###### **Spotify音乐软件系统调用分析：进程管理与I/O操作的深入探讨**

2023级 电子信息与电气工程学院 自动化 董宇彤

1. 测试目的

研究在使用spotify-launcher软件执行不同操作（例如播放音乐、添加到歌单等）时，系统调用的模式和行为。

1. **测试基本setup：**

* Arch Linux： 6.9.8-arch1-1 x86\_64
* spotify-launcher： 0.5.4-1
* strace： 6.9

1. **测试用例：**

* start.sh：

打开spotify-launcher软件，点击一下播放键，等待，再点击一下播放键

* search.sh：

用ctrl+L快捷键方式唤醒搜索功能，键入teeth

* play.sh：

按下播放键，播放音乐

* stop.sh:

按下播放键，暂停播放

* like.sh：

按下喜欢键，将歌曲添加入我喜欢；再按一次喜欢键，进入添加歌单界面将歌曲添加至 我的歌单，点击完成

* delete.sh:

按下喜欢键，取消勾选我喜欢和我的歌单，将歌曲从这两个歌单中删除，点击完成

* close.sh:

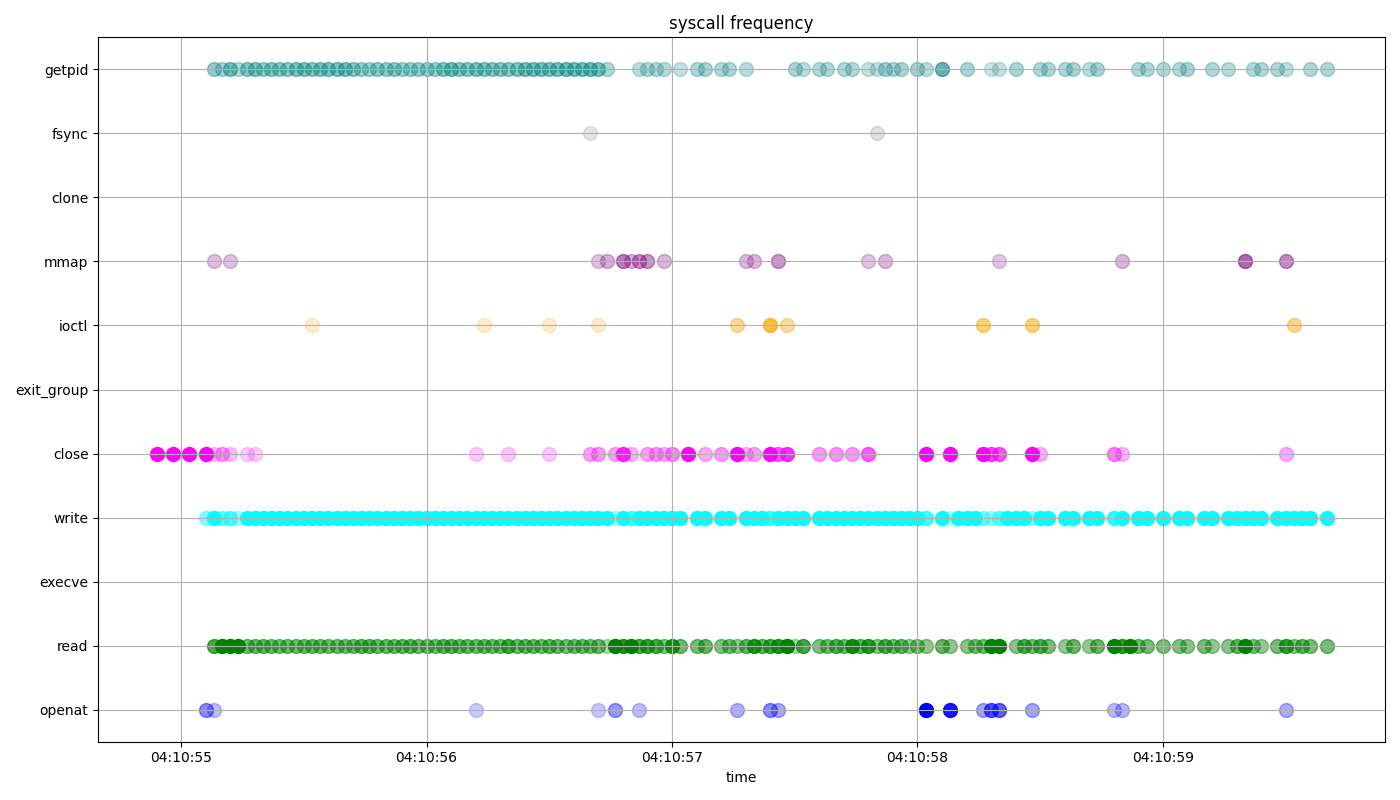
关闭软件

* example.sh:

上面所有操作的整合

