实验七 ：进程控制和通信

目录

一/二.实验目的与要求

1．完善实验 6 中的二状态进程模型，实现五状态进程模型，从而使进程可以分工合作，并发运行。
2．了解派生进程、结束进程、阻塞进程等过程中父、子进程之间的关系和分别进行的操作。
3．理解原语的概念并实现进程控制原语 do_fork()，do_exit()，do_wait()，wakeup，blocked。

三.实验方案

1.实验工具与环境（gcc+nasm+ld）

nasm 编译汇编代码生成.o 文件，gcc 编译 C 模块生成.o 文件；ld 链接.o 文件生成.bin 文件；dd 将.bin 写入映像文件.img；vmware 上添加.img 文件到软盘后

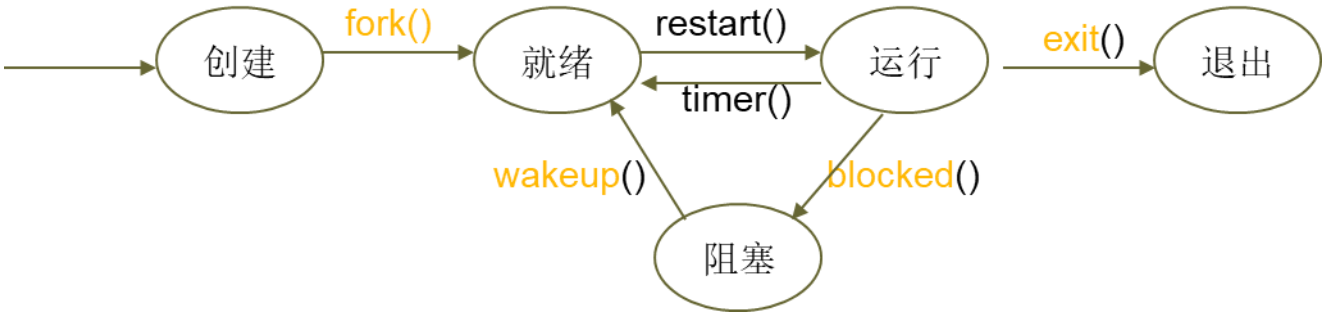
运行。

2.方案的思想

1.相关原理：

①关于五状态模型：

如图，进程可以分为5个状态，创建态，就绪态，运行态，阻塞态和退出态。各个状态之间的转化如图所示。除了就绪态和运行态，这些修改进程状态的操作都需要由操作系统内核来实现，所以需要设计对应的系统调用来实现。所以fork,exit和wait都是系统调用的一部分。



② 关于父子进程通信

父进程调用fork()原语，寻找空闲的pcb表项创建一个子进程。父子进程共享全局变量。父子进程就可以通过这个全局变量进行通信。另外通过设置阻塞态可以避免对全局变量的访问出现顺序错误。

③关于父子进程的关系：

父子进程共享全局变量以及代码段，这意味着父子进程的基址寄存器的值相等。另外父子进程拥有各自的堆栈和局部变量，即栈指针是不同的。然后父子进程可以并行执行。子进程的退出可以使得父进程被唤醒。

2.实验内容

（这次实验建立在上次实验的基础上，用户通过键盘输入来运行退出用户子程序。而子程序1就是这次实验要求的父进程与子进程合作运行。这里最主要讲的就是这个子程序。本次实验删除了测试系统调用的程序）

1. fork()原语，分配pcb表项。

在上个实验中，我是为每个子进程分配了固定的pcb表项，而这次的实验要求创建子进程时才在pcb表查找空闲的表项，这意味着动态的分配pcb表项。因此本次实验更改了对pcb表的分配。本次实验中，一个pcb表项对应于一个固定的堆栈地址。另外pcb表项中增加了一个基址寄存器，用以记录进程的起始地址，父子进程应该有相同的起始地址。Ip地址的初始值便是基址寄存器的值。

