BPF

Содержание

1 Введение

1.1 Требования к ядру

Рекомендуется использовать ядро Linux 4.9 (релиз в декабре 2016 года) или более новое. Некоторые параметры конфигурации ядра должны

быть включены. Вот эти параметры:

```
CONFIG_BPF=y
CONFIG_BPF_SYSCALL=y
CONFIG_BPF_JIT=y
CONFIG_HAVE_EBPF_JIT=y
CONFIG_BPF_EVENTS=y
```

1.2 Установка в Ubuntu

```
sudo apt-get update
```

bpftace:

```
sudo apt-get install bpftrace
```

Если при запуске bpf программ возникает ошибка:

```
ERROR: Could not resolve symbol: /proc/self/exe:BEGIN_trigger
```

то возможно поможет установка пакета: `bpftrace-dbgsym`

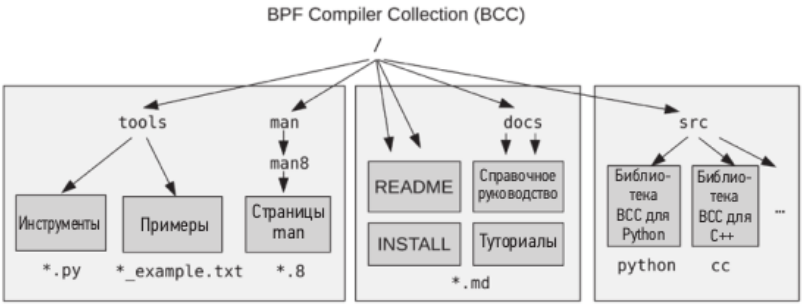
bcc:

```
sudo apt-get install bpfcc-tools linux-headers-$(uname -r)

# ls /sbin/*-bpfcc
```

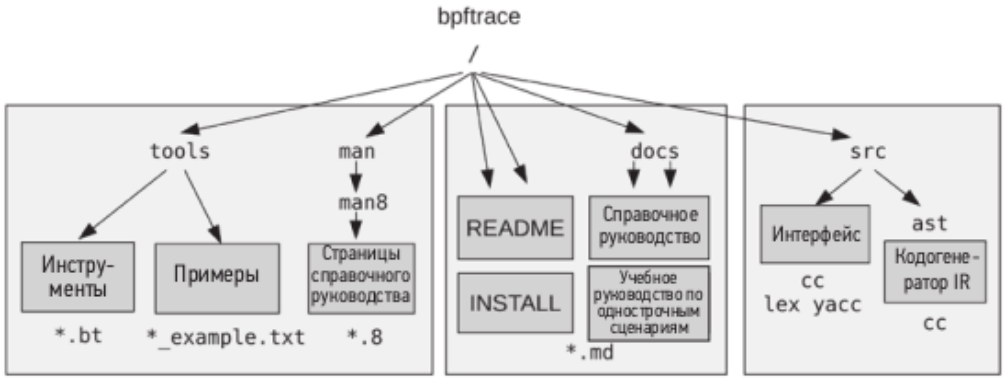
1.3 Схема BCC

`https://github.com/iovisor/bcc`



1.4 Схема bpftrace

`https://github.com/bpftrace/bpftrace`



1.5 Полезные ссылки

BPF Performance Tools - Материалы из книги Brendan Gregg:
`https://github.com/brendangregg/bpf-perf-tools-book/tree/master`

Docs от разработчиков Bpftrace:
`https://github.com/bpftrace/bpftrace/tree/master/docs`

BPF на сайте Brendan Gregg:
`https://brendangregg.com/ebpf.html`

Статья на хабре:
`https://habr.com/ru/articles/542560/`

2 bpftrace

2.1 Однострочные сценарии bpftrace

Показывает, кто и что выполняет:

```
bpftrace -e 'tracepoint:syscalls:sys_enter_execve
   { printf("%s -> %s\n", comm, str(args->filename)); }'
```

Показывает новые процессы с аргументами:

```
bpftrace -e 'tracepoint:syscalls:sys_enter_execve
   { join(args->argv); }'
```

Показывает, какие файлы открывает каждый процесс вызовом openat():

```
bpftrace -e 'tracepoint:syscalls:sys_enter_openat
```

```
   { printf("%s %s \n", comm, str(args->filename)); }'
```

Подсчитывает число системных вызовов, выполненных каждой программой:

```
bpftrace -e 'tracepoint:raw_syscalls:sys_enter
   { @[comm] = count(); }'
```

Подсчитывает число системных вызовов по их именам:

```
bpftrace -e 'tracepoint:syscalls:sys_enter_*
   { @[probe] = count(); }'
```