1. **测试结果分析**

* **图1：系统调用随时间变化的种类和次数分布图**



* ***用例***：example.sh
* ***横坐标***：时间
* ***纵坐标***：syscall种类
* ***圆点***：颜色深浅表示在这个时间点调用syscall次数的多少

**1. 关于 getpid 系统调用频率的变化：**

* 初始频率较高的原因：

getpid 系统调用用于获取当前进程的PID（进程ID）。在刚打开 spotify-launcher 软件时，操作系统可能需要多次调用 getpid 来为进程分配PID，并执行初始化和加载相关库的操作。这会导致 getpid 的频率较高。

* 之后频率减少的原因：

一旦 spotify-launcher 软件启动并完成初始化，它的主要进程可能会稳定在一个状态，并且不再需要频繁调用 getpid。此时操作系统可能会将一些进程信息缓存，而不需要再次查询。因此，导致 getpid 调用的频率逐渐减少。

**2. 运行过程中 write 和 read 频率高：**

* 数据交换与处理：

在软件运行过程中，频繁的 write 和 read 操作通常反映了程序与外部文件或设备进行数据交换或处理的需求。这些操作可能涉及到读取配置文件、日志文件的写入、临时数据的读取和写入等。频繁的数据交互可以是软件正常运行的必要组成部分。

