提交评测 2024-05-22 21:20:02 | 评测冷却: 120s

提交

更新

Lab 4-2 Extra

准备工作：创建并切换到 `lab4-2-extra` 分支

请在**自动初始化分支后**，在开发机依次执行以下命令：

```
$ cd ~/学号
$ git fetch
$ git checkout lab4-2-extra
```

初始化的 `lab4-2-extra` 分支基于课下完成的 `lab4` 分支，并且在 `tests` 目录下添加了 `lab4_clone` 样例测试目录。

题目背景

轻量级进程（Light-weight process）是建立在内核之上并由内核支持的用户线程，与普通进程相比，轻量级进程与其他进程**共享**所有（或大部分）它的逻辑地址空间和系统资源；与线程相比，轻量级进程有它自己的进程标识符，并和其他进程有着父子关系；线程既可由应用程序管理，又可由内核管理，而轻量级进程只能由内核管理并像普通进程一样被调度。

题目描述

在本题中，你需要实现一个简化版的轻量级进程。

简化的轻量级进程可以被理解为一种特殊的进程，它的调度、销毁等与进程没有区别，而它最显著的特点是**共享**父进程的**内存空间**。

在已经完成的进程创建流程中，我们会为每个进程分配独立的页表，而只有带有 `PTE_LIBRARY` 的页才会与其他子进程共享；轻量级进程在创建时不再会被分配页表，而是**直接使用父进程的页表**，从而实现了内存空间的共享。同时，为了防止大量轻量级进程耗尽资源，我们设定共享同一页表、同时存活的轻量级进程数量上限为64。

在题目表述中，任何创建了轻量级进程的进程也被视作轻量级进程。测试保证轻量级进程不会进一步 fork 子进程或进行 IPC 通信，轻量级子进程执行过程中也一定不出现写时复制（COW）相关异常的处理。

本题中，你需要实现系统调用函数 `syscall_clone` ，用于轻量级进程的创建。

```
int syscall_clone(void *func, void *child_stack)
```

该函数创建了一个当前进程的轻量级子进程，并将其加入进程的调度序列，参数详细说明如下：

- `func` ：新创建的轻量级进程开始执行的函数地址。
- `child_stack` ：新创建的轻量级进程的栈指针。轻量级进程具有自己的栈空间，保证不与父进程的栈空间重叠。
- 返回值：若创建成功则返回轻量级进程的 `env_id` 。如果与当前轻量级进程共享内存空间的、存活的轻量级进程 (包括当前进程) 的数目已超过 64 ，停止创建并返回 `-E_ACT_ENV_NUM_EXCEED` 即 **-14** 。

注意：你需要在 `include/error.h` 中追加定义 `#define E_ACT_ENV_NUM_EXCEED 14` 。

轻量级进程的调度、销毁需与进程的调度、销毁保持一致。

参考实现思路

为了实现内存的共享，参考实现思路如下，也可采取其他思路，满足题目要求即可。

为了确保页表仅在不再被任何进程需要时才被回收，延长页表的生命周期，系统需设立一种机制以跟踪每个页表被多少进程所共享。这可以通过为每个进程页表加入一个引用计数器来实现，该引用计数器负责记录此页表被进程引用的次数。

注意到由于**MIPS内存布局**的限制，页表中对应 `KSEG0` ， `KSEG1` 区域的表项永远不会被使用，因此可在页目录中对应在 `KSEG0` 、 `KSEG1` 区域的任意页目录项存储该计数器变量，无需更改其他数据结构。

整体题目具体实现步骤如下:

1. 在 `env.c` 中的 `env_setup_vm` 函数中，设置进程页表的页表引用计数值为 1 。
2. 在 `env.c` 中的 `env_free` 函数中，当且仅当页表的引用计数值为 1 时才释放 `asid` 与页表，否则仅减少页表引用计数器的值。
3. 在 `env.c` 中实现 `env_clone` 函数用于分配新的轻量级进程，该函数功能及参数返回值与 `env_alloc` 函数基本一致，可参考 `env_alloc` 的实现并对其加以修改，其中具体区别如下：

计数。

4. 在 include/syscall.h 中添加系统调用号 SYS_clone ，在 syscall_lib.c 中实现 syscall_clone 函数，并在 kern/syscall_all.c 中实现 sys_clone 系统调用，系统调用实现步骤如下：

   (1) 如果当前进程的页表被引用次数已大于等于64，则直接返回 -E_ACT_ENV_NUM_EXCEED 。

   (2) 通过 env_clone 函数分配新的轻量级进程。

   (3) 设置新轻量级进程的 trapframe 继承父亲的 trapframe，并根据系统调用传递的 func 及 child_stack 完成对新轻量级进程的 trapframe 部分字段修改。

   (4) 设置轻量级进程的运行状态为 ENV_RUNNABLE，并将其插入调度队列尾部。

本地测试说明

你可以使用：

- make test lab=4_clone && make run 在本地测试上述样例（调试模式）
- MOS_PROFILE=release make test lab=4_clone && make run 在本地测试上述样例（开启优化）

在本地测试样例中，我们在父进程中先创建了一个轻量级子进程。轻量级子进程创建之后，由父进程修改一个全局变量，并在轻量级子进程中检查是否能观察到相应全局变量的修改。

如果你的输出结果包含如下内容，说明你通过了本地样例测试。

```
They share same memory!
halt at test.c:26: child env ended
```

提交评测

请在开发机中执行下列命令后，**在课程网站上提交评测。**

```
$ cd ~/学号/
$ git add -A
```

提交评测 2024-05-22 21:20:02 | 评测冷却: 120s

提交

更新

评测说明

评测保证不出现轻量级进程与进程之间的通信，评测主要涉及对共享内存的测试、新分配二级页表的测试、多级 `syscall_clone` 测试等。

具体要求和分数分布如下：

测试点序号	评测说明	分值
1	与样例相同	10
2	共享内存测试	20
3	二级页表测试	10
4	多级 `syscall_clone` 测试	20
5	栈空间测试	10
6	线程数限制测试	10
7	综合测试	20