PR #29130 完整报告

vllm-project/vllm

[Test] Fix @create_new_process_for_each_test("fork") in interactive shell pipeline

合并时间：2026-04-16 00:22

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/29130

执行摘要

- 一句话：修复测试装饰器在交互式 shell 管道中因进程组信号传播导致的提前终止问题。
- 推荐动作：该 PR 值得快速浏览，重点关注进程组隔离的设计决策：将 os.setpgrp() 移到子进程是解决交互式 shell 中信号传播问题的关键技巧，展示了如何优雅处理多进程测试环境中的边缘情况。对于涉及多进程测试或信号处理的开发者，这段代码提供了实用参考。

功能与动机

PR body 中描述，当在交互式 shell 中运行使用 @create_new_process_for_each_test 装饰器的测试（如 tests/v1/engine/test_engine_core.py::test_engine_core）并通过管道（如 2>&1 | tee output.log）捕获输出时，测试会提前终止。原因是 tee 进程与 Python 进程属于同一进程组，父进程尝试忽略 SIGTERM 信号，但 tee 进程仍被杀死，导致整个管道失败。禁用作业控制（set +m）后测试正常，确认了进程组信号传播的问题。修复目标是使测试在交互式 shell 管道中稳定运行。

实现拆解

1. 移动进程组设置到子进程：在 tests/utils.py 的 fork_new_process_for_each_test 函数中，将 os.setpgrp() 调用从父进程（wrapper 函数开头）移到子进程分支（if pid == 0: 内部），使子进程成为自己进程组的领导者，隔离信号传播。
2. 简化父进程清理逻辑：移除父进程中忽略和恢复 SIGTERM 信号处理器的代码（signal.signal(signal.SIGTERM, signal.SIG_IGN) 和恢复操作），因为子进程独立后父进程不再需要处理 SIGTERM。同时，将获取进程组 ID（pgid）从 os.getpgid(pid) 改为直接使用 pid（因为 setpgrp() 后子进程 pgid 等于其 pid），并添加 contextlib.suppress(ProcessLookupError) 包装 os.killpg 调用，优雅处理子进程已退出的情况。
3. 测试配套：此变更仅涉及测试工具函数，没有新增测试文件或配置改动，但修复了现有测试在特定环境下的执行问题。

关键文件：

- tests/utils.py（模块 测试工具；类别 test；类型 test-coverage；符号 fork_new_process_for_each_test）：这是唯一变更的文件，包含修复测试装饰器进程组问题的核心逻辑。