**3. 打开软件时 close、mmap、openat 频率较高：**

* 初始化和资源准备：

openat：打开文件或目录，准备软件需要访问的资源。

mmap：内存映射文件，将文件映射到进程地址空间中，便于对文件内容的直接访问和处理。

close：关闭不再需要的文件描述符或资源，释放相关的系统资源。

* 在软件启动阶段，这些操作通常用于加载配置文件、读取必要的库文件、准备运行时数据结构等。

**4. 播放音乐时 mmap、ioctl、close、fsync 频率较高：**

* 音频数据处理和持久化：

mmap：可能用于将音频文件或数据映射到内存，以便音频播放程序能够直接访问数据。

ioctl：用于与设备驱动程序进行交互，可能用于音频设备的控制或设置。

close：在播放完成或关闭音频文件时，关闭相关的文件描述符或资源。

fsync：确保音频数据在写入磁盘前被同步，以避免数据丢失。

**5. 加入歌单和从歌单中删除时 fsync、close、openat 频率较高：**

* 文件系统操作：

fsync：确保对歌单文件或元数据文件的修改被写入磁盘，以保证数据的持久性。

close：关闭修改后的文件描述符或资源。

openat：重新打开或访问歌单文件或相关文件，以便于对其进行修改或读取操作。

**6. 停止播放时 mmap、ioctl、close、openat 频率较高：**

* 资源释放和清理：

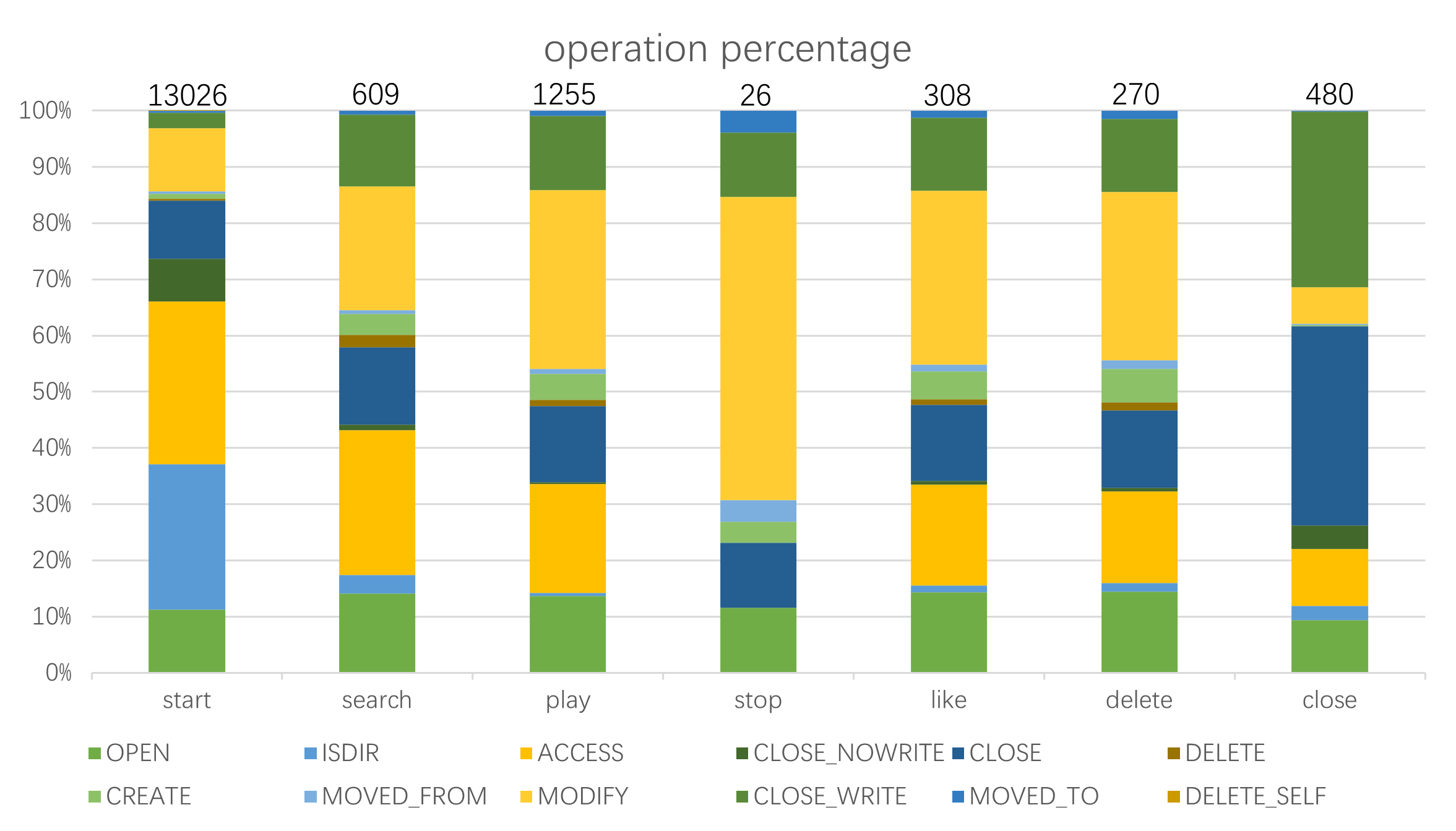
mmap：可能用于释放之前映射的音频数据文件。

ioctl：可能用于发送关闭播放设备或设置设备状态的命令。

close：关闭音频文件或相关的文件描述符。

openat：重新打开或访问需要关闭的资源或文件，进行相应的释放或清理操作。

* **图2：syscall类型比例和操作中涉及的总syscall数**



* ***用例***：所有的用例（除了example.sh）
* ***横坐标***：不同的操作
* ***纵坐标***：不同syscall的百分比
* ***柱顶注释***：在不同操作中syscall的总数

