山东大学 软件 学院

**操作系统** 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202000300125 | 姓名：贾星宇 | | 班级：2020级5班 |
| 实验编号：实验一 进程控制实验 | | | |
| 实验题目：进程控制实验 | | | |
| 实验学时：2 | | 实验日期：2021.3.29 | |
| 实验目的：  加深对于进程并发执行概念的理解。实践并发进程的创建和控制方法。观察和  体验进程的动态特性。进一步理解进程生命期期间创建、变换、撤销状态变换的过  程。掌握进程控制的方法，了解父子进程间的控制和协作关系。练习 Linux 系统中  进程创建与控制有关的系统调用的编程和调试技术。 | | | |
| 硬件环境：  宿主机：lntel(R) Core(TM) i5-10210U CPU @ 1.60G Hz 2 .1 1 G Hz  虚拟机：ubuntu-20.04.1-desktop-amd64 | | | |
| 软件环境：  宿主机：Windows 10 家庭中文版  虚拟机：Code::Blocks | | | |
| 实验步骤与内容：  1.新建文件夹text1，利用codeblocks编辑器编写文件main.c,main.h以及Makefile：        利用终端输入命令：  gcc -g -c main.c  gcc main.o -o main  ./main  运行得到结果：    以上程序的输出说明父进程 51725创建了一个子进程 51726，子进程执行被暂 停。父进程向子进程发出键盘中断信号唤醒子进程并与子进程并发执行。 父进程并没有等待子进程的结束继续执行先行结束了（此时的子进程成为了 孤儿进程，不会有父进程为它清理退出状态了）。而子进程继续执行，它变 成了列出当前目录所有文件名的命令 ls -a。在完成了列出文件名命令之 后，子进程的执行也结束了。此时子进程的退出状态将有初始化进程为它 清理。  再次运行，得到另一个结果：    独立实验：  父进程创建一个子进程并控制它每隔 3 秒显示一次当前目录中的文件名列表。  核心代码：  【核心思想：利用while循环父进程和子进程，子进程执行一次后继续pause，父进程继续发给子进程中断信号以让子进程再次执行ls命令；同时，子进程应使用system（“ls”）替换execve（）】        执行情况：    如图，每隔 3 秒子进程显示一次当前目录中的文件名列表。 | | | |
| 结论分析与体会：  【分析整个程序的运行过程】  首先父进程开始执行，执行到pid=fork（）；函数后，对父进程来说，pid=子进程的进程号，对子进程来说其pid=0；  （进程可以通过系统调用fork()创建子进程并和其子进程并发执行.**子进程初始的执行映像是父进程的一个复本.**子进程可以通过 exec()系统调用族装入一个新 的执行程序。父进程可以使用 wait()或 waitpid()系统调用等待子进程的结束并负 责收集和清理子进程的退出状态。  还有人可能疑惑为什么不是从#include处开始复制代码的，这是因为fork是把进程当前的情况拷贝一份，执行fork时，进程已经执行完了int count=0;fork只拷贝下一个要执行的代码到新的进程。  引用一位网友的话来解释fpid的值为什么在父子进程中不同。“其实就相当于链表，进程形成了链表，父进程的fpid(p 意味point)指向子进程的进程id, 因为子进程没有子进程，所以其fpid为0.”  ）  【父进程】因为子进程没有及时抢占父进程等可能原因，父进程首先进入最后一个else，打印了第一句话。此时子进程被fork函数调用了起来。其初始的执行映像是父进程的一个复本，而且从fork向后开始执行：  【子进程】  对子进程来说pid=fork导致的pid为0；随后打印“我是子进程”，然后被pause。  pause 挂起调用它的进程直到有任何信号到达。调用进程不自 定义处理方法，则进行信号的默认处理。只有进程自定义了信号处理方法 捕获并处理了一个信号后，pause 才会返回调进程。pause 总是返回-1,并设 置系统变量 errno 为 EINTR。  所以子进程处于挂起状态，父进程可以利用CPU并开始执行。  【父进程】  父进程执行而且argv【1】为空，所以首先sleep 3秒，随后发出信号kill（pid，SIGINT），而且pid就是子进程的pid。因此子进程收到了信号，pause失效，子进程继续执行。  