**《操作系统》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2020级计算机科学与技术01班**  **2020级计算机科学与技术02班** | | | **组员** | **陈鹏宇**  **徐小龙** |
| **实验题目** | **进程管理** | | | | | |
| **实验时间** | **2022.11.05** | | **实验地点** | **Ds3401** | | |
| **实验成绩** |  | | **实验性质** | **□验证性 □设计性 □综合性** | | |
| 教师评价：  **□**算法/实验过程正确； **□**源程序/实验内容提交 **□**程序结构/实验步骤合理；  **□**实验结果正确； **□**语法、语义正确； **□**报告规范；  其他：  评价教师签名： | | | | | | |
| 一、实验目的   1. 了解第一个用户进程创建过程； 2. 了解系统调用框架的实现机制； 3. 了解调用ucore如何实现系统调用sys\_fork/ sys\_exec/ sys\_exit/ sys\_wait来进行进程管理 | | | | | | |
| 二、实验项目内容  1. 加载应用程序并执行  do\_execve函数调用load\_icode（位于kern/process/proc.c中）来加载并解析一个处于内存中的ELF执行文件格式的应用程序，建立相应的用户内存空间来放置应用程序的代码段、数据段等，且要设置好proc\_struct结构中的成员变量trapframe中的内容，确保在执行此进程后，能够从应用程序设定的起始执行地址开始执行。需设置正确的trapframe内容。  (1) 请在实验报告中简要说明你的设计实现过程。  (2) 请在实验报告中描述当创建一个用户态进程并加载了应用程序后，CPU是如何让这个应用程序最终在用户态执行起来的。即这个用户态进程被ucore选择占用CPU执行（RUNNING态）到具体执行应用程序第一条指令的整个经过。  2. 父进程复制自己的内存空间给子进程  创建子进程的函数do\_fork在执行中将拷贝当前进程（即父进程）的用户内存地址空间中的合法内容到新进程中（子进程），完成内存资源的复制。具体是通过copy\_range函数（位于kern/mm/pmm.c中）实现的，请补充copy\_range的实现，确保能够正确执行。  (1) 请在实验报告中简要说明如何设计实现”Copy on Write 机制“，给出概要设计，鼓励给出详细设计。并指出实现COW时我们还需要添加哪种异常类型的处理？  3. 阅读分析源代码，理解进程执行 fork/exec/wait/exit 的实现，以及系统调用的实现  (1) 请在实验报告中简要说明你对 fork/exec/wait/exit函数的分析。并回答如下问题：  (2) 请分析fork/exec/wait/exit在实现中是如何影响进程的执行状态的？  (3)请给出ucore中一个用户态进程的执行状态生命周期图（包括执行状态，执行状态之间的变换关系，以及产生变换的事件或函数调用） | | | | | | |
| 三、实验过程或算法（源程序）  EX1: load\_icode      #ifdef LAB3\_EX1      /\* LAB3:EXERCISE1 YOUR CODE       \* should set tf\_era,tf\_regs.reg\_r[LOONGARCH\_REG\_SP],tf->tf\_prmd       \* NOTICE: If we set trapframe correctly, then the user level process can return to USER MODE from kernel and enable interrupt. So       \*          tf->tf\_prmd should be PLV\_USER | CSR\_CRMD\_IE       \*          tf->tf\_era should be the entry point of this binary program (elf->e\_entry)       \*          tf->tf\_regs.reg\_r[LOONGARCH\_REG\_SP] should be the top addr of user stack (USTACKTOP)       \*/      tf->tf\_era = elf->e\_entry; //era 指向ELF可执行文件加载到内存之后的入口处      tf->tf\_regs.reg\_r[LOONGARCH\_REG\_SP] = USTACKTOP; //将栈指针指向用户栈的顶部      uint32\_t status = 0; //      status |= PLV\_USER; // 将返回状态设置为用户特权级      status |= CSR\_CRMD\_IE;// 中断使能      tf->tf\_prmd = status;// 并写到PRMD中      #endif  EX2: copy\_range  int // 将实际的代码段和数据段搬到新的子进程里面去，再设置好页表的相关内容  copy\_range(pde\_t \*to, pde\_t \*from, uintptr\_t start, uintptr\_t end, bool share) {      assert(start % PGSIZE == 0 && end % PGSIZE == 