF
Map 1
(Shared memory)


Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
BPF library: libbpf
prog/map
load, attach, control
bptool
(User App)
BPF agent
(User App)
. . .
BPF_PROG_LOAD
bpf() SYSCALL
KERNEL SPACE
BPF CALLBACK !!
Callback / Hooked
func():
BPF Helper
Leveraging
BPF
Map 1
(Shared memory)


Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
BPF library: libbpf
prog/map
load, attach, control
bptool
(User App)
BPF agent
(User App)
. . .
bpf() SYSCALL
KERNEL SPACE
Map 1
(Shared memory)
BPF
user to kernel memory sharing


Wring C program
clang / llc
BPF .o
BPF
BPF bytecode
reading
BPF library: libbpf
prog/map
load, attach, control
bptool
(User App)
BPF agent
(User App)
. . .
bpf() SYSCALL
KERNEL SPACE
Map 1
(Shared memory)
BPF
user to kernel memory sharing
func()
BPF
func()
BPF
func()

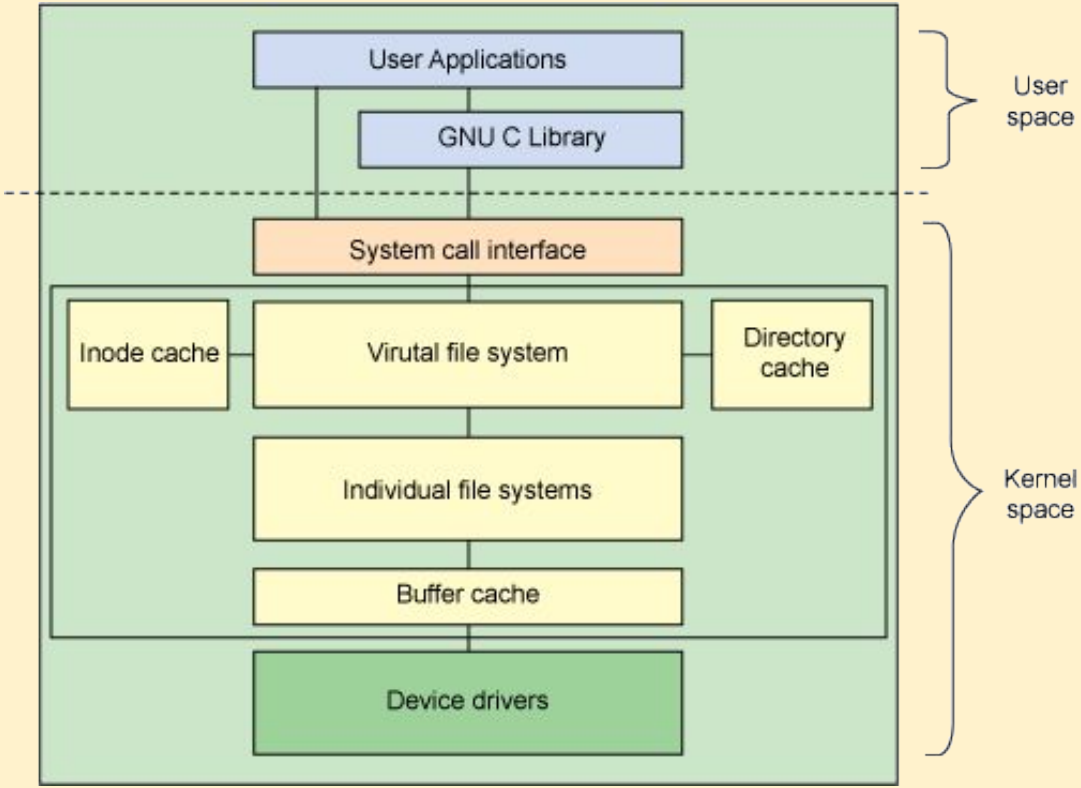
실습1

파일 write 과정

(추적용도 BPF 적용하기)

High-level architecture

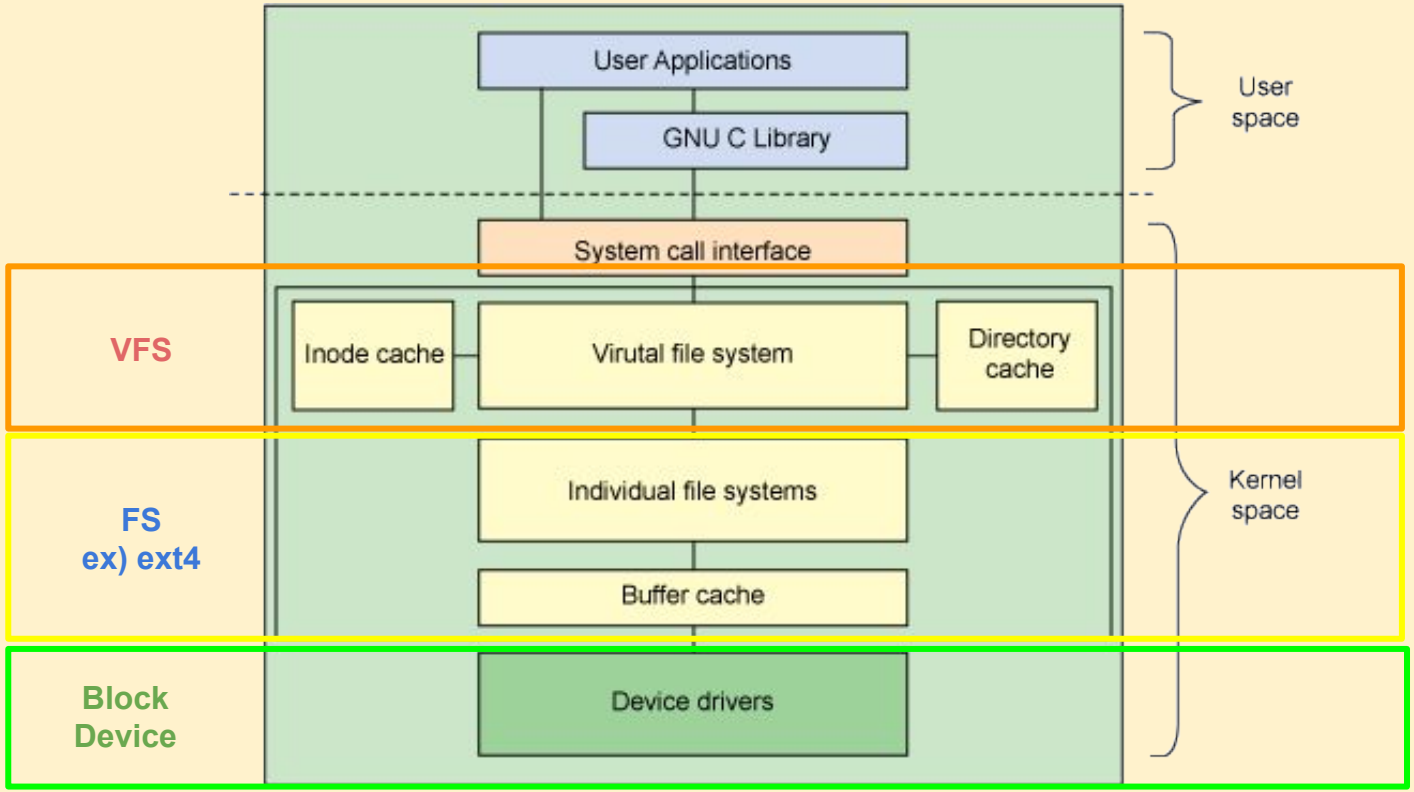
(Linux Filesystem components)



