

数据中心网络——实验手册2：eBPF Tracing教程

eBPF与动态跟踪

eBPF可以在内核中运行用户定义的代码，允许开发者在内核中动态插入代码以监测和调试系统行为。它可以用于性能分析、网络监控、安全审计等多种场景。

eBPF的强大之处在于它可以在**不修改内核源代码的情况下**，动态地插入和执行代码，从而实现对内核行为的实时监测和分析。

相比内核模块，eBPF具有**更高的灵活性和安全性**，因为它在内核中运行的代码是经过验证的，并且可以在运行时动态加载和卸载。

本文将介绍如何使用eBPF进行内核函数的动态跟踪，具体以`skb_clone()`函数为例，介绍如何使用bcc工具进行动态跟踪。

一、实验环境

- VMware Workstation Pro
- Ubuntu 22.04

二、BCC安装

采用源码编译的安装方式

1. 更换国内镜像源（可选） 编辑sources.list文件

```
sudo vim /etc/apt/sources.list
```

配置内容如下：

```
# 默认注释了源码仓库，如有需要可自行取消注释
deb https://mirrors.ustc.edu.cn/ubuntu/ jammy main restricted universe
multiverse
# deb-src https://mirrors.ustc.edu.cn/ubuntu/ jammy main restricted universe
multiverse

deb https://mirrors.ustc.edu.cn/ubuntu/ jammy-security main restricted
universe multiverse
# deb-src https://mirrors.ustc.edu.cn/ubuntu/ jammy-security main restricted
universe multiverse

deb https://mirrors.ustc.edu.cn/ubuntu/ jammy-updates main restricted
universe multiverse
# deb-src https://mirrors.ustc.edu.cn/ubuntu/ jammy-updates main restricted
universe multiverse
```

```
deb https://mirrors.ustc.edu.cn/ubuntu/ jammy-backports main restricted
universe multiverse
# deb-src https://mirrors.ustc.edu.cn/ubuntu/ jammy-backports main
restricted universe multiverse

# 预发布软件源，不建议启用
# deb https://mirrors.ustc.edu.cn/ubuntu/ jammy-proposed main restricted
universe multiverse
# deb-src https://mirrors.ustc.edu.cn/ubuntu/ jammy-proposed main restricted
universe multiverse
```

再更新软件列表以及软件包

```
sudo apt-get update
sudo apt-get upgrade
```

2. 安装相关依赖

要从源码构建工具链，需要：

- LLVM 3.7.1或者更高，编译时支持BPF(默认为on)
- Clang，由与LLVM相同的树构建
- cmake(>=3.1), gcc(>=4.7), flex, bison
- LuaJIT, 如果需要Lua

在终端执行以下命令：

```
sudo apt install -y bison build-essential cmake flex git libedit-dev \
libllvm14 llvm-14-dev libclang-14-dev python3 zlib1g-dev libelf-dev libfl-
dev python3-setuptools
```

对于其他版本的Ubuntu，可以参考 [https://github.com/iovisor/bcc/blob/master/INSTALL.md#ubuntu---](https://github.com/iovisor/bcc/blob/master/INSTALL.md#ubuntu---source) source

