# 1 perf

## 1.1. perf 相关基础知识

### 1.1.1. 什么是 perf

Perf 是 Linux 系统提供的一个强大的性能分析工具，它能够帮助开发者深入了解和优化应用程序的性能。Perf 基于 Linux 内核的性能事件子系统，可以收集各种性能数据，如 CPU 使用率、内存访问、函数调用等。本手册将介绍 Perf 工具的基本用法、常见的使用场景和一些高级功能。

### 1.1.2. 为什么要学会 perf

我们学习 perf 工具有以下几个重要原因：

- **性能分析和优化**：性能是软件开发中一个关键的方面。通过学习 perf 工具，您可以深入了解应用程序的性能瓶颈，找出性能瓶颈的根本原因，并采取相应的优化措施。这可以帮助您提高应用程序的响应速度、降低资源消耗，从而提升用户体验。
- **系统调优**：性能问题不仅局限于应用程序级别，还可以涉及到整个系统。perf 工具可以帮助您分析系统的性能状况，了解系统资源的使用情况，并进行系统调优。这对于确保系统的稳定性、提高系统的吞吐量和效率非常重要。
- **故障排查**：在开发过程中，可能会遇到各种性能问题和故障。perf 工具可以帮助您快速定位问题所在，了解程序的运行状况，并收集相关的性能数据和调试信息。这样可以加快故障排查的速度，提高开发效率。
- **深入了解内核**：perf 工具是基于 Linux 内核的性能事件子系统实现的，学习 perf 工具也可以帮助您更深入地了解 Linux 内核的工作原理和性能调优机制。这对于理解操作系统的运行机制和进行系统级的性能优化非常有帮助。

### 1.1.3. 什么是性能事件子系统

系统调用是应用程序在执行过程中向操作系统内核申请服务的方法，这可能包含硬件相关的服务、新进程的创建和执行以及进程调度。

性能事件子系统（Performance Event Subsystem）是 Linux 内核中的一个重要组成部分，用于收集和管理系统性能相关的数据。它提供了一种机制，通过配置硬件计数器和软件事件来捕获和记录各种性能事件，如 CPU 周期、缓存命中、指令执行等。

性能事件子系统的主要功能包括：

- 性能事件计数器：性能事件子系统允许开发者配置硬件计数器来测量各种硬件事件，如 CPU 周期、缓存命中、分支预测等。这些计数器位于处理器芯片中，并通过性能事件子系统进行控制和读取。
- 软件事件：除了硬件计数器，性能事件子系统还支持软件事件。软件事件是一种通

过编程方式定义的事件，可以跟踪特定的函数调用、指令执行等。开发者可以根据需要定义自己的软件事件，并利用性能事件子系统进行采集和分析。

- 性能事件采集：性能事件子系统提供了一套 API 和工具，用于配置、启动和停止性能事件的采集。开发者可以使用这些工具来指定要采集的事件和条件，并在运行时对性能事件进行采样和记录。
- 性能事件分析：采集到的性能事件数据可以通过性能事件子系统提供的工具进行分析和可视化。这些工具可以生成性能报告、火焰图等，帮助开发者深入理解应用程序或系统的性能状况，定位性能瓶颈，并进行优化。
- 通过性能事件子系统，开发者可以获取系统的详细性能数据，了解系统的运行状况，并进行性能分析和优化。它为开发者提供了一种强大的工具，使他们能够深入了解应用程序和系统的性能特性，从而提高应用程序的性能和系统的效率。

