软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(4):2055−2075 [doi: 10.13328/j.cnki.jos.006900]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

HiLog: OpenHarmony 的高性能日志系统*

吴圣垚[1], 王 枫[1], 武延军[1,2], 凌 祥[1], 屈 晟[1], 罗天悦[1], 吴敬征[1,2]

[1](中国科学院 软件研究所 智能软件研究中心, 北京 100190)

[2](计算机科学国家重点实验室 (中国科学院 软件研究所), 北京 100190)

通信作者: 武延军, E-mail: yanjun@iscas.ac.cn

摘 要: 日志是计算机系统中记录事件状态信息的的重要载体, 日志系统负责计算机系统的日志生成、收集和输出. OpenHarmony 是新兴的、面向全设备、全场景的开源操作系统. 在所述工作之前, 包括日志系统在内 OpenHarmony 有许多关键子系统尚未构建, 而 OpenHarmony 的开源特性使第三方开发者可以为其贡献核心代码. 为了解决 OpenHarmony 日志系统缺乏的问题, 主要开展如下工作: ① 分析当今主流日志系统的技术架构和优缺点; ② 基于 OpenHarmony 操作系统的异构设备互联特性设计 HiLog 日志系统模型规范; ③ 设计并实现第 1 个面向 OpenHarmony 的日志系统 HiLog, 并贡献到 OpenHarmony 主线; ④ 对 HiLog 日志系统的关键指标进行测试和对比试验. 实验数据表明, 在基础性能方面, HiLog 和 Log 的日志写入阶段吞吐量分别为 1 500 KB/s 和 700 KB/s, 相比 Android 日志系统吞吐量提升 114%; 在日志持久化方面, HiLog 可以 3.5% 的压缩率进行持久化, 并且丢包率小于 6‰, 远低于 Log. 此外, HiLog 还具备数据安全、流量控制等新型实用能力.

关键词: 操作系统; 日志系统; 开源软件; 数据安全; 流量控制

中图法分类号: TP316

中文引用格式: 吴圣垚, 王枫, 武延军, 凌祥, 屈晟, 罗天悦, 吴敬征. HiLog: OpenHarmony的高性能日志系统. 软件学报, 2024, 35(4): 2055–2075. http://www.jos.org.cn/1000-9825/6900.htm

英文引用格式: Wu SY, Wang F, Wu YJ, Ling X, Qu S, Luo TY, Wu JZ. HiLog: High Performance Log System of OpenHarmony. Ruan Jian Xue Bao/Journal of Software, 2024, 35(4): 2055–2075 (in Chinese). http://www.jos.org.cn/1000-9825/6900.htm

HiLog: High Performance Log System of OpenHarmony

WU Sheng-Yao[1], WANG Feng[1], WU Yan-Jun[1,2], LING Xiang[1], QU Sheng[1], LUO Tian-Yue[1], WU Jing-Zheng[1,2]

[1](Intelligent Software Research Center, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

[2](State Key Laboratory of Computer Science (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

Abstract: Log is an important carrier of a computer system, which records the states of events, and a log system is responsible for log generation, collection, and output. OpenHarmony is a new open-source, distributed operating system for smart devices in all scenarios of a fully-connected world. Prior to the work described in this study, many key subsystems of OpenHarmony, including the log system, had not been built. The open-source feature of OpenHarmony enables third-party developers to contribute core codes. To solve the problem of the lack of a log system of OpenHarmony, this paper mainly does the following work: ① It analyzes the technical architecture, advantages, and disadvantages of today's popular log systems. ② It clarifies the model specifications of the log system HiLog according to the interconnection feature of heterogeneous devices in OpenHarmony. ③ It designs and implements the first log system HiLog of OpenHarmony and contributes it to the OpenHarmony trunk. ④ It conducts comparative experiments on the key indicators of HiLog. The experimental data show that in terms of basic performance, the throughput of HiLog and Log is 1500 KB/s and 700 KB/s, respectively, which indicates that HiLog has a 114% improvement over the log system of Android. In terms of log persistence, the packet loss of

* 基金项目: 中国科学院战略性先导科技专项 (XDA0320000); 国家自然科学基金青年项目 (62202457); 中国博士后科学基金 (2022M713253)
收稿时间: 2022-09-15; 修改时间: 2022-10-24; 采用时间: 2022-12-08; jos 在线出版时间: 2023-06-28
CNKI 网络首发时间: 2023-06-29

HiLog is less than 6‰ with a compression rate of 3.5% for persistency, much lower than that of Log. In addition, HiLog also has some novel practical functions such as data protection and flow control.

Key words: operating system; log system; open-source software; data security; flow control

在计算机系统中, 日志记录了在操作系统中发生的事件或其他软件运行的事件[1]. 一方面, 日志具备极高的实用价值, 系统开发和运维人员需要通过日志对程序中存在的问题进行定位和分析[2−6], 提高工作效率. 另一方面, 在当今大数据时代, 日志又具备极高的商业价值[2−4], 日志记录了大量用户行为习惯信息, 这些信息通过大数据分析可用于了解用户需求, 作为改进产品或孵化新的商业项目的依据. 日志系统需要提供一整套日志相关的支撑能力[7]. 目前, 日志系统广泛采用的标准是 Syslog-RFC5424 (Syslog)[8], 标准规定日志系统需要具备生成、过滤、记录和消息分析的能力. 基于此标准, 目前产业界已经研究并开发了多种受到广泛应用和研究的日志系统, 如 Android 日志系统[9−11]、FTrace[12,13]、NanoLog[14]、Log4j2[15−17]等.

OpenHarmony 是开放原子开源基金会 (OpenAtom Foundation) 所维护的核心孵化项目, 目标是面向全场景、全连接、全智能时代, 基于开源的方式, 搭建一个智能终端设备操作系统的框架和平台, 促进万物互联产业的繁荣发展[18]. 在开源项目运营初期, OpenHarmony 操作系统有许多关键子系统尚未构建, 其中就包括操作系统必备的日志系统. 从 OpenHarmony 操作系统的需求出发, 其日志系统至少需要具备如下 3 个基本特性: ① 多进程日志读写: OpenHarmony 是支持多进程并发的操作系统, 其日志系统需要具备从多进程收集日志的能力. ② 实时日志读写: 作为操作系统的高效调试辅助工具, 日志系统需要具备事件发生-日志输出的实时响应能力. ③ 多内核适配: OpenHarmony 是多内核的操作系统, 其日志系统需要具备多种内核适配能力.

基于对主流日志系统的分析 (具体内容见第 1.2 节), 在技术架构层面, 当前主流日志系统都不适合作为 OpenHarmony 的日志系统, 原因如下: ① Log4j2 是单进程的日志框架[15], 不适用于操作系统这类多进程并发的状况. ② NanoLog 虽然拥有极高的日志写入速率, 但是日志读取需要复杂的后处理机制[14], 不能满足操作系统调试所需的实时读日志需求. ③ FTrace 日志系统仅适用于内核日志读写, 且使用 FTrace 则意味着操作系统的日志功能与 Linux 内核的强耦合[12], 不适用于适配多内核的 OpenHarmony 操作系统. Android 的日志系统 (Android version ⩾ 5.0.0) 的技术架构可以满足内核解耦、多进程、实时读写的需求. 但是该日志系统仍存在吞吐量不足、缺乏资源分配机制、缺乏数据安全能力和面向轻量设备的兼容性差 4 个关键问题 (具体内容见第 1.3 节), 不能满足 OpenHarmony 操作系统对日志能力的需求. 因此亟需为 OpenHarmony 操作系统研发一款日志系统, 解决上述关键问题.

基于以上分析, 本文设计了面向 OpenHarmony 操作系统的高性能日志系统 HiLog. 首先, 为了明确日志系统的研发目标与技术特点, 本文为 HiLog 设计了相应的模型规范. 包括性能原则、资源分配原则、设备兼容性原则和数据安全原则. 接下来, 遵循模型规范设计并实现了 HiLog 日志系统, 通过高效的 IPC 设计和缓冲区管理提高吞吐量、构建流量控制机制实现合理的日志资源分配、模块化设计模式提高轻量级设备的兼容性, 并对日志数据安全能力进行了初步的探索. 最后, 为了检验 HiLog 的表现, 对 HiLog 进行了功能和性能的多重测试, 并与 Android 日志系统进行了横向对比试验. 结果显示, HiLog 在基础性能方面相比 Android 的日志系统 Log 有较大的提升, 例如基于相同硬件平台, 在日志写入阶段 HiLog 的日志吞吐量比 Log 提升 114%; 在日志持久化阶段 HiLog 丢包率小于 6‰, 远低于 Log. 同时, HiLog 还提供 Log 所不具备的数据安全、流量控制、持久化压缩等实用能力.

本文第 1 节介绍日志和日志系统的相关概念, 分析相关的背景工作, 阐述研发 HiLog 日志系统的重要性和必要性. 第 2 节阐述面向 OpenHarmony 操作系统的 HiLog 日志系统的设计原则. 第 3 节描述 HiLog 日志系统的具体设计方案与实现方法. 第 4 节对 HiLog 进行性能测试和技术分析, 并和 Android 的日志系统 Log 进行横向的比较. 第 5 节总结本文工作的优缺点, 并给出下一步工作展望.

1 背景概述

1.1 日志与日志系统

日志是系统运行时数据的信息载体, 是基于时间序列的数据. 从功能角度分析, 日志可以在程序开发时提供问

题排查的指导, 也可以在程序运行时进行证据留痕, 还可以对丢失的数据进行恢复[19], 是当前计算机系统不可或缺的基础能力之一. 随着计算机软硬件的高速发展, 操作系统中的信息产生速度快速增长[20,21], 为了实现操作系统信息的有效捕获, 构建一个用于日志收集、排序、分类、保存、查看的系统十分必要. 因此, 日志系统就应运而生[1].

日志系统不但需要尽可能详细地记录计算机的每一条日志, 还需要为开发者提供便捷的查询服务. 随着计算机系统内程序的数量增多, 日志系统记录的日志逐步增多, 日志的写入速率愈发快速[20], 远远超出人工分析的范围. 因此, 对日志系统的功能提出了更多的要求, 如, 能够对日志进行分类、分级筛查, 关键字检索等.

1.2 主流日志系统

目前业界已存在多种日志系统, 例如 Android 日志系统、Fuchsia 日志系统、FTrace、NanoLog、Log4j2 等, 这些日志系统为操作系统和软件提供了不可或缺的日志能力, 提升了开发和维护的效率.

1.2.1 Android 日志系统

Android 5.0 之前版本的日志系统采用简单可靠的架构, 如图 1 所示, 其中, ① 代表日志写入过程, ② 代表日志读取 (打印) 过程, ③ 代表日志持久化过程. 日志的处理与暂存需要依靠 Linux Kernel 的 logger 模块, 其中, 不论是应用程序日志、系统日志还是内核日志, 都统一存储于位于内核的 ringbuffer 中. 写日志接口采用 Linux 原生的 ioctl (input/output control) 机制实现用户态向内核态传参[22], 读日志接口 logcat 实质是对 logger 命令的封装实现. 采用这种架构的优势在于内存 Copy 次数少; 但缺点也很明显, ioctl 机制需要解析命令行才能跳转到对应处理函数, 会增加日志接口延时和 CPU 负载[23]. 此外, 这种架构对 Linux 内核的耦合极强, 不能脱离 Linux 内核运行, 也必须跟随 Linux 版本进行迭代.

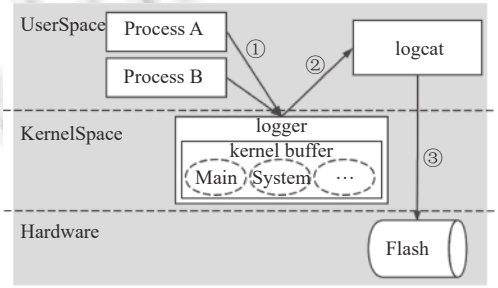


图 1 Android 5.0 之前版本的日志系统

Android 日志系统于 Android 5.0 版本后采用了新架构并一直沿用至今. 如图 2 所示, 其中, ①和②代表用户态日志写入过程, ③和④代表日志读取 (打印) 过程, ⑤代表日志持久化过程, ⑥代表内核态日志的写入过程. 新日志系统通过在用户态维护守护进程 logd 实现对应用日志、系统日志以及内核日志的自动化搜集与暂存. 在日志写入方面, 每一用户态写日志进程与 logd 的进程间通信 (inter process communication, IPC) 采用 Linux 原生套接字 (Socket) 实现; 内核态日志通过扫描/proc/kmsg 文件抓取. 在日志输出方面, logcat 采用 Socket 收发的方式实现传输.