2.wait()原语

Wait()原语是用来挂起进程的，这不仅要将进程状态置为挂起态，还要进行进程切换，即保存现有进程然后重启下一个就绪进程。

3.exit()原语

exit()原语意味着将一个运行态的进程转化为退出态。这个只需要把进程状态置为退出态，即将对应的pcb表项置位无效。当子进程退出时，父进程将被唤醒。

3.实现方法：

①fork()原语实现方法：

对于子进程，fork()直接返回0.对于父进程，fork()首先搜索一个空闲的pcb表项，若无则，直接返回-1，若有空闲的pcb表项，则将父进程的基址寄存器复制给子进程，子进程ip以基址寄存器的值为初始值。然后更新栈指针。最后将进程pcb

的id返回。

②wait()原语的方法：

Wait原语负责挂起当前进程，并进行进程切换。挂起进程就是将进程置为挂起态，然后进程切换时调用上个实验的save函数进行保存进程，然后在调用restart重启下一个进程。

③ exit()原语的方法：

Exit原语只需要把进程状态置为退出态即可。然后在子程序处，调用完exit后，进程进入死循环，那么返回进程后，在时钟中断进行一次进程切换后该进程就不会进入就绪态而是进入退出态了。

3.程序流程：

子程序(父子进程合作)的过程
1.父进程创建一个子进程，若创建失败，则结束进程。否则获得子进程 pcbid
2.父进程进入阻塞态等待子进程完成它的任务
3.子进程完成任务(计算字符串长度并赋值到一个全局变量)后退出程序以唤醒父进程
4.父进程被唤醒后，经过时钟中断进入运行态，完成自己的那部分任务(获得字符串长度并进行输入显示)。

4.程序关键模块：

1.fork()原语：

```
//无内存返回-1，父进程返回子进程id，子进程返回0
int do_fork()
{
    int i = 1;
    int stack_ip, proc_ip;
    if (father[running_proc] > 0) return 0;
    for (i = 1; i < 7; i++)
    {
        if (status[i] == 0)
        {
            father[i] = running_proc;
            proc_ip = proc_addr[running_proc];
            stack_ip = i * (0x200) - (0x200);
            stack_ip += 0x5000;
            SetProc(i, proc_ip, stack_ip);
            status[i] = 2;
            return i;
        }
    }
    return -1;
}
```

Father 数组代表进程的父亲进程。当它的值不为 0 时，代表有父亲。SetProc 是一个设置 pcb 的函数，由于我的 pcb 表用汇编实现，所以这里需要用函数来对 pcb 的 ip 和 sp 进行设置。

2.wait 原语：

```
int do_wait()
{
    status[running_proc] = 1;
    Schedule();
    return 1;
}
```

```
;调用wait函数后进行进程切换,要先压标志寄存器，cs，ip
_Schedule:
    pushf
    push cs
    call changeproc
    retf
changeproc:
    call save
    push ax
    xor eax, eax
    mov ax, _getnextproc
    call eax
    mov byte[cs:a_running_proc], al
    pop ax
    call restart
    iret
```

Schedule 函数用于切换进程，这里要注意，由于在时钟中断时，调用 save 函数时，栈顶是标志寄存器，cs，ip。所以这里要先压栈。之所以把进程切换写成函数，是为了将重启时的地址定在 retf 处。

3.exit 原语：

```
int do_exit()
{
    status[running_proc] = 0;
    if (status[father[running_proc]] == 1)
        status[father[running_proc]] = 2;
    father[running_proc] = 0;
}
```

Exit 原语需要注意的就是有父进程时，要唤醒父进程。

4.子程序 1

```
23      int pid;
24      pid = fork();
25      if (pid == -1) //内存失败
26      {
27          Putstr(memory_error);
28          exit();
29      }
30      if (pid > 0) //父进程
31      {
32          Putstr(father_str1);
33          wait();
34          size = 22;
35          num = letter_num;
36          while(num > 0)
37          {
38              father_str2[size] = num % 10 + '0';
39              num /= 10;
40              size++;
41          }
42          father_str2[size] = 0;
43          Putstr(father_str2);
44          exit();
45      }
46      else //子进程
47      {
48          Putstr(son_str);
49          letter_num = strlen(inf);
50          exit();
51      }
```