# 1.2. perf 的使用

直接运行 perf，查看到它的用法：

```
[root@imx6ull:~]# perf

 usage: perf [--version] [--help] [OPTIONS] COMMAND [ARGS]

 The most commonly used perf commands are:
   annotate        Read perf.data (created by perf record) and display annotated code
   archive         Create archive with object files with build-ids found in perf.data file
   bench           General framework for benchmark suites
   buildid-cache   Manage build-id cache.
   buildid-list    List the buildids in a perf.data file
   config          Get and set variables in a configuration file.
   data            Data file related processing
   diff            Read perf.data files and display the differential profile
   evlist          List the event names in a perf.data file
   inject          Filter to augment the events stream with additional information
   kmem            Tool to trace/measure kernel memory properties
   kvm             Tool to trace/measure kvm guest os
   list            List all symbolic event types
   lock            Analyze lock events
   mem             Profile memory accesses
   record          Run a command and record its profile into perf.data
   report          Read perf.data (created by perf record) and display the profile
   sched           Tool to trace/measure scheduler properties (latencies)
   script          Read perf.data (created by perf record) and display trace output
   stat            Run a command and gather performance counter statistics
   test            Runs sanity tests.
   timechart       Tool to visualize total system behavior during a workload
   top             System profiling tool.
   probe           Define new dynamic tracepoints

 See 'perf help COMMAND' for more information on a specific command.
```

## 1.2.1. perf 相关参数

下面是几个常用的选项：

- **perf record <options> <command>**：收集性能数据并保存到文件中。
- **perf report**：分析和查看收集的性能数据。
- **perf top**：实时显示 CPU 占用最高的函数和指令。
- **perf stat <command>**：统计程序运行时的性能事件。

1. 常用的 perf record 选项包括：

- -e <event>：指定要收集的性能事件，比如 CPU 周期计数、缓存命中率等。可以使用 perf list 命令查看可用的事件列表。
- -a：收集系统范围内的性能数据，而不仅仅是单个进程的数据。
- -g：收集调用图信息，用于显示函数调用关系。
- --call-graph <type>：指定调用图的类型，常见的选项有 dwarf 和 fp，分别表示使用 DWARF 调试信息和 Frame Pointer。
- -o <output-file>：指定保存性能数据的文件路径。

2. 常用的 perf report 选项包括：

- --sort <event>：按指定的性能事件对报告进行排序。
- --stdio：将报告输出到标准输出而不是交互式界面。
- --top：只显示前几个性能事件，类似于 perf top 的输出。

3. 常用的 perf stat 选项包括：

- -e <event>：指定要统计的性能事件，类似于 perf record 中的选项。
- --repeat <count>：多次运行命令并统计平均值。
- -p <pid>：统计指定进程的性能数据。

以上只是 perf 工具的一部分常用参数，perf 还有很多其他的选项和功能，可以通过 perf --help 命令或查阅官方文档来获取更详细的信息。使用 perf 需要具有 root 权限或者使用特定的性能事件时可能需要在系统中开启性能计数器功能。在使用 perf 时，建议先对性能数据进行分析和理解，以便更好地优化和改进程序的性能。

### 1.2.2. perf list 命令

Perf 这个工具最早是 Linux 内核著名开发者 Ingo Molnar 开发的，它的源代码在内核源码 tools 目录下。

使用 perf 之前，我们可以先运行一下 perf list 这个命令，然后就会看到 perf 列出了大量的 event，比如下面这个例子就列出了常用的 event。

```
[root@imx6ull:~]#  perf list

List of pre-defined events (to be used in -e):

  alignment-faults                                   [Software event]
  bpf-output                                         [Software event]
  context-switches OR cs                             [Software event]
  cpu-clock                                          [Software event]
  cpu-migrations OR migrations                       [Software event]
  dummy                                              [Software event]
  emulation-faults                                   [Software event]
  major-faults                                       [Software event]
  minor-faults                                       [Software event]
  page-faults OR faults                              [Software event]
  task-clock                                         [Software event]

  mmdc/busy-cycles/                                  [Kernel PMU event]
  mmdc/read-accesses/                                [Kernel PMU event]
  mmdc/read-bytes/                                   [Kernel PMU event]
  mmdc/total-cycles/                                 [Kernel PMU event]
  mmdc/write-accesses/                               [Kernel PMU event]
  mmdc/write-bytes/                                  [Kernel PMU event]

  rNNN                                               [Raw hardware event descriptor]
  cpu/t1=v1[,t2=v2,t3 ...]/modifier                  [Raw hardware event descriptor]
   (see 'man perf-list' on how to encode it)

  mem:<addr>[/len][:access]                          [Hardware breakpoint]
```