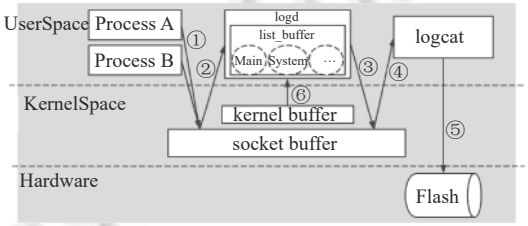


图 2 Android 5.0 及以后版本的日志系统架构 (Android version ⩾ 5.0.0)

新版 Android 日志系统相比旧版本最大的优势在于能够一定程度上与 Linux 内核解耦, 同时在用户态维护缓冲区所需的权限较低, 更为安全. 但是 Android 日志系统没有对日志资源的使用进行合理分配或约束, 因此写日志

程序之间会出现资源竞争, 导致日志显示缺失和 CPU 开销提升的问题. 此外, Android 日志系统未提供相应的敏感数据保护功能, 任何读权限日志用户均可阅读全部日志信息, 即使这些信息中存在其他用户的敏感数据信息; 最后, Android 日志系统对于日志的持久化缺乏优化, 导致持久化文件占用空间过大.

1.2.2 Fuchsia 日志系统

Fuchsia OS 作为 Google 开发的新一代开源操作系统, 其日志系统仍在不断地迭代开发中. 分析其设计思路可知, Fuchsia OS 的日志系统采用 Socket 实现进程间通信, 这一特征与 Android 日志系统的日志系统 (version ≥ 5.0.0) 一致. 两者的不同点在于 Fuchsia OS 日志系统以链表作为缓冲区, 可以有效利用碎片化的内存空间. 目前 Fuchsia 的日志系统暂未投入到实际产品中进行大规模应用, 经过对源码的初步分析, 该日志系统存在内存拷贝次数多、用户态内存频繁分配释放的问题.

1.2.3 FTrace 日志系统

FTrace 作为 Linux 内核日志系统, 用于协助开发人员了解 Linux 内核的运行时行为, 进行故障调试或性能分析. 与前述日志系统“为每个操作系统维护唯一缓冲区”的设计理念不同, FTrace 在架构设计上采用“为每 CPU 核维护一个缓冲区”的设计方案, 因此日志读/写接口延时较低. 同时, FTrace 采用 Page 结构组织数据, 单 Page 上记录一个时间戳, Page 仅内日志记录相对时间差, 节约记录时间所需的存储空间. 但是 FTrace 也存在缺陷, 例如由于 Page 数据存储结构的特性, 导致当前 Page 剩余空间不足仍写入一条日志时, 需要构造新的 Page, 造成内存空间浪费.

1.2.4 Log4j2 日志系统

Log4j2 是开源组织 Apache 维护的一款基于 Java 的单进程日志框架, 采用单进程维护一个缓冲区的设计方案, Log4j2 采用比较交换 (compare and swap, CAS) 方法实现缓冲区加解锁, 降低日志读写接口的延时[24]. 与此同时, Log4j2 使用缓存行填充解决伪共享问题, 隔离更新操作, 进一步提升运行效率[25]. Log4j2 存在的问题是 CAS 方法的 CPU 开销较大, 且存在日志缓冲区修改的 ABA 问题 (ABA problem)[26].

1.2.5 NanoLog 日志系统

NanoLog 日志系统由 Stanford 大学 Platform 实验室开发[14]. 如图 3 所示, 一方面该日志系统采用每线程对应一个缓冲区的设计理念, 另一方面在编译期和运行时分别对日志进行二进制预处理, 使大量数据操作以二进制形式完成. 因此, NanoLog 具备极低的时延和极高的日志吞吐量. NanoLog 读/写日志延时为 Log4j2 的 1/35, 数据吞吐量为 Log4j2 的 40 倍[14]. NanoLog 的缺陷在于采用空间换时间的策略, 内存消耗大, 同时在打印或持久化阶段需要进行复杂的反序列化、格式化、排序等后处理操作, 实质上是将复杂处理后移.

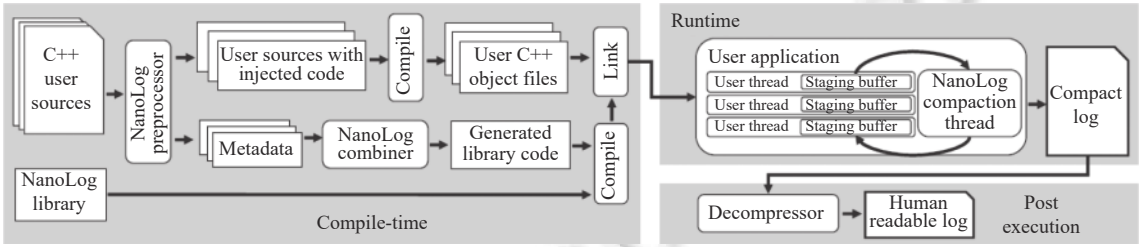


图 3 NanoLog 日志系统的日志写入流程[14]

1.3 HiLog 日志系统的必要性和重要性

OpenHarmony 是面向全设备、全场景的开源操作系统, 日志是 OpenHarmony 不可或缺的基础能力.

从技术架构层面分析, 大多数主流日志系统都不适合作为 OpenHarmony 的日志系统, 原因如下: ① FTrace 日志系统仅适用于内核日志读写, 且使用 FTrace 则意味着操作系统的日志功能与 Linux 内核的强耦合, 不适用于适配多内核的 OpenHarmony 操作系统; ② Log4j2 是单进程的日志框架, 不适用于操作系统这类多进程并发的状况. ③ NanoLog 虽然拥有极高的日志写入速率, 但是日志读取需要复杂的后处理机制, 不能满足操作系统调试所需的实时读日志需求.

因此在上述主流日志系统中, Android 的日志系统 (Android version ⩾ 5.0.0) 的技术架构脱颖而出, 可以满足内核解耦、多进程、实时读写的需求. 但是, 该日志系统仍存在一些关键问题亟待解决.

(1) 吞吐量 (throughput) 问题: Android 日志系统的吞吐量已经不能完全满足当下的丰富软件生态, 负载超过吞吐量将会导致严重的日志丢包问题. Yang 等人通过与 Google、VMware 工程师的交流, 发现在代码评审环节中, 有相当多的时间被用来讨论保留还是删除日志语句; 并且为了向吞吐量妥协, 剔除了大量有用的调试信息, 导致花费更多的调试时间[14], 形成恶性循环. 事实上, 主流日志系统的吞吐量瓶颈在当下已经切实导致了开发效率的下降.

(2) 资源分配问题: Android 日志系统仅对进入日志缓冲区的日志总流量做出限制, 而未对每个程序的日志流量进行限制. 从而导致各个程序会对写日志资源进行竞争, 竞争的结果是不确定、不公平的, 一个日志流量正常的程序很可能因为一个违规高速写日志的程序而失去日志记录. 并且作为单一程序的开发者没有责任和动力去规范程序的写日志速率, 从而节约日志总流量这项公共资源.

(3) 数据安全问题: 包含敏感数据的日志会流向日志系统, Android 日志系统没有相应的安全能力对数据进行保护. 通过与华为、润和等公司进行沟通, 发现各种厂家对于自身硬件关键参数的保护需求是十分迫切的. 但是基于对 Android 和 OpenHarmony 源码的日志语句分析, 在日志中普遍存在常用指针、内存地址、设备参数等关键信息; 同时也存在用户操作信息、用户名等用户敏感数据. 这些数据为开发者提供了便利, 但也允许任何日志的使用者都可以轻易获取上述数据, 带来严重的数据安全问题.

(4) 轻量设备兼容问题: Android 日志系统没有特别面向资源受限设备进行兼容设计. Android 日志系统在用户态维护独立缓冲区, 保存包括 Linux 内核日志在内的所有日志数据, 因此需要消耗大量的内存资源, 导致难以在资源受限设备上运行.

因此, 为了解决上述吞吐量、资源分配、数据安全和轻量设备兼容 4 个关键问题, 当前亟需为 OpenHarmony 操作系统设计与开发一款日志系统, 构成 OpenHarmony 操作系统软件供应链的重要节点[22].

2 HiLog 日志系统模型规范

尽管 HiLog 与 Android 日志系统有相同的基础架构, 但是 OpenHarmony 的面向全设备、面向全场景的独特性使得 HiLog 必须具备高性能、高设备兼容性等新特性. 为了明确开发目标与技术特点, 本节为 HiLog 设计了相应的模型规范, 包括性能原则、资源分配原则、设备兼容性原则和数据安全原则. 具体内容如下.

2.1 性能原则

衡量日志系统性能的指标主要是日志吞吐量, 吞吐量代表日志系统在不丢包的情况下日志收发的速率上限, 代表了日志系统在高速日志读写情境下精确捕获日志信息的能力. 随着应用生态的发展, 操作系统中各个应用的写日志需求逐渐增长, 对日志系统的吞吐量提出了更高的要求. 当操作系统中的写日志需求超过了日志系统的吞吐量时, 将会出现丢包问题. 丢包的日志无法被日志系统记录, 亦无法向程序开发者、维护人员提供任何有效信息. 事实上, 当前主流日志系统的吞吐量瓶颈问题已经切实地影响了程序的开发, 开发者们不得不向性能妥协, 减少日志的写入需求.

在吞吐量的影响因素方面, 一方面硬件平台的性能起到决定性的作用, 另一方面日志系统的合理设计和优化也可以使其在相同的硬件平台上达到更高的吞吐量. 摩尔定律失效警示软件开发者不能仅通过期待硬件性能的高速增长来满足软件的性能需求, 因此 HiLog 日志系统应当针对高吞吐量需求进行设计, 从软件层面解决吞吐量瓶颈问题.

2.2 资源分配原则

系统资源不是无限的, 因此需要考虑分配问题. 从操作系统层面看, 日志系统作为操作系统的信息记录者, 不应抢占过多的系统资源 (如 CPU 资源), 影响其他程序的正常运作. 从日志系统层面看, 一个程序的日志业务也不应占用过多的日志系统资源, 影响其他程序的日志业务. 因此, HiLog 应当在设计时考虑资源分配问题. 一方面在操作系统层面合理分配日志系统和其他程序占用的资源, 另一方面是在日志系统层面合理分配各个程序占用的日

志资源.

2.3 设备兼容性原则

近年来, 以智能手机为主的智能终端创新空间逐步收窄、市场增量接近饱和, 而智能可穿戴设备和物联网设备等轻量级设备逐渐成为智能终端产业下一个增长热点[27], 操作系统和相关软件对轻量设备的兼容性成为近年来的技术热点. OpenHarmony 操作系统即是一种面向全设备的操作系统, 全设备不但包括常见的手机、平板、计算机、智能手表等计算资源相对充足的电子产品, 也包括资源受限的轻量化设备, 如蓝牙耳机、键盘、智能音箱、传感器等. OpenHarmony 将不同设备依据内存大小等参数分为 5 个等级 (L1–L5), 并针对不同等级的设备开发了基于 LiteOS-m 内核的轻量系统 (mini system)、基于 LiteOS-a 内核的小型系统 (small system) 和基于 Linux 内核的标准系统 (standard system), 3 种操作系统具体划分情况如表 1 所示.

表 1 OpenHarmony 软硬件平台划分与对应关系

设备等级	内存范围	系统类型	代表设备
L1	128 KB–128 MB	轻量系统、小型系统	传感器、穿戴设备
L2	128 MB–4 GB	小型系统、标准系统	智能摄像头、路由器
L3–L5	>4 GB	标准系统	手机、平板

表 1 中, L1 级别的设备对应的就是轻量设备, 从表 1 数据可以看出, 轻量设备的内存非常小. 此外轻量级设备由于体积与功耗, 其计算资源和存储资源受到了较大的限制. 综上所述, HiLog 在设计时应当注意 3 个方面以提升对轻量设备的兼容性: ① 减少 CPU 占用, ② 减少内存占用, ③ 减少存储空间占用.