Letter_num 和 inf 是两个全局变量，子进程统计 inf 变量的长度并赋值给 letter_num 父进程则对 letter_num 进行输出。

四.实验过程与结果

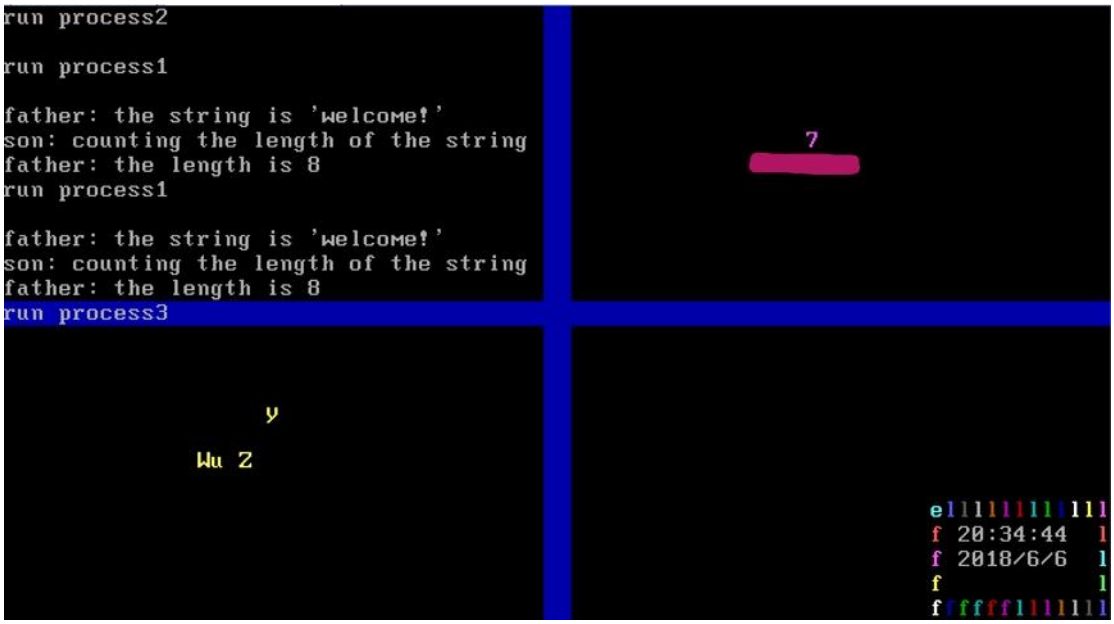
1.实验过程：

【编译链接的命令】
与上次实验一致。

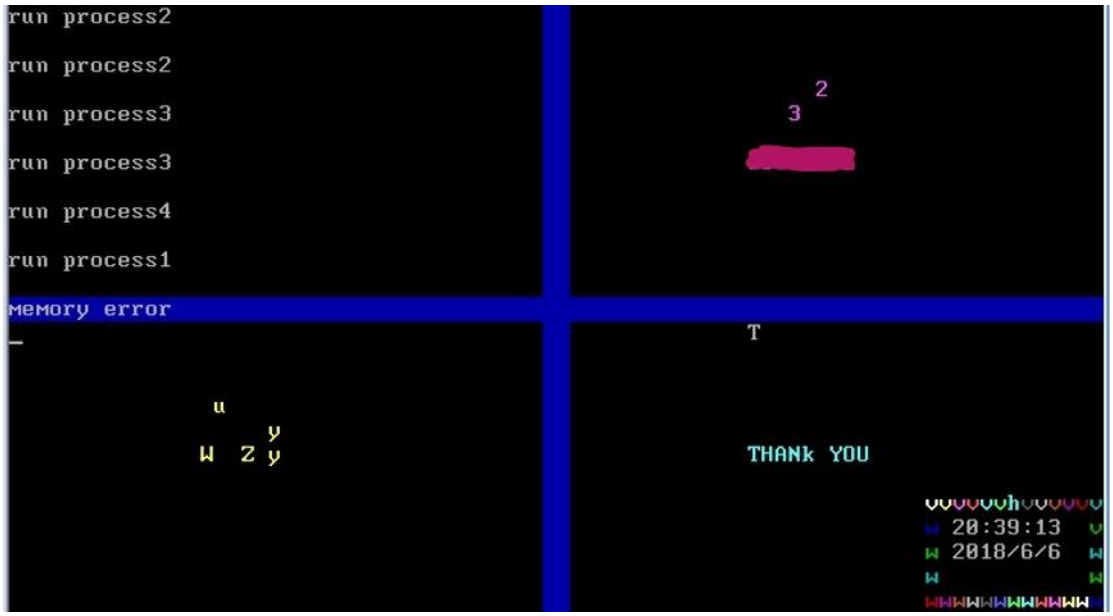
【磁盘与内存安排】
磁盘安排与上个实验一样。
内核的内存安排也与上个实验一样。
用户程序的内存安排则是由内核分配的。创建用户程序时，在 PCB 表中寻找空余的表项，表项与内存地址一一对应。