Подсчитывает число системных вызовов, выполненных каждым процессом:

```
bpftrace -e 'tracepoint:raw_syscalls:sys_enter
   { @[pid, comm] = count(); }'
```

Показывает общее число байтов, прочитанных каждым процессом:

```
bpftrace -e 'tracepoint:syscalls:sys_exit_read /args->ret/
   { @[comm] = sum(args->ret); }'
```

Показывает распределение размеров блоков, прочитанных каждым процессом:

```
bpftrace -e 'tracepoint:syscalls:sys_exit_read
   { @[comm] = hist(args->ret); }'
```

Показывает объемы дискового ввода/вывода для каждого процесса:

```
bpftrace -e 'tracepoint:block:block_rq_issue { printf("%d %s %d\n",
   pid, comm, args->bytes); }'
```

Подсчитывает число страниц, загруженных каждым процессом:

```
bpftrace -e 'software:major-faults:1 { @[comm] = count(); }'
```

Подсчитывает число отказов страниц для каждого процесса:

```
bpftrace -e 'software:faults:1 { @[comm] = count(); }'
```

Профилирует стек в пространстве пользователя для PID 189 с частотой 49 Гц:

```
bpftrace -e 'profile:hz:49 /pid == 189/ { @[ustack] = count(); }'
```

2.2 Пример программы на bpftrace

Программа измеряет время, затраченное на выполнение функции ядра vfs_read():

```
#!/usr/local/bin/bpftrace

kprobe:vfs_read
{
    @start[tid] = nsecs;
}

kretprobe:vfs_read
/@start[tid]/
{
    $duration_us = (nsecs - @start[tid]) / 1000;
    @us = hist($duration_us);
    delete(@start[tid]);
}
```

2.3 Развернутые циклы

`unroll (count) { statements }`
Аргумент count — это целочисленный литерал (константа) с максимально возможным значением 20.

2.4 Встроенные переменные

Встроенные переменные предопределены в bpftrace и обычно доступны только для чтения.
Наиболее важные встроенные переменные в bpftrace:

Переменная	Тип	Описание
pid	int	Идентификатор процесса (tgid в ядре)
tid	int	Идентификатор потока (pid в ядре)
uid	int	Идентификатор пользователя
username	string	Имя пользователя
nsecs	int	Отметка времени в наносекундах
elapsed	int	Время в наносекундах, прошедшее с начала инициализации bpftrace
cpu	int	Идентификатор процессора
comm	string	Имя процесса
kstack	string	Трассировка стека в пространстве ядра
ustack	string	Трассировка стека в пространстве пользователя
arg0, ..., argN	int	Аргументы зондов некоторых типов
args	struct	Аргументы зондов некоторых типов
retval	int	Возвращаемые значения для зондов некоторых типов
func	string	Имя трассируемой функции
probe	string	Полное имя текущего зонда
curtask	int	task_struct в ядре как 64-битное целое без знака (допускается приведение типа)
cgroup	int	Идентификатор cgroup
$1, ..., $N	int char*	Позиционные параметры программы bpftrace

2.5 Карты

Формат определения:

```
@name
@name[key]
@name[key1, key2[, ...]]
```

Примеры:

```
@start = nsecs;
@last[tid] = nsecs;
@bytes = hist(retval);
@who[pid, comm] = count();
```

2.6 Наиболее важные функции bpftrace

Функция	Описание
`printf(char *fmt [, ...])`	Форматированный вывод
`time(char *fmt)`	Форматированный вывод времени
`join(char *arr[])`	Выводит массив строк, объединяя их через пробел
`str(char *s [, int len])`	Возвращает строку, на которую ссылается указатель s, с необязательным ограничителем длины len
`kstack(int limit)`	Возвращает трассировку стека ядра с глубиной до limit
`ustack(int limit)`	Возвращает трассировку стека в пространстве пользователя с глубиной до limit
`ksym(void *p)`	Определяет символ по адресу в пространстве ядра и возвращает строку с ним
`usym(void *p)`	Определяет символ по адресу в пространстве пользователя и возвращает строку с ним
`kaddr(char *name)`	Определяет адрес символа в пространстве ядра
`uaddr(char *name)`	Определяет адрес символа в пространстве пользователя
`reg(char *name)`	Возвращает значение, хранящееся в указанном регистре
`ntop([int af,] int addr)`	Возвращает строковое представление IP-адреса
`system(char *fmt [, ...])`	Выполняет команду в командной оболочке
`cat(char *filename)`	Выводит содержимое указанного файла
`exit()`	Завершает выполнение bpftrace