2.4 数据安全原则

在操作系统的开发过程中, 开发人员会使用日志系统记录包括指针、内存地址、函数和错误代码在内的多种系统关键信息, 作为调试工作的辅助信息. 同时, 在操作系统的使用过程中, 日志记录了大量用户行为习惯等敏感数据. 如果不对这些信息进行保护, 则任何阅读日志的人都可以直接获取这些数据, 引发操作系统的安全问题和用户敏感数据泄露问题. 常见的隐私保护方法有匿名化[28]、同态加密[29]、差分隐私[30]等, 这些方法需要基于静态的、结构相同的数据集合计算数据之间的相关性, 构建字段隐藏规则, 可以有效利用在图、表结构表征的数据集合中. 但是日志是随时间变化的长文本数据集合, 难以适用这些方法, 原因在于: ① 基于时间序列意味着日志数据是随时间快速更新的, 每次更新都需要重新计算数据之间的关系, 计算开销是昂贵的, 特别对于移动设备来说是难以接受的. ② 长文本缺乏字段概念, 日志语句的长短、句式各不相同 (结构不同), 难以基于规则分辨需要保护的内容.

综上所述, HiLog 应当具备一定的日志数据安全能力, 但是同时需要保证轻量化, 不影响日志系统和操作系统的性能.

3 HiLog 日志系统设计实现

在实践层面上, 基于上述 4 种设计原则, 给出 HiLog 日志系统的设计方案. 下文将从日志的数据结构、日志系统的功能、日志系统模块设计 3 个方面描述 HiLog 日志系统.

3.1 日志数据结构定义

日志需要明确、详尽地体现程序运行时的相关信息. 作为 OpenHarmony 操作系统的日志, 为了协助开发、运维人员定位问题. 因此除了由使用者定义的日志标题和内容外, 还需要提供一些的辅助信息, 用于日志信息的分类和筛选. 同时, 为了减少内存和 IPC 负担, 这些信息需要尽可能精简.

3.1.1 日志类型

OpenHarmony 操作系统由下至上分为内核层、系统层和应用层. 内核层可由面向标准系统的 Linux 内核或面向轻量系统的 LiteOS 内核构成; 系统层主要由 OpenHarmony 操作系统的各个子系统构成; 应用层则由运行于

OpenHarmony 上的系统应用以及第三方应用构成. 针对这种分层架构, 操作系统的开发人员也分为 3 类, 分别为内核开发者、系统开发者和应用开发者, 不同开发者关心的信息是不同的. 因此为了方便开发者区分不同层级产生的日志, HiLog 将日志分为内核日志、系统日志和应用日志 3 类, 并实现日志的分类管理.

3.1.2 日志级别

为了方便开发者和运维人员快速分辨系统状态的严重程度, 日志应当基于记录事件的重要程度划分级别. 分级的标准如下: ① 日志的级别数目不应过多或过少, 防止检索困难或分类标准不明. ② 每个日志级别应当有清晰的使用标准, 开发者在开发时不可混用. ③ 在写入时, 每条日志都应当分配到一个日志级别. ④ 在输出时, 每个级别的日志都需要采用不同的字体或者颜色来区分. 基于上述标准, HiLog 将日志分为调试 (Debug)、通知 (Info)、告警 (Warn)、错误 (Error) 和致命错误 (Fatal) 这 5 个等级, 相应的使用标准和输出特征如后文表 2 所示.

3.1.3 日志数据结构

HiLog 日志除了提供日志类型 (Type)、级别 (Level) 以外, 还需要提供版本 (Version)、时间 (Time)、进程 ID (PID)、线程 ID (TID)、业务 (Domain) 共计 5 项辅助信息. 为了减少 IPC 开销和存储开销, 提高日志系统的日志吞吐量, HiLog 日志在传输和存储时采用位域定义结构体, 减少信息所占用的空间. 相较于全部使用 C 语言基本数据类型定义的结构体, 每条日志体积减小 134 bits, 体积最高缩小 59.8 % (Tag 和 Content 为空时), HiLog 日志数据结构如表 3 所示.

表 2 HiLog 日志级别定义

HiLog日志级别	使用标准	输出特征
Debug	展示调试时用于详细了解系统运行状态的信息	紫色字体
Info	告知操作执行或状态变化	白色字体
Warn	警示不正确的状态, 但是系统可以继续运行	黄色字体
Error	报告错误已经发生, 不确定系统是否可以继续运行	红色字体
Fatal	报告严重错误发生, 系统继续运行会存在极高风险	红棕色字体

表 3 HiLog 日志数据结构

序号	信息	位域类型	位域宽度 (bits)
1	Version	unsigned int	8
2	Time	unsigned int	40
3	PID	unsigned int	5
4	Level	unsigned int	3
5	TID	unsigned int	8
6	Domain	unsigned int	8
7	Type	unsigned int	2
8	Tag	char[]	8–256
9	Content	char[]	8–8192

3.2 日志系统功能定义

基于日志系统的使用需求, HiLog 需要提供日志写入、日志读取、日志控制 3 类能力. 下面将分别对 3 类能力做出说明.

(1) HiLog 提供日志写入能力, 包括日志生成、日志排序、日志暂存. 日志生成的作用是在系统运行时产生日志信息; 日志排序功能按照时间戳对写入的日志内容进行排序; 日志暂存就是维护一个内存缓冲区, 用于将完成排序的日志存储在其中. 在开发时 HiLog 的使用者通过引入 libhilog 的头文件, 使用 libhilog 提供的写日志接口编写程序, 在程序运行时 libhilog 即可生成日志. libhilog 在生成日志过程中还提供数据保护、进程流控等辅助能力. 接下来, 在标准 HiLog 中, hilogd 收集来自各个 libhilog 的日志信息, 按时间进行排序, 并进入缓冲区暂存. 在轻量 HiLog 中, 日志缓冲区是 LiteOS 的 kernel_log_buffer, 相应的日志排序和暂存能力由 kernel_log_buffer 实现.

(2) HiLog 提供日志输出能力, 包括日志打印、日志持久化的能力. 日志打印功能可以读取暂存的日志写入到标准输出 (std_out), 并且支持通过辅助信息等特征进行筛选; 日志持久化功能可以将暂存的日志写入文件, 并进一步地提供压缩功能. 在运行时 HiLog 使用者可以通过 hilogtool 命令行工具执行日志打印、持久化等输出工作.

(3) HiLog 提供日志系统控制能力, 其中包括提供数据安全、进程流控、业务流控、缓冲区以及持久化的配置能力. 通过对 hilogtool 输入相应指令, 系统开发者即可在 HiLog 运行时实现上述能力的自定义, 以满足在不同的硬件平台和业务场景的适配需求. 例如, 当操作系统内存空间紧张, 可以缩小日志缓冲区空间, 为其他程序让出更多内存. 又例如, 当操作系统 CPU 负载较高, 可以降低流量控制阈值, 减少 HiLog 日志处理消耗的 CPU 资源.

3.3 架构与模块设计

以上为 HiLog 日志系统的日志数据结构定义和功能定义, 下文将依据 HiLog 日志系统的模型规范, 给出 HiLog 日志系统的架构与模块设计, 并对其中重要的模块给出解析和说明.

3.3.1 HiLog 日志系统体系结构

HiLog 日志系统的架构设计如图 4 所示, 主要包括 libhilog、hilogtool 和 hilogd 这 3 个模块, 其中, 由蓝色和红色的箭头分别代表标准和轻量 HiLog 的日志流动方向. 开发者通过 libhilog 进行日志写入, hilogtool 进行日志输出. HiLog 对于计算资源充沛的系统实现了标准 HiLog, 通过维护守护进程 hilogd 实现高性能的日志缓冲区管理; 而对于计算资源受限的轻量、小型系统设计了轻量 HiLog, 直接将日志写入内核的缓冲区中. 现将每个模块描述如下.

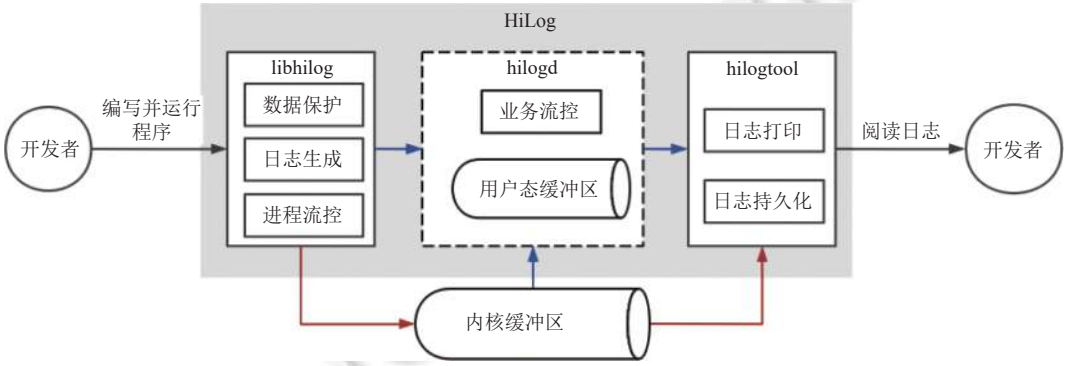


图 4 HiLog 日志系统整体架构

(1) libhilog 模块提供写日志能力, 一方面提供静态写日志接口, 另一方面负责运行时日志生成. 在附加功能方面, libhilog 创新性地提供写日志接口的敏感数据标识, 实现数据安全; 同时 libhilog 还提供基于进程的日志流控机制, 实现对所有进程日志写入资源的合理分配. 对于 L2–L5 级别的平台, 日志经过添加敏感数据标识和流控后, 被发送至 hilogd 模块; 对于 L1 平台, 日志经过添加敏感数据标识和流控后, 将日志直接写入内核缓冲区. libhilog 生成各类型日志的原理: 在开发时, 需要 HiLog 的使用者在开发程序时引入 libhilog 的头文件, 调用 HiLog 的写日志接口. 在编译构建时, 将开发的程序模块链接到 libhilog 的动态链接库. 在程序运行时, 对应进程会加载动态链接库. 当程序运行至写日志接口时, 会调用动态链接库中的写日志方法, 生成日志.

(2) hilogtool 模块提供读日志能力, 一方面提供与操作系统 Shell 交互的能力, 另一方面负责执行读日志任务. 开发者通过 Shell 命令控制日志打印或日志持久化任务. 对于 L2–L5 级别的平台, hilogtool 从 hilogd 读取日志; 对于 L1 平台, hilogtool 从内核缓冲区读取日志.

(3) hilogd 模块是面向 L2–L5 平台设计的高性能日志缓冲区 (hilog_buffer) 及其管理模块. 提供日志监听、排序和存储的功能, 其运行时具备系统守护进程的特性. hilogd 与系统的其他模块是解耦的, 在面向 L1 级别的 OpenHarmony 平台时, 可以不加载 hilogd 模块, 以达到节省系统资源的目标, 增强 HiLog 对资源受限设备的兼容性.

