Runqueue 长度观测实验

一、代码截图及运行截图

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

struct cfs_rq {
    struct load_weight load;
    unsigned int nr_running;
    unsigned int h_nr_running;
};

BEGIN
{
    printf("获取运行队列长度，按 Ctrl-C 结束 \n");
}

profile:hz:99
{
    $my_q = (struct cfs_rq *)curtask->se.cfs_rq;
    $length = $my_q->nr_running;
    printf("队列长度: %u\n", $length);
}
```

```
hacker@ok:~/桌面$ sudo bpftrace ./rqlen.bt
Attaching 2 probes...
获取运行队列长度，按 Ctrl-C 结束
队列长度: 1
队列长度: 0
队列长度: 0
队列长度: 1
队列长度: 0
队列长度: 1
队列长度: 1
队列长度: 1
队列长度: 1
队列长度: 1
队列长度: 1
队列长度: 0
队列长度: 0
队列长度: 0
```

二、ebpf 程序执行的时机

profile:hz:99 这个探针类型是一个周期性采样探针。

profile: 表示这是一个采样探针。

hz:99: 表示每秒钟触发 99 次，也就是每 10.1 毫秒触发一次。

这个 eBPF 程序在每秒钟执行 99 次，周期性地检查当前的 CFS 调度队列长度。

三、关键语句

1.profile:hz:99: 定义一个 profile 探针，每秒运行 99 次。

2.$my_q = (struct cfs_rq *)curtask->se.cfs_rq;: 通过当前任务的调度实体 (curtask->se) 获取 CFS 运行队列。

(1) curtask 是当前正在执行的任务，即当前 CPU 上的进程。

(2)Linux 内核通过一个被称为进程描述符的 task_struct 结构体来管理进程。

(3)curtask->se 是当前任务的调度实体（sched_entity）。

(4)curtask->se.cfs_rq 是当前任务所属的 CFS 运行队列。

(6)包含调度策略字段 struct sched_entity se。

(7)struct sched_entity 描述一个调度实体，它包含了一个实体所有运行时的参数。

(8)其中就包含 struct cfs_rq 结构

-$my_q 通过类型转换将 curtask->se.cfs_rq 转换为 struct cfs_rq 类型的指针。

-$len 获取该运行队列中的任务数量 nr_running。

-通过该结构直接获取运行队列长度 nr_running 即可

curtask: 这是 bpftrace 提供的内置变量，表示当前正在调度的任务 (task_struct)。

curtask->se: 访问当前任务的调度实体 (sched_entity)。

curtask->se.cfs_rq: 通过调度实体访问该任务所属的 CFS 运行队列 (cfs_rq)。

3.$len = $my_q->nr_running;: 获取当前 CFS 运行队列的长度。

4.printf("获取队列长度为： %u\n", $len);: 打印当前 CFS 运行队列的长度。