顺便安装一下后面要用到的curl：

```
apt install curl
```

3. 安装编译BCC

在终端依序执行以下命令：

```
git clone https://github.com/iovisor/bcc.git
mkdir bcc/build; cd bcc/build
cmake ..
make
sudo make install
cmake -DPYTHON_CMD=python3 ..
pushd src/python/
make
sudo make install
popd
```

```
forward@ubuntu:/usr/share/bcc/build$ sudo cmake -DPYTHON_CMD=python3 ..
-- Latest recognized Git tag is v0.24.0
-- Git HEAD is 8f40d6f57a8d94e7aee74ce358572d34d31b4ed4
-- Revision is 0.24.0-8f40d6f5
-- Found LLVM: /usr/lib/llvm-6.0/include 6.0.0 (Use LLVM_ROOT environment variable for another version of LLVM)
-- Could NOT find LibDebuginfod (missing: LIBDEBUGINFOD_LIBRARIES LIBDEBUGINFOD_INCLUDE_DIRS)
-- Using static-libstdc++
-- Could NOT find LuaJIT (missing: LUAJIT_LIBRARIES LUAJIT_INCLUDE_DIR)
CMake Warning at tests/python/CMakeLists.txt:10 (message):
  Recommended test program 'netperf' not found

CMake Warning at tests/python/CMakeLists.txt:16 (message):
  Recommended test program 'iperf' or 'iperf3' not found

-- Configuring done
-- Generating done
-- Build files have been written to: /usr/share/bcc/build
forward@ubuntu:/usr/share/bcc/build$ pushd src/python/
/usr/share/bcc/build/src/python /usr/share/bcc/build
forward@ubuntu:/usr/share/bcc/build/src/python$ sudo make
[100%] Built target bcc_py_python3
forward@ubuntu:/usr/share/bcc/build/src/python$ sudo make install
[100%] Built target bcc_py_python3
Install the project...
-- Install configuration: "Release"
running install
running build
running build_py
running install_lib
copying build/lib/bcc/table.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/__init__.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/libbcc.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/syscall.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/utls.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/containers.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/disassembler.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/perf.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/tcp.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/usdt.py -> /usr/lib/python3/dist-packages/bcc
copying build/lib/bcc/version.py -> /usr/lib/python3/dist-packages/bcc
byte-compiling /usr/lib/python3/dist-packages/bcc/table.py to table.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/__init__.py to __init__.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/libbcc.py to libbcc.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/syscall.py to syscall.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/utls.py to utls.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/containers.py to containers.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/disassembler.py to disassembler.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/perf.py to perf.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/tcp.py to tcp.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/usdt.py to usdt.cpython-36.pyc
byte-compiling /usr/lib/python3/dist-packages/bcc/version.py to version.cpython-36.pyc
running install_egg_info
Removing /usr/lib/python3/dist-packages/bcc-0.24.0-8f40d6f5.egg-info
Writing /usr/lib/python3/dist-packages/bcc-0.24.0-8f40d6f5.egg-info
forward@ubuntu:/usr/share/bcc/build/src/python$ popd
/usr/share/bcc/build
```

至此BCC安装完成。

注意修改python软链接指向python3，即可直接使用/usr/share/bcc/tools中的现有工具。

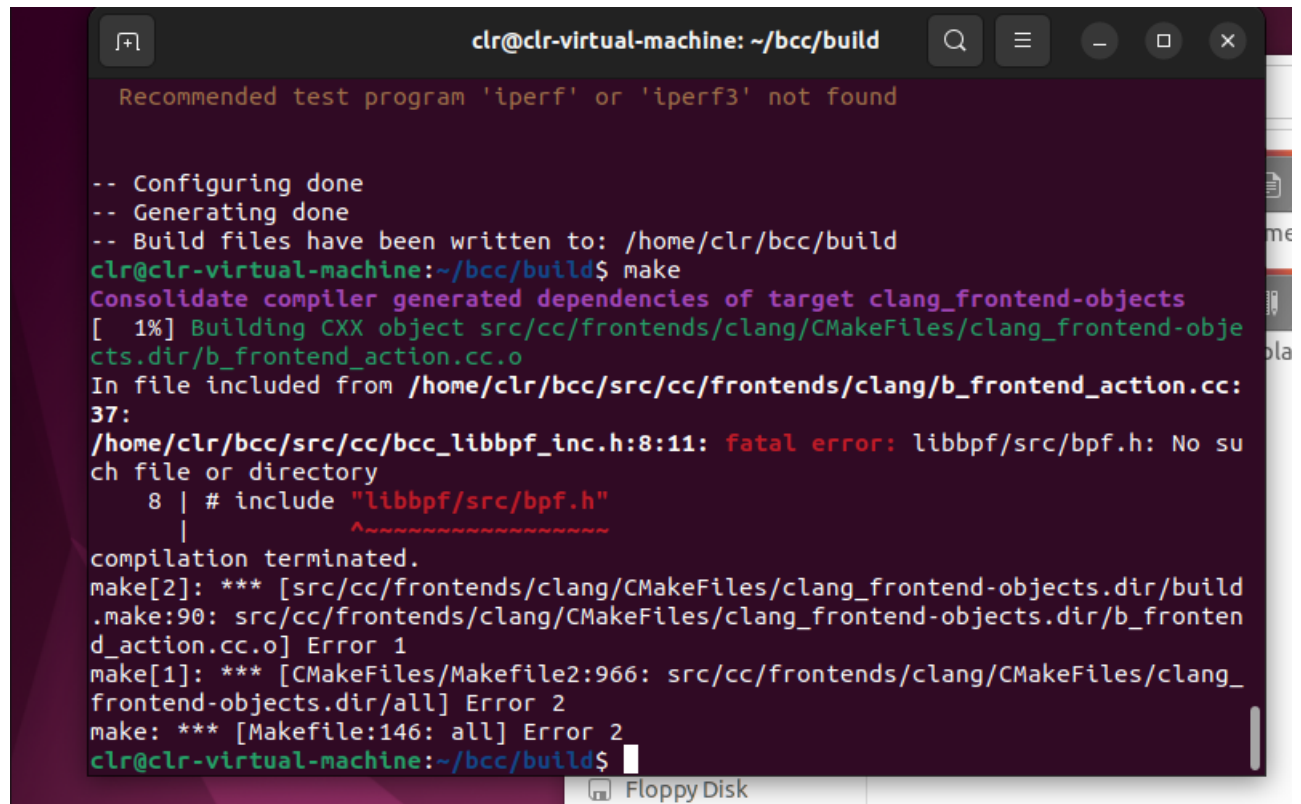
```
cd /usr/bin
sudo ln -s python3 python
```

