**实验三\_14011907\_2019302789\_梁芮槐**

**实验要求1：**

3）使用系统调用fork()创建两个子进程，在此程序运行时，系统中就有一个父进程和两个子进程在活动。让每一个进程在屏幕上显示一个字符：父进程显示字符a，两个子进程分别显示字符b和子符c。试观察、记录并分析屏幕上进程调度和并发执行的情况。若在程序中使用系统调用nice()来改变各进程的优先级，会出现什么现象？

答：

无nice()，fork\_learn.c：

#include <stdio.h>

#include <errno.h>

#include <stdlib.h>

#include <unistd.h>

int main(int argc, char\* argv[])

{

pid\_t fpid\_1;

pid\_t fpid\_2;

(fpid\_1 = fork()) && (fpid\_2 = fork());

if(fpid\_1 == 0)

{

printf("I'm child b ppid=%d pid=%d\n", getppid(), getpid());

}

else if(fpid\_2 == 0)

{

printf("I'm child c ppid=%d pid=%d\n", getppid(), getpid());

}

else

{

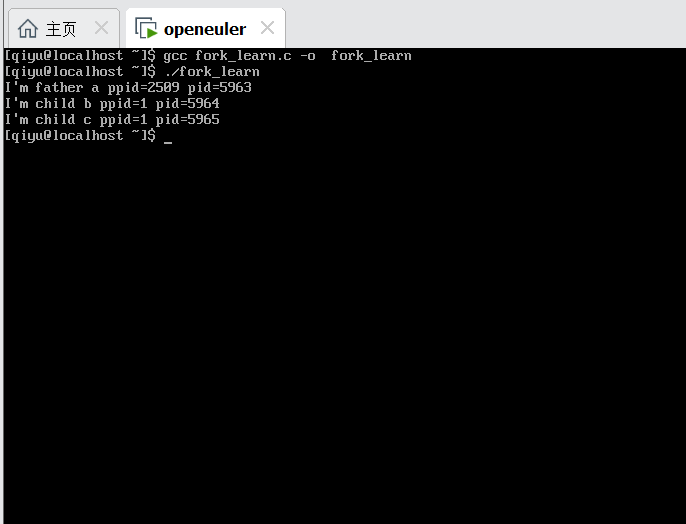
printf("I'm father a ppid=%d pid=%d\n", getppid(), getpid());

}

return 0;

}

无nice()运行结果：



有nice()，fork\_learn.c：

#include <stdio.h>

#include <errno.h>

#include <stdlib.h>

#include <unistd.h>

int main(int argc, char\* argv[])

{

pid\_t fpid\_1;

pid\_t fpid\_2;

(fpid\_1 = fork()) && (fpid\_2 = fork());

if(fpid\_1 == 0)

{

int ret = nice(0);

printf("child b priority=%d\n", ret);

if(ret == -1)

{

if(errno == EACCES) printf("cannot set priority\n");

}

}

else if(fpid\_2 == 0)

{

int ret = nice(10);

printf("child c priority=%d\n", ret);

if(ret == -1)

{

if(errno == EACCES) printf("cannot set priority\n");

}

}

else

{

int ret = nice(19);

printf("father a priority=%d\n", ret);

if(ret == -1)

{

if(errno == EACCES) printf("cannot set priority\n");

}

}

if(fpid\_1 == 0)

{

printf("I'm child b ppid=%d pid=%d\n", getppid(), getpid());

}

else if(fpid\_2 == 0)

{

printf("I'm child c ppid=%d pid=%d\n", getppid(), getpid());

