附件（四）

**深 圳 大 学 实 验 报 告**

**课程名称： 操作系统**

**实验名称： 并发程序设计**

**学院： 计算机与软件学院**

**专业： 计算机科学与技术**

**指导教师： 张 滇**

**报告人： 缪克达 学号：2020274045 班级： 二班**

**实验报告提交时间： 2023年2月26日星期日**

**教务处制**

|  |
| --- |
| **一、实验目的与要求：**   1. 掌握在linux中编译运行的方法。 2. 阅读例程，理解函数fork()、execl()、exit()、getpid()和waitpid()的功能和用法 3. 运行例程，分析例程中关键代码的功能，给出运行结果并对运行结果进行分析说明。 4. 模仿例程，编写一段程序实现以下功能：    1. 使用系统调用fork()创建两个子进程    2. 各个子进程显示和输出一些提示信息和自己的进程标识符。    3. 父进程显示自己的进程ID和一些提示信息，然后调用waitpid()等待多个子进程结束，并在子进程结束后显示输出提示信息表示程序结束。 5. 创建多个（3个以上）进程并发运行，控制好各个子进程输出自己的进程标识符和一些提示信息，对程序运行结果进行分析说明。观察各个子进程并发执行的顺序，输出结果是否与设想中的顺序不同，并分析原因。 |
| **二．实验过程及内容：**(对程序代码进行说明和分析，越详细越好，代码排版要整齐，可读性要高)  **1. 编写一个最简单的程序，检查并发环境是否正确**  鉴于clang的优越性，我们将采用llvm的标准来编译程序，首先确保环境配置正确，clang编译器是否正确的安装在Linux系统上。  输入bash指令$ clang test.c -o test，然后执行：    可以发现我们的环境配置是没有问题的，可以进入下一步了。   1. **运行例程，分析例程中关键代码的功能，给出运行结果并对运行结果进行分析说明。**   **首先键入例程1的代码：**    上面的代码简单来说，实现了父进程/子进程之间的用户级别的进程复制，并使用waitpid()函数让父进程不断等待其子进程退出。tprintf函数用于替换常规的printf，功能是在普通的格式字符串fmt前面增加显示当前的时间。getpid函数在main中用于获取正在执行当前程序的进程ID，而fork就是创建子进程，其数据拷贝自当前进程，父进程结束时，子进程也会相应结束。  为了弄清楚这个程序究竟做了什么，我们需要用到lldb去debug。首先是用clang编译程序，但这个时候需要增加参数-g，这样允许我们在debug的时候设置断点：    接下来键入$ lldb main进入debug：    设置在fork之后的断点，然后查看当前pid的值：    可以发现，在fork之后（也就是创建了子进程之后），pid = 83228，这说明创建的子进程代码为83228，而父进程的代码为83225. 并且，从输出的提示来说，创建的子进程并没有从main的开头执行，而是从fork()函数后开始执行。所以并没有打印”Hello from Parent Process, PID is 83228.”的输出。这也很好的印证了这一点。  由于我们设置的断点是在父进程，所以无法具体观测到子进程是一种什么样的情况。但是幸运的是可以通过子进程的输出来判断。从输出：  Hello from Child Process 83228. 1 times  Hello from Child Process 83228. 2 times  Hello from Child Process 83228. 3 times  可以看出来，子进程是从fork后开始执行的，并且在子进程中，pid是等于0的。这是因为在C语言中，pid等于0通常是在fork()系统调用返回的子进程中使用的。这是因为在子进程中，fork()系统调用的返回值是0。当父进程调用fork()创建子进程时，fork()会返回子进程的进程ID（PID），而在子进程中，fork()会返回0。这样，我们可以根据返回值判断当前进程是父进程还是子进程。因此，**当pid等于0时，通常表示当前代码正在子进程中执行。**  当子进程结束之后，父进程也将跳出等待，而结束程序。因此最终的输出也非常简单（忽略PID的改变，因为我重新运行了程序）：    接下来我们看看例程2的程序：    这段代码其实是例程1的修改，最主要的修改是在子进程添加了调用系统二进制程序ps。