PR #28089 完整报告

sgl-project/sglang

[CI] Reclaim leaked /dev/shm segments on server startup

合并时间：2026-06-16 07:14

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/28089

执行摘要

- 一句话：CI 启动时自动清理泄漏的共享内存段
- 推荐动作：值得精读，尤其是其降级策略设计：通过命名嵌入 PID、仅 CI 激活、最佳努力清理不阻塞启动等。可作为容器中资源泄漏处理的参考模式。建议开发者和基础设施维护者关注 cleanup_stale_shm 的门控机制和异常处理模式。

功能与动机

CI job 中通过 kill_process_tree 和 PR_SET_PDEATHSIG 以 SIGKILL 终止进程，跳过了所有Python级别的共享内存清理路径（finally、atexit、multiprocessingresource_tracker等）。泄漏的 psm_* 段在长期运行的 runner 容器中累积，直到 /dev/shm（1-GPU runner 上仅 1 GiB）100% 占满，导致下一个调度器初始化时出现 'Fatal Python error: Bus error' 崩溃。实际在 h100-novita-host3-gpu-2 上观察到 base-b-test-1-gpu-large 失败，/dev/shm 被数百个多 MB 段填满。

实现拆解

1. 新增 stale_shm_cleanup 模块(python/sglang/srt/utils/stale_shm_cleanup.py)：定义 make_shm_name(kind) 生成 sgl_shm_<kind>_<pid>_<rand> 格式的段名；实现 cleanup_stale_shm() 遍历 /dev/shm，对匹配段提取 PID 并通过 os.kill(pid,0) 判断进程是否存活，若已死则 unlink；所有操作以 SGLANG_IS_IN_CI 环境变量为门控（仅 CI 容器设置），异常容错绝不阻塞启动。
2. 改造四个匿名 SharedMemory 创建点：在 ShmRingBuffer (shm_broadcast.py)、in_the_same_node_as (parallel_state.py)、ShmPointerMMData (mm_utils.py)、ShmSyncBuffer (cuda_ipc_transport_utils.py) 的 SharedMemory(create=True) 调用中增加 name=make_shm_name(...) 参数，使这些段遵循可回收命名规范。
3. 集成到 CI 脚本：在 ci_install_dependency.sh 中（kill_existing_processes 之后、依赖安装之前）通过 python path/to/module.py 执行清扫模块。此时 sglang 尚未安装，该模块保持 import-free 且可通过 __main__ 入口运行。
4. 添加完整单元测试：新建 test_stale_shm_cleanup.py，包含名称格式、PID 解析、清扫场景（清理死段、保留活段、不碰外来段）、CI 外无操作、生产绑定（ShmRingBuffer 名称校验）、离线路径执行等 7 个测试用例，注册到 CPU CI suite。

关键文件：

- python/sglang/srt/utils/stale_shm_cleanup.py（模块 共享内存清扫；类别 source；类型 core-logic；符号 make_shm_name, _creator_pid, _pid_alive, cleanup_stale_shm）：核心新模块：定义共享内存命名规则与清扫逻辑，是本次变更的核心实现。
- test/registered/utils/test_stale_shm_cleanup.py（模块 单元测试；类别 test；类型 test-coverage；符号 _spawn_dead_pid, TestMakeShmName, test_embeds_pid_and_is_unique, test_creator_pid_parsing）：完整单元测试覆盖：包括命名解析、场景测试（清理 / 不清理 / 不误删）、生产站点绑定测试、离线执行测试等
- python/sglang/srt/distributed/device_communicators/shm_broadcast.py（模块 通信层；类别 source；类型 dependency-wiring）：ShmRingBuffer 创建点改造：传入 name=make_shm_name('mq') 使共享内存段可被回收
- python/sglang/srt/distributed/parallel_state.py（模块 并行状态；类别 source；类型 dependency-wiring）：in_the_same_node_as 函数创建点改造：传入 name=make_shm_name('nodecheck') 使其可回收
- python/sglang/srt/managers/mm_utils.py（模块 MM 工具；类别 source；类型 dependency-wiring）：ShmPointerMMData 创建点改造：传入 name=make_shm_name('mm') 使其可回收
- python/sglang/srt/utils/cuda_ipc_transport_utils.py（模块 IPC 传输；类别 source；类型 dependency-wiring）：ShmSyncBuffer 创建点改造：传入 name=make_shm_name('sync') 使其可回收
- scripts/ci/cuda/ci_install_dependency.sh（模块 CI 脚本；类别 infra；类型 infrastructure）：CI 脚本集成清扫调用：在 job 开始时运行 cleanup_stale_shm 模块

关键符号：make_shm_name, _creator_pid, _pid_alive, cleanup_stale_shm, _is_in_ci, _cleanup_stale_shm_impl, in_the_same_node_as, ShmRingBuffer, ShmPointerMMData, ShmSyncBuffer

评论区精华

PR 页面无公开审核评论，但从 commit 信息 “address review: crash-proof sweep, log failures, cover remaining anonymous shm sites” 可以推断，评审中重点关注了：（1）清扫失败不应阻塞服务器启动（最终实现最外层 try/except + logging.warning）；（2）失败日志应记录而非静默忽略；（3）所有匿名共享内存站点都应被覆盖。PR 描述也明确包含了设计原则：CI-only、Best-effort、PID 重用降级为少清理而非误删。

- 清扫失败不应阻塞启动 (design): 已实现：cleanup_stale_shm() 不抛出异常，失败仅记录 warning。
- PID 重用导致漏清理的设计接受 (design): 接受退化，维持安全偏向。

风险与影响

- 风险：
  - PID 重用风险：死进程的 PID 可能被新进程重用，此时 _pid_alive 返回 True 导致对应段不被清理。设计接受此退化：最多漏清理，不会误删。

- CI 环境假阳性：若用户在非 CI 环境设置 SGLANG_IS_IN_CI=true 并运行，可能误清理其他用户段。但实际该变量仅在 CI 容器内由系统设置，共享机器上用户通常不会设置。
- /dev/shm 不存在：代码已判断 is_dir()，不会异常。
- 清扫全异常捕获：cleanup_stale_shm 外层捕获所有异常，不会阻止启动。
- 对生产环境无影响。
- 影响：直接影响 CI runner 可靠性：新 job 启动时会自动清理遗留共享内存段，预计可消除因 /dev/shm 满导致的 CI 失败，减少假阳性，提升 CI 稳定性。对用户无感知，对系统架构无侵入。
- 风险标记：仅 CI 适用，PID 重用退化，最佳努力清扫

关联脉络

- 暂无明显关联 PR