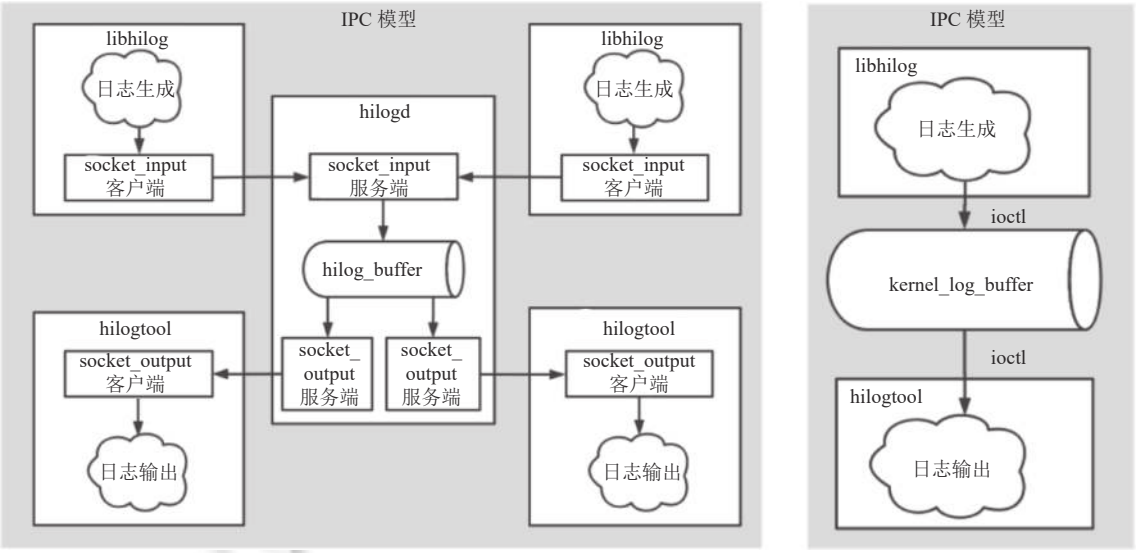
图 4 中的内核缓冲区在不同 OpenHarmony 系统中拥有不同的意义. 在标准 OpenHarmony 系统中内核缓冲区是指 Linux 的内核日志缓冲区, hilogd 将会读取其中的日志信息到 hilog_buffer, 保证 hilog_buffer 中拥有系统的全量日志信息. 在轻量、小型 OpenHarmony 系统中内核缓冲区是指 LiteOS 内核的内核日志缓冲区, 负责暂存全量的日志信息.

在编译构建时, 可以通过自定义选择构建标准 HiLog 或轻量 HiLog, 目的在于实现良好的设备兼容性. 标准 HiLog 在用户态维护高性能的日志缓冲区 (hilog_buffer), 适用日志流量较大的场景, 对于设备的内存需求较高, 其硬件约束条件为 OpenHarmony 硬件标准的 L2 及以上设备 (即内存大于 128 MB 的设备). 轻量 HiLog 利用操作系

统内核日志缓冲区 (kernel_log_buffer) 作为 HiLog 的缓冲区, 适用日志流量较小的场景, 对于设备的内存需求量极小, 其硬件约束条件为 OpenHarmony 硬件标准的 L1 设备 (即内存介于 128 KB–128 MB 之间的设备).

3.3.2 HiLog 日志系统 IPC

图 5(a) 描绘了标准 HiLog 进程间通信模型. 一方面各个不同进程产生的日志需要通过 IPC 收集到 hilogd 进程, 另一方面读日志进程又需要通过向 hilogd 进程请求日志数据, 因此日志系统的 IPC 效率对日志吞吐量存在重要影响. 标准 HiLog 采用套接字 (Socket) 作为 IPC 实现方案, 在写入阶段构造 socket_input 客户端/服务端; 在输出阶段构造 socket_output 客户端/服务端.



(a) 标准 HiLog (b) 轻量 HiLog

图 5 HiLog 日志系统的进程间通信模型

socket_input 服务端采用多路 IO 复用模型 (IO multiplexing) 构建. 由于一般情况下, 系统中会存在多个系统进程和第三方应用进程, 因此服务端需要同时应对多个客户端, 日志的写入存在并发特征; 同时, 每个进程在每个时间段产生的日志数量是不定的, 且每一条日志的长度 (字节数) 也是不等的, 因此日志的写入数据量存在时间分布不均匀特征. 此时如果采用多线程策略, 对于每一个客户端都使用独立线程进行处理无疑是资源消耗的. 基于以上情景, 综合考虑性能和系统资源消耗, 采用多路 IO 复用模型构建 socket_input 服务端, 使服务端同时监听所有客户端的文件描述符, 一旦有客户端就绪就进行日志的接收, IO 复用使连接在服务端和多个客户端之间流转, 适用于数据的收发时间不定、收发量不均的情况.

socket_input 客户端采用非阻塞 IO 模型 (non-blocking input/output) 构建. 由于 socket_input 服务端采用单线程的多路 IO 复用模型构建, 因此不能保证对每个客户端日志的实时接收, 如果此时客户端采用同步调用方式会导致进程阻塞 (服务端在完成数据接受前不会相应其他客户端的请求), 造成性能的下降并影响后续日志的写入. 因此 libhilog 的客户端采用异步调用方式构建, 即不论服务端是否接收完成, 都持续发送后续日志.

socket_output 客户端/服务端均采用阻塞 IO (blocking input/output) 模型构建. 由于读日志事件的数据量较大且需要确保数据到达的先后顺序, 且从需求分析不会同时存在太多读日志进程, 因此阻塞 IO 不会给系统带来过大的负担. 因此在输出阶段, 每个读日志进程加载 hilogtool, 维护各自的 socket_output 客户端向 hilogd 发送读日志请求, hilogd 对于每一个客户端创建一个线程操作 socket_output 服务端.

轻量 HiLog 的 IPC 模型如图 5(b) 所示, 受计算资源的限制, 在用户态维护一个系统守护进程用于日志收发和存储是奢侈的, 因此面向 L1 的 HiLog 日志系统的用户态和内核态日志都将写入内核态的 kernel_log_buffer 中, 读

日志进程也直接从 kernel_log_buffer 中读取日志, IPC 机制为 ioctl.

3.3.3 HiLog 日志系统数据安全

如第 2.4 节所述, 常见的隐私保护方法有匿名化、同态加密、差分隐私等, 但是受限于计算开销等问题, 这些方法难以应用到日志这种基于时间序列的长文本数据结构上. 需要探索一种轻量化的日志数据安全能力. 为了平衡安全性和性能开销, 在设计 HiLog 的数据安全能力时重点考虑了变量的安全问题. 因为在程序开发的过程中, 开发者可以将变量作为日志参数进行打印输出. 相比于常量, 变量仅在程序运行时产生并且会不断变化, 仅基于静态的源码分析难以捕捉是否有敏感数据会被日志系统记录, 因此变量是敏感数据泄露的重要风险因素.

HiLog 通过识别敏感数据标识提供数据安全能力, 敏感数据标识分为 2 种, 分别是公开标识{public}和隐藏标识{private}. 如图 6 所示, 写日志用户需要在格式化占位符 (Format Placeholder, 如 %s, %d 等) 中修饰{public}方能将对应参数值写入到 hilog_buffer 中. 若修饰{private}, 在开启数据安全能力的情况下, libhilog 会以字符串“<private>”替换原参数后, 再将对应日志发送到 hilog_buffer, 这样对应参数的真实值就不会记录在 hilog_buffer 中. 后续读取到或者持久化的日志中, 将不会存在该参数的真实值, 相应的信息得到了保护.

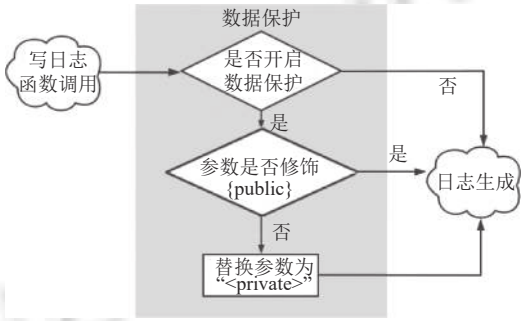


图 6 HiLog 日志系统的数据安全能力

一般地, 用于开发和调试的 OpenHarmony 开发者版本可以关闭数据安全能力, 即便添加了{private}标识, 开发者仍可以便利地利用全部的日志信息完成开发和调试工作. 而 OpenHarmony 发行版系统强制开启数据安全能力, 相同的程序运行在 OpenHarmony 发行版系统时, 不必修改代码, 即可实现敏感数据的保护. 如需在发行版提取后期运维工作必要的日志信息, 则需要开发者将相应的日志参数添加{public}标识并提交到 OpenHarmony 社区进行审核, 审核人员可以通过简单的代码静态分析找到所有{public}标识, 和开发者进行有针对性的沟通和风险分析. 参数记录规则如表 4 所示.

表 4 HiLog 日志参数记录规则

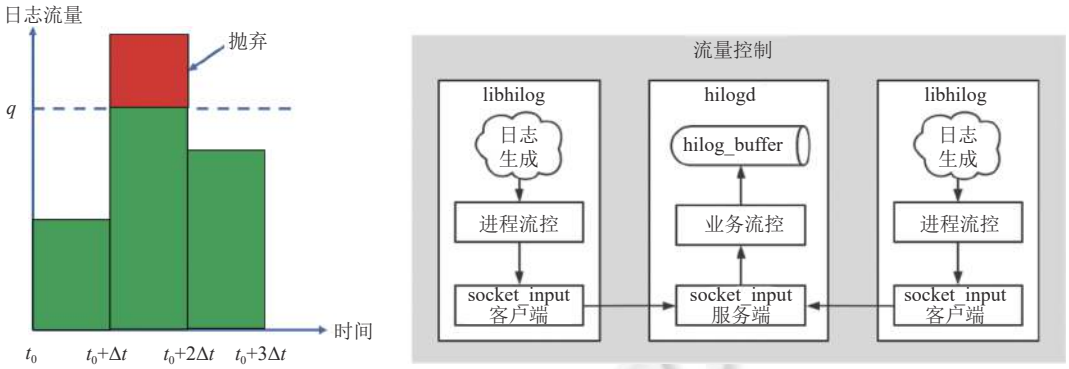
系统版本	数据安全能力	修饰{public}	修饰{private}	无修饰
开发者版本	关闭	记录	记录	记录
开发者版本	开启	记录	不记录	不记录
发行版本	强制开启	记录	不记录	不记录

3.3.4 HiLog 日志系统流量控制

作为实现系统资源合理分配的手段, HiLog 提供日志流量控制 (简称流控) 机制, 避免部分进程日志流量过大造成的系统负载过高和日志丢包问题. 流量控制的基本原理如图 7(a) 所示, 即设置一个日志流量阈值 q, 在每个时间片段 Δt 内统计日志流量, 当某个时间片段内的日志流量超出阈值 q 时, 抛弃超出部分的日志. 图 7(b) 以标准 HiLog 为例, 描述了流量控制模型. 流量控制主要分为在 libhilog 中进行的进程流控和在 hilogd 中进行的业务流控.

实现进程流控可以平衡 IPC 资源的使用并降低性能开销. 一方面, 高速写日志进程会持续占用 socket_input 服务端, 这是一种对日志系统 IPC 资源的不公平占用, 侵占了其他进程的写日志资源. 另一方面, 高速写日志进程

会占用系统的 CPU 资源, 侵占了其他系统服务的资源. 因此 libhilog 提供进程流控功能, libhilog 会统计本进程在一定时间内的日志流量, 按照默认配额或者进程白名单设置的配额进行控制, 对超出配额的日志进行抛弃.



(a) 原理图 (b) 模型图

图 7 流量控制原理和标准 HiLog 的流量控制模型

实现业务流控可以有效管理多进程业务的日志流量, 并进一步减少 hilog_buffer 对系统资源的占用. 在 OpenHarmony 系统中可以通过领域标识 (Domain) 将进程进行归类, 具备相同 Domain 的进程被归纳为同一业务. 因此当某个多进程业务总的日志量过大时, 一方面会造成不同业务之间的日志资源不公平占用, 另一方面也会增加 hilog_buffer 的覆盖写操作频率. 因此 hilogd 提供业务流控功能, 依据日志的 Domain 信息统计一定时间单业务的日志写入量, 按照缺省配额或者系统开发者设置的配额进行控制, 如果超过相应配额则丢弃, 不写入 hilog_buffer. 轻量 HiLog 日志系统不设置 hilogd 模块, 因此仅具备 libhilog 提供的进程流控能力. 同时由于轻量设备的特性, 系统不会运行大量的进程, 因此减少业务流控也是没有问题的.

3.3.5 HiLog 日志系统缓冲区管理

图 8 描绘了标准 HiLog 的缓冲区 hilog_buffer 的结构, 所有进程的日志都将写入用户态 hilog_buffer. 为了高效利用碎片化的内存空间, hilog_buffer 采用链表作为数据结构. 同时, 将链表设计为双向循环结构, 这样可以有效降低日志的排序、插入、读取等操作时指针需要跳转的链表节点数目, 提高性能. hilog_buffer 的特性如下.