从这里我们可以了解到 event 都有哪些类型， perf list 列出的每个 event 后面都有一个"[]"，里面写了这个 event 属于什么类型，比如"Hardware event"、"Software event"等。完整的 event 类型，我们在内核代码枚举结构 perf_type_id 里可以看到。

我们可以通过 ag 命令找到这个枚举结构：

```
book@100ask:~/100ask_imx6ull-sdk/Linux-4.9.88/include$ ag perf_type_id
uapi/linux/perf_event.h
28:enum perf_type_id {
```

```
/*
 * attr.type
 */
enum perf_type_id {
        PERF_TYPE_HARDWARE                      = 0,
        PERF_TYPE_SOFTWARE                      = 1,
        PERF_TYPE_TRACEPOINT                    = 2,
        PERF_TYPE_HW_CACHE                      = 3,
        PERF_TYPE_RAW                           = 4,
        PERF_TYPE_BREAKPOINT                    = 5,

        PERF_TYPE_MAX,                          /* non-ABI */
};
```

- Hardware Event：由 PMU 硬件产生的事件，比如 cache 命中，用于了解程序对硬件特性的使用情况；
- Software Event：内核软件产生的事件，比如进程切换，tick 数等；
- Tracepoint event：内核中静态 tracepoint 所触发的事件，这些 tracepoint 用来判断程序运行期间内核的行为细节，比如 slab 分配器的分配次数等；

### 1.2.3. perf stat 命令

当我们接到一个性能优化任务时，最好采用自顶向下的策略。先整体看看该程序运行时各种统计事件的汇总数据，再针对某些方向深入处理细节，而不要立即深入处理琐碎的细节，这样可能会一叶障目。

有些程序运行得慢是因为计算量太大，其多数时间在使用 CPU 进行计算，这类程序叫

作 CPU-Bound 型；而有些程序运行得慢是因为过多的 I/O，这时其 CPU 利用率应该不高，这类程序叫作 I/O-Bound 型。对 CPU-Bound 型程序的调优和 I/O-Bound 型的调优是不同的。

perf stat 命令是一个通过概括、精简的方式提供被调试程序运行的整体情况和汇总数据的工具。perf stat 命令的选项如下所示。

- -a：显示所有 CPU 上的统计信息
- -c：显示指定 CPU 上的统计信息
- -e：指定要显示的事件
- -p：指定要显示的进程的 ID

```
[root@imx6ull:/home/book]# perf stat -a
^C
 Performance counter stats for 'system wide':

       2346.289001      cpu-clock (msec)          #    1.000 CPUs utilized
               175      context-switches          #    0.075 K/sec
                 0      cpu-migrations            #    0.000 K/sec
                 0      page-faults               #    0.000 K/sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

       2.346441334 seconds time elapsed
```

perf stat 会输出如下:

- task-clock：CPU 利用率，此值越高说明程序的多数时间花费在 CPU 计算上而非 IO；
- context-switches：进程切换次数，记录程序运行过程中发生了多少次进程切换，频繁的进程切换是应该避免的；CPU-migrations：程序在运行过程中发生的处理器迁移次数。
- cache-misses：程序运行过程中总体的 cache 利用情况，如果该值过高，说明程序的 cache 利用不好；
- CPU-migrations：表示进程 t1 运行过程中发生了多少次 CPU 迁移，即被调度器从一个 CPU 转移到另外一个 CPU 上运行；
- cycles：处理器时钟，一条指令可能需要多个 cycles；
- instructions:机器指令数目；
- IPC：instructions/cycles 的比值，该值越大越好，说明程序充分利用了处理器的特性；
- cache-references：cache 命中的次数；
- cache-misses：cache 失效的次数；