**1. 总syscall次数的差异**

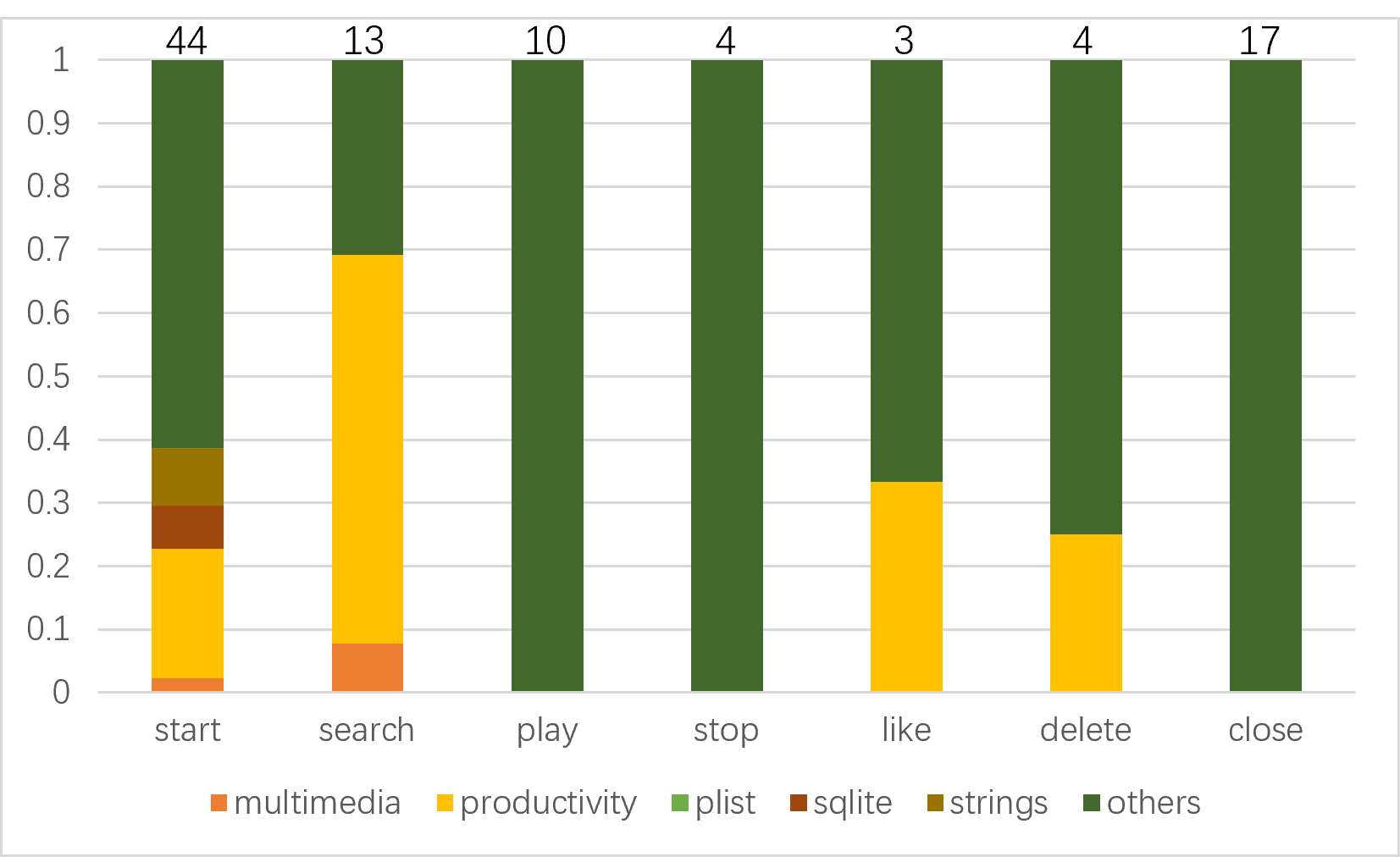
* start 操作具有最高的syscall总数（13026次），这表明在启动或初始化过程中，系统需要执行大量的系统调用来完成各种任务。
* close 操作虽然syscall总数相对较低（480次），但可能表示在系统关闭或资源释放阶段，系统调用次数有所减少，符合常规的资源管理逻辑。
* play、delete 和 search 操作也有较高的syscall次数，分别表明在播放、删除和搜索过程中，系统也需要频繁地与操作系统交互。