https://www.ibm.com/developerworks/library/l-linux-filesystem/

High-level architecture

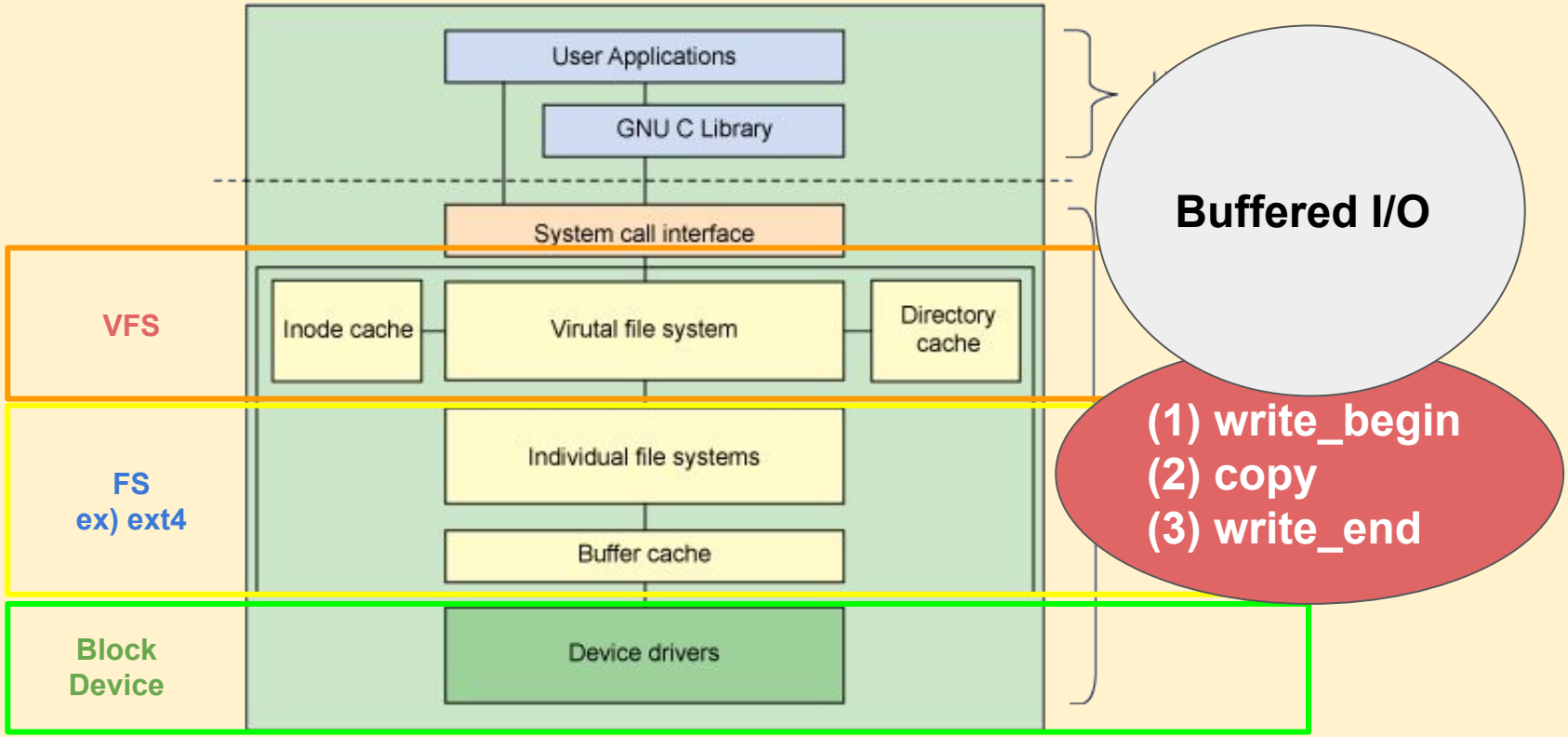
(Linux Filesystem components)



https://www.ibm.com/developerworks/library/l-linux-filesystem/

High-level architecture

(Linux Filesystem components)



