lab5

编程题

1.实现一个使用nice,fork,exec,spawn等与进程管理相关的系统调用的linux应用程序。

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(void)
{
   int childpid;  // 存储子进程的进程ID
   int i;         // 未使用的整型变量i

   if (fork() == 0)  // 子进程
   {
         char * execv_str[] = {"echo", "child process, executed by execv",NULL};
 // 定义一个字符串数组，用于存储要通过execv()执行的命令和参数
         if (execv("/usr/bin/echo",execv_str) <0 ){  // 使用execv()函数执行指定的命令（这里是echo），若返回值小于0表示执行出错
            perror("error on exec\n");  // 打印错误信息
            exit(0);  // 退出子进程
         }
   }
   else  // 父进程
   {
         wait(&childpid);  // 父进程等待子进程结束，使用wait()函数，并通过&childpid参数存储子进程的退出状态
         printf("parent process, execv done\n");  // 子进程执行完毕后，父进程打印消息
   }
   return 0;
}
```

输出结果为：

```
child process, executed by execv
parent process, execv done
```

2.请阅读下列代码，分析程序的输出 `A` 的数量：（已知 `&&` 的优先级比 `||` 高）

```
int main() {
    fork() && fork() && fork() || fork() && fork() || fork() && fork();
    printf("A");
    return 0;
}
```

22个。

由于&&比||优先级高于是可以使用||作为分割，可以分成三部分，第一部分3个fork，第二、三部分2个fork。由于&&和||的特性决定每部分的可能情况为：如果有n个fork，则每个fork的真假情况为：F、TF、TTF……(n-1)TF、n个T，共n+1种情况。同时当全是T时候便不再执行下一部分。于是可以分成全 T（1种情况）和非全 T（n种情况）两类，然后逐部分相乘想加，就变成了
`1+3*（1+2*（1+2））= 22` 也可以写成`1+3*1+3*2*1+3*2*2 = 22`。
按照此思路可以写出计算程序，首先以||截取子串，然后计算每个部分fork数量然后依次相乘相加。&&优先级高于||，根据fork子进程返回值为0父进程返回pid和逻辑运算符的短路现象（&&左边为F即短路，||左边为T即短路）， 可以按||分割来进行判断，共`1+1*3+3*2+3*2*2=22`.

问答题

1.如何查看Linux操作系统中的进程?

使用ps命令，常用方法：

```
$ ps aux
```

2.简单描述一下进程的地址空间中有哪些数据和代码。

代码(text)段，数据(data)段： 已初始化的全局变量的内存映射，bss段：未初始化或默认初始化为0的全局变量，堆(heap)，用户栈(stack)，共享内存段

3.进程控制块保存哪些内容?

进程标识符、进程调度信息（进程状态，进程的优先级，进程调度所需的其它信息）、进程间通信信息、内存管理信息（基地址、页表或段表等存储空间结构）、进程所用资源（ I/O 设备列表、打开文件列表等）、处理机信息（通用寄存器、指令计数器、用户的栈指针）

4.进程上下文切换需要保存哪些内容?

页全局目录、部分寄存器、内核栈、当前运行位置

5.fork 为什么需要在父进程和子进程提供不同的返回值?

可以根据返回值区分父子进程，明确进程之间的关系，方便用户为不同进程执行不同的操作。

6.fork + exec 的一个比较大的问题是 fork 之后的内存页/文件等资源完全没有使用就废弃了，针对这一点，有什么改进策略?

采用COW(copy on write),或使用使用vfork等。

7.其实使用了6的策略之后，fork + exec 所带来的无效资源的问题已经基本被解决了，但是近年来fork 还是在被不断的批判，那么到底是什么正在"杀死"fork？可以参考 论文 。

fork 和其他的操作不正交,也就是 os 每增加一个功能,都要改 fork, 这导致新功能开发困难,设计受限.有些和硬件相关的甚至根本无法支持 fork.

fork 得到的父子进程可能产生共享资源的冲突;

子进程继承父进程，如果父进程处理不当，子进程可以找到父进程的安全漏洞进而威胁父进程;

还有比如 fork 必须要虚存, SAS 无法支持等等.

8.请阅读下列代码，并分析程序的输出，假定不发生运行错误，不考虑行缓冲，不考虑中断：