**2. syscall类型的分布**

* CLOSE\_WRITE 和 CLOSE\_NOWRITE 类型的syscall在多个操作中均占有较高比例，尤其是在 close 操作中 CLOSE\_WRITE 占比极高（31.25%），这表明文件或资源的关闭操作是常见的系统调用之一。
* ISDIR 和 ACCESS 在 start 操作中占据一定比例，可能用于检查目录存在性和文件访问权限。
* DELETE 和 DELETE\_SELF 在 delete 操作中显著，符合预期，因为删除操作需要这些syscall来删除文件或资源。
* CREATE 和 MOVED\_TO 在 start 和 play 操作中也有一定占比，可能用于创建新文件或移动文件。

**3. 操作的特定模式**

* stop 操作中 CLOSE\_WRITE 占比极高（53.85%），且没有 ISDIR、ACCESS、CREATE 等与文件创建或检查相关的syscall，表明停止操作主要是关闭已打开的资源。
* like 操作中syscall类型分布相对均匀，可能表示该操作涉及多种系统资源的访问和管理。
* **图3：文件类型比例和操作中涉及的总文件数**



* ***用例***：所有的操作
* ***横坐标***：不同的操作
* ***纵坐标***：不同文件类型的百分比
* ***柱顶注释***：在不同操作中涉及文件总数

**1.操作类型与文件数量：**

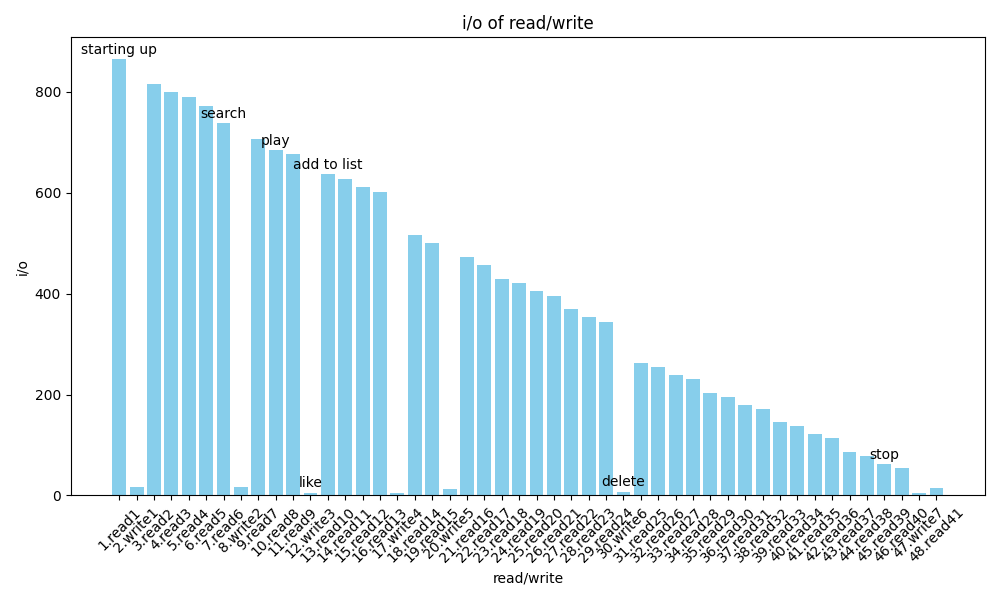
* start 和 search 操作涉及的文件数量最多，分别为44和13，这表明这两种操作在系统中较为核心且频繁。
* play、stop、like、delete 和 close 操作涉及的文件数量相对较少，可能表明这些操作的作用范围较小或每次操作影响的文件数量有限。

**2.文件类型分布：**

* productivity 文件在 search 操作中有显著比例（61.54%），但在其他操作中几乎不出现，说明它可能与搜索功能紧密相关。
* plist、strings 文件类型在大多数操作中占比很低或为0，表明它们可能不是主要操作对象。
* others 文件类型在多个操作中都有一定比例，尤其是在 like、delete 和 close 操作中占据主要比例。进一步分析others包含的的文件类型和涉及到的次数，有以下观察：
  + 临时文件和缓存文件在输出中最为常见，且数量可能较多，因为它们用于存储操作过程中的临时数据和频繁访问的数据。
  + 配置文件和库文件的数量相对较少，但它们是程序正常运行所必需的。
  + 日志文件的数量取决于程序的使用频率和日志记录策略。
  + 系统文件和主题文件的数量通常较少且固定。