https://www.ibm.com/developerworks/library/l-linux-filesystem/

실습1: tracex1_kern.c 예제 파일을 만들고 컴파일 하자

```
$ cd ~/git/linux/samples/bpf
$ cat tracex1_kern.c
/* Copyright (c) 2013-2015 PLUMgrid, http://plumgrid.com
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of version 2 of the GNU General Public
 * License as published by the Free Software Foundation.
 */
#include <linux/fs.h>
#include <linux/file.h>
#include <linux/namei.h>
#include <uapi/linux/bpf.h>
#include <linux/version.h>
#include "bpf_helpers.h"
#include "../../fs/ext4/ext4_extents.h"
#include <linux/uio.h>

#define _(P) ({typeof(P) val = 0; bpf_probe_read(&val, sizeof(val), &P);
val;})
/* kprobe is NOT a stable ABI
 * kernel functions can be removed, renamed or completely change semantics.
 * Number of arguments and their positions can change, etc.
 * In such case this bpf+kprobe example will no longer be meaningful
 */

SEC("kprobe/ext4_da_write_begin")
int bpf_prog5(struct pt_regs *ctx)
{
    char fmt [] = "ext4: da_write_begin  \n";

    bpf_trace_printk(fmt, sizeof(fmt));
    return 0;
}

SEC("kprobe/ext4_da_write_end")
int bpf_prog6(struct pt_regs *ctx)
{
    char fmt [] = "ext4: da_write_end\n";

    bpf_trace_printk(fmt, sizeof(fmt));
    return 0;
}

SEC("kprobe/mark_buffer_dirty")
int bpf_prog7(struct pt_regs *ctx)
{
    char fmt1 [] = "buffer on pagecache(disk block):%s\n";
    struct buffer_head *bh = (struct buffer_head *) PT_REGS_PARM1(ctx);
    char *b_data = _(bh->b_data);

    bpf_trace_printk(fmt1, sizeof(fmt1), b_data);
    return 0;
}

char _license[] SEC("license") = "GPL";
u32 _version SEC("version") = LINUX_VERSION_CODE;
```

실습1: hello.txt 파일 write 과정중 write_begin / write_end / copy 결과 확인

```
$ cd ~/git/linux/samples/bpf
$ make -j4
$ sudo ./tracex1
...
write-15764 [002] ....  7580.240905: 0: ext4: da_write_begin

write-15764 [002] ....  7580.241251: 0: ext4: da_write_end
write-15764 [002] .N..  7580.241362: 0: buffer on pagecache(disk block):hello linux filesystem

# 다른 터미널에서 write 예제 프로그램으로 "hello linux filesystem" 내용의 파일(hello.txt) 작성 테스트
$ ./write
$ cat hello.txt
hello linux filesystem
```

실습1: hello.txt 파일 write 과정중 write_begin / write_end / copy 결과 확인

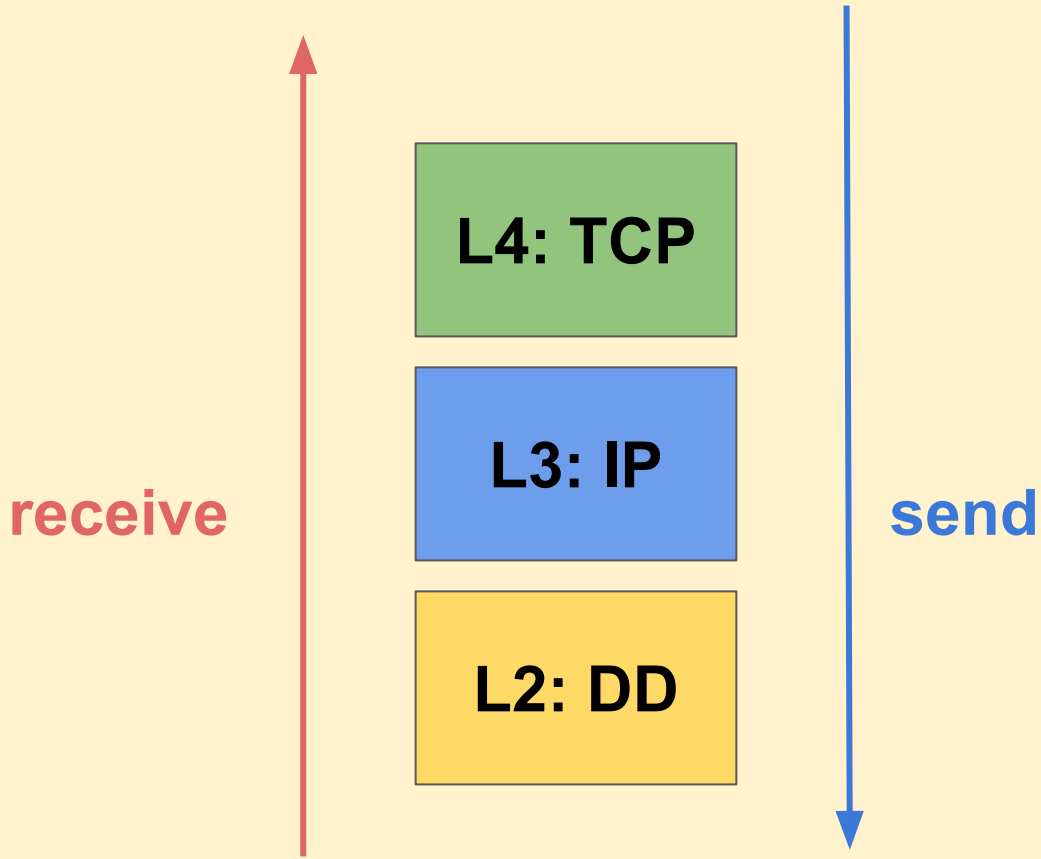
```
$ cd ~/git/linux/samples/bpf
$ make -j4
$ sudo ./tracex1
...
write-15764 [002] ....  7580.240905: 0: ext4: da_write_begin

write-15764 [002] ....  7580.241251: 0: ext4: da_write_end
write-15764 [002] .N..  7580.241362: 0: buffer on pagecache(disk block):hello linux filesystem

# 다른 터미널에서 write 예제 프로그램으로 "hello linux filesystem" 내용의 파일(hello.txt) 작성 테스트
$ ./write
$ cat hello.txt
hello linux filesystem
```