通过-e 选项，可以改变 perf stat 的缺省事件(perf list 查看)，perf stat -h 查看帮助信息：

```
[root@imx6ull:~]# perf stat -h

 Usage: perf stat [<options>] [<command>]

    -a, --all-cpus        system-wide collection from all CPUs
    -A, --no-aggr         disable CPU count aggregation
    -B, --big-num         print large numbers with thousands' separators
    -C, --cpu <cpu>       list of cpus to monitor in system-wide
    -c, --scale           scale/normalize counters
    -D, --delay <n>       ms to wait before starting measurement after program start
    -d, --detailed        detailed run - start a lot of events
    -e, --event <event>   event selector. use 'perf list' to list available events
    -G, --cgroup <name>   monitor event in cgroup name only
    -g, --group           put the counters into a counter group
    -I, --interval-print <n>
                          print counts at regular interval in ms (>= 10)
    -i, --no-inherit      child tasks do not inherit counters
    -n, --null            null run - dont start any counters
    -o, --output <file>   output file name
    -p, --pid <pid>       stat events on existing process id
    -r, --repeat <n>      repeat command and print average + stddev (max: 100, forever: 0)
    -S, --sync            call sync() before starting a run
    -t, --tid <tid>       stat events on existing thread id
    -T, --transaction     hardware transaction statistics
    -v, --verbose         be more verbose (show counter open errors, etc)
    -x, --field-separator <separator>
                          print counts with custom separator
        --append          append to the output file
        --filter <filter>
                          event filter
        --log-fd <n>      log output to fd, instead of stderr
        --metric-only     Only print computed metrics. No raw values
        --per-core        aggregate counts per physical processor core
        --per-socket      aggregate counts per processor socket
        --per-thread      aggregate counts per thread
        --post <command>  command to run after to the measured command
        --pre <command>   command to run prior to the measured command
        --topdown         measure topdown level 1 statistics
```

```
perf stat -p $pid -d      #进程级别统计
perf stat -a -d sleep 5   #系统整体统计
perf stat -p $pid -e 'syscalls:sys_enter' sleep 10   #分析进程调用系统调用的情形
perf stat -a sleep 5 #收集整个系统的性能计数，持续 5 秒
perf stat -C 0   #统计 CPU 0 的信息
```

例如：

```
[root@imx6ull:~]# perf stat -p 273 -d




^C
 Performance counter stats for process id '273':

     <not counted>      task-clock
     <not counted>      context-switches
     <not counted>      cpu-migrations
     <not counted>      page-faults
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses
   <not supported>      L1-dcache-loads
   <not supported>      L1-dcache-load-misses
   <not supported>      LLC-loads
   <not supported>      LLC-load-misses

       2.177491334 seconds time elapsed
```

### 1.2.4. perf top 命令

当你有一个明确的优化目标或者对象时，可以使用 perf stat 命令。但有些时候系统性能会无端下降。此时需要一个类似于 top 的命令，以列出所有值得怀疑的进程，从中快速定位问题和缩小范围。