例如执行funccount工具，进行简单监测，验证安装成功。

```
forward@ubuntu:/usr/share/bcc/tools$ sudo ./funccount "vfs_*"
[sudo] password for forward:
Tracing 67 functions for "b'vfs_*'"... Hit Ctrl-C to end.
^C
FUNC                                COUNT
b'vfs_statfs'                        1
b'vfs_readlink'                     3
b'vfs_statx_fd'                     18
b'vfs_statx'                        30
b'vfs_getattr'                      46
b'vfs_getattr_nosec'                46
b'vfs_open'                         113
b'vfs_write'                        556
b'vfs_read'                         767
b'vfs_writev'                      1155
Detaching...
```

4. 问题解决

有的同学在安装bcc时会遇到如下的问题：



```
clr@clr-virtual-machine: ~/bcc/build
Recommended test program 'iperf' or 'iperf3' not found

-- Configuring done
-- Generating done
-- Build files have been written to: /home/clr/bcc/build
clr@clr-virtual-machine:~/bcc/build$ make
Consolidate compiler generated dependencies of target clang_frontend-objects
[ 1%] Building CXX object src/cc/frontends/clang/CMakeFiles/clang_frontend-objects.dir/b_frontend_action.cc.o
In file included from /home/clr/bcc/src/cc/frontends/clang/b_frontend_action.cc:
37:
/home/clr/bcc/src/cc/bcc_libbpf_inc.h:8:11: fatal error: libbpf/src/bpf.h: No such
file or directory
    8 | # include "libbpf/src/bpf.h"
      |           ^~~~~~
compilation terminated.
make[2]: *** [src/cc/frontends/clang/CMakeFiles/clang_frontend-objects.dir/build
.make:90: src/cc/frontends/clang/CMakeFiles/clang_frontend-objects.dir/b_fron
d_action.cc.o] Error 1
make[1]: *** [CMakeFiles/Makefile2:966: src/cc/frontends/clang/CMakeFiles/clang_
frontend-objects.dir/all] Error 2
make: *** [Makefile:146: all] Error 2
clr@clr-virtual-machine:~/bcc/build$
```

这可能是由于网络问题，在git clone子模块时失败了，如果遇到了可以通过依次执行如下命令尝试解决。

该解决方案本质上是下载一个打包了子模块的源码包。

```
mkdir ~/bcc_submodule
cd ~/bcc_submodule
wget https://github.com/iovisor/bcc/releases/download/v0.27.0/bcc-src-with-submodule.tar.gz
tar xzvf bcc-src-with-submodule.tar.gz
mkdir bcc/build; cd bcc/build
cmake ..
make
sudo make install
cmake -DPYTHON_CMD=python3 ..
pushd src/python/
make
sudo make install
popd
```

三、skb_clone实验

1. 实验要求

本机用curl请求一个网上的文件，用bcc测量一下这个过程中skb_clone()这个函数被调用的次数以及调用的进程id。

2. 实验思路

要实现上述实验要求，需要关注以下三点任务。

2.1. 使用kprobe对内核进行动态tracing

```
from bcc import BPF

BPF(text = """
#C语言代码段
""")

#对bpf的处理代码
```

