## 在文件系统的lookup过程中，利用eBPF map进行优化的操作主要涉及以下几个阶段：

1. 用户请求传递到VFS层： 在这一阶段，可以利用eBPF map进行缓存优化。具体而言，可以使用eBPF map来存储已经查找过的路径和对应的inode信息，以避免重复的IO操作和VFS层的开销。
2. VFS层处理请求后，将其传递给FUSE内核： 在这一阶段，可以利用eBPF map进行预取优化。通过eBPF map，可以提前将相关的文件或目录元数据加载到内存中，从而减少后续访问时的IO操作和延迟。
3. FUSE 请求分配和初始化完成后，将FUSE请求传递给适当的FUSE内核队列： 在这一阶段，可以利用eBPF map进行请求过滤和分发优化。通过eBPF map中存储的规则，可以根据请求的属性和优先级，选择性地将请求发送到不同的FUSE内核队列中，以实现更好的请求调度和负载均衡。
4. FUSE 请求被复制到用户空间守护进程： 在这一阶段，可以利用eBPF map进行零拷贝优化。通过eBPF map映射的共享内存，可以避免将请求数据从内核空间复制到用户空间的开销，提高数据传输效率。
5. FUSE 守护进程处理请求并将其传递给底层文件系统： 在这一阶段，可以利用eBPF map进行缓存和读写合并优化。通过eBPF map的缓存功能，可以保存常访问的文件或目录数据，避免重复的底层文件系统访问。同时，eBPF map还可以用于合并多个相邻的读写请求，减少与底层文件系统的通信次数。
6. 底层文件系统执行后，将回复传递给FUSE守护进程： 在这一阶段，可以利用eBPF map进行回复处理优化。通过eBPF map，可以对来自底层文件系统的回复进行解析和转换，以满足特定的文件系统扩展需求或提供额外的功能。

综上所述，利用eBPF map在lookup过程中可以进行缓存、预取、请求分发、零拷贝、缓存、读写合并和回复处理等优化，从而提高文件系统的性能和效率。

## FUSE daemon实现的基本过程：

1. 用户空间程序启动FUSE守护进程：用户空间的程序首先启动FUSE守护进程，通常通过命令行或者其他配置方式指定文件系统的实现及其选项。
2. FUSE守护进程与内核通信：FUSE守护进程会创建一个FUSE文件系统实例，并向内核注册该文件系统。这样，内核就可以将文件系统相关的请求转发给FUSE守护进程处理。
3. 内核将请求传递给FUSE守护进程：当用户在终端或其他应用程序中对文件系统进行操作时，例如读取、写入、创建、删除文件等操作，这些操作都会被内核截获，并传递给FUSE守护进程。
4. FUSE守护进程处理请求：FUSE守护进程接收到内核传递的文件系统请求后，根据具体的操作类型和路径，调用相应的回调函数来处理请求。例如，如果是读取文件内容的请求，守护进程会调用对应的读取文件回调函数来读取文件数据。
5. 文件系统操作的实现：在回调函数中，用户空间程序可以根据具体的文件系统逻辑，执行相应的操作，例如读取文件、写入文件、修改文件属性等。用户空间程序也可以与其他应用程序或存储系统进行交互，以提供所需的功能和数据。
6. FUSE守护进程返回结果给内核：处理完文件系统请求后，FUSE守护进程将处理结果返回给内核。内核会将结果传递给发起请求的应用程序，这样应用程序就可以获得文件系统操作的结果。

## 在libfuse库中，FUSE守护进程的实现过程如下：

1. 调用fuse_main()函数：你的代码需要使用libfuse库提供的fuse_main()函数作为入口点来启动FUSE守护进程。该函数接受命令行参数、FUSE操作结构体以及可选的挂载点参数，并负责与内核通信以处理文件系统请求。
2. 与内核进行通信：在fuse_main()函数内部，libfuse库会与内核建立通信通道。这通常是通过打开/dev/fuse设备文件或使用其他适当的机制来完成的。一旦通信通道建立，libfuse就开始监听来自内核的文件系统请求。