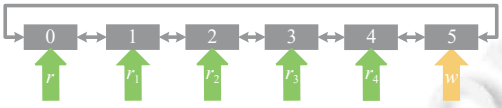


图 8 HiLog 用户态日志缓冲区 (hilog_buffer) 示意图

(1) 时间戳有序: hilog_buffer 中的数据按照日志数据的时间戳排序. 排序是日志系统中非常重要的功能, 因为正确的时间顺序是提取正确逻辑的前提. 为了给开发和维护人员提供符合逻辑的信息, 日志在输出时需要具备正确时序. 在 HiLog 中, 多进程需要通过 Socket 将日志数据传输到 hilog_buffer, 这一过程在 OpenHarmony 这种分时操作系统中可能会存在随机的延时, 进而导致日志产生时间顺序与到达缓冲区的时间顺序不一致的问题. 如果不进行排序, 在打印时时就会以错误的顺序输出, 造成使用者阅读困难甚至误解. 排序功能存在“先排序”和“后排序”两种方案, “先排序”是指日志在写入日志缓冲区时进行排序; “后排序”是指从缓冲区读取日志进行输出时排序. 考虑到 HiLog 缓冲区的单生产者、多消费者模型, 先排序方案更优. 原因在于: ① “先排序”方案执行一次排序即可解决顺序问题, 由于缓冲区内日志有序, 排序是较为简单的, 只需将新到日志的时间戳依次与缓冲区日志的时间戳进行比较 (由新到旧) 然后插入即可. ② “后排序”需要每个消费者都进行排序, 由于缓冲区内日志无序, 消费者需要遍历一次链表才能排序一条日志, 效率较低.

(2) 读写指针: hilog_buffer 包含 3 类读写指针成员. 如图 8 所示, 3 类指针成员分别为: 写指针 w, 指向当前

hilog_buffer 中最新的日志节点; 公用读指针 r, 指向当前 hilog_buffer 中最旧的日志节点; 读者指针 r_i ($i \in N+$), 会指向介于最新和最旧之间的日志节点.

(3) 单生产者: 在 hilogd 中只运行一个生产者线程, hilog_buffer 中只有一个写指针 w. 当生产者线程收集到新的日志数据时会操作写指针 w 将新数据插入到链表中, 这一过程包括以下 3 种情况: ① 插入数据比 w 所指数据更新, 则在链表最新节点后插入数据, 移动 w 指针到最新节点. ② 插入数据比 w 所指数据更旧, 但比 r 所指数据更新, 则操作 w 指针向前遍历, 找到合适的位置插入数据, 移动 w 指针回到最新节点. ③ 插入数据比 r 所指数据更旧, 则在②的基础上, 还需将 r 指向新插入的节点.

(4) 多消费者: 消费者线程是日志打印线程或日志持久化线程, 在 hilogd 中可能同时运行多个消费者线程. 每个消费者拥有自己的读指针, 通过读指针按顺序读取 log_buffer 中的数据. 第 i 个消费者提供服务时, 起始时将读指针 r_i 指向公共读指针 r 所指节点, 接下来操作 r_i 依次读取后续节点.

(5) 同步问题: 单生产者和多个消费者访问 hilog_buffer 过程中, 可能会出现并发同步问题, 即消费者进行读取的时候, 生产者可能同时对链表进行结构调整. 需要注意的是, 仅需处理生产者和消费者之间的同步问题即可, 消费者之间不存在同步问题. 因此只需对生产者调整链表结构、消费者进行链表节点跳转这两类过程使用同一个线程互斥锁 (pthread_mutex), 即可保证同步. 在锁的选择上, 考虑了线程互斥锁和 CAS 锁两种方案. CAS 锁的优势在于不需要上下文切换, 速度快; 缺点在于需要轮询锁状态, CPU 消耗大, 并且存在 ABA 风险. 线程互斥锁的优势在于 CPU 占用低、不存在 ABA 问题; 缺点在于需要额外消耗上下文切换的时间. 基于两者的特性分析, hilog_buffer 使用线程互斥锁更为合适, 原因如下: ① 资源消耗问题. hilog_buffer 的写日志线程与多个读日志线程需要互斥. 在使用 CAS 锁的情况下, 若写日志线程取得锁, 则会同时存在多个线程轮询锁状态, 消耗 CPU 资源, 对于计算能力不丰富的移动终端设备不是好的选择. ② 临界区问题. hilog_buffer 的写日志操作较为复杂, 在加锁后需要判断 hilog_buffer 是否已满, 如果已满需要执行删除操作, 然后执行排序插入操作. 临界区执行时间较长, 因此整体延时对线程互斥锁额外引入的上下文切换时间不敏感.

(6) 缓冲区容量问题: 缓冲区容量上限可配置, 当未达到上限时, 插入数据会导致 hilog_buffer 长度增长, 删除数据会导致 hilog_buffer 长度减小. 当 hilog_buffer 容量已达上限时, 插入数据会导致时间戳最早的一部分数据被删除, 以存储最新的数据. 删除的过程为: 首先将公用读指针 r 指向剩余数据中最旧的节点, 然后检查读者列表中每个读者的读指针是否指向将被删除的节点, 如果是, 则将该读者的读指针指向公用读指针 r 所指结点, 最后将需要删除的节点内存释放.

3.3.6 HiLog 日志系统日志持久化与压缩

日志的持久化能够有效地防止由于系统崩溃或意外断电导致的日志丢失问题, 且崩溃或断电前持久化的日志往往能够成为追溯操作系统问题的重要依据, 因此日志的持久化功能是日志系统的重要功能之一. HiLog 提供日志持久化机制, 即读取 hilog_buffer 内容, 写入到文件系统中.

为了减小持久化阶段的内存开销, 同时又保证日志数据量, HiLog 采用了多文件轮转的持久化机制, 即规定日志文件的总数和每个日志文件的大小, 这样仅需维持一个日志文件大小的内存空间, 即可实现数倍于日志文件大小的持久化操作. 将内存中的日志数据写入文件之前, 检查当前目录下对应的文件数量和将要写入的文件的大小. 如果即将写入的文件大小超过规定阈值, 切换要写入的文件编号. 如果当前目录下文件数量超过规定阈值, 先删除序号最小的文件, 然后对所有文件进行名称修改, 序号依次减 1. 新建序号最大的文件, 作为写入目标, 如图 9 所示.

HiLog 日志系统提供两种不同的日志压缩方法, 一种是日志流压缩, 另一种是日志文件压缩.

日志流压缩方法首先将从日志缓冲区读取日志, 接下来将日志作为比特流输入压缩算法接口, 最后当将压缩算法的输出数据量达到单文件大小阈值时写入文件. 日志作为高度格式化的数据, 可以以极低的压缩率执行压缩, 节省存储空间. 综合考虑了压缩时间和压缩率等压缩重要指标[31], HiLog 采用 zlib[32]作为流压缩算法库.

日志文件压缩方法是对日志文件进行压缩的手段, 是日志流压缩方法的补充. 日志文件压缩方法主要面向小流量的写日志场景. 当日志流量较小时, 如果采用流压缩方法, 压缩算法的输出数据量达到单文件大小阈值需要很长的时间, 期间一旦出现系统崩溃或断电问题, 就会导致内存中的日志数据丢失. 因此, 对于这种场景, 首先使用多

文件轮转的持久化机制构建临时的日志文件, 后续通过日志文件压缩方法将临时日志文件压缩处理为压缩文件, 最后删除临时日志文件. 期间出现意外情况, 仍可读取临时的日志文件.

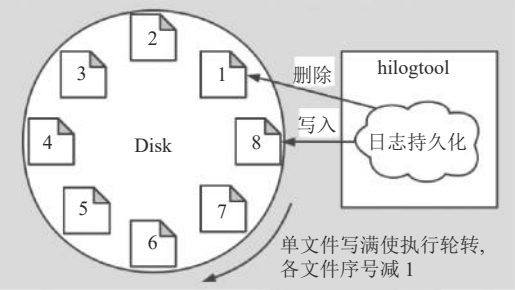


图 9 HiLog 日志系统的多文件轮转的持久化机制

4 HiLog 日志系统实验分析

为了检验标准 HiLog 的表现, 本文基于多种硬件平台对 HiLog 进行实验, 并且和在相同平台上和 Android 的 Log 日志系统进行横向的对比实验. 相应的实验条件如下所示.

(1) 在硬件平台方面: 在 HiLog 性能实验中, 使用的硬件平台包括润和 Hi3516DV300 开发板 (Hi3516)、树莓派基金会 Raspberry Pi Model B V1.2 开发板 (RPI3B)、润和 HH-SCDAYU200 开发板 (DAYU200) 和润和 HiKey960 开发板 (HiKey960). 在 HiLog 和 Log 的对比实验中, 目前同时能够支持 OpenHarmony 操作系统和 Android 操作系统的硬件平台并不多, 本实验挑选了 RPI3B 和 HiKey960 作为对比实验的硬件平台.

(2) 在软件平台 (操作系统) 方面: OpenHarmony 操作系统仍属于快速迭代期, 目前并非所有的系统版本都能够适配以上的全部硬件平台, 因此考虑到操作系统的适配性问题, 本实验的使用的 OpenHarmony 操作系统是 OpenHarmony3.0 版本. Android 的 Log 日志系统在 Android 5.0.0 版本进行了整体架构的更新, 并一直沿用至今, 因此本实验使用的 Android 操作系统为 Android 6.0.0 版本.

(3) 在实验数据方面: 为了使实验数据更贴近实际使用场景, 收集了在 OpenHarmony 操作系统和 Android 操作系统运行过程中产生的共计 15 368 条日志数据, 获取其 Tag 和 Content, 作为数据集. 利用 Tag 和 Content 作为参数分别调用 Log 和 HiLog 的写日志接口, 构建日志写入程序.

为了检验轻量 HiLog 对资源受限设备的兼容性, 实验在多种轻量设备上安装基于 LiteOS-m 内核的 OpenHarmony3.0 轻量操作系统, 并测试轻量 HiLog 的功能. 测试涉及的硬件平台有海思 HiSpark Hi3861V100 开发板 (Hi3861)、小熊派 BearPi-HM_Nano 开发板 (BearPi)、旗点科技 GD32F303 开发板 (GD32F303). 测试使用基于 LiteOS-m 内核的 OpenHamrony3.0 的轻量级系统.

4.1 HiLog 基本性能分析

前述吞吐量是日志系统性能的重要指标, 然而吞吐量仅代表日志未出现丢包情况的最高速率, 不能描述日志系统在各种工况下的表现. 因此除了吞吐量外, 还需引入丢包率指标, 考虑到大部分开发人员对日志的使用需求是将日志打印到窗口或文件进行分析, 日志的丢包率的计算方法定义如下:

$$L = \frac{S_{\text{input}} - S_{\text{print}}}{S_{\text{input}}} \tag{1}$$

其中, S_{print} 代表打印到文件的日志大小, S_{input} 代表写日志进程写入日志的数据大小, 为了考察日志系统在不同写入速率下的丢包率表现, S_{print} 可表达为:

$$S_{\text{input}} = V_{\text{input}} \times t \tag{2}$$

其中, V_{input} 表示写日志进程写入速率, t 为写入的持续时间.