ps的作用是查看进程。这个在Linux开发中非常常用，我们一般会使用ps -ef来输出当前系统的所有进程，并且这种输出是比较规范的，下面以我的Linux系统为例子展示了ps的最基础的使用：    如果要筛选出特地的进程，可以使用管道grep，这里展示筛选出wine进程的ps指令：    了解了ps之后，我们就可以大概看看这个程序的执行结果了。因为比较简单，大部分功能在例程1已经debug过了，所以例程2不需要debug了。我们来看看执行结果：    我们发现，在执行ps之后，输出了：    其实就是执行了ps -a，那为什么要调用/bin/ps -a呢？因为我们这个程序并没有引入环境变量。-a这个参数，表示输出当前终端的所有进程。这就说明为什么输出的进程那么少了，只有zsh shell（bash的平替，语法一样）、concurrent\_prog（当前的程序）和ps。  另外，如果执行的不是ps，而是ls，那么输出如下：    上述的输出等效于在bash(zsh) shell中输入ls -l /etc。结果是一样的。   1. **模仿例程，编写一段程序实现以下功能：**   **a)使用系统调用fork()创建两个子进程**  **b)各个子进程显示和输出一些提示信息和自己的进程标识符。**  **c)父进程显示自己的进程ID和一些提示信息，然后调用waitpid()等待多个子进程结束，并在子进程结束后显示输出提示信息表示程序结束。**  我们在原有程序的基础上进行修改。首先是main函数，为了创建两个线程，我们用一个相对特殊的创建方式，这种创建方式并不会显式的定义两个pid，而是一种可以用一个pid\_t变量表示无穷个进程代码的方式（理论上），这种创建方式的伪代码如下：  # Algorihtm：Create a child process  for i = 1 to n:  pid = CreateAChildProcess();  if pid is Child Or Error:  break;  这个创建子进程的算法的空间复杂度为，无论有多少个子进程被创建。而传统的创建方式需要的空间复杂度。  具体用C语言实现如下：    特别注意的是，当当前线程是子线程的时候，就跳出循环而执行输出子线程的相关信息及其标识符。所以exec函数具体如下：    我们尝试用clang编译然后运行，得到的结果如下：     1. **创建多个（3个以上）进程并发运行，控制好各个子进程输出自己的进程标识符和一些提示信息，对程序运行结果进行分析说明。观察各个子进程并发执行的顺序，输出结果是否与设想中的顺序不同，并分析原因。**   进程的创建有三种：直接创建、嵌套创建和二叉树创建，如图所示：  20210316154625413   * 1. 直接创建   直接创建是指所有的子进程都是由一个父进程创建而来的：    这种创建方式最简单，一般用我们刚刚的Algorithm：Create a child process算法就可以创建出来，代码如下：    这样，创建所有子进程的父亲都是父进程了。我们用clang编译运行看看：    **我们发现，不管设定的顺序是什么样的，最终这个程序的所有子进程都是随机顺序执行的。并没有按照我们的预期。**  为了验证是否是直接创建，那么就要验证每个子进程的父亲是不是父进程。Linux中有个工具叫做pstree，这个工具可以以树的形式输出进程及其子进程的关系，非常适合验证。只需要输入pstree -p 父进程代号即可。另外，为了方便使用这个指令，我们有比要将程序放到后台执行，因此我们可以使用&参数，Linux会把这个运行的程序放到后台执行，于是我们可以继续输入bash指令了。  并且在结尾加上pause()，这样程序就不会暂停了。方便我们去使用pstree。  另外，我们要在源代码中删掉waitpid()函数，因为这个函数会使得第一个子进程结束时父进程才能继续创建子进程。这说明如果不删掉waitpid()函数的话，由于pause的作用，整个程序都不会停止。这样父进程就会永远等待第一个子进程执行完毕，导致父进程永远只有一个子进程。  稍作修改，得到如下的版本：    输入  $ clang concurrent\_program\_4\_1.c -o concurrent\_program\_4\_1; ./concurrent\_program\_4\_1 &  然后得到如下的输出结果：    可以知道当前的父进程是38368，于是我们输入$ pstree -p 38368：    可以看出来，我们的创建方式是正确的。  