3. 处理文件系统请求：当内核发送文件系统请求时，libfuse库会根据请求类型调用相应的回调函数。你的代码需要在实现的回调函数中执行相应的操作来处理请求。常见的回调函数包括getattr、readdir、read、write等，你可以根据需求实现这些回调函数。
4. 回复内核：处理完请求后，你的代码需要根据实际情况向内核发送响应。libfuse库提供了一些函数（如fuse_reply_*系列函数）来生成并发送响应给内核。你可以使用这些函数来构造合适的响应并将其发送给内核。
5. 守护进程退出：当用户请求终止FUSE守护进程时，比如按下Ctrl+C键，libfuse会收到相应的信号并进行清理操作。在清理过程中，你的代码有机会执行一些清理任务，释放资源等。

### 在libfuse库中，以下是一些与FUSE守护进程实际函数相关的常用函数：

1. fuse_main(): 这是FUSE守护进程的入口函数，用于启动FUSE并处理文件系统请求。
2. fuse_operations 结构体：该结构体定义了回调函数的集合，你需要实现这些回调函数以处理特定的文件系统操作请求。常见的回调函数包括getattr、readdir、read、write等。
3. fuse_reply_* 系列函数：这些函数用于生成响应并将其发送给内核。例如，fuse_reply_attr()用于回复getattr操作的结果，fuse_reply_open()用于回复open操作的结果等。
4. fuse_get_context()：此函数可用于获取当前请求的上下文信息，如进程ID、用户ID等。
5. fuse_mount() 和 fuse_unmount()：这些函数用于挂载和卸载FUSE文件系统。
6. fuse_notify_inval_entry() 和 fuse_notify_inval_inode()：这些函数用于向内核发送无效通知，以使内核刷新缓存。

### 在修改内核代码之后，需要运行以下命令来编译、安装和配置新的内核：

1. `make`： 运行 `make` 命令会编译修改后的内核源代码并生成可执行文件。
2. `make modules`： 使用 `make modules` 命令编译内核模块。如果你修改了内核的某些模块，这一步是必需的。
3. `sudo make modules_install`： 使用 `sudo make modules_install` 命令安装编译好的内核模块。这将把模块文件复制到系统的适当目录中，以便在运行时加载使用。
4. `sudo make install`： 运行 `sudo make install` 命令将编译好的内核文件安装到系统中。它会将内核镜像、initramfs 和相关的引导配置文件复制到适当的位置，并更新引导加载程序 (如 GRUB) 的配置。
5. 重启系统： 修改内核代码后，通常需要重新启动系统以使更改生效。在重新启动后，系统将加载新的内核镜像和模块，以及应用相关的配置。

### ftrace跟踪器的使用：

查看内核配置，确定是否支持跟踪器（tracer）tracing：

```
//  /usr/src/$(uname -r)/.config文件中查看系统配置
CONFIG_FTRACE=y
CONFIG_STACK_TRACER=y
CONFIG_FUNCTION_TRACER=y
CONFIG_FUNCTION_GRAPH_TRACER=y
CONFIG_HAVE_DYNAMIC_FTRACE=y
CONFIG_HAVE_FUNCTION_TRACER=y
CONFIG_IRQSOFF_TRACER=y
CONFIG_SCHED_TRACER=y
CONFIG_FTRACE_SYSCALLS=y
CONFIG_PREEMPTIRQ_EVENTS=y
CONFIG_TRACER_SNAPSHOT=y
```

在ubuntu16.04.3中，上述的大部分配置已被启用，但有两个没有启用：

```
asdf@ubuntu:/usr/src/linux$ cat .config | grep "CONFIG_IRQSOFF_TRACER"
# CONFIG_IRQSOFF_TRACER is not set
asdf@ubuntu:/usr/src/linux$ cat .config | grep "CONFIG_PREEMPTIRQ_EVENTS"
# CONFIG_PREEMPTIRQ_EVENTS is not set
```

没有启用 `CONFIG_IRQSOFF_TRACER` 选项。这意味着 IRQSOFF 跟踪器在你的系统中不可用。

**IRQSOFF 跟踪器是用于跟踪中断禁用期间发生的事件的一种跟踪器。**它需要相应的内核配置和支持才能启用和使用。

没有启用 `CONFIG_PREEMPTIRQ_EVENTS` 选项。这意味着 PreemptIRQ Events（中断抢占事件）在你的系统中不可用。

**PreemptIRQ Events 是一种与中断相关的跟踪功能**，它可以帮助你了解中断处理程序的调度情况。它是基于抢占式内核的功能，需要相应的内核配置和支持才能启用和使用。

Ftrace的使用教程请看：https://zhuanlan.zhihu.com/p/611163404