**3.操作与文件类型的关联性：**

* 某些操作与特定文件类型有强关联性。例如，start 操作与多种文件类型都有涉及，而 search 操作则与 productivity 和 others 文件类型相关。
* 某些操作（如 play、stop）没有显示与任何文件类型的直接关联，这可能意味着这些操作不直接处理文件，或者它们处理的文件类型未明确列出。
* **图4：read和write系统调用顺序及其IO量分布**



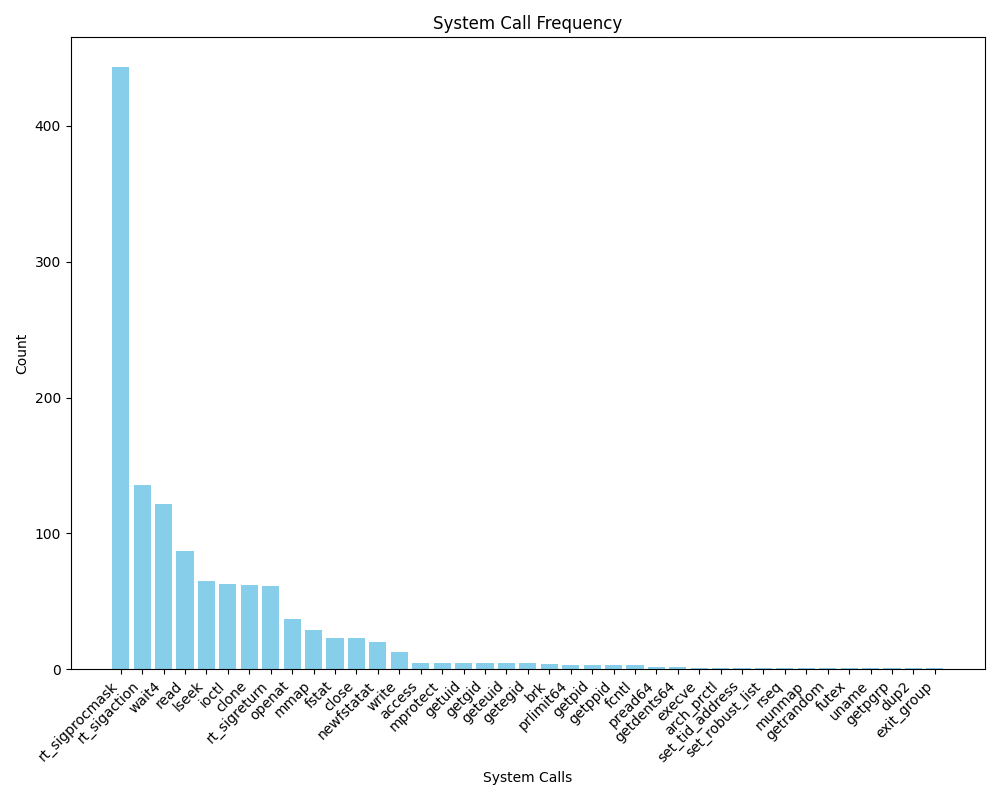
* ***用例***：example.sh
* ***横坐标***：随着时间进行发生的read/write syscall
* ***纵坐标***：i/o字节数
* ***柱顶注释***：当前时间点对软件做的操作

**1.数据规律：**

* Read IO量逐渐减少： 脚本在开始时从外部读取数据的频率减少。可能是因为初始阶段需要加载大量数据，随后脚本将数据缓存到内存中，减少磁盘读取次数。
* Write IO量保持较低水平： 脚本的写操作相对较少，表明主要在内存中进行数据处理、计算或输出，而不是频繁地写入大量数据。