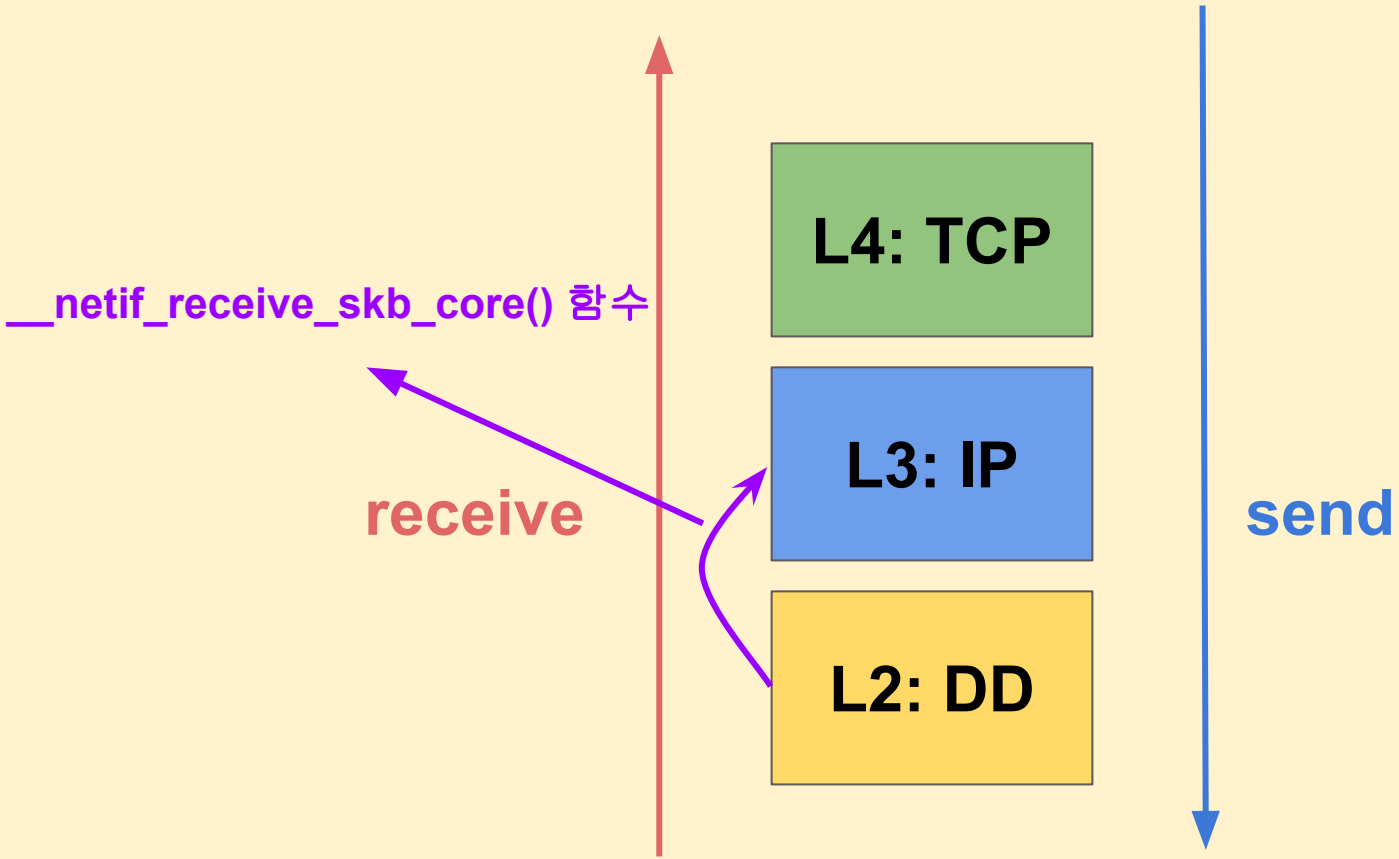
실습2

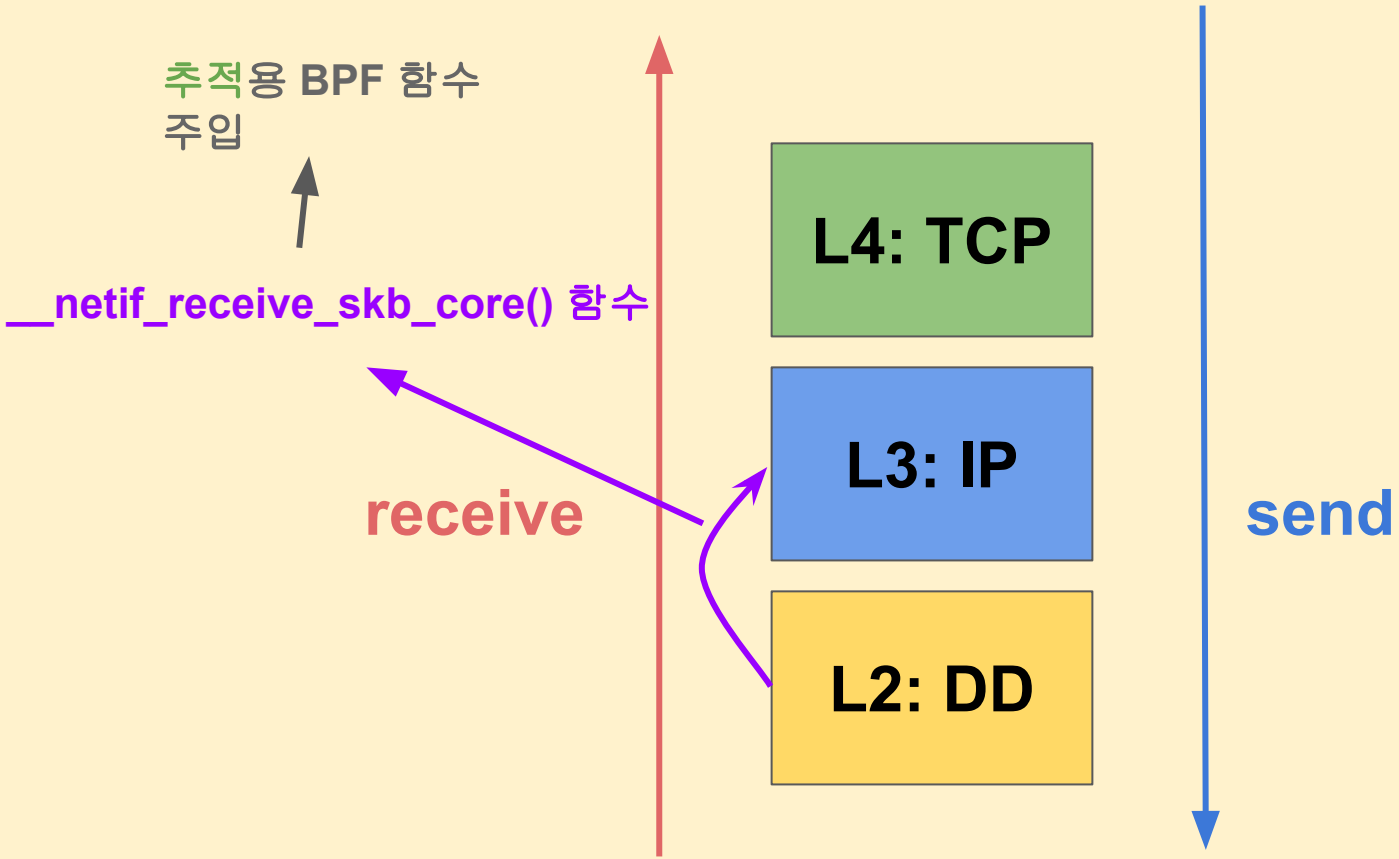
네트워크 패킷 receive 과정

(추적용도 BPF 적용하기)


L4: TCP
L3: IP
L2: DD


L4: TCP
L3: IP
L2: DD
receive
send


__netif_receive_skb_core() 함수
receive
L4: TCP
L3: IP
L2: DD
send


추적용 BPF 함수
주입
__netif_receive_skb_core() 함수
receive
L4: TCP
L3: IP
L2: DD
send

실습2: 네트워크 패킷 receive 과정 중 loopback 에서 오는 패킷만 모니터링 하기

```
$ cd ~/git/linux/samples/bpf
# 아까 수정했던 내용 복구 해서 원래 tracex1_kern.c 로 변경
$ git checkout -- tracex1_kern.c
# BPF 프로그램 (tracex1_kern.o) 재컴파일 하기
$ make -j4
$ sudo ./tracex1
...
ping-16548 [000] ..s1  8007.417583: 0: skb 000000009218db0b len 84