0); //确保start和end可以整除PGSIZE      assert(USER\_ACCESS(start, end));      //以页为单位进行复制      do {          //得到A&B的pte地址          pte\_t \*ptep = get\_pte(from, start, 0), \*nptep;          if (ptep == NULL) {              start = ROUNDDOWN\_2N(start + PTSIZE, PGSHIFT);              continue ;          }          if (\*ptep & PTE\_P) {            // 获取子进程start地址对应的页表项的线性地址            if ((nptep = get\_pte(to, start, 1)) == NULL) {              return -E\_NO\_MEM;            }            uint32\_t perm = (\*ptep & PTE\_USER); // 获取父进程的权限            struct Page \*page = pte2page(\*ptep); // 获取父进程物理页的描述符            struct Page \*npage=alloc\_page(); // 分配子进程物理页            assert(page!=NULL); // 断言父子进程的物理页都存在            assert(npage!=NULL);            int ret=0;          #ifdef LAB3\_EX2          /\* LAB3 EXERCISE2: YOUR CODE           \* replicate content of page to npage, build the map of phy addr of nage with the linear addr start           \*           \* Some Useful MACROs and DEFINEs, you can use them in below implementation.           \* MACROs or Functions:           \*    page2kva(struct Page \*page): return the kernel vritual addr of memory which page managed (SEE pmm.h)           \*    page\_insert: build the map of phy addr of an Page with the linear addr la           \*    memcpy: typical memory copy function           \* (1) find src\_kvaddr: the kernel virtual address of page           \* (2) find dst\_kvaddr: the kernel virtual address of npage           \* (3) memory copy from src\_kvaddr to dst\_kvaddr, size is PGSIZE           \* (4) build the map of phy addr of  nage with the linear addr start           \*/            void \* kva\_src = page2kva(page);// 找寻父进程需要复制的虚拟页在内核虚拟页地址            void \* kva\_dst = page2kva(npage);// 找寻子进程需要被填充的内核虚拟页地址          memcpy(kva\_dst, kva\_src, PGSIZE);// 复制父进程内容到子进程中          ret = page\_insert(to, npage, start, perm);// 建立子进程的物理页与虚拟页的映射关系          #endif            assert(ret == 0);          }          start += PGSIZE;      } while (start != 0 && start < end);      return 0;  } | | | | | | |
| 1. 实验结果及分析   命令行输出如下：  IMG_256  练习一：   1. 设计实现过程：   结合实验背景介绍，代码阅读和注释，我们能直到load\_icode函数的主要工作就是给用户进程建立一个能让用户进程正常运行的用户环境。而我们要补充的代码就是伪造中断返回现场，是的系统调用之后可以正确跳转到需要运行的程序入口。根据注释提示，我们需将era指向e\_entry也就是ELF可执行文件加载到内存之后的入口处；将栈指针指向用户栈的顶部；设置返回状态为用户特权级；中断使能并写到PRMD中。   1. 描述当创建一个用户态进程并加载了应用程序后，CPU是如何让这个应用程序最终在用户态执行起来的：  * 调用schedule函数，经过调度器占用了CPU的资源之后，用户态进程调用了exec系统调用，从而转入到了系统调用的处理例程。 * 之后进行正常的中断处理例程，然后控制权转移到了syscall函数，根据系统调用号转移给了sys\_exec函数，在该函数中调用了do\_execve函数来完成指定应用程序的加载   调用kernel\_execve通过系统调用加载用户进程，会进入到syscall    Do\_execve主要工作就是先回收自身所占用户空间，然后调用load\_icode       * 使用 load\_icode 函数，完成了对整个用户线程内存空间的初始化，包括堆栈的设置以及将ELF可执行文件的加载，之后通过 current->tf指针修改了当前系统调用的trapframe，使得最终中断返回的时候能够切换到用户态，并且同时可以正确地将控制权转移到应用程序的入口处 * 完成sys\_exec后将进行正常的中断返回，执行中断返回指令“ertn”，并且完成特权级的切换，并且跳转到要求的应用程序的入口处，开始执行应用程序的第一条代码   练习二：  进程创建相关函数如下     * 父进程调用fork系统调用，进入正常的中断处理机制，最终交由syscall函数进行处理，在syscall函数中，根据系统调用，交由sys\_fork函数处理      * 该函数进一步调用了do\_fork函数，这个函数是主要的创建子进程、并且将父进程的内存空间复制给子进程。   分配并初始化进程控制块    分配并初始化内核栈    按clong\_flags复制父进程文件信息    复制或共享进程的内存管理结构    设置中断帧和执行上下文    把进程控制块放入到hash\_proc全局链表中    放入大链表    唤醒新程序，加入就绪序列     * 在do\_fork函数中，调用copy\_mm进行内存空间的复制，在该函数中，进一步调用了dup\_mmap，在这个函数中，遍历了父进程的所有合法虚拟内存空间，并且调用copy\_range将这些空间的内容复制到子进程的内存空间中去        * 在copy\_range函数中，对需要复制的内存空间按照页为单位从父进程的内存空间复制到子进程的内存空间中去     在所要填写部分，首先需找到父进程需要复制的虚拟页在内核虚拟页地址和子进程需要被填充的内核虚拟页地址，之后复制父进程内容到子进程中，建立子进程的物理页与虚拟页的映射关系，循环，直到父进程内容复制完，完成了内容的复制和映射的建立   * 请在实验报告中简要说明如何设计实现”Copy on Write 机制“，给出概要设计，鼓励给出详细设计。并指出实现COW时我们还需要添加哪种异常类型的处理？   共享父进程的内存空间，从而免除重新拷贝，除非要执行写操作。当一个用户父进程创建自己的子进程时,父进程会把其申请的用户空间设置为只读,子进程可共享父进程占用的用户内存空间中的页面。当其中任何一个进程修改此用户内存空间中的某页面时,ucore会通过page fault异常获知该操作,并完成拷贝内存页面,使得两个进程都有各自的内存页面。这样一个进程所做的修改不会被另外一个进程可见了。  练习三：   1. 请在实验报告中简要说明你对 fork/exec/wait/exit函数的分析。  * Fork   do\_fork函数完成进程的拷贝，内部具体细节在练习二中提到过。首先检测当前总进程数目是否到达限制，之后分配并初始化进程控制块，内核栈，复制或共享进程内存管理结构，设置中断帧和执行上下文，放入全局进程链表中，叫醒进程进入就绪态。   * Exec   Do\_execve函数完成用户进程的创建工作，同时使用户进程进入执行。在练习一中也展示过。首先检查进程名称的地址和长度是否合法，将cr3页表基址指向内核页表，然后实现对进程的内存管理区域的释放，调用load\_icode将代码加载进内存并建立新的内存映射关系，调用set\_proc\_name重新设置进程名称。   * Wait   Do\_wait函数完成对子进程的内核栈和进程控制块所占内存空间的回收。首先检查返回码指针地址位于合法的范围内    根据PID找到需要等待的子进程PCB，循环询问正在等待的子进程的状态，直到有子进程状态变为僵尸态。  如果pid！=0，则找到进程id为pid的处于退出状态的子进程    如果pid==0，则随意找一个处于退出状态的子进程    如果没找到，则父进程重新进入睡眠，并重复寻找的过程     * Exit   Do\_exit函数完成当前进程执行退出过程中的部分资源回收。  释放进程的虚拟内存空间    设置当期进程状态为PROC\_ZOMBIE即标记为僵尸进程    如果父进程处于等待当期进程退出的状态，则将父进程唤醒    如果当前进程有子进程，则将子进程设置为initproc的子进程    选择新的进程去执行     1. 请分析fork/exec/wait/exit在实现中是如何影响进程的执行状态的？  * fork将创建新的子线程，将子线程的状态由UNINIT态变为RUNNABLE态，不改变父进程的状态 * exec完成用户进程的创建工作，同时使用户进程进入执行，不改变进程状态 * wait完成子进程资源回收，如果有已经结束的子进程或者没有子进程，那么调用会立刻结束，不影响进程状态；否则，进程需要等待子进程结束，进程从RUNNIG态变为SLEEPING态。 * exit完成对资源的回收，进程从RUNNIG态变为ZOMBIE态。  1. 请给出ucore中一个用户态进程的执行状态生命周期图（包括执行状态，执行状态之间的变换关系，以及产生变换的事件或函数调用）   wps | | | | | | |
| 五、组内分工  20204227陈鹏宇：完成练习一二的设计、代码、测试与报告撰写  20204137徐小龙：完成练习三的设计、代码、测试与报告撰写 | | | | | | |