}

else

{

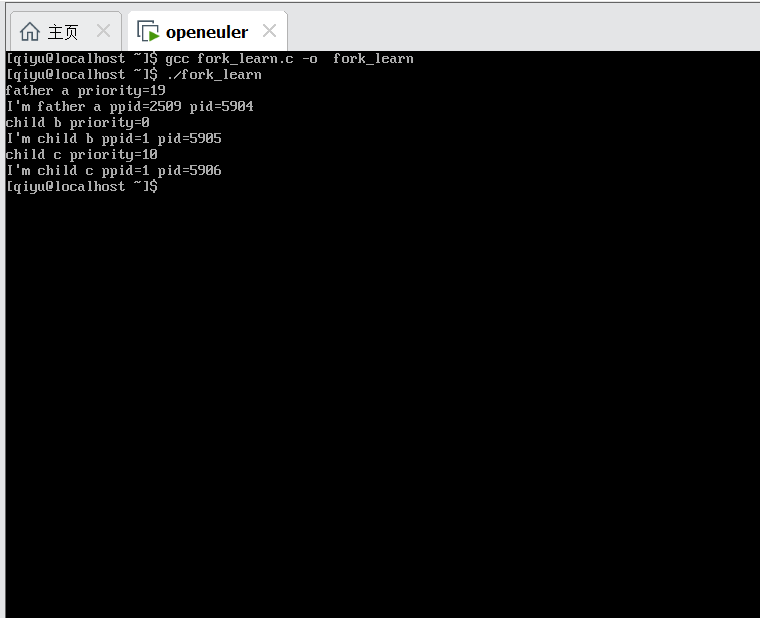
printf("I'm father a ppid=%d pid=%d\n", getppid(), getpid());

}

return 0;

}

有nice()，运行结果：



4）提交源程序清单，并附加流程图与注释

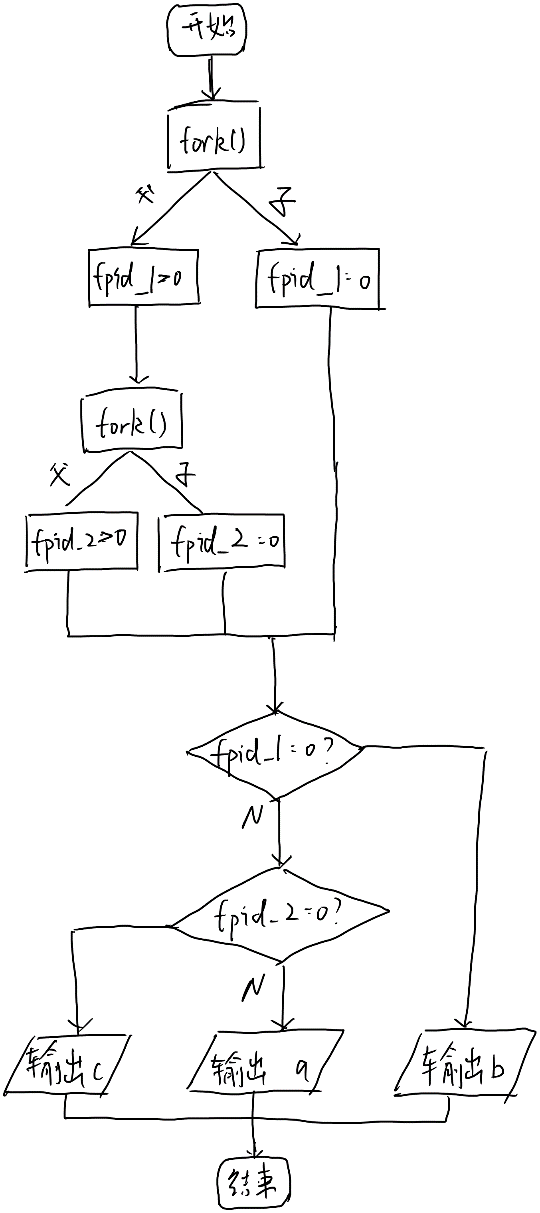
并回答以下问题：

1. 系统调用fork()是怎样创建进程的？

2）当首次调用新创建的子进程时，其入口在哪里？

3）分析进程调度和进程并发执行的关系？

答：流程图：



注释：第一个子进程里fpid\_1是0，故&&后的运算没有意义便没有执行，但父进程里fpid大于0，&&后的运算执行了，父进程便又创建了一个子进程，这样就是一个父进程和两个子进程了；

而父进程里fpid\_1和fpid\_2都大于0，最后逻辑分支输出父进程信息，两个子进程fpid\_1和fpid\_2分别为0，逻辑分支输出相应子进程的信息；

第二步加入了nice()函数操作，使子进程b的优先级最高，并且检验优先级设置结果是成功的，但是这并没有影响三个进程的运行顺序和输出顺序，我想是因为它们没有争抢资源所以在输出上并没有体现优先顺序。

