深 圳 大 学 实 验 报 告

课程名称： 操作系统

实验项目名称：综合实验二 xv6线程+虚存

学院： 计算机与软件学院

专业： 计算机科学与技术

指导教师： 罗秋明

报告人：谭嘉豪 学号： 2020152087 班级： 数计班

实验时间： 2023年6月1日

实验报告提交时间： 2023年6月15日

教务部制

|  |
| --- |
| 一、实验目的与要求   1. 加深对线程和虚存的直观认识； 2. 掌握xv6操作系统中实现线程的核心机制； 3. 掌握xv6操作系统中实现虚存的基本方法；   二、实验内容   1. 修改xv6内核代码实现简单线程；（40%） 2. 修改xv6内核代码实现文件访问权限控制；（30%） 3. 修改xv6内核代码实现简单的页帧交换。（20%） 4. 修改xv6内核代码实现（有名）管道（10%）   三、实验步骤及说明   1. 实现xv6内核线程   修改PCB结构体，如下：  添加两个变量pthread和ustack。    在 proc.c 的 allocproc()中将 pthread 初始化为 0。    接下来实现sys\_clone和sys\_join的系统调用。   * sys\_clone   线程的创建由 clone() 系统调用实现。clone() 还要负责初始化用户 栈，使得线程回到用户态后能找到对应的入口。  在kernel/proc.c文件在实现clone方法。由于新创建线程的内核栈 trapframe 里的 era 被设置成传入的参数 fcn（即线程函数），因 此当 clone()返回到用户态后，将执行 fcn 所指定的函数。用户态的线程栈则填写了对应的线程 函数的参数以及返回地址。这里的简化实现中，要求线程结束处必须显式调用 exit()。当然也 可以将返回地址填上 exit()的地址，从而不必显式调用。 这里的简化实现方案中，并没有对页表的引用进行计数，如果主线程提前结束释放内存空 间后会造成子线程的异常。  实现clone方法的代码如下。    将clone方法封装成sys\_clone方法，在sysproc.c文件中实现。     * sys\_join   在这里，子进程和子线程的结束时的资源回收过程有所不同：  （1）子进程的 PCB 由父进程调用 wait() 回收；  （2）子线程的 PCB/TCB 由父进程/线程调用 join() 回收。  由于两者差异很小，因此 join() 的实现和 wait() 极其相似。join() 可看作 clone() 的逆过程，由当前线程回收状态为 ZOMBIE 的子线程。子线程需要调用 exit() 才会变成 ZOMBIE 线 程。  实现join方法的代码如下。    将join方法封装成sys\_join方法，在sysproc.c文件中实现。    可以看出，如果主线程调用 join()的时候，子线程还未执行 exit()，那就找不到状态为 ZOMBIE 子线程，主线程将会进入睡眠状态。因此子线程退出的时候需要唤醒对应的主线程。 我们需要在 proc.c 的 exit() 中增加唤醒主线程的功能，具体代码如下。    在user/user.h中声明clone方法和join方法。    在syscall.h文件中添加系统调用编号。    在syscall.c文件的跳转表添加如下代码，并添加代码“extern uint64 sys\_clone(void); extern uint64 sys\_join(void);”声明这两个函数是外部函数。    在user/usys.pl在添加如下代码。    为了方便用户程序调用。我们还需要实现相应的用 户线程库，帮忙管理用户栈和线程的协调。用户库为 uthread.c，需要将其添加到 Makefile 的 ULIB 变量中。    create\_thread()用于创建线程，需要提供待执行的线程函数和运行参数。create\_thread()通过clone()创建线程，需要提前用 malloc()分配一个线程栈，最后借助于 add\_thread()将线程记 录在本进程的 TCB 数组 threads[NTHREAD]中。对应地有一个 thread\_join()用于等待线程结束，它通过 join()系统调用回收已经停止的线程，然后通过 remove\_thread()从本进程的线程数组 threads[NTHEAD]中删除。  uthread.c的代码如下。    创建uthread.h文件，声明以下五个函数。    注意：在从内核态回到用户态时，需要判断 proc->pthread 保存对应的 trapframe。具体需要修改 kernel/trap.c 的usertrapret()函数。    编写thread\_test.c程序对实现的线程库进行测试。    在Makefile文件中添加如下代码。    测试结果如下，成功创建了线程并将参数传入，线程通过递归计算得到的斐波那契数列也正确，global 变量的使用结果也验证了线程间共享内存的事实，检查 tmp 文件也可以看到相应的字符串。     