ping-16548 [000] ..s1  8007.417618: 0: skb 0000000064bf60cb len 84

ping-16548 [000] ..s1  8008.441838: 0: skb 000000009218db0b len 84
ping-16548 [000] ..s1  8008.441982: 0: skb 0000000064bf60cb len 84
...

# 다른 터미널에서 ping 127.0.0.1 테스트를 통해서 loopback 인터페이스로 84 length 의 패킷확
$ ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.322 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.223 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.192 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.191 ms
...
```

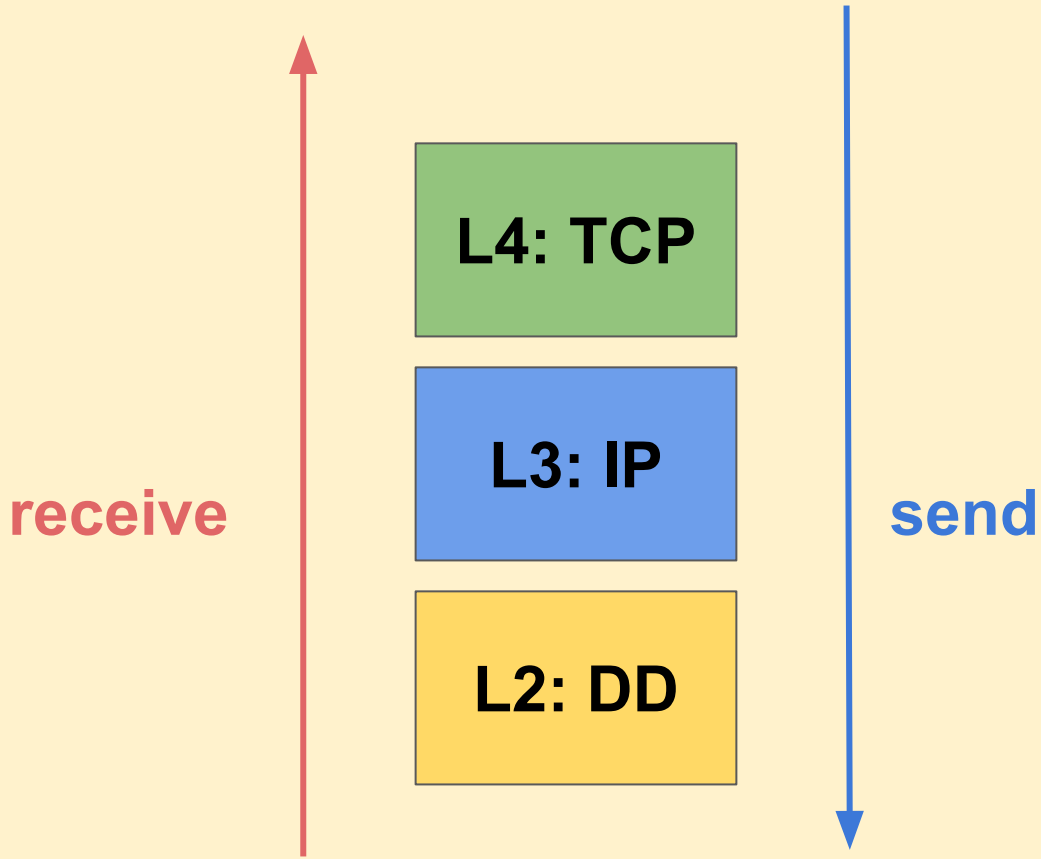
실습2: 네트워크 패킷 receive 과정 중 loopback 에서 오는 패킷만 모니터링 하기

