|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **武汉大学国家网络安全学院课程实验报告** | | | | | | | |
| 课程名称 | 操作系统原理 | | | | | | |
| 实验名称 | 创建线程与跟踪进程 | | | 实验序号 | 4 | 实验日期 | 20.03.22 |
| 姓 名 | 沈思源 | 学 号 | 2018302070001 | 专 业 | 信息安全 | 年级-班 | 2018级-2班 |
| 1. 实验目的及实验内容   学习fork函数， exec函数和pthread函数的使用，阅读源码，分析三个函数的机理  代码实现   1. 进程A 创建子进程B 2. 子进程B与父进程A分别对应不同的exe(进程A打印Hello word, 进程B实现sum累加 3. 进程B具有两线程，其中主线程创建新的线程来实现sum累加（从1累加到参数x)   分析运行中各个执行体的处理器使用、内存使用等基本信息 | | | | | | | |
| 【实验内容】   1. 学习fork函数， exec函数和pthread函数的使用和运行机理 2. 创建程序实现实验要求 | | | | | | | |
| 1. 实验步骤及结果分析   （本次实验所使用的软件、工具的情况；具体的实验步骤） | | | | | | | |
| 实验环境  Ubuntu 18.04.2虚拟机环境Oracle VM VirtualBox  工具: gcc7.0编译器，vim  实验步骤:   1. 编写Hello world程序用于进程调用     GCC编译器编译并进行测试     1. 编写主体程序框架     Main()函数主体框架:  定义一个Pid变量，创建子进程B，根据Pid值确定进程的父子关系，根据是A还是B进程来分别设置不同的功能。  当pid<0时，fork()失败，打印错误信息，返回错误状态码，结束  当pid==0时，表示当前是B(子进程)，根据实验要求设置进程功能  当pid>0时，表示当前是A(父进程)，设置相应功能.  3、设置A进程打印Hello World!    系统调用Wait(NULL)，这里用于等待子进程B结束  子进程B结束后，调用execlp加载二进制文件hello到内存中执行，破坏原来程序对应的内存内容。如果加载失败，打印错误信息。  4、设置B进程，创建两个线程计算求和    睡眠10s，使进程在系统中存在一定的时间。  5、编写求和函数，用于线程运行    6、编写Makefile文件，默认测试求和1-100    7、编译生成文件测试    8、分析各个进程的处理器和内存使用情况  启动程序，然后将进程挂起    开启另一个Shell,根据进程信息查询子进程B的资源利用状态      可以看到，子进程B的Pid是4154，父进程A的pid是4153  进而查询父进程A的状态    根据说明文档信息  Name:       gedit             /\*进程的程序名\*/ State:   S(sleeping) /\*进程的状态信息\*/ Tgid:         9744            /\*线程组号\*/ Pid:           9744            /\*进程pid\*/ PPid:         7672           /\*父进程的pid\*/ TracerPid: 0                 /\*跟踪进程的pid\*/ Uid:           1000    1000    1000    1000       /\*uid euid suid fsuid\*/ Gid:           1000    1000    1000    1000       /\*gid egid sgid fsgid\*/ FDSize:     256             /\*文件描述符的最大个数，file->fds\*/ Groups:     0 4 20 24 25 29 30 44 46 107 109 115 124 1000     /\*启动该进程的用户所属的组的id\*/ VmPeak:   60184 kB     /\*进程地址空间的大小\*/ VmSize:    60180 kB     /\*进程虚拟地址空间的大小reserved\_vm：进程在预留或特殊的内存间的物理页\*/ VmLck:     0 kB             /\*进程已经锁住的物理内存的大小.锁住的物理内存不能交换到硬盘\*/ VmHWM: 18020 kB      /\*文件内存映射和匿名内存映射的大小\*/ VmRSS:   18020 kB     /\*应用程序正在使用的物理内存的大小，就是用ps命令的参数rss的值 (rss)\*/ VmData:   12240 kB    /\*程序数据段的大小（所占虚拟内存的大小），存放初始化了的数据\*/ VmStk:      84 kB         /\*进程在用户态的栈的大小\*/ VmExe:     576 kB       /\*程序所拥有的可执行虚拟内存的大小,代码段,不包括任务使用的库 \*/ VmLib:      21072 kB   /\*被映像到任务的虚拟内存空间的库的大小\*/ VmPTE:    56 kB         /\*该进程的所有页表的大小\*/ Threads:   1                /\*共享使用该信号描述符的任务的个数\*/ SigQ:        0/8183       /\*待处理信号的个数/目前最大可以处理的信号的个数\*/ SigPnd:    0000000000000000 /\*屏蔽位，存储了该线程的待处理信号\*/ ShdPnd:   0000000000000000 /\*屏蔽位，存储了该线程组的待处理信号\*/ SigBlk:      0000000000000000 /\*存放被阻塞的信号\*/ SigIgn:      0000000000001000 /\*存放被忽略的信号\*/ SigCgt:     0000000180000000 /\*存放被俘获到的信号\*/ CapInh:    0000000000000000 /\*能被当前进程执行的程序的继承的能力\*/ CapPrm:   0000000000000000 /\*进程能够使用的能力，可以包含CapEff中没有的能力，这些能力是被进程自己临时放弃的\*/ CapEff:     0000000000000000 /\*是CapPrm的一个子集，进程放弃没有必要的能力有利于提高安全性\*/ Cpus\_allowed:    01            /\*可以执行该进程的CPU掩码集\*/ Mems\_allowed:   1              /\*\*/ voluntary\_ctxt\_switches:    1241      /\*进程主动切换的次数\*/ nonvoluntary\_ctxt\_switches:   717   /\*进程被动切换的次数\*/    9、分析函数机理  系统调用: fork()；  fork系统调用用于创建一个新进程，称为子进程，它与进程（称为系统调用fork的进程）同时运行，此进程称为父进程。创建新的子进程后，两个进程将执行fork（）系统调用之后的下一条指令。子进程使用相同的pc（程序计数器），相同的CPU寄存器，在父进程中使用的相同打开文件。  它不需要参数并返回一个整数值。下面是fork（）返回的不同值。  负值：创建子进程失败。  　　 零：返回到新创建的子进程。  　　 正值：返回父进程或调电者。该值包含新创建的子进程的进程pid(进程表示符)  头文件: #include<unistd.h>/\*#包含<unistd.h>\*/  #include<sys/types.h>/\*#包含<sys/types.h>\*/  fork调用的一个特殊之处就是它仅仅被调用一次，却能够返回两次，它可能有三种不同的返回值：因为调用后有了两个进程，返回时两个进程都有返回值  　　(1)在父进程中，fork返回新创建子进程的进程ID；  (2)在子进程中，fork返回0；  (3)如果出现错误，fork返回一个负值。  函数原型:  pid\_t fork( void);  （pid\_t 是一个宏定义，其实质是int 被定义在#include<sys/types.h>中）  Exec()函数族；  execlp()函数说明  函数说明：  execlp()会从PATH 环境变量所指的目录中查找符合参数file的文件名，找到后便执行该文件，然后将第二个以后的参数当做该文件的argv[0]、argv[1]……，最后一个参数必须用空指针(NULL)作结束。如果用常数0来表示一个空指针，则必须将它强制转换为一个字符指针，否则它将解释为整形参数，如果一个整形数的长度与char \* 的长度不同，那么exec函数的实际参数就将出错。如果函数调用成功,进程自己的执行代码就会变成加载程序的代码,execlp()后边的代码也就不会执行了.  头文件：  #include<unistd.h>  定义函数：  int execlp(const char \* file,const char \* argv,....,NULL);  Pthread()函数族  头文件 #include<pthread.h> 链接编译参数 -l pthread  Pthread\_t sum\_id //声明创建一个线程标识符变量  Pthread\_attr\_t attr; //设置一种线程属性变量  Pthread\_attr\_init(); //用于设置线程属性变量  Pthread\_attr\_init(&attr);  Pthread\_create()  pthread\_create是类Unix操作系统（Unix、Linux、Mac OS X等）的创建线程的函数。它的功能是创建线程（实际上就是确定调用该线程函数的入口点），在线程创建以后，就开始运行相关的线程函数。  pthread\_create的返回值 表示成功，返回0；表示出错，返回-1。  函数声明  int pthread\_create(pthread\_t \*tidp,const pthread\_attr\_t \*attr,(void\*)(\*start\_rtn)(void\*),void \*arg);  参数  第一个参数为指向线程标识符的指针。  第二个参数用来设置线程属性。  第三个参数是线程运行函数的起始地址(函数名称)  最后一个参数是创建的线程要运行函数的参数。  运行函数声明  Void \* function\_name(void\*param)；  Pthread\_join()  函数pthread\_join用来等待一个线程的结束,线程间同步的操作。  头文件 ： #include <pthread.h>  函数定义： int pthread\_join(pthread\_t thread, void \*\*retval);  描述 ：pthread\_join()函数，以阻塞的方式等待thread指定的线程结束。当函数返回时，被等待线程的资源被收回。如果线程已经结束，那么该函数会立即返回。并且thread指定的线程必须是joinable的。  参数 ：thread: 线程标识符，即线程ID，标识唯一线程。retval: 用户定义的指针，用来存储被等待线程的返回值。  返回值 ： 0代表成功。 失败，返回的则是错误号。  Pthread\_exit();  终止进程 | | | | | | | |
| 1. 总结 2. 在操作系统种，进程是资源分配单位，线程是CPU调度单位 3. 合理利用多线程编程可以增强响应性 4. Pthreads线程API能够便捷的创建多线程，在创建多线程和进程时要合理运用 5. 进程跟踪可以通过挂起和Linux资源管理工具进行 | | | | | | | |
|  | | | | | | | |