2.7 Наиболее важные функции-карты в bpftrace

Функция	Описание
`count()`	Подсчитывает число вхождений
`sum(int n)`	Подсчитывает сумму значений
`avg(int n)`	Вычисляет среднее значение
`min(int n)`	Запоминает минимальное значение
`max(int n)`	Запоминает максимальное значение
`stats(int n)`	Возвращает общее количество, среднее и сумму
`hist(int n)`	Выводит гистограмму значений с шагом, равным степени двойки
`lhist(int n, int min, int max, int step)`	Выводит линейную гистограмму значений
`delete(@m[key])`	Удаляет пару ключ — значение из карты
`print(@m [, top [, div]])`	Выводит содержимое карты с необязательными ограничением на вывод определенного числа наибольших значений и делителем
`clear(@m)`	Удаляет все пары ключ — значение из карты
`zero(@m)`	Сбрасывает все значения в карте в ноль

Вывод частоты вхождений в течение каждого интервала:

```
# bpftrace -e 'tracepoint:block:block_rq_i* {@[probe] = count();}
      interval:s:1 { print(@); clear(@); }'
```

Подсчет общего числа байтов, прочитанных системным вызовом read(2):

```
# bpftrace -e 'tracepoint:syscalls:sys_exit_read /args->ret>0/ {
      @bytes = sum(args->ret); }'
```

Гистограммы размеров блоков, успешно прочитанных системным вызовом read(2):

```
# bpftrace -e 'tracepoint:syscalls:sys_exit_read {@ret = hist(args->ret);}'
```

3 Зонды

Поиск зондов:

```
# bpftrace -l 'kprobe:vfs_*'
```

Подробная информация о зонде:

```
# bpftrace -lv tracepoint:syscalls:sys_enter_accept
```

Типы зондов в bpftrace:

Тип (Псевдоним)		Описание
tracepoint	t	Инструментируют статические точки трассировки в ядре
usdt	U	Инструментируют статические точки трассировки в пространстве пользователя
kprobe	k	Инструментируют динамические точки вызовов функций ядра
kretprobe	kr	Инструментируют динамические точки возврата из функций ядра
uprobe	u	Инструментируют динамические точки вызовов функций в пространстве пользователя
uretprobe	ur	Инструментируют динамические точки возврата из функций в пространстве пользователя
software	s	Программные события в пространстве ядра
hardware	h	Инструментируют аппаратные счетчики
profile	p	Производят выборку по времени для всех процессоров
interval	i	Производят выборку в течение интервала (для одного процессора)
BEGIN		Запуск bpftrace
END		Завершение bpftrace

3.1 tracepoint

Зонды типа tracepoint инструментируют статические точки трассировки в ядре. Формат определения:

```
tracepoint:tracepoint_name
```

Страница справочного руководства:

```
ssize_t read(int fd, void *buf, size_t count);
```

В точке трассировки sys_enter_read эти аргументы должны быть доступны как args-> fd, args-> buf и args->count.

```
# bpftrace -lv tracepoint:syscalls:sys_enter_read
tracepoint:syscalls:sys_enter_read
   int __syscall_nr;
   unsigned int fd;
   char * buf;
   size_t count;
```

3.2 usdt

Зонды этого типа инструментируют статические точки трассировки в пространстве пользователя. Формат определения:

```
usdt:binary_path:probe_name
usdt:library_path:probe_name
usdt:binary_path:probe_namespace:probe_name
usdt:library_path:probe_namespace:probe_name
```

Получить список зондов, доступных в файле, можно с помощью параметра `-l`, например:

```
# bpftrace -l 'usdt:/usr/local/cpython/python'
   usdt:/usr/local/cpython/python:line
   usdt:/usr/local/cpython/python:function__entry
   usdt:/usr/local/cpython/python:function__return
   usdt:/usr/local/cpython/python:import__find__load__start
   usdt:/usr/local/cpython/python:import__find__load__done
   usdt:/usr/local/cpython/python:gc__start
   usdt:/sur/local/cpython/python:gc__done
```

Можно получить и список зондов USDT в выполняющемся процессе, в этом случае вместо имени файла следует использовать параметр `-p PID`.

3.3 kprobe и kretprobe

Зонды этого типа используются для динамической инструментации ядра. Формат определения:

```
kprobe:function_name
kretprobe:function_name
```

Зонды kprobe имеют аргументы arg0, arg1, ..., argN — входные аргументы функции, как целые 64-битные целые без знака. Если какой-то из них является указателем на структуру языка C, его можно привести к типу этой структуры.

Единственный аргумент kretprobe - встроенная переменная retval - содержит возвращаемое значение функции. retval всегда имеет тип uint64.