perf top 命令类似于 Linux 系统中的 top 命令，可以实时分析系统的性能瓶颈。perf top 命令常见的选项如下所示：

- -e：指定要分析的性能事件
- -p：仅分析目标进程
- -k：指定带符号表信息的内核映像路径
- -K：不显示内核或者内核模块的符号
- -U：不显示属于用户态程序的符号
- -g：显示函数调用关系图

```
PerfTop:    4051 irqs/sec  kernel:84.6%  exact:  0.0% [4000Hz cpu-clock],  (all, 1 CPU)
-------------------------------------------------------------------------------------------------------------

  81.48%  [kernel]           [k] cpuidle_enter_state
   0.84%  libc-2.23.so       [.] strcmp
   0.71%  libc-2.23.so       [.] memset
   0.37%  libpthread-2.23.so [.] __pthread_rwlock_unlock
   0.34%  [kernel]           [k] _raw_spin_unlock_irq
   0.34%  [kernel]           [k] format_decode
   0.32%  [kernel]           [k] kallsyms_expand_symbol.constprop.1
   0.22%  libc-2.23.so       [.] __libc_calloc
   0.21%  [kernel]           [k] _raw_spin_unlock_irqrestore
   0.21%  [kernel]           [k] number
   0.19%  [kernel]           [k] vsnprintf
   0.19%  perf               [.] 0x000c3d8c
   0.18%  libpthread-2.23.so [.] pthread_mutex_lock
   0.17%  libpthread-2.23.so [.] __pthread_rwlock_rdlock
   0.16%  perf               [.] 0x00080370
   0.16%  perf               [.] 0x000c0278
   0.15%  libpthread-2.23.so [.] __pthread_rwlock_wrlock
   0.14%  perf               [.] 0x0010c2cc
   0.13%  libc-2.23.so       [.] getdelim
   0.13%  libpthread-2.23.so [.] __pthread_mutex_unlock_usercnt
   0.12%  [kernel]           [k] string
   0.12%  perf               [.] 0x000c3d40
   0.11%  libc-2.23.so       [.] memchr
   0.10%  perf               [.] 0x000bc288
   0.10%  perf               [.] 0x000be6ac
   0.10%  libc-2.23.so       [.] strchr
   0.10%  libc-2.23.so       [.] free
   0.09%  perf               [.] 0x000c0230
   0.08%  [kernel]           [k] tick_nohz_idle_enter
   0.08%  perf               [.] 0x000c0250
   0.08%  perf               [.] 0x000c0254
```

## 1.2.5. perf record 命令

perf record 命令可以用来采集数据，并且把数据写入数据文件中，随后可以通过 perf report 命令对数据文件进行分析。

perf record 命令有不少的选项，常用的选项如下：

- -e：选择一个事件，可以是硬件事件也可以是软件事件
- -a：全系统范围的数据采集
- -p：指定一个进程的 ID 来采集特定进程的数据
- -o：指定要写入采集数据的数据文件
- -g：使能函数调用图功能
- -C：只采集某个 CPU 的数据

使用案例：

```
perf record -a -g -- sleep 60
```

```
[root@imx6ull:~]# perf record -a -g -- sleep 60
[ perf record: Woken up 76 times to write data ]
Warning:
Processed 220825 events and lost 1 chunks!

Check IO/CPU overload!

[ perf record: Captured and wrote 19.079 MB perf.data (219223 samples) ]
```

可以看出上述命令生成了 perf.data 文件，这个 perf.data 是 perf 工具收集的性能数据保存的文件，它包含了程序运行期间收集到的各种性能事件信息，如 CPU 使用情况、函数调用图、硬件性能计数器等。要对 perf.data 进行分析，可以使用 perf report 命令或其他相关工具。