2.运行结果：

正常运行时。子程序 1 就是父子进程合作通信的程序。



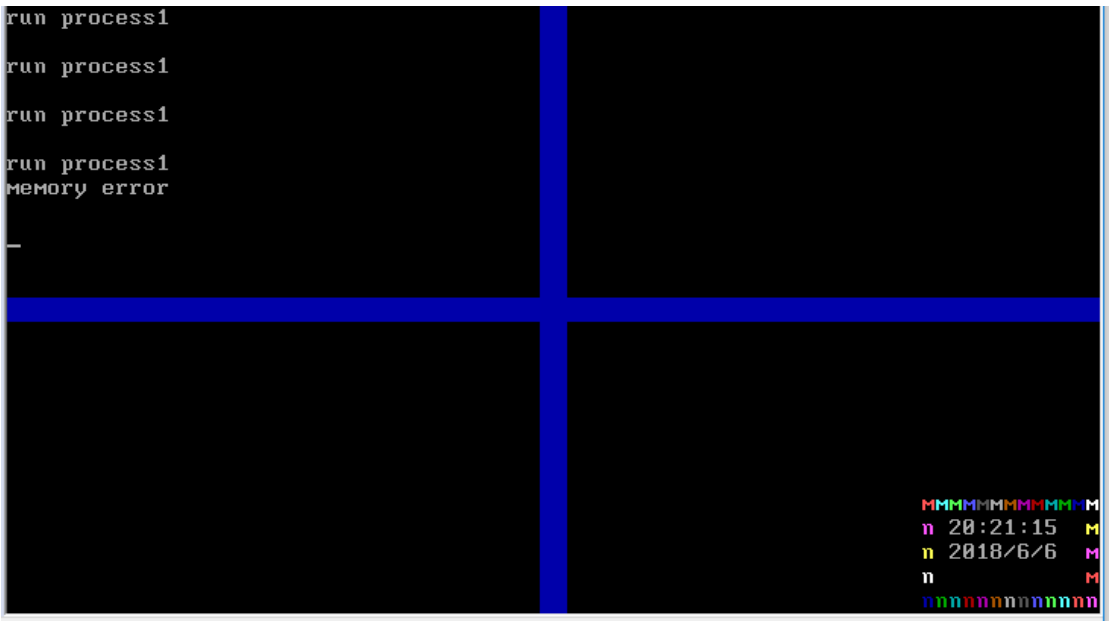
子进程没有空闲的pcb表项时，在这里pcb表项一共有6项，所以第六个进程是子程序1时，无法生成子进程：