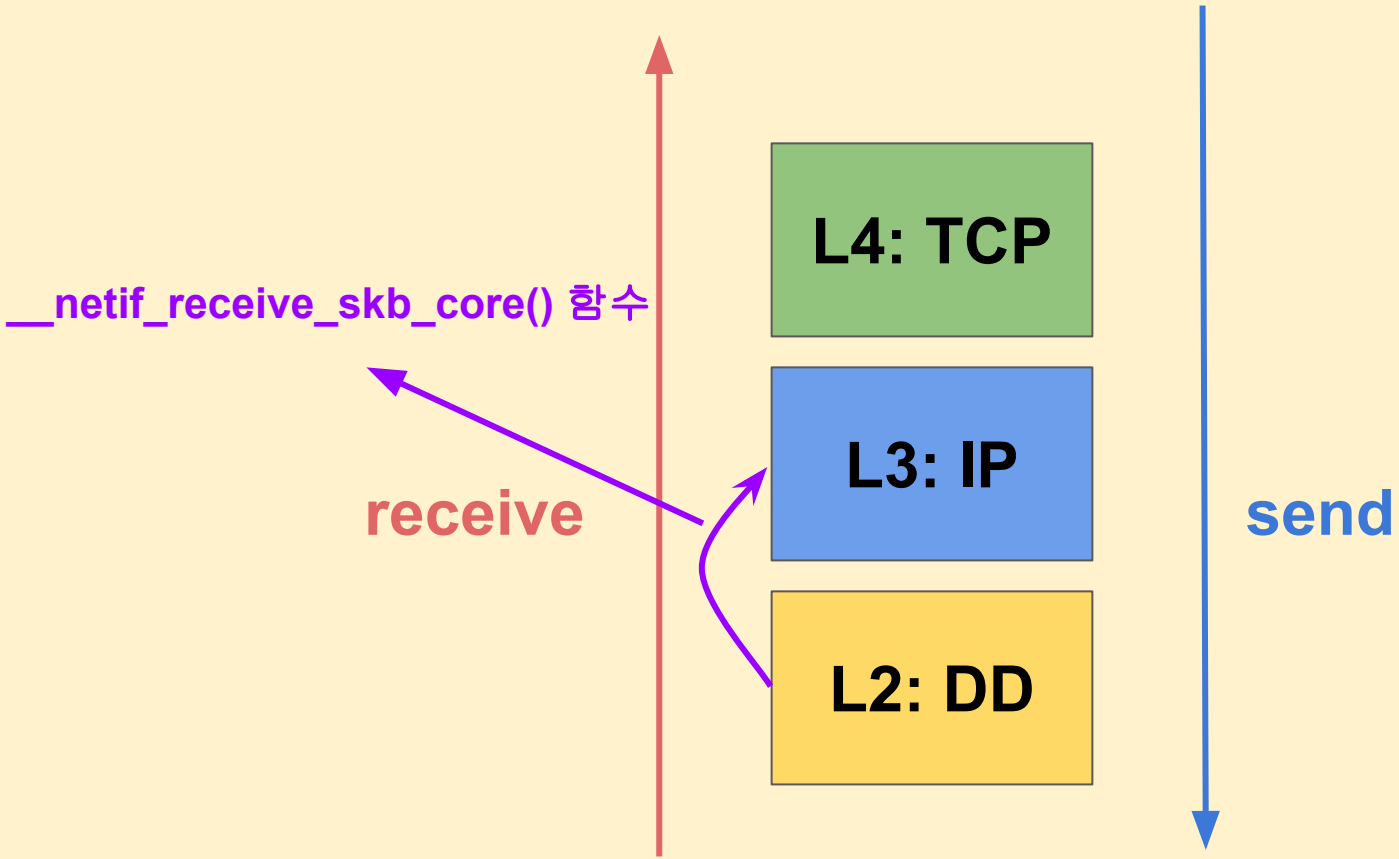
```
$ cd ~/git/linux/samples/bpf
# 아까 수정했던 내용 복구 해서 원래 tracex1_kern.c 로 변경
$ git checkout -- tracex1_kern.c
# BPF 프로그램 (tracex1_kern.o) 재컴파일 하기
$ make -j4
$ sudo ./tracex1
...
ping-16548 [000] ..s1  8007.417583: 0: skb 000000009218db0b len 84

ping-16548 [000] ..s1  8007.417618: 0: skb 0000000064bf60cb len 84

ping-16548 [000] ..s1  8008.441838: 0: skb 000000009218db0b len 84
ping-16548 [000] ..s1  8008.441982: 0: skb 0000000064bf60cb len 84
...

# 다른 터미널에서 ping 127.0.0.1 테스트를 통해서 loopback 인터페이스로 84 length 의 패킷확
$ ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.322 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.223 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.192 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.191 ms
...
```