## 1.2.6. perf report 命令

perf report 是 perf 工具的一个命令，用于分析和查看收集的性能数据（保存在 perf.data

文件中）。它以交互式的界面呈现，提供了丰富的性能数据信息，帮助开发者定位程序的性能瓶颈。

```
perf report
```

```
[root@imx6ull:~]# perf report
no symbols found in /sbin/init, maybe install a debug package?
Warning:
Processed 220825 events and lost 1 chunks!

Check IO/CPU overload!

# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 219K of event 'cpu-clock'
# Event count (approx.): 54805750000
#
# Children      Self  Command          Shared Object           Symbol
# ........  ........  ...............  ......................  ....................................
#
    96.31%     0.00%  swapper          [kernel.kallsyms]       [k] start_kernel
            |
            ---start_kernel
               |
                --96.31%--cpu_startup_entry
                          |
                           --96.03%--cpuidle_enter_state

    96.31%     0.00%  swapper          [kernel.kallsyms]       [k] cpu_startup_entry
            |
             --96.31%--cpu_startup_entry
                       |
                        --96.03%--cpuidle_enter_state

    96.03%    96.00%  swapper          [kernel.kallsyms]       [k] cpuidle_enter_state
            |
             --96.00%--start_kernel
                       cpu_startup_entry
                       cpuidle_enter_state

     2.03%     0.00%  perf             [unknown]               [k] 0x7eb22914
            |
            ---0x7eb22914
               |
                --2.03%--__GI___libc_write
```

下面介绍一些 perf report 界面显示的内容：

1. 总体信息：

- Samples: 总样本数，表示收集的性能事件的数量。
- Period: 采样周期，表示每个性能事件被采样的频率。
- Duration: 总采样时长，表示性能数据收集的时间跨度。
- Overhead: 性能数据采样本身的开销占比。

2. 报告窗口：

perf report 的主要界面是报告窗口，它会显示各个函数的性能统计信息，包括：

- Percent: 函数运行时间占比。
- Self: 函数自身的运行时间。
- Children: 被该函数调用的子函数的运行时间。
- Call Graph: 函数调用图，显示函数之间的调用关系。

3. 调用图（Call Graph）：

perf report 提供了调用图的功能，用于显示函数之间的调用关系。可以通过上下箭头键和回车键在不同的函数之间进行切换和查看。调用图可以帮助开发者理解函数调用链，找出函数调用开销较大的路径。

4. 性能事件分布：

在报告窗口下方，perf report 会显示性能事件的分布图表。这些图表显示了不同的性能事件，例如 CPU 周期计数、缓存命中率等，随时间的变化情况。通过这些图表，可以直观地了解性能事件的趋势和变化情况。

5. 交互式功能：

perf report 界面支持交互式操作，你可以使用回车键用于查看更多细节。还可以使用快捷键切换不同的视图和排序方式，帮助更好地分析性能数据。

```
    1.82%     0.00%  perf            [kernel.kallsyms]       [k] sys_write
            |
             --1.81%--sys_write
                       |
                        --1.81%--vfs_write
                                  |
                                   --1.80%--__vfs_write
(Enter:next line Space:next page Q:quit R:show the rest)
```

### 1.2.7. 火焰图

终端执行如下：

```
perf record -a -g -- sleep 60
perf script > out.perf
```

ubuntu 进行生成火焰图
先下载 FlameGraph：
下载网页：https://github.com/brendangregg/FlameGraph

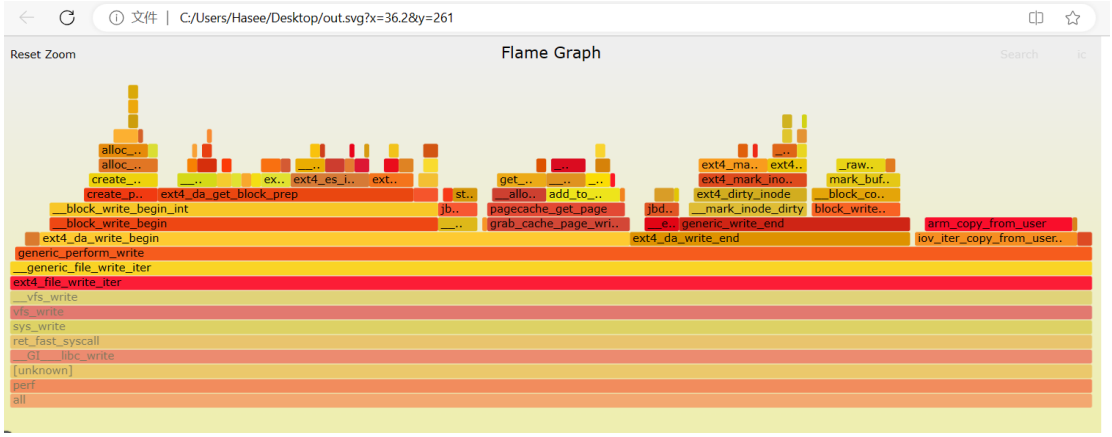
```
FlameGraph-master/stackcollapse-perf.pl out.perf > out.folded
FlameGraph-master/flamegraph.pl out.folded > out.svg
```

```
book@100ask:~$ FlameGraph-master/stackcollapse-perf.pl out.perf  > out.folded



book@100ask:~$
book@100ask:~$
book@100ask:~$
book@100ask:~$
book@100ask:~$ FlameGraph-master/flamegraph.pl out.folded > out.svg
```