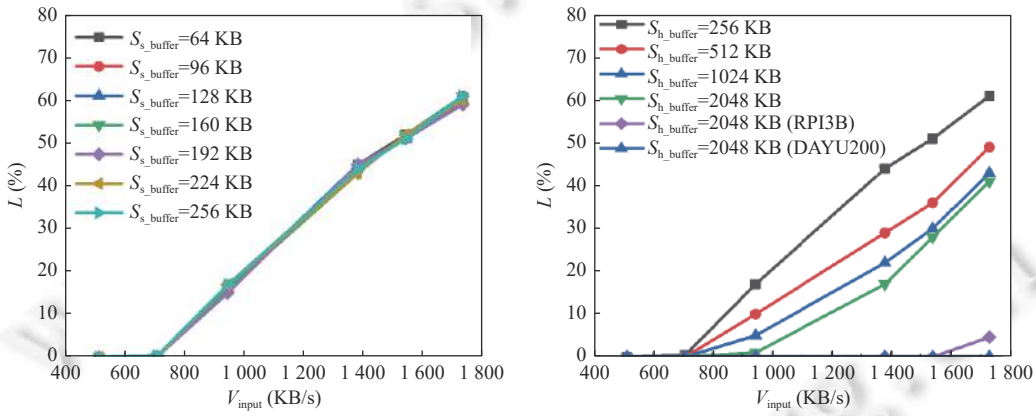
在丢包率的影响因素方面, 除了日志系统的架构设计, 硬件平台的性能以外, 日志系统的负载 (日志写入速

率) 和一些日志系统的参数也是十分重要的因素. 基于对丢包的原理分析, 可能影响丢包的参数有 socket_input 服务端的 Socket 缓冲区空间 S_{s_buffer} 和 hilog_buffer 的空间 S_{h_buffer}.

为了使本实验设计的写日志速率范围更加合理, 本文统计了在 OpenHarmony 操作系统中日志的使用情况, 以下平均速率由 60 s 内的写入数据量计算, 峰值速率由 1 s 内的写入数据量计算: ① 在操作系统重新启动后, 保持无操作状态, 此时日志写入平均速率约为 10 KB/s, 峰值可达 31 KB/s. ② 在操作系统 UI 界面被频繁点击 (约 1 次/s), 此时日志的写入平均速率约为 230 KB/s, 峰值可达 330 KB/s. ③ 在操作系统 UI 界面被频繁点击, 并且开启了数个应用及其后台服务 (新闻、视频播放、应用市场、系统管理), 此时日志写入平均速率约为 500 KB/s, 峰值速率可达为 720 KB/s.

基于上述数据可以看出, 当前 OpenHarmony 操作系统的写日志需求一般不超过 500 KB. 但是在选取研究的写日志速率范围时, 需要考虑以下几点: ① 采集以上数据使用的系统、应用和服务均为发行版. 事实上在系统、应用的开发和调试过程中, 为了提供更加详细的信息, 开发者会写入更多的日志语句, 写日志速率会更高. ② 有必要考虑软件生态快速发展的趋势带来的写日志需求增加, 作为面向未来设计的日志系统, 有必要保留余量. 综上, 本文选择 $V_{input} \geqslant 500$ KB/s 的写日志速率区间作为研究范围.

图 10(a) 描绘了在 Hi3516 开发板上, 采用系统默认的参数 S_{s_buffer}=64 KB、S_{h_buffer}=256 KB, 日志写入速率对丢包率的影响. 可以看出, 在日志写入速率小于 700 KB/s 时, 无日志丢包现象; 而当日志写入速率进一步提升时, 丢包率开始逐渐增长; 当写入速率达到 1 730 KB/s 时, 丢包率已达到 60%.



(a) 探究 S_{s_buffer} 的影响 (S_{h_buffer}=256 KB)　　(b) 探究 S_{h_buffer} 的影响 (S_{s_buffer}=256 KB)

图 10　HiLog 丢包率 L 随 V_{input} 增长的变化趋势, 并探究 HiLog 参数的影响

为了分析丢包的成因, 通过调整 Socket 缓冲区的大小, 发现当 S_{s_buffer} 从 64 KB 逐渐增大到 256 KB 时, 如图 10(a) 所示, 丢包率没有明显的变化, 说明丢包可能并不是由于 Socket 缓冲区挤占导致的. 接下来固定 S_{s_buffer} 为 256 KB, 调整 S_{h_buffer} 从 256 KB 逐渐增大到 2 048 KB, 如图 10(b) 所示, 发现随 S_{h_buffer} 的增大丢包率有明显的下降, 在写入速率为 1 730 KB/s 时, 丢包率为 41%, 降幅 31.67%, 说明丢包与 hilog_buffer 有关.

经过理论分析, 丢包现象应当和 hilog_buffer 的覆盖写操作有关. 当 hilog_buffer 的双向循环链表空间已满时, 需要执行覆盖写操作后才可以继续进行数据写入: 即选取最旧的 5% 日志数据进行删除、移动公读指针和部分读者指针到指定节点. 因此, 当日志写入速率过快, 就会频繁触发覆盖写操作, 消耗 CPU 资源, 当覆盖写速度低于写入速度时, 就会产生日志的丢包情况. 通过增大 hilog_buffer 的大小, 可以增加每次覆盖写删除的日志数据量, 事实上增加了覆盖写的速率, 进而减少丢包率. 同理, 更大的 hilog_buffer 可以在面对相同的写入速率时, 降低覆盖写的频率, 降低 CPU 资源的消耗, 但是会增加内存资源的占用.

随后, 使用 S_{s_buffer}=256 KB、S_{h_buffer}=2048 KB 的参数配置, 分别在 CPU 性能更强的 RPI3B 和 DAYU200 开发板上测试了 HiLog 在各种负载下的丢包情况, 结果如图 10(b) 所示. 可以看出, 相比 Hi3516 开发板, RPI3B 和

DAYU200 开发板在面对相同的日志写入速率时, 丢包率有了明显的下降; 在面对 1 730 KB/s 的写入速率时, 仅分别有 4.6% 和 0.1% 的数据丢包. 这一定程度上印证了丢包是由于覆盖写操作造成的 CPU 瓶颈导致的.

最后, 在 RPI3B 开发板上比较了 Android 的日志系统 Log 和 OpenHarmony 的日志系统 HiLog 的丢包率随写日志速率的变化. 在选择参数 S_{s_buffer}、S_{h_buffer} 时, 参考使用 Android 日志系统 Log 的默认 socket_buffer 空间大小 (64 KB) 和 list_buffer 空间大小 (256 KB). 结果如图 11 所示, 可以看出 Log 在写入速率达到 700 KB/s 时开始出现丢包情况, 而 HiLog 直到 1 500 KB/s 时才出现丢包; 当日志写入速率达到 1 730 KB/s 时, Log 的丢包率达到 19%, 而 HiLog 的丢包率仅为 8%. 上述结果表明, HiLog 相比 Log 具有更高的日志吞吐量, 且在写日志速率超过吞吐量的情况下具备更低的丢包率, 可以有效地保留更完备的操作系统事件信息提供给系统开发者或运维人员使用.

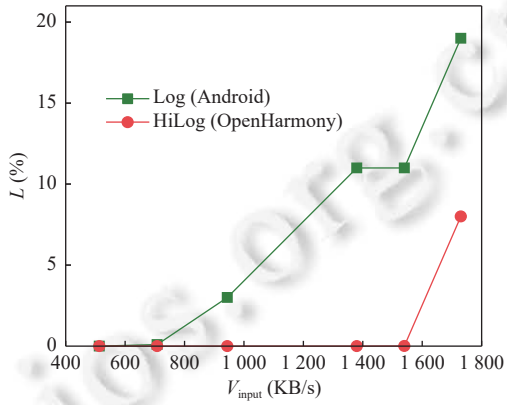


图 11 Log 和 HiLog 的丢包率 L 随 V_{input} 增长的变化趋势 (硬件平台: RPI3B)

上述实验首先分析了 HiLog 日志系统在 500 KB/s–1 730 KB/s 写入速率范围的丢包情况和原因, 发现在高日志写入速率下的丢包是由于 hilog_buffer 覆盖写操作造成的 CPU 瓶颈导致的, 可以通过增大 S_{h_buffer} 降低丢包率. 接下来对比了 HiLog 和 Log 在 500 KB/s–1 730 KB/s 写入速率范围内的丢包率情况, 结果表明在相同的硬件条件下, HiLog 相比 Log 具有更低的丢包率, 这说明 HiLog 日志系统可以更好地保留操作系统中的事件信息, 为使用者提供更详尽的日志数据用于开展工作. 在如今这种由于丰富的软件带来日志数量持续膨胀而硬件摩尔定律失效的状态下, HiLog 的性能优势是十分有意义的.

4.2 HiLog 流控性能分析

流量控制机制是 HiLog 实现系统资源合理分配的重要手段. 对于日志系统的基本性能分析表明, 当日志写入速率过快时, 会导致丢包的情况, 同时也会大量占用 CPU 等系统资源. 为了避免这一问题, HiLog 设计了日志的流量控制功能, 它可以对每进程的日志流量进行限制, 避免由于部分程序写日志速率过快占用 HiLog 的处理资源.

为了测试流量控制功能的效果, 实验基于 HiKey960 开发板搭建了测试环境, 构建了 2 个写日志进程, 其中进程 A 是正常写日志进程, 其写日志速率 V_{input_A} 在正常速率范围内写日志 (1–20 KB/s); 进程 B 是违规写日志进程, 持续以极高速度写日志 (4 096 KB/s). 实验目的是分析在开启 (每进程写日志流量上限为 13 KB/s)/关闭流量控制的情况下, 进程 A 的日志丢包率 L_A 和 hilogd 进程的 CPU 占用率.

实验结果如图 12 所示, 其中, 黑色虚线代表单进程写日志流量上限, 可以看到当系统中存在一个高速写日志进程时, 如果关闭流量控制, HiLog 几乎无法收集到其余进程的日志信息, 且 hilogd 进程的 CPU 占用率达到了 44%, 严重侵占了系统的资源. 流量控制功能开启时, 情况发生了变化, 当进程 A 写日志速率低于设定的流量上限 (13 KB/s) 时, 几乎不会出现丢包情况, 且 hilogd 的 CPU 占用率也下降到 3% 左右. 说明流量控制功能抛弃了进程 B 的绝大多数日志, 使它们不会进入 IPC 和缓冲区处理环节, HiLog 能够有效处理进程 A 的日志信息. 流量控制功能同时降低了 hilog_buffer 达到容量上限进行的覆盖写操作频率, 降低丢包率和 CPU 资源的损耗.

上述实验说明了 HiLog 流量控制功能的有效性, 可以有效降低违规高速写日志进程对其他进程日志资源的侵

占, 保障日志资源的合理分配. 流量控制功能将日志系统写日志资源从公共资源转化为了每个程序的私有资源, 一方面有助于保护各个程序的日志资源使用权, 另一方面可以增加开发者的责任和动力去规范程序的写日志速率.

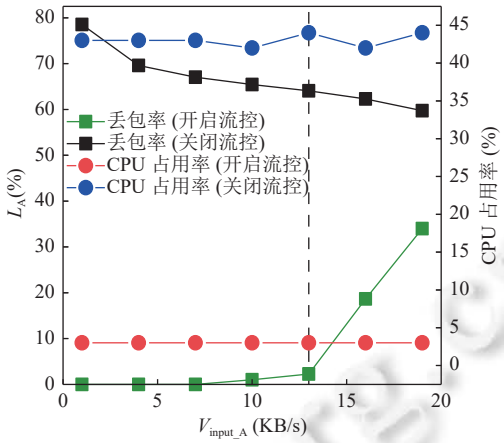


图 12 开启/关闭 HiLog 的流量控制功能时的日志丢包率 L_A 和 hilogd 的 CPU 占用率

4.3 HiLog 持久化性能分析

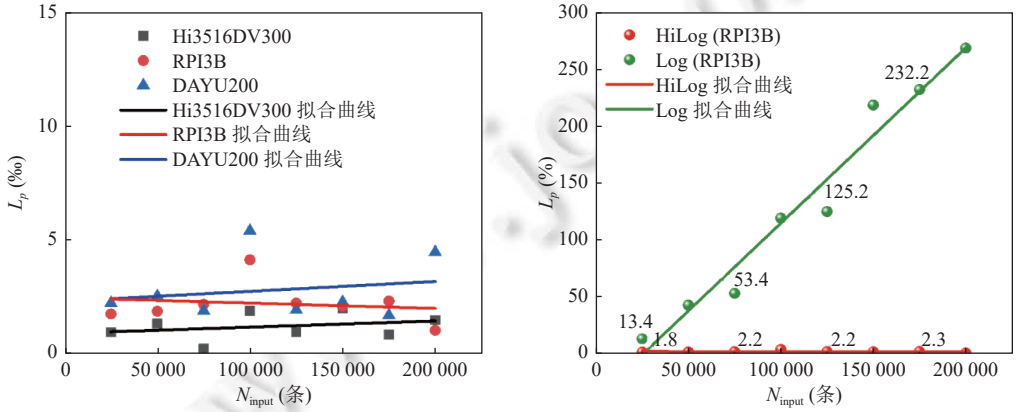
日志持久化是日志系统的重要功能之一, 其中重要的指标是持久化丢包率和压缩率. 持久化丢包率越低说明在持久化阶段的信息损失量越少, 日志系统可以将更全面的日志保存到存储空间中; 而越低的压缩率说明在保存相同信息的条件下, 日志系统可以节省更多的磁盘空间. 在以下测试中, 持久化丢包率 L_p 和持久化压缩率 C_p 的计算公式为:

$$L_p = \frac{N_{\text{input}} - N_{\text{disk}}}{N_{\text{input}}} \tag{3}$$

$$C_p = \frac{S_{\text{file}}}{(1 - L_p) \times S_{\text{input}}} \tag{4}$$

其中, N_{input} 和 N_{disk} 分别代表写入的日志数量和持久化到硬盘上的日志数量, S_{input} 和 S_{file} 分别代表写入的日志大小和持久化到硬盘上的日志文件 (或压缩文件) 大小, 压缩策略采用流压缩.