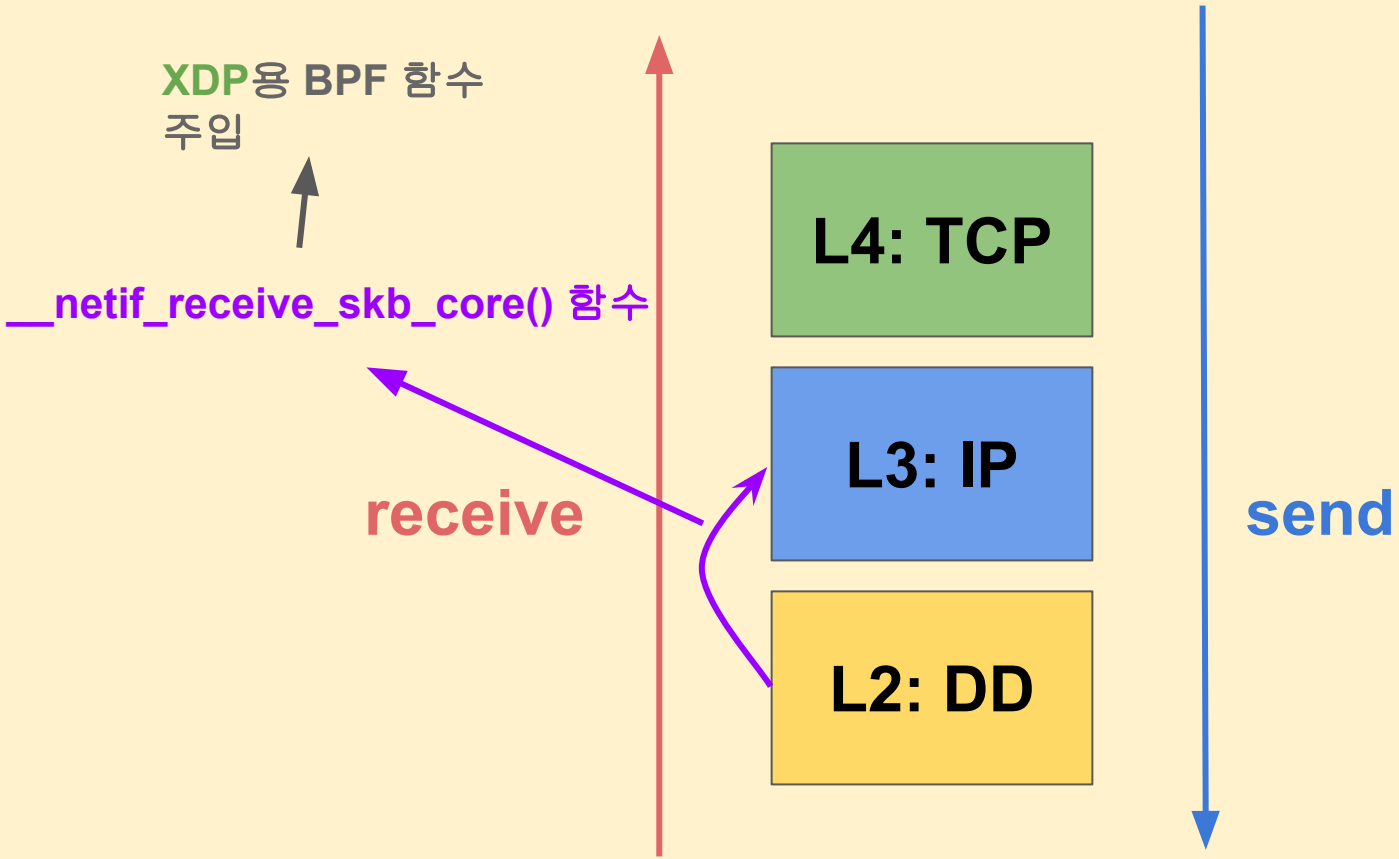
실습3

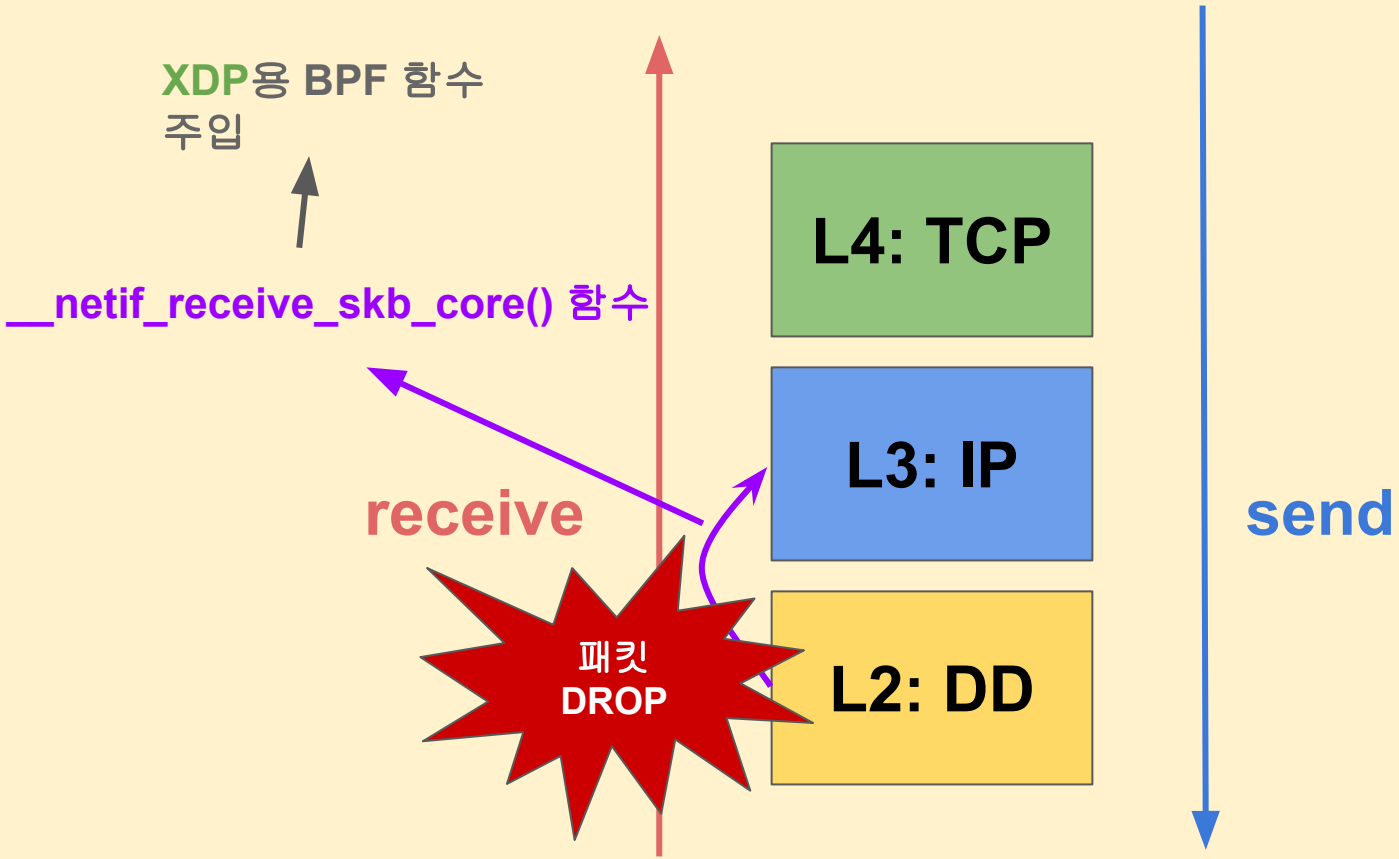
네트워크 패킷 receive 과정

(XDP용도 BPF 적용하기)


L4: TCP
L3: IP
L2: DD
receive
send


__netif_receive_skb_core() 함수
receive
L4: TCP
L3: IP
L2: DD
send


XDP용 BPF 함수
주입
__netif_receive_skb_core() 함수
receive
L4: TCP
L3: IP
L2: DD
send


XDP용 BPF 함수
주입
__netif_receive_skb_core() 함수
receive
패킷
DROP
L4: TCP
L3: IP
L2: DD
send

실습3: 예제소스 xdp-drop.c 를 만들자

```
$ vim xdp-drop.c
$ cat xdp-drop.c
#include <linux/bpf.h>

#ifndef __section
# define __section(NAME)                  \
   __attribute__((section(NAME), used))
#endif

__section("prog")
int xdp_drop(struct xdp_md *ctx)
{
    return XDP_DROP;
}

char __license[] __section("license") = "GPL";
```

실습3: 예제소스 xdp-drop.c 를 clang으로 컴파일 하자

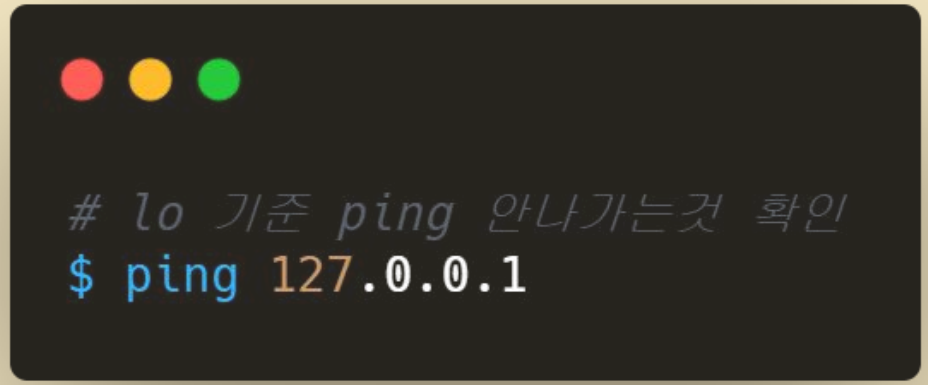
```