由于我们有意让程序不终止，所以当查看完进程结构后，需要我们手动杀死这些没有用的进程。输入$ ps -ef | grep concurrent\_prog：    可以看到有很多进程，所以需要杀死，利用$ kill $(pgrep concurrent)指令杀死含有concurrent的进程。之后再次查看进程：    成功了杀死了无用进程。   * 1. 嵌套创建   我们在4.1实现了直接创建，而嵌套创建的含义如下：    也就是说，每个子进程都是上一个子进程的孩子，其第一个子进程的父亲是父进程。实现这种方式很简单，只需要将创建语句中的break放到父进程里面，这样就可以让子进程继续创建进程，而父进程创建完一个子进程后将不再创建子进程。    我们键入  $ clang concurrent\_program\_4\_2.c -o concurrent\_program\_4\_2; ./concurrent\_program\_4\_2 &  然后看看输出结果：    接下来验证是否为嵌套创建：$ pstree -p 40579    因为输出空间不够了，所以后面省略了。可以看得出来，这就是嵌套创建。   * 1. 二叉树创建   二叉树创建，也就是让每个进程都创建两个子进程。既然如此，我们可以继续修改for里面的代码，得到如下简单的创建代码：    一共三层循环，创建四层满二叉树结构的进程树。  其原理是，第一层循环内，创建了子进程X，然后因为当前是父进程，所以此时pid > 0，因此进入if语句，再创建了一个子进程Y，此时pid还是大于0，于是跳出循环，父进程不再创建子进程。第二层循环内，原先创建好的两个子进程X和Y分别执行第二层循环里的操作，所以我只举其中一个子进程X的例子，这个子进程X继续创建下一个子进程Z，但是因为代码是在子进程运作，所以此时pid等于0，于是进入第三层循环，第三层循环中子进程X又创建了一个进程W。所以子进程X创建了两个子进程Z和W，而与X互为兄弟的子进程Y也是同理。子进程Z和W创建后就从第二层循环开始，也是同理。最终这样形成了一个满二叉树结构的进程树。  我们键入  $ clang concurrent\_program\_4\_3.c -o concurrent\_program\_4\_3; ./concurrent\_program\_4\_3 &  然后看看输出结果：    然后用$pstree -p 45244看看结构：    可以看出来是二叉树创建。 |
| **三、实验结论：**（提供运行结果，对结果进行探讨、分析、评价，并提出结论性意见和改进想法）   1. 进程的运行顺序是完全随机的，这完全基于OS的考虑而设定的。并发控制的顺序是我们难以预测的。 2. 利用fork函数可以创建该进程（父进程）的子进程，在父进程调用fork得到的进程代号是大于0的，而在子进程中调用fork函数得到的是等于0的。所以可以利用这个特性去判断当前进程是否是子进程。需要注意的是，子进程调用fork函数也可以创建子进程，即便fork的调用结果是0。 3. 进程的创建大致有三种创建方式：直接创建、嵌套创建和二叉树创建。这三种创建的方式可以利用pstree -p 进程号来查看进程树。 4. 杀死进程的方式很简单，首先需要ps aux | grep 进程所在的程序名称来获取进程，然后kill指令杀死即可。或者可以采用kill $(pgrep concurrent)直接杀死也可以。concurrent是程序名称的前缀，不同程序不一样，这点需要注意查看。   **四、实验体会：**（根据自己情况填写）   1. 更加熟悉Linux环境，并且有将Linux作为主力系统的打算。 2. 基本掌握了ps指令、ls指令、kill指令。 3. 更加熟悉使用clang来编译C语言程序，其语法与gcc差不多，但更高级且高效。同理，我也更加熟悉了lldb调试器。 4. 初步了解了进程的概念，也学会使用fork函数、getpid函数、waitpid函数、exit函数和execl函数。   注：“指导教师批阅意见”栏请单独放置一页 |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  年 月 日 |
| 备注： |