本文首先对 HiLog 的持久化丢包率进行了测试, 重点考量了硬件性能和日志条数对持久化丢包率的影响, 如图 13(a) 所示. 可以看到总体上 HiLog 日志持久化丢包率低于 6‰, 通过对 L_p-N_{input} 的最小二乘法拟合, 发现 HiLog 日志系统在所有测试平台上都表现出较高的稳定性, 即持久化丢包率随日志数量没有明显的变化.



(a) 探究硬件平台对 HiLog 的 L_p 的影响 (b) 在相同硬件平台比较 HiLog 和 Log 的 L_p

图 13 HiLog 和 Log 持久化丢包率 L_p 随 N_{input} 的变化趋势

为了横向比较 HiLog 日志系统和 Log 日志系统的持久化丢包率, 在相同平台 RPI3B 平台上分别安装 Android 5.1.1 操作系统和 OpenHarmony3.0 操作系统, 分别测试 Log 和 HiLog 的持久化丢包率, 测试结果如图 13(b) 所示. 可以看到 Log 日志系统在持久化 25 000 条日志时已有 13.4‰的丢包率, 且这一数值随日志数量的增加有明显的增长. 而 HiLog 日志系统在持久化相同日志数量时丢包率仅有 1.8‰, 仅为前者的 13%, 且随日志数量的增加没有明显的变化.

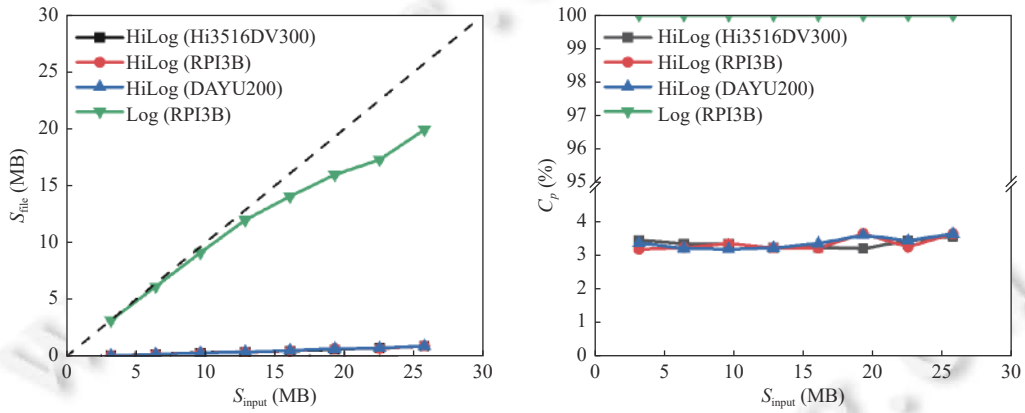
为了量化 Log 和 HiLog 的持久化稳定性差距, 通过最小二乘法分别对 Log 和 HiLog 进行了 L_p-N_{input} 线性拟合, 发现丢包率随日志条数的变化具备线性特征, 曲线表达式为:

$$R_l^{\text{Log}}(n) = k^{\text{Log}} \times n + b^{\text{Log}} \tag{5}$$

$$R_l^{\text{HiLog}}(n) = k^{\text{HiLog}} \times n + b^{\text{HiLog}} \tag{6}$$

其中, n 代表日志系统内存 Buffer 中的日志数量. 拟合结果显示, Android 的 Log 日志系统 k^{Log} 值为 1.54×10^{-3}, 而 OpenHarmony 的 HiLog 日志系统 k^{HiLog} 值为 2.34×10^{-6}, 仅为前者的 1.52×10^{-3} 倍, 说明 HiLog 的落盘丢包率随日志数量的增长是极为缓慢的, 可以适应大量日志持久化的场景需求.

接下来在不同的硬件平台上测试了 HiLog/Log 的持久化存储占用情况. 由于 Android 的 Log 日志系统不存在压缩持久化功能, 理论上其持久化日志文件大小应等于缓冲区中的日志大小. 然而从图 14(a) 可以看出, Log 的日志文件总是小于缓冲区中的日志大小, 且随着日志大小的增加偏差在增大, 基于上述丢包率分析, 可以将这种现象归因为丢包. 图 14(b) 表明 HiLog 的持久化压缩率约为 3.5%, 这一结果在不同的设备上是较为稳定的.



(a) S_{file} 随 S_{input} 的变化　　(b) C_p 随 S_{input} 的变化

图 14　HiLog 和 Log 的持久化性能对比

上述分析表明, OpenHarmony 的 HiLog 日志系统相比 Android 的 Log 日志系统, 在保持低持久化丢包率的基础上, 还能够对日志进行压缩以节省大量的存储空间 (约 96.5%).

4.4　HiLog 设备兼容性分析

后文表 5 列出了 1 种 L2 硬件平台和 3 种 L1 硬件平台的硬件参数, 可以看到 L1 平台的 CPU 主频、内存空间、存储空间均远小于 L2 平台. 轻量 HiLog 通过编译时参数控制, 不编译 hilogd 模块达到减少缓冲区维护所需的计算量和内存空间, 同时 LiteOS 内核的日志缓冲区将会替代 hilog_buffer 作为日志暂存的缓冲区, hilogtool 将保留持久化压缩的能力, 节约 L1 硬件平台本不富裕的存储空间.

兼容性实验测试了 HiLog 和 Log 在 4 种设备上的运行情况, 相应结果如表 5 所示. 测试结果表明, HiLog 在内存最小为 96 KB 的设备上依然可以完成日志的写入、打印、持久化功能. 由于 Log 无法脱离 logd 运行, 因此在 L1 设备上是无法安装和运行的. 综上所述, HiLog 具备更强的设备兼容能力, 对于计算资源丰富的硬件可以提供标准 HiLog 作为高性能的日志系统, 对于计算资源受限的硬件可以提供轻量 HiLog 实现基本的日志功能.

4.5　HiLog 数据安全能力分析

作为对轻量化日志数据安全能力的初步探索, 在设计 HiLog 的数据安全能力时重点考虑了变量的安全问题.

开发者基于代码规范对参数加入敏感数据标识, HiLog 可以识别敏感数据标识进行数据遮蔽, 在不影响性能的前提下提供日志安全能力. 图 15 展示了使用示例和输出结果. 使用示例如代码 2–9 行所示, 在参数 name 和 pass 的格式化占位符中加入{private}, 参数 errorcode 的格式化占位符加入{public}. 输出结果如代码 11–14 行所示, 在开启数据安全能力的情况下 errorcode 的值正常写入, name 和 pass 的值被替换为了“<private>”, 避免相关信息在后续的日志分析中被获取和利用; 在关闭数据安全能力的情况下, 则相应参数值均可以被正常记录.

表 5 HiLog 和 Log 的设备兼容性比较

开发板名称	CPU	内存	存储	HiLog版本	Log兼容性
润和HH-SCDAYU200	四核 @ 2.0 GHz	2 GB	32 GB+扩展	标准	可兼容
HiSpark Hi3861V100	单核 @ 160 MHz	352 KB	2 048 KB	轻量	不兼容
BearPi-HM_Nano	单核 @ 160 MHz	352 KB	2 048 KB	轻量	不兼容
旗点科技GD32F303	单核 @ 120 MHz	96 KB	256 KB	轻量	不兼容

```
1.  ------------------------------------------------ Source Code ------------------------------------------------
2.  std::string name = GetUsernameFromInput();
3.  std::string pass = GetPasswordFromInput();
4.  int errorcode = TryToLogin(name, pass);
5.  if(errorcode){
6.      OHOS::HiviewDFX::HiLog::Info(LABEL,
7.      “HiLog private log test: Username=%{private}s, Password=%{private}s, Errorcode=%{public}d”,
8.      name,pass, errorcode);
9.  }
10. ------------------------------------------------ Outputs ------------------------------------------------
11. With data protection on>>> “HiLog private log test: Username=<private>, Password=<private>, Errorcode=403”
12. With data protection off>>> “HiLog private log test: Username=Zhangsan, Password=123abc, Errorcode=403”
```

图 15 HiLog 日志敏感数据安全能力使用示例

为了考量 HiLog 数据安全能力的轻量化程度, 本文测试了 HiLog 数据安全能力对写日志代码执行时间的影响. 相关结果如图 16 所示, 可以看出在打开/关闭数据安全能力时, 写日志代码的平均执行时间 T_{func} 没有明显变化. 可以说明 HiLog 的数据安全能力是一种轻量化的日志安全保护方法, 不会对日志系统和操作系统的性能造成影响. 据统计在 OpenHarmony 系统代码中, 总共有了 418 145 条 HiLog 写日志代码, 其中 48 999 条写日志代码使用了 HiLog 数据安全能力, 占比 11.7%. HiLog 的日志数据安全能力已在 OpenHarmony 主线系统代码中逐渐得到推广和应用.

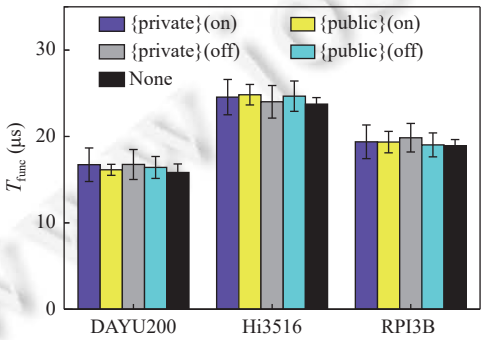


图 16 HiLog 日志敏感数据安全能力对写日志函数执行时间的影响

5 总结与讨论

本文主要工作是为开源操作系统 OpenHarmony 设计并实现了 HiLog 日志系统. HiLog 负责存储并管理包括内核日志、系统日志以及三方日志等各种类型的日志信息, 为系统开发者和应用开发者提供日志读写、压缩、存储等功能. 目前, HiLog 已在包括 HarmonyOS 在内的多个 OpenHarmony 发行版上进行了稳定的运行, 为系统开发者、应用开发者和数据分析师提供了重要的日志数据. 同时, 作为开源项目, HiLog 日志系统已受到国内外开发者的广泛关注, 相关代码已合入 OpenHarmony 主线. 经过本文的理论和实验分析, HiLog 的优势如下.

(1) HiLog 拥有很高的日志吞吐量和稳定性, 通过高效的 IPC 模型和缓冲区结构设计提高日志读写效率. 在日志写入阶段, HiLog 具备极高的日志吞吐量; 在日志持久化阶段, HiLog 具备远低于 Log 的丢包率. 满足了性能需求.

(2) HiLog 创新性地加入了流量控制功能, 可以有效地平衡写日志资源的占用并降低 CPU 资源消耗. 同时将写日志资源从一种公共资源转化为了每个程序的私有资源, 鼓励开发者规范程序的写日志速率, 促进操作系统生态和谐. 满足了资源分配需求.

(3) HiLog 提供轻量化的数据安全能力, 未使用公开标识标记的参数都会被掩码隐藏, 降低系统敏感数据或用户的隐私信息的泄露风险. 并且此功能对系统资源消耗极小, 不会影响写日志速率. 一定程度上满足了数据安全需求.