3.遇到的问题及解决情况：

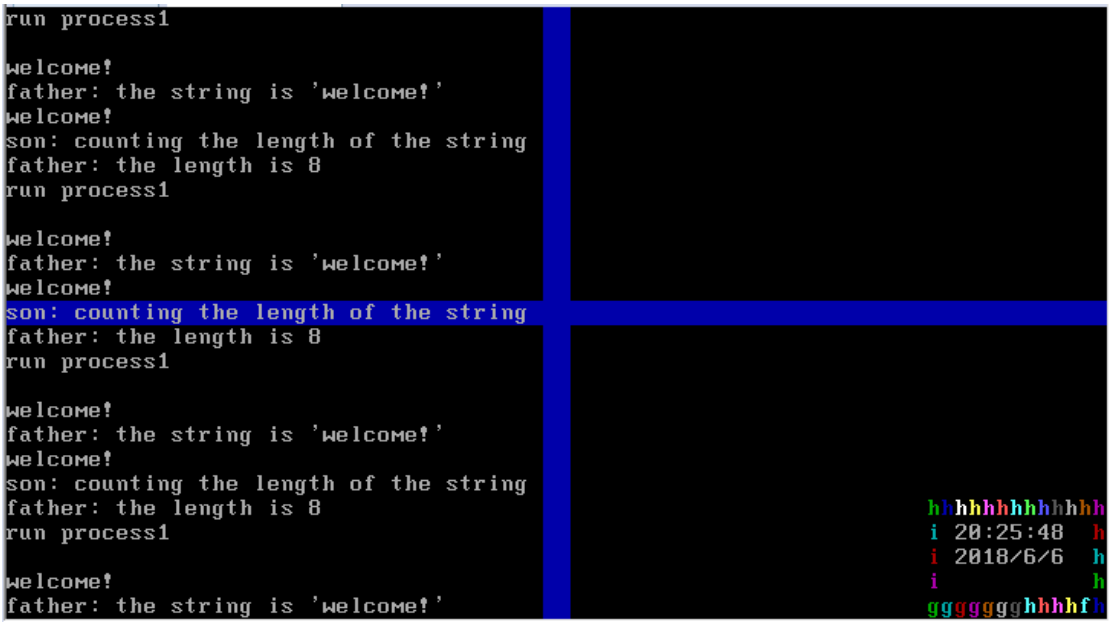
①一个我无法解释的问题。

```
void drawLfUp()
{

    // int i, j;
    // for (i = 5; i < 8; i++)
    // →for(j = 14; j < 30; j++)
    // →→    Setcolor(i, j, 7);
    int pid;
    pid = fork();
    Putstr(inf);
    if (pid == -1) //内存失败
```



当我注释掉这一段没有意义的代码后，子程序的输出不见了。而当有这段代码时，子程序是可以正常运行的。如下图，这两份代码除了这处注释没有不同，却导致结果很大差异。而这个Setcolor函数只是一个对屏幕颜色的设置。

```
void drawLfUp()
{

    int i, j;
    for (i = 5; i < 8; i++)
        for(j = 14; j < 30; j++)
            Setcolor(i, j, 7);
    int pid;
    pid = fork();
    Putstr(inf);
    if (pid == -1) //内存失败
    {
```



②问题：在wait函数，我原本是直接将进程状态设置为挂起态。结果父进程无法输出字符串长度。

问题的原因是虽然我把状态改成挂起态了，但是我没有切换进程，当CPU控制权从内核返回到子程序时，并不会检测进程状态，只有在时钟中断中切换进程时，才会检测进程状态，所以子程序会继续运行。但是这时候子进程还没运行，即没有计算出字符串长度。而父进程已经进行了输出，这就是运行顺序出错了，所以wait函数挂起进程状态后，要进行进程切换。

五.实验创新点

六.实验总结：

本次实验实现了五进程模型。其主要内容是进程合作。具体的应用是父进程创建子进程，父进程阻塞等待子进程，子进程完成父进程安排的任务后唤醒父进程。

这次实验和上次实验的时间隔得有点久，其中很多的内容我都忘记了自己是怎么写的。

这次实验我在 wait 函数上遇到了不少的问题。首先是进程切换的时候，由于在时钟中断中进行进程切换时，属于调用中断，所以是先把标志寄存器，cs，ip 压入栈中的。重启进程的时候也应该是要将 ip，cs，标志寄存器压入栈。由于一开始没有意识到 save 和 restart 的特性，所以在这里就卡 bug 了。

这次实验比较花费我的时间的一个东西是，修改上次实验的内容。因为上次

实验我没有动态的分配 pcb 表项，上次我是每个 pcb 表项固定一个进程，运行和退出的时候就修改进程状态。并且上次要另写一个子程序测试中断是否仍然可用，而这个测试程序要占用键盘输入，因此我要屏蔽其他进程。而这次实验要求动态分配 pcb，所以我要把之前的硬编码的 pcb 表项重写。

说说这次的用户程序，我把父子进程都包含进了同一个用户程序，然后在程序开头用条件判断语句判断当前程序是父程序还是子程序，然后执行相应的模块。

参考文献：

1．《80x86 汇编语言程序设计教程》 （杨季文）
2．《nasm 中文手册》 （csdn）
3．《一个操作系统的实现》 于渊
4．《汇编语言（第三版）》 王爽