# xdp-drop.c 예제소스 코드 clang으로 컴파일 하기
$ clang -O2 -Wall -target bpf -I/usr/include/x86_64-linux-gnu/ -c xdp-drop.c -o xdp-drop.o
```

실습3: 예제 BPF프로그램 xdp-drop.o 를 ip link 명령으로 커널에 주입하자

* BPF로 짠 새로운 함수(xdp_drop)가 커널코드로 실시간 주입된다.

```
# ip link 명령을 통해서 xdp-drop.o BPF 프로그램을 주입하자
$ sudo ip link set dev lo xdp obj xdp-drop.o
```

실습3: ping 127.0.0.1 을 통해서 (주입된)xdp_drop 함수 call-back 테스트



실습3: 예제 BPF프로그램 xdp-drop.o 를 ip link 명령으로 커널에 제거하자

* BPF로 짠 새로운 함수(xdp_drop)가 커널코드에서 제거 된다.

```
# xdp 제거후 확인
$ sudo ip link set dev lo xdp off
$ ip a
```

참고: XDP가 지원이 Network Driver 들을 위해 L2 -> L3 로 올라오면서

호출되는 __netif_receive_skb_core() 함수에

do_xdp_generic() 함수가 있고

여기에 주입된 bpf함수를 호출해주는 hook 포인트가 있다.

추가 BPF 적용 사례 소개

1. 적외선 리모트 컨트롤러 decode 함수 주입용

   (LIRC - Linux Infrared Remote Control)

2. 특정 커널 함수 return 값 변경하기

   (error 테스트 목적: bpf_override_return 함수 활용)