根据BPF程序结构，bcc有如下两种写法：

- C函数以**kprobe__**开头，其余部分被视为要监测的内核函数名，如**kprobe__skb_clone()**
- C函数为自定义函数，利用**b.attach_kprobe(event=("func_1"), fn_name="func_2")**建立

kprobe，内核系统出现func_1操作时执行func_2函数，即自定义函数。

2.2. 获取调用skb_clone()函数的进程id

提供如下两种思路：

- 从**trace_fields()**中获取诸如pid和时间戳的信息
- 使用**BPF_PERF_OUTPUT()**接口

2.3. 记录skb_clone()函数的调用次数

需要用变量count记录次数，实现count的更新累加，并能返回给用户态。

利用BPF map对象，实现用户态和内核态的数据交互，这里使用BPF_HASH。

3. 实验过程

根据两种获取信息的思路，提供两种编写方式：

3.1. trace_fields()

核心代码如下：

```
#include <uapi/linux/ptrace.h>

//创建一个BPF map对象
BPF_HASH(count);

//出现skb_clone操作时执行以下函数，记录次数
int do_trace(struct pt_regs *ctx){
    u64 cur_count = 1, key = 0;
    u64 *cur_ptr;
    cur_ptr = count.lookup(&key);
    if (cur_ptr == NULL){

        count.update(&key, &cur_count);
        bpf_trace_printk("%d\\n", cur_count);

    }else{

        cur_count = *cur_ptr + 1;
        count.update(&key, &cur_count);
        bpf_trace_printk("%d\\n", cur_count);

    }
    return 0;
}
```

其中，

- `count.lookup(&key)`：查看hash中key对应的值，如果存在返回值的指针
- `count.update(&key, &cur_count)`：将value与key关联，并覆盖之前的value，即更新次数
- `bpf_trace_printk()`：打印的信息会由`trace_fields()`中的`msg`字段取得

当出现skb_clone()操作时，就会执行do_trace()函数，用变量cur_count记录次数。

每出现一次skb_clone()操作时，`cur_ptr = count.lookup(&key)`就会查看hash中key对应的值，返回值的指针，`cur_count = *cur_ptr + 1`即可更新cur_count数值，并通过hash：`count.update(&key, &cur_count)`实现次数的更新覆盖。

最后，输出相关信息即可。

3.2. BPF_PERF_OUTPUT()

该方法的逻辑思路一致，区别在于不再从**trace_fields()**中获取信息。

需要自己定义C struct，将data从kernel传到user space。

```
struct data_t{
    u32 pid;
    u64 ts;
    u64 cur_count;
    char comm[TASK_COMM_LEN];
};
```

我们选择记录pid、时间戳、次数以及进程名，通过相应bpf函数获取。

获取并计算得出相应数据后，通过**events.perf_submit(ctx, &data, sizeof(data))**提交，使得user space能够通过per ring buffer读取。

输出信息部分代码如下：

```
start = 0
def print_event(cpu, data, size):
    global start
    event = b["events"].event(data) #获取由bpf函数perf_submit输出的数据
    if start == 0:
        start = event.ts
    time_s = (float(event.ts-start)) / 1000000000
    print("%-18.9f %-19s %-6d %d"%(time_s, event.comm, event.pid,
    event.cur_count))

#打开perf缓冲区，并指定一个回调函数，当有数据时，自动使用该函数处理
b["events"].open_perf_buffer(print_event)
while True:
    try:
        b.perf_buffer_poll() #等待perf缓冲区内容
    except KeyboardInterrupt:
        exit()
```