**2.可能原因：**

* 数据缓存： 脚本可能将读取的数据缓存到内存中，减少对磁盘的频繁访问。这种缓存可以是操作系统提供的文件系统缓存，也可能是脚本自身维护的数据结构。
* 文件句柄复用： 脚本可能会复用已打开的文件句柄来减少实际的文件IO操作次数。
* 写操作频率低： 脚本主要在内存中执行计算、分析或输出，因此写操作的次数较少，通常仅在特定事件或条件下进行。
* 脚本执行阶段： 脚本不同阶段对IO的需求不同，初始阶段可能涉及大量数据加载，而后续阶段可能更多地依赖于内存中的数据。
* **图5：脚本运行过程中各系统调用及其数量分布**



* ***用例***：example.sh
* ***横坐标***：随着时间进行发生的read/write syscall
* ***纵坐标***：syscall的次数
* ***柱顶注释***：当前时间点对软件做的操作

**1. 启动和初始化相关的系统调用**

* execve: 1次，这是用于执行新程序的调用，这里表明Spotify Launcher被启动了1次。
* brk, mmap, munmap: 这些调用与内存管理相关。brk被调用了4次，可能用于调整堆的大小；mmap被调用了29次，用于映射内存区域，这通常用于加载程序和数据；munmap仅1次，用于释放之前mmap映射的内存。
* set\_tid\_address, set\_robust\_list, arch\_prctl, rseq: 这些调用与线程和进程的初始化及保护相关，每个调用仅发生1次，表明这些是在程序启动时设置的基础配置。

**2. 文件操作和权限检查**

* access: 5次，用于检查调用进程是否可以对指定的文件执行某种操作（如读、写、执行）。
* openat, fstat, close: openat用于打开文件或目录，共37次，表明在Spotify Launcher的运行过程中，有多个文件或目录被打开；fstat用于获取打开文件的状态，共23次，与openat次数相近；close用于关闭文件描述符，也是23次，与fstat次数相同，这表明每打开一个文件后，都进行了状态检查和随后的关闭。
* getdents64: 2次，用于读取目录内容。
* newfstatat: 20次，类似于fstat，但允许指定目录的入口点，用于获取文件或目录的状态。

**3. 信号处理和同步**

* rt\_sigprocmask, rt\_sigaction: 分别被调用了443次和136次，表明Spotify Launcher对信号处理非常频繁，可能是因为它涉及到多线程或多进程间的同步，以及处理各种异步事件（如用户输入、网络响应等）。
* futex: 1次，用于在用户空间与内核空间之间进行同步，可能是为了等待或唤醒线程。

**4. 用户和系统信息获取**

* getuid, getgid, geteuid, getegid: 每种都被调用了5次，用于获取当前进程的用户ID和组ID，可能是出于安全或权限检查的目的。
* uname: 1次，用于获取系统信息。
* getpid, getppid, getpgrp: 分别被调用了3次、3次和1次，用于获取当前进程ID、父进程ID和进程组ID。

**5. I/O操作**

* read, write: 分别是87次和13次，表明Spotify Launcher进行了大量的读操作（如从文件、网络读取数据）和少量的写操作（如写入日志文件或用户配置）。
* pread64: 2次，类似于read，但允许指定文件偏移量，可能用于读取特定位置的数据。
* lseek: 65次，用于移动文件读取指针的位置，与I/O操作紧密相关。

**6. 进程和线程管理**

* clone: 62次，用于创建新进程或线程，表明Spotify Launcher在运行过程中创建了多个子进程或线程。
* wait4: 122次，用于等待子进程结束，与clone调用次数相关，表明父进程需要等待多个子进程完成。
* rt\_sigreturn: 61次，与信号处理相关，当信号处理函数执行完毕后，会返回到原来的执行点。

**7. 退出和清理**

* exit\_group: 1次，用于终止当前进程组中的所有进程，表明Spotify Launcher正常退出。

1. **总结**

这些数据表明，Spotify Launcher在运行过程中涉及到大量的文件操作、内存管理、信号处理、进程和线程管理以及I/O操作。特别是信号处理（rt\_sigprocmask, rt\_sigaction）和I/O操作（read, write, openat等）非常频繁，这可能是由于Spotify Launcher需要处理用户输入、网络请求、文件读写等多种异步和同步事件。此外，多次的文件打开和关闭操作，以及内存映射和释放，表明Spotify Launcher在资源管理上也非常活跃。