1. 实现文件访问权限控制 2. 添加访问权限位   xv6 的 inode 结构体定义在 kernel/file.h，其中一部分信息是要存储在硬盘上的，这部分 内容用 dinode（定义在 kernel/fs.h） 描述。由于磁盘布局是很严格的，所以结构体的大小都 是设计好的。xv6 中用来描述文件类型的变量是 short 类型，为了增加文件权限功能，且不影响文件系统布局，我们将 short 类型拆成两个 char 类型，一个用来当作 mode 来描述文件权限，此时新的 dinode 定义如下。同时 kernel/file.h 中的 inode 结构体也要做同样的修改。      在 mkfs/mkfs.c 中，对函数 ialloc() 进行修改，将参数中的 type 全部改为 uchar 类 型，并在函数中初始化 mode 为 3。    修改 kernel/fs.c 中的 ialloc() 中参数 type 改为 char 类型，并在函数中初始化 mode 为 3。同时修改在 defs.h 中的声明。      修改 kernel/sysfile.c 中的 create() 函数，将其参数 type 类型改为 char。    在 kernel/fs.c 中的 ilock() 中，将dinode->mode 传递给 inode->mode。    在 kernel/fs.c 中的 iupdate() 中，将 inode->mode 传递给 dinode->mode。    在 kernel/stat.h 中，修改 stat 结构体，添加 char 类型的 mode，并把 type 修改为 char 类型。    在 kernel/fs.c 中的 stati() 中，将 inode->mode 传递给 stat->mode。    在 user/ls.c 中，修改其中的几处 printf 输出语句，使得能打印文件的访问权限 mode。  运行xv6，执行ls命令，得到如下，从左到右分别是：文件名、访问权限、文件类型、索引节点、文件大小。       1. 设置权限的系统调用   xv6 的文件有 3 种类型，分别是目录、文件、设备。我们只限制普通文件的读写权限， 即 T\_FILE 类型的文件读写才受到控制。而且将 mode 的最低位作为读位，次低位作为写位， 则有   * 3 表示可读可写。 * 2 表示可写。 * 1 表示可读。 * 0 表示不可读不可写。   为了支持文件权限，且不改变 xv6 的接口，需要实现专门的系统调用来改变文件的读写权限。用户接口定义如下：    新增 sys\_chmod()来修改 inode->mode，该函数可以放到 kernel/sysfile.c 中实现，代码如下。     1. 读写前判断   接下来可以使用新增的权限来限制读写操作了。文件的读写函数分别是 fileread() 和 filewrite()，它们位于 kernel/file.c 中。  其中 fileread() 中的修改部分如下。    filewrite() 中的修改部分如下。    编写chmod\_test.c文件进行测试。    首先在默认读写权限（可读+可写）情况下，用 echo hello > content 将数据写入到 content 文件中，此时用 ls 查看 content 文件为访问权限为 3（表 示可读+可写），修改文件内容为world，修改成功。          重复上面的操作，但是第一次写入 hello 后用 chmod 命令将 content 修改为可读不可写的权限，然后执行第二次写入 world。由于第二次写操作前关闭了写入权限，预料到的情况应为：数据仍是原来的旧数据 hello，但是实际的情况却是只读文件content的数据全部清零。    最后，将 hello 写入到 content，然后用 chmod 命令将 content 设置为可写不可读的权限，再用用 cat 命令读取 content 文件内容时提示出错失败。     1. 实现简单的页帧交换   修 改 kernel/memlayout.h 中的参数 RAMSTOP 为 RAMBASE + 300\*4\*1024。    为了记录剩余页帧数量，需要修改 kernel/kalloc.c 中的 kmem 结构体，加上一个 count 计 数值（=freelist 长度），每次 kalloc()和 kfree()后以做相应修改，在kinit()在初始化为0。          在 proc.h 的 proc 结构体中添加一个变量 uint64 swap\_start， 用来记录 sbrk() 的起始地址。    在 exec() 和 fork() 中对 swap\_start 进行初始化。      bio.c 文件主要负责磁盘的读写，实现 write\_page\_to\_disk和 read\_page\_from\_disk操作用于承担换进换出时的磁盘读写操作，具体代码如下。。