(4) HiLog 考虑了面向资源受限设备的兼容性, 通过良好的模块化设计可以在轻量设备上分离用户态缓冲区及其管理模块, 降低系统资源占用. 同时 HiLog 提供日志持久化压缩功能, 可以有效节约设备的存储空间, 空间节约效率可达 96.5%. 满足了设备兼容性需求.

同时, 也反映出 HiLog 的一些问题与改进空间.

(1) 目前业界对于日志系统的数据安全的研究较少, HiLog 的轻量化数据安全能力是对于日志数据安全问题的初步探索, 虽然可以一定程度上遮蔽敏感数据信息, 但是仍需要开发者和审核人员投入精力对日志参数进行修改和验证, 具备一定的人工成本. 后续希望能够引入一些自动化的判断机制, 例如采用机器学习的手段对日志参数的上下文进行分析, 进而自动化地判断该参数是否属于隐私信息, 减少人工审核成本, 提高数据安全能力的易用性.

(2) OpenHarmony 作为分布式操作系统, 原生支持分布式能力. 分布式能力涉及多台设备的协同运作, 即日志信息会在多台设备上产生, 然而目前 HiLog 尚不具备从多设备统一收集日志并进行管理的能力. 这种缺陷对于分布式能力的开发和调试造成了一定的不便, 具备优化的空间. 构造分布式日志系统有两个重要的问题需要解决, 其一是设备间高速、高稳定的连接问题, 其二是多设备的时钟同步问题. 对于第 1 个问题, 可以等待 OpenHarmony 的软总线 (SoftBus) 技术成熟后, 利用 SoftBus 作为稳定高速的日志传输的通道; 对于第 2 个问题, 可以考虑基于精确时间协议 (precision time protocol, PTP) 实现无线局域网内的多设备时钟同步. 使 HiLog 成为更具 OpenHarmony 特色的分布式的日志系统.

References:

[1] Beckett D. Combined log system. Computer Networks and ISDN Systems, 1995, 27(6): 1089–1096. [doi: 10.1016/0169-7552(95)00013-W]

[2] He PJ, Zhu JM, He SL, Li J, Lyu MR. Towards automated log parsing for large-scale log data analysis. IEEE Trans. on Dependable and Secure Computing, 2018, 15(6): 931–944. [doi: 10.1109/TDSC.2017.2762673]

[3] Landauer M, Skopik F, Wurzenberger M, Rauber A. System log clustering approaches for cyber security applications: A survey. Computers & Security, 2020, 92: 101739. [doi: 10.1016/j.cose.2020.101739]

[4] Dumais S, Jeffries R, Russell DM, Tang D, Teevan J. Understanding user behavior through log data and analysis. In: Olson JS, Kellogg WA, eds. Ways of Knowing in HCI. New York: Springer, 2014. 349–372. [doi: 10.1007/978-1-4939-0378-8_14]

[5] Zhao X, Rodrigues K, Luo Y, Stumm M, Yuan D, Zhou YY. Log20: Fully automated optimal placement of log printing statements under specified overhead threshold. In: Proc. of the 26th Symp. on Operating Systems Principles. Shanghai: Association for Computing

Machinery, 2017. 565–581. [doi: 10.1145/3132747.3132778]

[6] Shekhtman L, Waisbard E. EngraveChain: A blockchain-based tamper-proof distributed log system. Future Internet, 2021, 13(6): 143. [doi: 10.3390/fi13060143]

[7] Liao XK, Li SS, Dong W, Jia ZY, Liu XD, Zhou SL. Survey on log research of large scale software system. Ruan Jian Xue Bao/Journal of Software, 2016, 27(8): 1934−1947 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4936.htm [doi: 10.13328/j.cnki.jos.004936]

[8] The Syslog Protocol. 2009. https://datatracker.ietf.org/doc/html/rfc5424

[9] Chowdhury S, Di Nardo S, Hindle A, Jiang ZM. An exploratory study on assessing the energy impact of logging on Android applications. Empirical Software Engineering, 2018, 23(3): 1422–1456. [doi: 10.1007/s10664-017-9545-x]

[10] Zhauniarovich Y, Gadyatskaya O. Small changes, big changes: An updated view on the Android permission system. In: Proc. of the 19th Int'l Symp. on Research in Attacks, Intrusions, and Defenses. Paris: Springer, 2016. 346–367. [doi: 10.1007/978-3-319-45719-2_16]

[11] Elgharabawy M, Kojusner B, Mannan M, Butler KRB, Williams B, Youssef A. SAUSAGE: Security analysis of UNIX domain socket usage in Android. In: Proc. of the 7th IEEE European Symp. on Security and Privacy (EuroS&P). Genoa: IEEE, 2022. 572–586. [doi: 10.1109/EuroSP53844.2022.00042]

[12] Fukui D, Shimaoka M, Mikami H, Hillenbrand D, Yamamoto H, Kimura K, Kasahara H. Annotatable systrace: An extended Linux FTrace for tracing a parallelized program. In: Proc. of the 2nd Int'l Workshop on Software Engineering for Parallel Systems. Pittsburgh: Association for Computing Machinery, 2015. 21–25. [doi: 10.1145/2837476.2837479]

[13] Kilbury J, Bontcheva K, Samih Y. FTrace: A tool for finite-state morphology. In: Proc. of the 9th Int'l Workshop on Finite State Methods and Natural Language Processing. Blois: Association for Computational Linguistics, 2011. 88–92.

[14] Yang S, Park SJ, Ousterhout J. NanoLog: A nanosecond scale logging system. In: Proc. of the 2018 USENIX Annual Technical Conf. Boston: USENIX Association, 2018. 335–349.

[15] Gupta S. Pro Apache Log4j. 2nd ed., Berkeley: Apress, 2005.

[16] Srinivasa S, Pedersen JM, Vasilomanolakis E. Deceptive directories and "vulnerable" logs: A honeypot study of the LDAP and Log4j attack landscape. In: Proc. of the 2022 IEEE European Symp. on Security and Privacy Workshops (EuroS&PW). Genoa: IEEE, 2022. 442–447. [doi: 10.1109/EuroSPW55150.2022.00052]

[17] Liu WZ, Tao QY, He Q, Yu LJ. Application of Log4j in e-commerce services. Applied Mechanics and Materials, 2014, 635–637: 1517–1521. [doi: 10.4028/www.scientific.net/AMM.635-637.1517]

[18] OpenHarmony. 2009. https://gitee.com/openharmony

[19] Xu YX. Research and design of high performance distributed log system. Industrial Control Computer, 2020, 33(12): 44–46 (in Chinese with English abstract). [doi: 10.3969/j.issn.1001-182X.2020.12.019]

[20] You Y, Wang H, Ren T, Gu SH, Sun JL. Storage design of tracing-logs for application performance management system. Ruan Jian Xue Bao/Journal of Software, 2021, 32(5): 1302−1321 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6234.htm [doi: 10.13328/j.cnki.jos.006234]

[21] Guo JW, Cai P, Qian WN, Zhou AY. Accurate and efficient follower log repair for Raft-replicated database systems. Frontiers of Computer Science, 2021, 15(2): 152605. [doi: 10.1007/s11704-019-8349-0]

[22] Liang GY, Wu YJ, Wu JZ, Zhao C. Open source software supply chain for reliability assurance of operating systems. Ruan Jian Xue Bao/Journal of Software, 2020, 31(10): 3056−3073 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6070.htm [doi: 10.13328/j.cnki.jos.006070]

[23] Neira-Ayuso P, Gasca RM, Lefevre L. Communicating between the kernel and user-space in Linux using Netlink sockets. Software: Practice and Experience, 2010, 40(9): 797–810. [doi: 10.1002/spe.981]

[24] Shakya B, Xu XL, Tehranipoor M, Forte D. CAS-Lock: A security-corruptibility trade-off resilient logic locking scheme. IACR Trans. on Cryptographic Hardware and Embedded Systems, 2020, 2020(1): 175–202. [doi: 10.13154/tches.v2020.i1.175-202]

[25] Torrellas J, Lam MS, Hennessy JL. False sharing and spatial locality in multiprocessor caches. IEEE Trans. on Computers, 1994, 43(6): 651–663. [doi: 10.1109/12.286299]

[26] Dechev D. The ABA problem in multicore data structures with collaborating operations. In: Proc. of the 7th Int'l Conf. on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom). Orlando: IEEE, 2011. 158–167. [doi: 10.4108/icst.collaboratecom.2011.247161]

[27] China Academy of Information and Communications. White Paper of IOT. 2020 (in Chinese). http://www.caict.ac.cn/english/research/whitepapers/202012/P020201223330673521178.pdf

[28] Murthy S, Bakar AA, Rahim FA, Ramli R. A comparative study of data anonymization techniques. In: Proc. of the 5th IEEE Int'l Conf.

on Big Data Security on Cloud (BigDataSecurity), IEEE Int'l Conf. on High Performance and Smart Computing, (HPSC) and IEEE Int'l Conf. on Intelligent Data and Security (IDS). Washington: IEEE, 2019. 306–309. [doi: 10.1109/BigDataSecurity-HPSC-IDS.2019.00063]

[29] Alloghani M, Alani MM, Al-Jumeily D, Baker T, Mustafina J, Hussain A, Aljaaf AJ. A systematic review on the status and progress of homomorphic encryption technologies. Journal of Information Security and Applications, 2019, 48: 102362. [doi: 10.1016/j.jisa.2019.102362]

[30] Ul Hassan M, Rehmani MH, Chen JJ. Differential privacy techniques for cyber physical systems: A survey. IEEE Communications Surveys & Tutorials, 2020, 22(1): 746–789. [doi: 10.1109/COMST.2019.2944748]

[31] Rahman MA, Hamada M. Lossless image compression techniques: A state-of-the-art survey. Symmetry, 2019, 11(10): 1274. [doi: 10.3390/sym11101274]

[32] zlib. 2022. https://github.com/madler/zlib

附中文参考文献:

[7] 廖湘科, 李姗姗, 董威, 贾周阳, 刘晓东, 周书林. 大规模软件系统日志研究综述. 软件学报, 2016, 27(8): 1934−1947. http://www.jos.org.cn/1000-9825/4936.htm [doi: 10.13328/j.cnki.jos.004936]

[19] 徐永新. 高性能分布式日志系统研究与设计. 工业控制计算机, 2020, 33(12): 44–46. [doi: 10.3969/j.issn.1001-182X.2020.12.019]

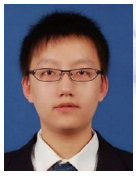
[20] 尤勇, 汪浩, 任天, 顾胜晖, 孙佳林. 一种监控系统的链路跟踪型日志数据的存储设计. 软件学报, 2021, 32(5): 1302−1321. http://www.jos.org.cn/1000-9825/6234.htm [doi: 10.13328/j.cnki.jos.006234]

[22] 梁冠宇, 武延军, 吴敬征, 赵琛. 面向操作系统可靠性保障的开源软件供应链. 软件学报, 2020, 31(10): 3056–3073. http://www.jos.org.cn/1000-9825/6070.htm [doi: 10.13328/j.cnki.jos.006070]

[27] 中国信通院. 物联网白皮书. 2020. http://www.caict.ac.cn/english/research/whitepapers/202012/P020201223330673521178.pdf

吴圣垚(1996－), 男, 工程师, 主要研究领域为软件工程, 人工智能.

屈晟(1989－) 男, 工程师, CCF 专业会员, 主要研究领域为开源软件供应链, 开源操作系统.



王枫(1985－), 男, 工程师, 主要研究领域为操作系统, 云计算, 数据存储.

罗天悦(1990－) 男, 工程师, CCF 专业会员, 主要研究领域为操作系统安全分析, 代码漏洞挖掘, 人工智能安全.

武延军(1979－), 博士, 研究员, 博士生导师, CCF 杰出会员, 主要研究领域为操作系统, 系统安全.

吴敬征(1982－), 男, 博士, 研究员, 博士生导师, CCF 高级会员, 主要研究领域为系统安全, 漏洞挖掘, 操作系统安全.

凌祥(1992－), 男, 博士, CCF 专业会员, 主要研究领域为软件安全, 人工智能安全.