PR #26964 完整报告

sgl-project/sglang

Fix kill_process_tree reap wait crashing on pidfd EINVAL

合并时间：2026-06-02 04:56

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26964

执行摘要

- 一句话：修复 kill_process_tree 因 pidfd EINVAL 崩溃
- 推荐动作：该 PR 是典型的系统兼容性修复，值得合并。建议精读 _still_holding_resources 和新的 _wait_for_reap_or_raise 实现，理解如何用轮询替代 psutil.wait_procs 绕过内核限制。该模式在未来处理类似 os.pidfd_open 兼容性问题时可复用。

功能与动机

psutil.wait_procs 内部对非子进程使用 os.pidfd_open，在某些内核上对刚被 SIGKILL 的进程会抛出 OSError(EINVAL)，导致测试 tearDownClass 中的 reap 等待意外中止，即使测试本身已通过。PR body 明确描述了这一问题。

实现拆解

1. 新增 _still_holding_resources 函数（python/sglang/srt/utils/common.py）：遍历进程列表，通过 p.is_running() 和 p.status() != psutil.STATUS_ZOMBIE 判断进程是否仍持有资源。僵尸进程（Zombie）视为资源已释放，因为内核已回收其 GPU context 等。对 psutil.NoSuchProcess 和 OSError 静默跳过。
2. 重写 _wait_for_reap_or_raise 函数：将原本基于 psutil.wait_procs 的两阶段等待替换为轮询循环。计算 deadline 和 warn_deadline，每轮调用 _still_holding_resources 检查存活进程。若超过 deadline 仍有进程未退出则抛出 RuntimeError；若首次达到 warn_deadline 则打印警告日志。每次轮询后 time.sleep(0.1) 避免忙等待。
3. 更新函数文档字符串：在 _wait_for_reap_or_raise 的 docstring 中说明改用轮询 /proc 而非 psutil.wait_procs 的原因（绕过 pidfd EINVAL）。
4. 调用方 kill_process_tree 无需修改：其接口与行为完全不变，仅底层等待逻辑被替换。

关键文件：

- python/sglang/srt/utils/common.py（模块 工具函数；类别 source；类型 core-logic；符号 _still_holding_resources）：包含全部变更：新增 _still_holding_resources 函数，重写 _wait_for_reap_or_raise 用轮询替代 psutil.wait_procs，修复 OSError(EINVAL) 崩溃。

关键符号：_still_holding_resources, _wait_for_reap_or_raise, kill_process_tree

关键源码片段

python/sglang/srt/utils/common.py

包含全部变更：新增 _still_holding_resources 函数，重写 _wait_for_reap_or_raise 用轮询替代 psutil.wait_procs，修复 OSError(EINVAL) 崩溃。

```
def _still_holding_resources(procs):
    """Procs still holding GPU context, pinned memory or fds.

    A zombie has already had its resources freed by the kernel (only the exit
    status lingers), so it counts as gone; NoSuchProcess / OSError (see
    _wait_for_reap_or_raise) mean the same.
    """
    alive = []
    for p in procs:
        try:
            # 调用 psutil.Process.is_running() 会检查 /proc/<pid> 是否存在；
            # status() 读取 /proc/<pid>/status，Zombie 状态表示进程已死但保留退出码。
            # 只有真正还活着的非 Zombie 进程才视为仍持有资源。
            if p.is_running() and p.status() != psutil.STATUS_ZOMBIE:
                alive.append(p)
        except (psutil.NoSuchProcess, OSError):
            # 进程已消失或 pidfd_open 失败（如 EINVAL），视为已释放。
            pass
    return alive


def _wait_for_reap_or_raise(procs, wait_timeout: float) -> None:
    """Wait for `procs` to exit; warn at ~10s, raise on `wait_timeout`.

    SIGKILL is asynchronous -- children hold GPU context, pinned memory and
    fds until the kernel reaps them. Raise on timeout so a stuck process
    surfaces instead of leaving a latent race.

    Polls /proc via is_running()/status() rather than psutil.wait_procs, whose
    os.pidfd_open path (used for non-child procs) raises OSError(EINVAL) against
    a just-killed process on some kernels and aborts the whole wait.
    """
    warn_at = min(10.0, wait_timeout / 2)
    deadline = time.monotonic() + wait_timeout
    warn_deadline = time.monotonic() + warn_at
    warned = False
    while True:
        alive = _still_holding_resources(procs)
        if not alive:
            return
        now = time.monotonic()
        if now >= deadline:
            raise RuntimeError(
                f"kill_process_tree: {len(alive)} process(es) not reaped within "
```

```
                f"{wait_timeout}s after SIGKILL; pids={[p.pid for p in alive]}"
            )
        if not warned and now >= warn_deadline:
            logger.warning(
                "kill_process_tree: %d process(es) still alive after %.1fs SIGKILL; "
                "continuing to wait up to %.1fs total. pids=%s",
                len(alive),
                warn_at,
                wait_timeout,
                [p.pid for p in alive],
            )
            warned = True
        # 短 sleep 避免忙等；轮询间隔 100ms 可接受。
        time.sleep(0.1)
```

评论区精华

PR 没有 review 评论或讨论线程。开发者独立完成了分析和修复。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 轮询开销：_still_holding_resources 在 while True 循环中每 0.1 秒调用一次 is_running() 和 status()，在进程数量多时可能有一定 CPU 开销，但通常 kill_process_tree 的调用频率很低（仅测试清理场景）。
  2. 行为改变：新逻辑采用固定轮询间隔而非内核通知，可能导致释放检测延迟（最多 100ms），但绝大多数场景仍能正常工作。
  3. psutil.wait_procs 被完全替换：原本 wait_procs 利用内核事件驱动，新实现为纯用户态轮询，但考虑到原行为在特定内核上根本不可用，这一 trade-off 可以接受。
  4. 僵尸进程处理：Zombie 被视为已释放资源（status() == psutil.STATUS_ZOMBIE 时跳过），这是正确的——僵尸进程只保留 exit status，不持有 GPU 资源。 - 影响：影响范围仅限于 python/sglang/srt/utils/common.py 中的 _wait_for_reap_or_raise 和新增的 _still_holding_resources。影响用户为所有调用 kill_process_tree(wait_timeout=...) 的代码路径，主要是测试清理和进程管理场景。修复了在某些内核版本上测试 teardown 因 OSError(EINVAL) 意外失败的问题，提升测试稳定性和开发体验。 - 风险标记：缺少测试覆盖

关联脉络

- 暂无明显关联 PR