小问1）：fork（）函数通过系统调用创建一个与原来进程几乎完全相同的进程，也就是两个进程可以做完全相同的事，但如果初始参数或者传入的变量不同，两个进程也可以做不同的事。一个进程调用fork（）函数后，系统先给新的进程分配资源，例如存储数据和代码的空间。然后把原来的进程的所有值都复制到新的新进程中，只有少数值与原来的进程的值不同。相当于克隆了一个自己。

小问2）：fork是把进程当前的情况拷贝一份，故新进程的入口是fork()调用时当前进程的状态。

小问3）：通过进程调度才能实现进程并发执行。

1. **实验要求2：**读取文件“sort.txt”，文件第一行是十个数字 “65,43,79,21,298,12,3,54,44,1”；

冒泡排序，BubbleSort.c：

#include <stdio.h>

void BubbleSort(int A[], int s, int m)

{

int i, j, t;

for(j = 0; j <= m - s; j++)

for(i = s; i < m - j; i++)

if(A[i] < A[i + 1])

t = A[i], A[i] = A[i + 1], A[i + 1] = t;

}

int main()

{

int A[10];

FILE \*fr = fopen("/home/qiyu/code\_1/sort.txt", "r");

FILE \*fw = fopen("/home/qiyu/code\_1/result.txt", "w");

for(int i = 0; i < 10; i++)

fscanf(fr, "%d", &A[i]);

fclose(fr);

BubbleSort(A, 0, 9);

for(int i = 0; i < 10; i++)

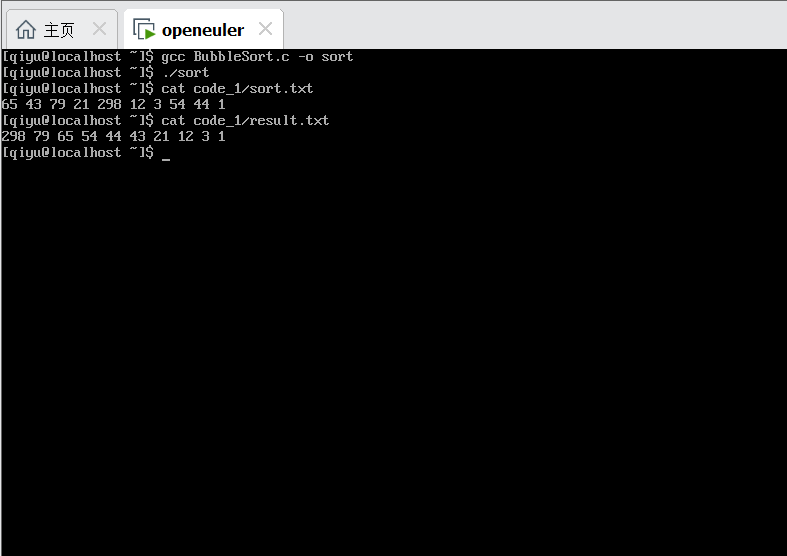
fprintf(fw, "%d ",A[i]);

fclose(fw);

return 0;

}

冒泡排序结果：



快速排序，QuickSort.c：

#include <stdio.h>

void QuickSort(int A[], int s, int m)

{

int i, j, t;

if(s < m)

{

i = s;

j = m + 1;

while(1)

{

while(i + 1 < s + m && A[++i] > A[s]);

while(j - 1 > -1 && A[--j] < A[s]);

if(i >= j) break;

t = A[i], A[i] = A[j], A[j] = t;

}

t = A[s], A[s] = A[j], A[j] = t;

QuickSort(A, s, j - 1);

QuickSort(A, j + 1, m);

}

}

int main()

{

int A[10];

FILE \*fr = fopen("/home/qiyu/code\_1/sort.txt", "r");

FILE \*fw = fopen("/home/qiyu/code\_1/result.txt", "w");

for (int i = 0; i < 10; i++)

{

fscanf(fr, "%d", &A[i]);

}

fclose(fr);

QuickSort(A, 0, 9);

for(int i = 0; i < 10; i++)

{

fprintf(fw, "%d ", A[i]);

}

fclose(fw);

return 0;

}

快速排序结果：