关键符号：fork_new_process_for_each_test

关键源码片段

tests/utils.py

这是唯一变更的文件，包含修复测试装饰器进程组问题的核心逻辑。

```
def fork_new_process_for_each_test(func: Callable[_P, None]) -> Callable[_P, None]:
    """Decorator to fork a new process for each test function.
    See https://github.com/vllm-project/vllm/issues/7053 for more details.
    """

    @functools.wraps(func)
    def wrapper(*args: _P.args, **kwargs: _P.kwargs) -> None:
        from _pytest.outcomes import Skipped

        # 创建临时文件存储子进程异常信息
        with (
            tempfile.NamedTemporaryFile(
                delete=False,
                mode="w+b",
                prefix=f"vllm_test_{func.__name__}_{os.getpid()}_",
                suffix=".exc",
            ) as exc_file,
            ExitStack() as delete_after,
        ):
            exc_file_path = exc_file.name
            delete_after.callback(os.remove, exc_file_path)

            pid = os.fork()
            print(f"Fork a new process to run a test {pid}")
            if pid == 0:
                # 关键变更：将子进程设置为独立进程组的领导者，隔离信号传播
                os.setpgrp()
                # 父进程负责删除临时文件，子进程中不删除
                delete_after.pop_all()
                try:
                    func(*args, **kwargs)
                except Skipped as e:
                    print(str(e))
                    os._exit(0)
                except Exception as e:
                    # 异常处理逻辑（序列化异常并写入文件）
                    import traceback
                    tb_string = traceback.format_exc()
                    exc_to_serialize: dict[str, Any]
                    try:
                        exc_to_serialize = {"pickled_exception": e}
                        cloudpickle.dumps(exc_to_serialize)
                    except (Exception, KeyboardInterrupt):
                        exc_to_serialize = {
```

```
                "exception_type": type(e).__name__,
                "exception_msg": str(e),
                "traceback": tb_string,
            }
        try:
            with open(exc_file_path, "wb") as f:
                cloudpickle.dump(exc_to_serialize, f)
        except Exception:
            print(tb_string)
        os._exit(1)
    else:
        os._exit(0)
else:
    # 子进程调用setpgrp()后，其进程组ID等于pid
    pgid = pid
    _pid, _exitcode = os.waitpid(pid, 0)
    # 清理子进程组，但可能已正常退出，使用suppress避免ProcessLookupError
    with contextlib.suppress(ProcessLookupError):
        os.killpg(pgid, signal.SIGTERM)
    if _exitcode != 0:
        # 读取子进程异常信息并重新抛出
        exc_info = {}
        if os.path.exists(exc_file_path):
            with contextlib.suppress(Exception), open(exc_file_path, "rb") as f:
                exc_info = cloudpickle.load(f)
        if (original_exception := exc_info.get("pickled_exception")) is not None:
            assert isinstance(original_exception, Exception)
            raise original_exception
        if (original_tb := exc_info.get("traceback")) is not None:
            raise AssertionError(
                f"Child process failed with:\n{original_tb}"
            )
        raise AssertionError("Child process failed with unknown error")
```

评论区精华

review 中只有两条评论：

- gemini-code-assist[bot]指出 PR 提供了稳健的修复，正确地将 os.setpgrp() 移到子进程，这是隔离子进程及其后代以进行基于组信号的标准且正确方法，简化了代码并移除了有问题的信号忽略逻辑，同时添加 contextlib.suppress(ProcessLookupError) 使清理过程更具弹性。
- robertgshaw2-redhat简单赞赏调查工作（"nice investigation!"）。 没有争议点或未解决疑虑，讨论聚焦于对修复方案的认可。
- 修复方案的正确性与简化 (design): 修复方案被认可为稳健且正确，无争议。

风险与影响

- 风险：技术风险较低：
- 回归风险：变更仅影响测试装饰器的进程管理逻辑，不涉及核心业务代码；但若 os.setpgrp() 在子进程中调用失败或平台不支持，可能影响测试执行，不过原逻辑已使用此 API，风险可控。
- 兼容性风险：os.setpgrp() 和 os.killpg 是标准 POSIX API，在支持 fork 的平台上应保持兼容；但 Windows 平台可能不适用，不过 vLLM 主要面向 Linux/Unix 环境。
- 性能风险：无显著性能影响，仅微调进程创建和清理顺序。 主要风险在于如果子进程未能正确成为进程组领导者，信号传播问题可能残留，但 review 中已确认方案正确。
- 影响：影响范围有限：
- 用户影响：无直接影响，仅涉及内部测试执行环境。
- 系统影响：修复后，使用该装饰器的测试在交互式 shell 管道中能正常完成，提升测试可靠性和开发者体验。
- 团队影响：简化了测试装饰器逻辑，移除冗余信号处理代码，便于维护；关联历史 PR（#23795 和 #7054）显示这是长期问题的持续改进。
- 风险标记：进程管理变更，信号处理调整

关联脉络

- PR #23795 [Test] Fix @create_new_process_for_each_test("fork") in interactive shell pipeline: PR body 中提及此 PR 为相关历史 PR，可能涉及同一问题的先前修复或讨论。
- PR #7054 未知（根据 PR body 引用）：PR body 中提及 #7054 和 #7053，关联到原始 issue，显示这是长期测试环境问题的持续改进。