PR #36589 完整报告

sgl-project/sglang

fix: kill_process_tree waits for the reap by default

合并时间：2026-08-27 17:34

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36589

执行摘要

- 一句话：kill_process_tree 默认等待回收，修复 GPU 资源泄漏
- 推荐动作：值得精读，尤其是 kill_process_tree 的语义变更和进程生命周期管理逻辑，对理解 SGLang 的进程模型和资源清理有重要参考价值。

功能与动机

在 CI 日志中，test_metrics 返回时服务器子进程仍占用 29.14 GiB GPU 显存，导致后续任务等待超时。作者指出 SIGKILL 不直接释放资源，只有进程退出时才在 exit_files 关闭 /dev/nvidia* 时释放，因此 kill_process_tree 必须等待回收。文档也已说明 '>30s observed on GB300'，但默认 None 使大量调用点不安全。

实现拆解

1. 修改默认等待：在 python/sglang/srt/utils/common.py 中，将 kill_process_tree 的 wait_timeout 参数默认值从 None 改为 60，并扩展 docstring 说明原因。
2. 调整异常处理：将 children() 调用移入 NoSuchProcess 守卫内，避免目标进程在查找和遍历之间死亡导致异常逃逸。
3. tokenizer_manager 适配：在 sigterm_watchdog 中改为 include_parent=False 并显式传递 wait_timeout=60，使子进程被收割后再退出，而不是交给 init 收养。
4. 保持 __del__不阻塞：在 runtime_endpoint.py 的 shutdown 中显式传递 wait_timeout=None，避免在垃圾回收时阻塞。

关键文件：

- python/sglang/srt/utils/common.py（模块 工具库；类别 source；类型 core-logic；符号 kill_process_tree）：修改 kill_process_tree 默认等待，是本次变更核心。
- python/sglang/srt/managers/tokenizer_manager.py（模块 管理器；类别 source；类型 core-logic；符号 sigterm_watchdog）：修改 sigterm_watchdog 的进程回收方式，使子进程被收割。
- python/sglang/lang/backend/runtime_endpoint.py（模块 后端；类别 source；类型 core-logic；符号 shutdown）：显式保持 __del__路径不阻塞。

关键符号：kill_process_tree, sigterm_watchdog, shutdown

关键源码片段

python/sglang/srt/utils/common.py

修改 kill_process_tree 默认等待，是本次变更核心。

```
# python/sglang/srt/utils/common.py

def kill_process_tree(
    parent_pid,
    include_parent: bool = True,
    skip_pid: int = None,
    wait_timeout: Optional[float] = 60, # 默认等待，避免资源残留
):
    """Kill the process and all its child processes.

    `wait_timeout` (seconds) blocks until every killed process is reaped and
    raises `RuntimeError` on timeout. SIGKILL only queues the teardown, so
    returning without waiting leaves the GPU context, the pinned host memory
    and the ports held for seconds; pass `None` only where blocking is
    unacceptable, such as a `__del__`.
    """
    logger.info(
        f"kill_process_tree called: parent_pid={parent_pid}, "
        f"include_parent={include_parent}, pid={os.getpid()}"
    )

    if parent_pid is None:
        parent_pid = os.getpid()
        include_parent = False

    try:
        itself = psutil.Process(parent_pid)
        # 在守卫内获取 children，避免目标进程在查找和遍历之间死亡
        children = itself.children(recursive=True)
    except psutil.NoSuchProcess:
        return

    killed = []
    for child in children:
        if child.pid == skip_pid:
            continue
        try:
            child.kill()
            killed.append(child)
        except psutil.NoSuchProcess:
            pass

    if include_parent:
        try:
```

```
            if parent_pid == os.getpid():
                itself.kill()
                sys.exit(0) # 无法等待自身，直接退出

            itself.kill()
            itself.send_signal(signal.SIGQUIT) # 补充信号确保杀死
            killed.append(itself)
        except psutil.NoSuchProcess:
            pass

    if wait_timeout is not None and killed:
        _wait_for_reap_or_raise(killed, wait_timeout)
```

python/sglang/srt/managers/tokenizer_manager.py

修改 sigterm_watchdog 的进程回收方式，使子进程被收割。

```
# python/sglang/srt/managers/tokenizer_manager.py

async def sigterm_watchdog(self):
    # ... 健康检查和退出逻辑
    # 等待调度器释放资源
    self._dispatch_to_scheduler(ShutdownReq())
    deadline = time.monotonic() + 15
    while time.monotonic() < deadline and collect_scheduler_processes():
        time.sleep(0.1)
    # 收割所有子进程，再退出自身，避免进程被重新收养导致资源残留
    kill_process_tree(os.getpid(), include_parent=False, wait_timeout=60)
    sys.exit(0)
```

python/sglang/lang/backend/runtime_endpoint.py

显式保持 __del__路径不阻塞。

```
# python/sglang/lang/backend/runtime_endpoint.py

def shutdown(self):
    from sglang.srt.utils import kill_process_tree

    if self.pid is not None:
        # 注意：__del__ 会调用此方法，阻塞回收会卡住 GC 线程
        kill_process_tree(self.pid, wait_timeout=None)
        self.pid = None
```

评论区精华

作者在 PR body 中提出开放问题：_wait_for_reap_or_raise 在超时时抛出异常，而部分新等待点是 finally 块，会掩盖原始异常。作者认为大声报错更合适，但也愿意改为 log-and-continue。

- 超时异常处理 (design): 作者倾向保留大声报错，但也愿意改为 log-and-continue，最终代码未显式处理，可能默认保留抛错。

风险与影响

- 风险：风险点集中在默认行为变更可能影响调用方：kill_process_tree 现在默认阻塞，若调用方不期望等待可能挂起。但调用限定了超时，且有显式 None 用于 __del__。另外，tokenizer_manager 的 include_parent 修改可能改变退出行为，需验证信号处理。
- 影响：影响范围主要在 SGLang 内部进程管理和测试基础设施：默认等待能防止 GPU 显存和端口资源残留，减少 CI 超时和僵尸进程。对用户而言，尤其影响使用 kill_process_tree 的脚本和测试，可能有额外等待时间。
- 风险标记：默认行为变更，可能掩盖异常

关联脉络

- PR #23213 kill_process_tree wait_timeout: 该 PR 引入了 wait_timeout 机制，本 PR 在其基础上修改默认值。