3.4 uprobe и uretprobe

Зонды этого типа используются для динамической инструментации кода в пространстве пользователя. Формат определения:

```
uprobe:binary_path:function_name
```

```
uprobe:library_path:function_name
uretprobe:binary_path:function_name
uretprobe:library_path:function_name
```

3.5 software и hardware

Зонды этого типа инструментируют предопределенные программные (software) и аппаратные (hardware) события. Формат определения:

```
software:event_name:count
software:event_name:
hardware:event_name:count
hardware:event_name:
```

Программные события:

Имя события / Псевдоним Значение счетчика по умолчанию			Описание
cpu-clock	cpu	10^6	Фактическое время процессора
task-clock		10^6	Время использования процессора задачей (увеличивается, только когда задача выполняется на процессоре)
page-faults	faults	100	Отказы страниц
context-switches	cs	1000	Переключение контекста
cpu-migrations		1	Миграция потоков процессора
minor-faults		100	Незначительные отказы страниц: устранены за счет памяти
major-faults		1	Значительные отказы страниц: устранены за счет ввода/вывода из хранилища
alignment-faults		1	Ошибки выравнивания
emulation-faults		1	Ошибки эмуляции
dummy		1	Искусственное событие для тестирования
bpf-output		1	Канал вывода BPF

Аппаратные события:

Имя события / Псевдоним		Значение счетчика по умолчанию	Описание
cpu-cycles	cycles	10^6	Такты процессора
instructions		10^6	Инструкции процессора
cache-references		10^6	Обращения к кэшу процессора последнего уровня
cache-misses		10^6	Промахи кэша процессора последнего уровня
branch-instructions	branches	10^5	Инструкции ветвления
bus-cycles		10^5	Такты шины
frontend-stalls		10^6	Пропуски циклов внешнего интерфейса процессора (например, на время выборки инструкций)
backend-stalls		10^6	Пропуски внутренних циклов процессора (например, на время загрузки/сохранения данных)
ref-cycles		10^6	Циклы обращения к процессору (не масштабируется в турборежиме)

3.6 profile и interval

Зонды этого типа инструментируют события, имеющие отношение к времени. Формат определения:

```
profile:hz:rate
profile:s:rate
profile:ms:rate
profile:us:rate
interval:s:rate
interval:ms:rate
```

Зонды profile срабатывают для всех процессоров, interval только для одного процессора. Во втором поле можно указать:

- hz: герцы (событий в секунду);
- s: секунды;
- ms: миллисекунды;
- us: микросекунды.

4 Специализированные инструменты

4.1 funccount

Подсчитывает события, в частности вызовы функций.

Примеры использования:
Вызывается ли функция tcp_drop()?
```
# funccount tcp_drop
```
Какая функция из подсистемы VFS в ядре вызывается чаще всего?
```
# funccount 'vfs_*'
```
Сколько раз в секунду вызывается функция pthread_mutex_lock() в пространстве пользователя?
```
# funccount -i 1 c:pthread_mutex_lock
```
Какая из строковых функций в libc вызывается чаще всего в системе в целом?
```
# funccount 'c:str*'
```
Какой системный вызов вызывается чаще всего?
```
# funccount 't:syscalls:sys_enter_*'
```
Подсчет вызовов функций виртуальной файловой системы в ядре:
```
# funccount 'vfs_*'
```
Подсчет вызовов функций TCP в ядре:
```
# funccount 'tcp_*'
```
Определение частоты вызовов в секунду функций TCP:
```
# funccount -i 1 'tcp_send*'
```
Определение частоты операций блочного ввода/вывода в секунду:
```
# funccount -i 1 't:block:*'
```
Определение частоты запуска новых процессов в секунду:
```
# funccount -i 1 t:sched:sched_process_fork
```
Определение частоты вызовов в секунду функции getaddrinfo() (разрешение имен) из библиотеки libc:
```
# funccount -i 1 c:getaddrinfo
```
Подсчет вызовов всех функций «os.*" в библиотеке libgo:
```
# funccount 'go:os.*'
```

4.2 stackcount

Подсчитывает трассировки стека, которые привели к событию.