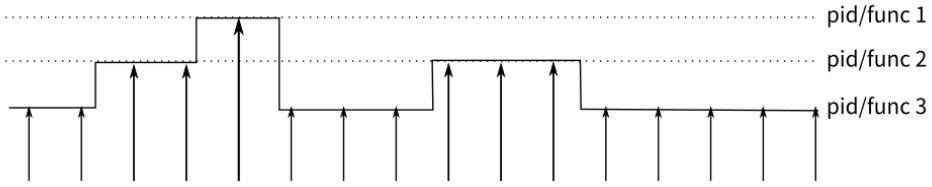
我们用浏览器打开生成的 out.svg，就可以看见下面的图了




# 1.3. perf 的工作原理

### 1.3.1. perf 工作原理

perf 的原理是这样的：每隔一个固定的时间，就在 CPU 上（每个核上都有）产生一个中断，在中断上看看，当前是哪个 pid，哪个函数，然后给对应的 pid 和函数加一个统计值，这样，我们就知道 CPU 有百分几的时间在某个 pid，或者某个函数上了。这个原理图示如下：




以下是 perf 工作的基本原理：

- **性能计数器（Performance Counter）：** Linux 内核中的性能计数器是一组特殊的寄存器，用于记录处理器的硬件事件，如指令执行次数、缓存命中率、分支预测失败次数等。每个处理器都有一组独立的性能计数器。
- **perf 命令行工具：**perf 工具是通过命令行来操作和配置的。通过运行 perf 命令，可以选择不同的子命令和选项，用于指定要监测的性能事件、输出结果的格式等。
- **事件选择：** 使用 perf，可以选择感兴趣的性能事件，例如指令数、缓存命中、缓存失效、分支预测失败等。每种事件对应着硬件性能计数器中的一个计数器。
- **采样和收集：** 当 perf 开始监测一个程序时，它会周期性地对硬件性能计数器进行采样，记录性能事件的计数值。这些采样数据保存在内核空间的缓冲区中。
- **输出结果：** perf 可以将采样数据从内核空间复制到用户空间，并根据用户的选项生成相应的报告和分析结果。可以输出到控制台、文件或者其他工具进行进一步的处理。
- **符号解析：** perf 会对采样数据中的函数地址进行符号解析，将地址转换为对应的函数名和源代码位置，从而提供更易读的分析结果。

perf 可以在用户空间和内核空间运行，因此它能够捕获到包括用户程序和内核函数在内的所有程序执行期间的性能信息。通过 perf，开发者可以深入了解程序在硬件层面上的执行情况，找到性能瓶颈和优化的可能点，从而提高程序的性能和效率。