全部示例代码参考附录。

四、skb_clone实验结果

以curl www.baidu.com为例，监测**skb_clone()**的调用情况。

1. trace_fields


```
forward@ubuntu: ~/Desktop
File Edit View Search Terminal Help
forward@ubuntu:~/Desktop$ sudo python skb_clone_count.py
[sudo] password for forward:
Tracing for skb_clone... Ctrl-C to end
TIME(s)      COMM      PID      COUNT
6337.167259000 b'systemd-resolve' 973 b'1'
6337.170892000 b'systemd-resolve' 973 b'2'
6337.182433000 b'systemd-resolve' 973 b'3'
6337.196506000 b'curl' 9752 b'4'
6337.196525000 b'curl' 9752 b'5'
6337.207929000 b'ldt' 0 b'6'
6337.208149000 b'curl' 9752 b'7'
6337.208155000 b'curl' 9752 b'8'
6337.221946000 b'ksoftirqd/0' 10 b'9'
6337.222314000 b'curl' 9752 b'10'
6337.222331000 b'curl' 9752 b'11'
6337.233578000 b'gnome-shell' 1784 b'12'
```

```
forward@ubuntu: ~
File Edit View Search Terminal Help
.com/img/bd_logo1.png width=270 height=129> </div> <form id=form name=f action=
/www.baidu.com/s class=fn> <input type=hidden name=bdorz_come value=1> <input ty
pe=hidden name=ie value=utf-8> <input type=hidden name=f value=8> <input type=hi
dden name=rsv_bp value=1> <input type=hidden name=rsv_idx value=1> <input type=h
idden name=tn value=baidu><span class="bg s ipt_wr"><input id=kw name=wd class=s
ipt value maxlength=255 autocomplete=off autofocus></span><span class="bg s btn
_wr"><input type=submit id=su value=百度一下 class="bg s_btn"></span> </form> </
div> <div id=u1> <a href=http://news.baidu.com name=tj_trnews class=mnav>
新闻</a> <a href=http://www.hao123.com name=tj_trhao123 class=mnav>hao123</a> <a
href=http://map.baidu.com name=tj_trmap class=mnav>地图</a> <a href=http://v.ba
idu.com name=tj_trvideo class=mnav>视频</a> <a href=http://tieba.baidu.com name=
tj_trtieba class=mnav>贴吧</a> <noscript> <a href=http://www.baidu.com/bdorz/log
in.gif?login&tpl=mn&u=http%3A%2F%2Fwww.baidu.com%2F%3Fbdorz_come%3D1 nam
e=tj_login class=lb>登录</a> </noscript> <script>document.write('<a href="http:/
/www.baidu.com/bdorz/login.gif?login&tpl=mn&u='+ encodeURIComponent(window.locat
ion.href+ (window.location.search === "" ? "?" : "&")+ "bdorz_come=1")+ "' name=
"tj_login" class="lb">登录</a>');</script> <a href=/www.baidu.com/more/ name=tj
_brilcon class=bri style="display: block;">更多产品</a> </div> </div> <dl
v id=ftCon> <div id=ftConw> <p id=lh> <a href=http://home.baidu.com>关于百度</a>
<a href=http://ir.baidu.com>About Baidu</a> </p> <p id=cp>&copy;2017&nbsp;Baidu
&nbsp;&nbsp;<a href=http://www.baidu.com/duty/>使用百度前必读</a>&nbsp;&nbsp;<a href=http://
jianyi.baidu.com/ class=cp-feedback>意见反馈</a>&nbsp;&nbsp;京ICP证030173号&nbsp;&nbsp;<im
g src=/www.baidu.com/img/gd.gif> </p> </div> </div> </div> </body> </html>
forward@ubuntu:~$
```