Пример использования stackcount для определения путей выполнения

кода, ведущих к вызову ktime_get():

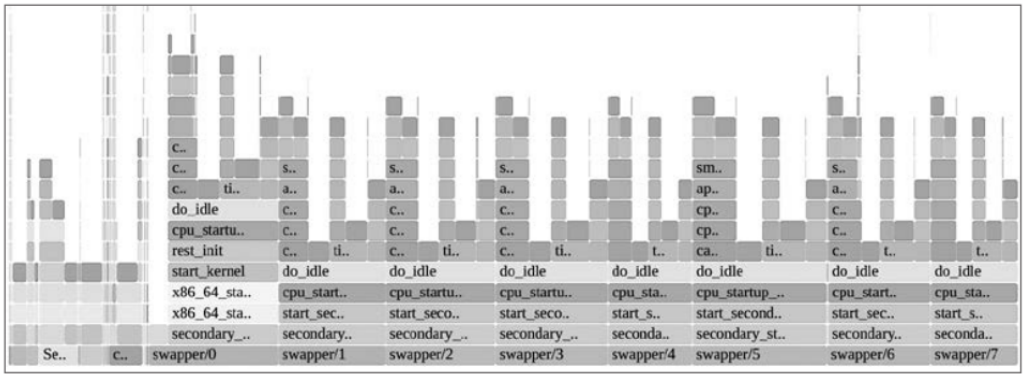
```
# stackcount ktime_get
```

Параметр -P позволяет добавить имена и идентификаторы PID процессов:

```
# stackcount -P ktime_get
```

Создание флейм-графиков:

```
# stackcount -f -P -D 10 ktime_get > out.stackcount01.txt
$ git clone http://github.com/brendangregg/FlameGraph
$ cd FlameGraph
$ ./flamegraph.pl -hash -bgcolors=grey \
   < ../out.stackcount01.txt > out.stackcount01.svg
```



Однострочные сценарии stackcount

Подсчет трассировок стека, приводящих к операции блочного ввода/вывода:

```
stackcount t:block:block_rq_insert
```

Подсчет трассировок стека, приводящих к отправке IP-пакетов:

```
stackcount ip_output
```

Подсчет трассировок стека, приводящих к отправке IP-пакетов, с разделением по PID:

```
stackcount -P ip_output
```

Подсчет трассировок стека, приводящих к блокировке потока и переходу в режим ожидания:

```
stackcount t:sched:sched_switch
```

Подсчет трассировок стека, приводящих к системному вызову read():

```
stackcount t:syscalls:sys_enter_read
```

4.3 trace

Многофункциональный инструмент BCC для трассировки отдельных событий из разных источников: kprobes, uprobes, tracepoints и USDT.

Перехват событий открытия файлов:

```
# trace 'do_sys_open "%s arg2'
```

Трассировка вызовов функции do_sys_open() с выводом имен открываемых файлов:

```
# trace 'do_sys_open "%s arg2'
```

Трассировка возврата из функции ядра do_sys_open() с выводом возвращаемого значения:

```
# trace 'r::do_sys_open "ret: %d retval'
```

Трассировка функции do_nanosleep() с выводом аргумента режима и трассировкой стека в пространстве пользователя:

```
# trace -U 'do_nanosleep "mode: %d arg2'
```

Трассировка запросов в библиотеку pam на аутентификацию:

```
# trace 'pam:pam_start "%s: %s arg1, arg2'
```

trace и структуры:

```
# trace -I 'net/sock.h' \
   udpv6_sendmsg(struct sock *sk) (sk->sk_dport == 13568)'
```

Использование trace для отладки утечек дескрипторов файлов:

```
# trace -tKU 'r::sock_alloc "open %llx retval' \
   '__sock_release "close %llx arg1'
```

4.4 argdist

Многофункциональный инструмент, который суммирует аргументы.

Гистограмма распределения размеров окна:

```
# argdist -H 'r::__tcp_select_window():int:$retval'
```

Вывести гистограмму результатов (размеров), возвращаемых функцией ядра vfs_read():

```
# argdist.py -H 'r::vfs_read()'
```

Вывести гистограмму результатов (размеров), возвращаемых функцией read() из библиотеки libc в пространстве пользователя для PID 1005:

```
# argdist -p 1005 -H 'r:c:read()'
```

Подсчитать число обращений к системным вызовам по их идентификаторам с ис- пользованием точки трассировки raw_syscalls:sys_enter:

```
# argdist.py -C 't:raw_syscalls:sys_enter():int:args->id'
```

Подсчитать значения аргумента size для tcp_sendmsg():

```
# argdist -C 'p::tcp_sendmsg(struct sock *sk, \
      struct msghdr *msg, size_t size):u32:size'
```

Вывести гистограмму распределения значений аргумента size в вызовах tcp_ sendmsg():

```
# argdist -H 'p::tcp_sendmsg(struct sock *sk, \
      struct msghdr *msg, size_t size):u32:size'
```

Подсчитать количество вызовов функции write() из библиотеки libc для PID 181 по дескрипторам файлов:

```
# argdist -p 181 -C 'p:c:write(int fd):int:fd'
```