```
int main(){
    int val = 2;

    printf("%d", 0);
    int pid = fork();
    if (pid == 0) {
        val++;
        printf("%d", val);
    } else {
        val--;
        printf("%d", val);
        wait(NULL);
    }
    val++;
    printf("%d", val);
    return 0;
}
```

如果 fork() 之后主程序先运行，则结果如何？如果 fork() 之后 child 先运行，则结果如何？

01342 03412

9.为什么子进程退出后需要父进程对它进行 wait，它才能被完全回收?

当一个进程通过exit系统调用退出之后，它所占用的资源并不能够立即全部回收，需要由该进程的父进程通过wait收集该进程的返回状态并回收掉它所占据的全部资源，防止子进程变为僵尸进程造成内存泄漏。同时父进程通过wait可以获取子进程执行结果，判断运行是否达到预期，进行管理。

10.有哪些可能的时机导致进程切换?

进程主动放弃cpu：运行结束、调用yield/sleep等、运行发生异常中断

进程被动失去cpu：时间片用完、新进程到达、发生I/O中断等

11.请描述在本章操作系统中实现本章提出的某一种调度算法（RR调度除外）的简要实现步骤。

可降低优先级的MLFQ：将manager的进程就绪队列变为数个，初始进程进入第一队列，调度器每次选择第一队列的队首进程执行，当一个进程用完时间片而未执行完，就在将它重新添加至就绪队列时添加到下一队列，直到进程位于底部队列。

12.非抢占式的调度算法，以及抢占式的调度算法，他们的优点各是什么?

非抢占式：中断响应性能好、进程执行连续，便于分析管理

抢占式：任务级响应时间最优，更能满足紧迫作业要求

13.假设我们简单的将进程分为两种：前台交互（要求短时延）、后台计算（计算量大）。下列进程/或进程组分别是前台还是后台？a) make 编译 linux; b) vim 光标移动; c) firefox 下载影片; d) 某游戏处理玩家点击鼠标开枪; e) 播放交响乐歌曲; f) 转码一个电影视频。除此以外，想想你日常应用程序的运行，它们哪些是前台，哪些是后台的?

前台：b,d,e

后台：a,c,f

14.RR 算法的时间片长短对系统性能指标有什么影响?

时间片太大，可以让每个任务都在时间片内完成，但进程平均周转时间会比较长，极限情况下甚至退化为FCFS；

时间片过小，反应迅速，响应时间会比较短，可以提高批量短任务的完成速度。但产生大量上下文切换开销，使进程的实际执行时间受到挤占。

因此需要在响应时间和进程切换开销之间进行权衡，合理设定时间片大小。

15.MLFQ 算法并不公平，恶意的用户程序可以愚弄 MLFQ 算法，大幅挤占其他进程的时间。（MLFQ 的规则：“如果一个进程，时间片用完了它还在执行用户计算，那么 MLFQ 下调它的优先级”）你能举出一个例子，使得你的用户程序能够挤占其他进程的时间吗?

每次连续执行只进行大半个时间片长度即通过执行一个IO操作等让出cpu，这样优先级不会下降，仍能很快得到下一次调度。

16.多核执行和调度引入了哪些新的问题和挑战?

多处理机之间的负载不均问题：在调度时，如何保证每一个处理机的就绪队列保证优先级、性能指标的同时负载均衡

数据在不同处理机之间的共享与同步问题：除了Cache一致性的问题，在不同处理机上同时运行的进程可能对共享的数据区域产生相同的数据要求，这时就需要避免数据冲突，采用同步互斥机制处理资源竞争；

线程化问题：如何将单个进程分为多线程放在多个处理机上

Cache一致性问题：由于各个处理机有自己的私有Cache，需要保证不同处理机下的Cache之中的数据一致性

处理器亲和性问题：在单一处理机上运行的进程可以利用Cache实现内存访问的优化与加速，这就需要我们规划调度策略，尽量使一个进程在它前一次运行过的同一个CPU上运行，也即满足处理器亲和性。

通信问题：类似同步问题，如何降低核间的通信代价