在write\_page\_to\_disk()中用到的 DMWIN\_MASK 宏定义在 kernel/memlayout.h 中，所以还要在 bio.c 中包含 memlayout.h。    由于是简化实现，并没有专门建立交换区，而是直接在普通文件区找到连续 4 个盘块（对应一个页，共 4KB）来存储一个换出的页帧。xv6 磁盘读写是建立在缓存块已经分配的基础上，所以我们还要负责缓存块的分配和释放，借用 bfree() 和 balloc() 实现两个函数 bfree4() 和 balloc4()，为数据交换提供容量为 4KB的缓存块，具体如代码如下。    在 sys\_sbrk() 函数中会调用 growproc(n) 来申请物理页帧， 将其注释掉，重新运行 xv6，执行某个可执行文件（例如 ls）后会出现如下语句。      这是因为 sh（shell）在执行外部命令 ls 的时候会调用 sbrk ()函数分配内存并使用，但由于我们注释掉了 growproc()并没有为之分配内存，因此导致缺页异常。  修改 kernel/trap.c 的 usertrap()，添加缺页中断处理，注意将上述sys\_sbrk()函数注释代码恢复。    将中断处理 pgfault() 函数定义在 kernel/vm.c（因为涉及 mappages()），其工作步骤如下：  （1）如果缺页地址大于 sz 则表示非法地址（未分配），终止程序；  （2）如果地址小于 proc->sz 且大于 swap\_start，则合法的可交换地址，需要进行处理。 首先检查引起缺页的 PTE 中 SWAPPED 位，判定该页是否被调出内存。如果是被换出则启动交 换机制；如果不是，则调用 kalloc() 生成一个物理页帧以供使用。如果 kaclloc() 没有分配到 物理页帧，则从进程空间中 swap\_start～sz 之间找一个物理页帧调出磁盘。如果没有找到可 供换出的页帧，则撤销本进程。  缺页时候需要找到一块已经有映射的页换出去，由于对应功能函数 pagfault()和进程空间 有关，我们将 pgfault()代码放到 vm.c 中。当需要执行页帧交换功能时，从 swap\_start 开始遍 历，找到一个直接返回，中断程序 pgfault()实现如代码如下。    pgfault()中用到的 r\_csr\_badv()函数需要在 kernel/loongarch.h 中添加实现。    其中的换出和换入功能由 swapout() 和 swapin()负责，我们将它们实现在 vm.c 中，具体 如代码 4-18 所示，同时需要在 vm.c 中包含 spinlock.h 和 proc.h。此处 swapout()换出的时候我 们从进程地址最高端开始往低地址扫描，直到 proc-> swap\_start。后面进行验证的时候，我们 会反过来扫描体现不同交换算法的差异。    swapout()函数中用到的 PTE\_SWAPPED 宏需要在 kernel/loongarch.h 中定义。    由于前面使用了延迟分配内存以及 swapout()函数中修改了换出的 pte 的 PTE\_V 标志，所 以需要在 uvmunmap()中做适当的修改，避免一些系统报错。    对uvmalloc方法修改如下。    在vm.c中加入以下头文件，头文件的顺序不可改变，否则编译出现以下报错。      在defs.h头文件中添加上述的函数声明。        编写测试程序swap\_demo.c，其内部操作，安排如下：  （1）为了便于实验观察，我们先分配并使用掉系统的大部分物理页帧，直到剩下 1 个物理页帧。  （2）然后分配 4个页，用于验证延时分配。  （3）分别给这 4个页的第一字节写入数值 a～d。访问第 4个页的第一字节并打印输出，然后访问第 1 到第 4个页的第一字并输出，验证缺页功能和交换功能。其中的 bstat()用与打印系统剩余物理页帧数量，也就是前面提到的 kmem 结构体新增的 count，增加其bstat()系统调用即可。    测试结果如下：    反转 swapout()的扫描过程，从低地址 proc-> proc-> swap\_start 到高地址 proc->sz 方 向查找换出页，则发现最后分配的 4 个页帧第一次访问时有四次缺页，但第二次访问时不会引 起缺页——因为换出的时前面分配的那些映射到低地址处页帧。    结果如下。    四、感想及其他  本次xv6综合实验二，我主要为xv6实现了简单的线程、文件访问权限控制、简单的页帧交换，在尝试实现简单的页帧交换时遇到了较大困难，但也对课本上的理论知识有了更深的体会。 |

深圳大学学生实验报告用纸

|  |
| --- |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  2023年 月 日 |
| 备注： |

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后10日内。