2. BPF_PERF_OUTPUT()

```
forward@ubuntu: ~/Desktop
File Edit View Search Terminal Help
forward@ubuntu:~/Desktop$ sudo python skb_perf_output.py
Tracing for skb_clone... Ctrl-C to end
TIME(s)      COMM      PID      COUNT
0.000000000 b'systemd-resolve' 973 1
0.001520756 b'systemd-resolve' 973 2
0.008737358 b'systemd-resolve' 973 3
0.027825965 b'curl' 9770 4
0.027852635 b'curl' 9770 5
0.040397840 b'swapper/0' 0 6
0.040975771 b'curl' 9770 7
0.040999006 b'curl' 9770 8
0.053844822 b'swapper/0' 0 9
0.054281767 b'curl' 9770 10
0.054303731 b'curl' 9770 11
0.065193183 b'swapper/0' 0 12
```

```
forward@ubuntu: ~
File Edit View Search Terminal Help
.com/img/bd_logo1.png width=270 height=129> </div> <form id=form name=f action=
/www.baidu.com/s class=fn> <input type=hidden name=bdorz_come value=1> <input ty
pe=hidden name=ie value=utf-8> <input type=hidden name=f value=8> <input type=hi
dden name=rsv_bp value=1> <input type=hidden name=rsv_idx value=1> <input type=h
idden name=tn value=baidu><span class="bg s ipt_wr"><input id=kw name=wd class=s
ipt value maxlength=255 autocomplete=off autofocus></span><span class="bg s btn
_wr"><input type=submit id=su value=百度一下 class="bg s_btn"></span> </form> </
div> <div id=u1> <a href=http://news.baidu.com name=tj_trnews class=mnav>
新闻</a> <a href=http://www.hao123.com name=tj_trhao123 class=mnav>hao123</a> <a
href=http://map.baidu.com name=tj_trmap class=mnav>地图</a> <a href=http://v.ba
idu.com name=tj_trvideo class=mnav>视频</a> <a href=http://tieba.baidu.com name=
tj_trtieba class=mnav>贴吧</a> <noscript> <a href=http://www.baidu.com/bdorz/log
in.gif?login&tpl=mn&u=http%3A%2F%2Fwww.baidu.com%2F%3Fbdorz_come%3D1 nam
e=tj_login class=lb>登录</a> </noscript> <script>document.write('<a href="http:/
/www.baidu.com/bdorz/login.gif?login&tpl=mn&u='+ encodeURIComponent(window.locat
ion.href+ (window.location.search === "" ? "?" : "&")+ "bdorz_come=1")+ "' name=
"tj_login" class="lb">登录</a>');</script> <a href=/www.baidu.com/more/ name=tj
_brilcon class=bri style="display: block;">更多产品</a> </div> </div> <dl
v id=ftCon> <div id=ftConw> <p id=lh> <a href=http://home.baidu.com>关于百度</a>
<a href=http://ir.baidu.com>About Baidu</a> </p> <p id=cp>&copy;2017&nbsp;Baidu
&nbsp;&nbsp;<a href=http://www.baidu.com/duty/>使用百度前必读</a>&nbsp;&nbsp;<a href=http://
jianyi.baidu.com/ class=cp-feedback>意见反馈</a>&nbsp;&nbsp;京ICP证030173号&nbsp;&nbsp;<im
g src=/www.baidu.com/img/gd.gif> </p> </div> </div> </div> </body> </html>
forward@ubuntu:~$
```

示例中两种方法测出的调用次数为12，可以多次测量观察差别。

不同版本、不同环境下会有不同，可以根据具体情况、结合输出内容深入分析相应网络行为。

五、参考资料及建议

- bcc仓库
<https://github.com/iovisor/bcc>
https://github.com/iovisor/bcc/blob/master/docs/reference_guide.md
- 推荐使用手册中对应版本，版本不一致的可参考仓库中相关资料

六、新尝试

在大家的实际操作中，可能会发现：

```
0.899249318      b'node'      123273 1366
0.899376884      b'sshd'      226904 1367
0.899548227      b'node'      123306 1368
0.899921183      b'node'      123273 1369
0.899994353      b'sshd'      226904 1370
0.900143034      b'node'      123306 1371
0.901585063      b'node'      123273 1372
0.901699166      b'sshd'      226904 1373
0.901906800      b'node'      123306 1374
0.902720698      b'swapper/1' 0       1375
0.902749130      b'swapper/1' 0       1376
0.902803610      b'sshd'      226904 1377
0.902818662      b'sshd'      226904 1378
0.903352049      b'node'      123273 1379
0.903455699      b'sshd'      226904 1380
0.903616923      b'node'      123306 1381
^Czhaidoudou123@mbp-Parallels-ARM-Virtual-Machine:~/eBPF/tracing$ ^C
```

由于VSCode、SSH等服务的运行，导致调用skb_clone函数的进程数量有很多，我们没有办法准确定位到curl命令。我们应当尽可能排除这些影响，即，关闭ssh连接、关闭VSCode，选择直接从虚拟机本身的terminal进行测试。

但是，如果我们只想测试curl自身的函数调用情况，且就是懒得进虚拟机搞呢？

我们可以通过上周介绍的grep命令筛选出curl命令相关的结果：

```
sudo python skb_clone_count.py | grep 'curl'
```

但是，如此运行后有几个问题：

```
zhaidoudou123@mbp-Parallels-ARM-Virtual-Machine:~/eBPF/tracing$ sudo python skb_clone_count.py | grep 'curl'
79006.639028000    b'curl'      229593 b'62'
79006.662987000    b'curl'      229593 b'63'
```