### 1.3.2. perf 源码

perf 源码位于 tools/perf:

```
book@100ask:~/100ask_imx6ull-sdk/Linux-4.9.88/tools/perf$ ls
arch                    builtin-mem.c        Makefile.perf
bench                   builtin-probe.c      MANIFEST
Build                   builtin-record.c     perf-archive.sh
builtin-annotate.c      builtin-report.c     perf.c
builtin-bench.c         builtin-sched.c      perf-completion.sh
builtin-buildid-cache.c builtin-script.c     perf.h
builtin-buildid-list.c  builtin-stat.c       perf-read-vdso.c
builtin-config.c        builtin-timechart.c  perf-sys.h
builtin-data.c          builtin-top.c        perf-with-kcore.sh
builtin-diff.c          builtin-trace.c      pmu-events
builtin-evlist.c        builtin-version.c    python
builtin.h               command-list.txt     scripts
builtin-help.c          CREDITS              tests
builtin-inject.c        design.txt           trace
builtin-kmem.c          Documentation        ui
builtin-kvm.c           jvmti                util
builtin-list.c          Makefile
builtin-lock.c          Makefile.config
```

在 perf 工具的源代码中，perf.c 是整个工具的主要入口文件，其中包含了 main 函数和一些全局变量的定义。perf.c 负责处理命令行参数解析、子命令分发以及整个 perf 工具的初始化和执行。

- main 函数：main 函数是 C 语言程序的入口函数，perf.c 中的 main 函数负责启动 perf 工具的执行。在 main 函数中，首先进行初始化操作，然后解析命令行参数，接着根据用户提供的子命令执行相应的功能。
- 参数解析：perf.c 中包含一些函数用于解析命令行参数和选项。它们会解析用户输入的命令行参数，检查选项，根据用户的选择调用相应的子命令处理函数。
- 子命令处理：perf 是一个多功能工具，支持多个子命令，如 record、stat、report 等。在 perf.c 中，可以找到这些子命令的定义和初始化，以及它们各自的处理函数。

```
static struct cmd_struct commands[] = {
        { "buildid-cache", cmd_buildid_cache, 0 },
        { "buildid-list", cmd_buildid_list, 0 },
        { "config",     cmd_config,     0 },
        { "diff",       cmd_diff,       0 },
        { "evlist",     cmd_evlist,     0 },
        { "help",       cmd_help,       0 },
        { "list",       cmd_list,       0 },
        { "record",     cmd_record,     0 },
        { "report",     cmd_report,     0 },
        { "bench",      cmd_bench,      0 },
        { "stat",       cmd_stat,       0 },
        { "timechart",  cmd_timechart,  0 },
        { "top",        cmd_top,        0 },
        { "annotate",   cmd_annotate,   0 },
        { "version",    cmd_version,    0 },
        { "script",     cmd_script,     0 },
        { "sched",      cmd_sched,      0 },
#ifdef HAVE_LIBELF_SUPPORT
        { "probe",      cmd_probe,      0 },
#endif
        { "kmem",       cmd_kmem,       0 },
        { "lock",       cmd_lock,       0 },
        { "kvm",        cmd_kvm,        0 },
        { "test",       cmd_test,       0 },
#ifdef HAVE_LIBAUDIT_SUPPORT
        { "trace",      cmd_trace,      0 },
#endif
        { "inject",     cmd_inject,     0 },
        { "mem",        cmd_mem,        0 },
        { "data",       cmd_data,       0 },
};
```

初始化：在 main 函数中，可能会进行一些初始化操作，如设置默认值、检查必要的文件和

硬件支持等。这些初始化操作会确保 perf 工具在执行时处于正确的状态。

输出处理：在一些子命令执行完成后，perf.c 可能会负责处理和输出性能分析的结果，这些结果可以在终端或输出文件中显示。

错误处理：perf.c 中可能包含一些错误处理代码，用于捕获和处理运行时错误，以保证工具的稳定性和健壮性。

值得注意的是，perf.c 是整个 perf 工具的主要入口文件，但 perf 工具的功能远不止于此，其他文件和模块也负责实现更多的性能分析功能和细节。对于想深入了解 perf 工具的实现和细节的开发人员，建议详细阅读整个源代码，并查阅相关文档和注释，以便全面理解其内部工作原理和设计思路。