对父进程来说，它亦继续向后执行而且打印“唤醒子进程”以及“不等待子进程结束”，随后进入下一次循环：打印“我是父进程”，并开始三秒的sleep  【子进程】  对子进程来说，它继续执行并打印“子进程将会执行”、命令以及执行命令（利用system（“ls”）,利用此函数不会导致子进程被覆盖，因此保证了子进程可以继续下一次循环  开始子进程下一次循环，执行“我是子进程”后被暂停，等待父进程sleep三秒后被中断信号唤醒，继续执行ls命令  至此，实现了每隔三秒父进程控制子进程ls的命令。  【反思记录】  进程的特征：  1.动态性，进程是程序的一次执行，有创建、活动、暂停、终止等过程。  2.并发性，多个进程实体同存于内存中，能在一段时间内同时运行。  3.独立性，进程实体是一个能独立运行独立获得资源和独立接受调度的基本单位  4.异步性，各进程按各自独立的不可预知的速度向前推进  如何控制进程：  fork（）函数创建一个子进程并行执行  Exec 执行成功后将用一个新的程序代替原进程，但进程号不变，它绝不会再  返回到调用进程了。  wait 和 waitpid 执行成功将返回终止的子进程的进程号，不成功返回-1。  getpid 返回当前进程的进程号，getppid 返回当前进程父进程的进程号kill能够发送除杀死一个进程(SIGKILL、SIGTERM、  SIGQUIT) 之外的其他信号，例如键盘中断(Ctrl+C)信号 SIGINT,进程暂停(Ctrl+Z)信号SIGTSTP 等等。调用 Pause 函数会令调用进程的执行挂起直到一个任意信号到来后再继  续运行。  调用 sleep 函数会令调用进程的执行挂起睡眠指定的秒数或一个它可以  响应的信号到来后继续执行。  进程的并发：  进程的并发性是指一组进程的执行在时间上是重叠的，重叠是指一个进程执行的第一条指令是在另一个进程执行的最后一条指令完成之前开始的  子进程创建和执行新程序：  子进程可以通过 exec()系统调用族装入一个新 的执行程序。父进程可以使用 wait()或 waitpid()系统调用等待子进程的结束并负 责收集和清理子进程的退出状态。  信号机理及实现进程控制：  整形信号量（经典的PV操作：P(wait)：申请资源 V(signal)：释放资源）  记录型信号量  AND型信号量  信号量集  SIGINT：ctrl+c 终止信号  　　SIGQUIT：ctrl+\ 终止信号  　　SIGTSTP:ctrl+z 暂停信号  　　SIGALRM：闹钟信号 收到此信号后定时结束，结束进程  　　SIGCHLD：子进程状态改变，父进程收到信号  　　SIGKILL：杀死信号  1）int kill(pid\_t pid, int sig);  　　功能：信号发送  　　参数：pid：指定进程  　　sig：要发送的信号  　　返回值：成功 0；失败 -1  2）int raise(int sig);  　　功能：进程向自己发送信号  　　参数：sig：信号  　　返回值：成功 0；失败 -1  3）unsigned int alarm(unsigned int seconds)  　　功能：在进程中设置一个定时器  　　参数：seconds：定时时间，单位为秒  　　返回值：如果调用此alarm()前，进程中已经设置了闹钟时间，则返回上一个闹钟时间的剩余时间，否则返回0。  注意：一个进程只能有一个闹钟时间。如果在调用alarm时已设置过闹钟时间，则之前的闹钟时间被新值所代替  4）int pause(void);  　　功能：用于将调用进程挂起直到收到信号为止。  5）void (\*signal(int signum, void (\*handler)(int)))(int);  　　或者：  　　typedef void (\*sighandler\_t)(int);  　　sighandler\_t signal(int signum, sighandler\_t handler);  　　功能：信号处理函数  　　参数：signum：要处理的信号//不能是SIGKILL和SIGSTOP  　　　　handler：SIG\_IGN：忽略该信号。  　　　　SIG\_DFL：采用系统默认方式处理信号。  　　　　自定义的信号处理函数指针  　　返回值：成功：设置之前的信号处理方式；失败：SIG\_ERR  6）void abort(void);  　　给自己发送异常终止信号，（6.SIGABRO）终止并产生core文件。 | | | |