1. 调用次数统计是针对所有进程的，没有办法单独针对curl命令统计
2. 运行速度很慢，在调用函数的进程很多的情况下从curl命令发出到出现结果要过很久

提示：

通过bpf_get_current_comm()、bpf_get_current_pid_tgid()等函数就可以获取特定进程的进程名、进程PID等。

具体函数用法如下，且可以参考skb_perf_output.py文件中的使用方法：

```
char comm[TASK_COMM_LEN];
bpf_get_current_comm(char * COMM, sizeof(COMM)); //将进程名赋给字符数组COMM

u64 pid = bpf_get_current_pid_tgid(); //将进程pid赋给u64变量pid
```

对于当前代码中的哈希表BPF_HASH(count)来说，key定义为0，可以通过将pid设置为key来解决第一个问题。

而运行速度很慢的原因是因为输出的太多，如果可以控制只输出进程名为curl，即可加速。

我们的目标主要是实现第一个问题的解决，随后调用grep命令即可，慢的问题的解决有一些复杂，因为涉及到内核中如何实现字符串比较的问题，大家如果有兴趣可以课后自己尝试。

七、代码附录

1. skb_clone_count.py

```
from __future__ import print_function
from bcc import BPF

b = BPF(text="""
#include <uapi/linux/ptrace.h>

BPF_HASH(count);

int do_trace(struct pt_regs *ctx){
    u64 cur_count = 1, key = 0;
    u64 *cur_ptr;
    cur_ptr = count.lookup(&key);
    if (cur_ptr == NULL){

        count.update(&key, &cur_count);
        bpf_trace_printk("%d\\n", cur_count);

    }else{

        cur_count = *cur_ptr + 1;
        count.update(&key, &cur_count);
        bpf_trace_printk("%d\\n", cur_count);

    }
    return 0;
}
""")

b.attach_kprobe(event="skb_clone", fn_name="do_trace")
print("Tracing for skb_clone... Ctrl-C to end")
print("%-18s %-19s %-6s %s" % ("TIME(s)", "COMM", "PID", "COUNT"))
while True:
    try:
        (task, pid, cpu, flags, ts, msg) = b.trace_fields()
    except KeyboardInterrupt:
        exit()
    print("%-18.9f %-19s %-6d %s" %(ts, task, pid, msg))
```

2. skb_perf_output.py

```
from __future__ import print_function
from bcc import BPF

prog = """

#include <linux/sched.h>
#include <uapi/linux/ptrace.h>
```

```

struct data_t{
    u32 pid;
    u64 ts;
    u64 cur_count;
    char comm[TASK_COMM_LEN];
};

BPF_PERF_OUTPUT(events);
BPF_HASH(count);

int do_trace(struct pt_regs *ctx){
    struct data_t data = {};

    data.pid = bpf_get_current_pid_tgid();
    data.ts = bpf_ktime_get_ns();
    bpf_get_current_comm(&data.comm, sizeof(data.comm));

    u64 cur_count = 1, key = 0;
    u64 *cur_ptr;
    cur_ptr = count.lookup(&key);

    if (cur_ptr == NULL){

        count.update(&key, &cur_count);
        data.cur_count = cur_count;
        events.perf_submit(ctx, &data, sizeof(data));

    }else{

        cur_count = *cur_ptr + 1;
        count.update(&key, &cur_count);
        data.cur_count = cur_count;
        events.perf_submit(ctx, &data, sizeof(data));

    }
    return 0;
}
""""

b = BPF(text = prog)
b.attach_kprobe(event = "skb_clone", fn_name="do_trace")
print("Tracing for skb_clone... Ctrl-C to end")
print("%-18s %-19s %-6s %s" % ("TIME(s)", "COMM", "PID", "COUNT"))

start = 0
def print_event(cpu, data, size):
    global start
    event = b["events"].event(data)
    if start == 0:
        start = event.ts
    time_s = (float(event.ts-start)) / 1000000000
    print("%-18.9f %-19s %-6d %d" % (time_s, event.comm, event.pid,
event.cur_count))

```

```
b["events"].open_perf_buffer(print_event)
while True:
    try:
        b.perf_buffer_poll()
    except KeyboardInterrupt:
        exit()
```