第一次大作业

杨燕

1752669

题目要求

进程1，计算，输出斐波那契数列。每50ms输出一个数。

进程2，按1，2，3，4，.....的顺序输出100000个自然数，每10ms输出一个数。

进程3，输出200个对角线长度为随机数的正方形，每100ms输出一个正方形。

父进程计算输出所有的质数，不需要定时输出。

注意：

创建子进程后，父进程投入自己的运算和所有子进程并发；处理$SIGCHLD$信号时，捎带回收子进程的$task_struct(PCB)$.用户用$ctrl+c$终止父进程，但父进程终止前要回收所有子进程的$task_struct(PCB)$.

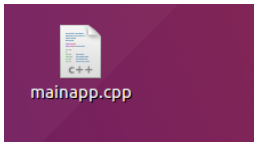
4个进程的输出不要搅在一起。

可以让子进程3做任何有意思的事情。

作业设计

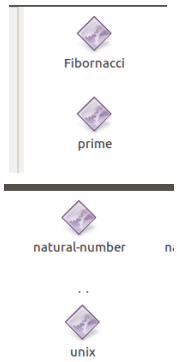
一：总体架构

主程序：$mainapp$



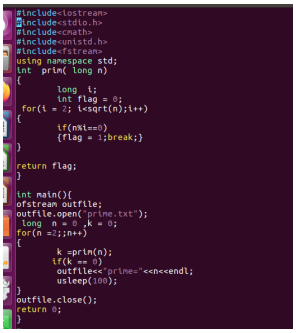
三个子进程和父进程

解释：$unix$：就是进程3.



二：代码展示

1：父进程的代码



```
#include<iostream>
#include<stdio.h>
#include<cmath>
#include<unistd.h>
#include<fstream>
using namespace std;
int  prim( long n)
{
        long  i;
        int flag = 0;
 for(i = 2; i<sqrt(n);i++)
{
        if(n%i==0)
        {flag = 1;break;}
}

return flag;
}

int main(){
ofstream outfile;
outfile.open("prime.txt");
 long  n = 0 ,k = 0;
for(n =2;;n++)
{
       k =prim(n);
       if(k == 0)
        outfile<<"prime="<<n<<endl;
        usleep(100);
}
outfile.close();
return 0;
}
```

2：子进程1的代码

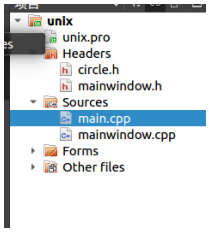
```cpp
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<iostream>
#include<fstream>
using namespace std;

int  main()
 {
unsigned long long tmp2 = 1, tmp1 = 1, sum =
 ofstream outfile;
        outfile.open("Fibornacci.txt");
        outfile<<"sum="<<tmp1<<endl;
        usleep(50);
        outfile<<"sum="<<tmp2<<endl;
        usleep(50);
     while(1)
     {
         sum = tmp1 + tmp2;
         tmp1 = tmp2;
         tmp2 = sum;
         outfile<<"sum="<<sum<<endl;
        usleep(50);
     }
outfile.close();
return 0;
}
```

3：子进程2的代码

```cpp
#include<stdio.h>
#include<iostream>
#include<unistd.h>
#include<fstream>
using namespace std;

int main()
{ofstream outfile;
outfile.open("natural_number.txt");
int count = 100000;
for(int i = 1; i<count+1;i++)
{ outfile<<"number="<<i<<endl;

 usleep(10);
}
outfile.close();
return 0;
}
```

4：子进程3的代码

子进程3主要是用QT来实现图形界面以及动态图：



$circle.h$这个头文件是用来画正方形的$(yellow)$，设置相关颜色以及边框颜色$(black)$

```cpp
#include<QWidget>
#include<QPainter>
#include<QPen>

struct Circle: QWidget{
    Q_OBJECT
public:
    explicit Circle(QWidget *parent = nullptr) :QWidget(parent){
        resize(0,0);
    }
    void paintEvent(QPaintEvent *) override{
        QPen pen;
        pen.setWidth(5);
        pen.setColor(Qt::black);
        pen.setStyle(Qt::SolidLine); //设置风格
        QPainter painter(this);
        painter.setPen(pen);
        painter.setBrush(Qt::yellow);
        painter.drawRect(50,50,100,100);
    }
};
```

然后就是主函数的代码

$number_rand()$函数就是生成随机数，我把这个生成的随机数传输到$addAnimation()$这个函数中，来随机表示正方形的边长。

图中注释的那个$delay100ms$就是延迟函数，每隔$100ms$输出一个正方形。

其他的就是画动态图的一些代码，不过多解释。

```cpp
#include <iostream>
#include <QApplication>
#include <QMessageBox>
#include <QDebug>
#include <QLabel>
#include <QRect>
#include <QMainWindow>
#include "circle.h"
#include <QSequentialAnimationGroup>
#include <QGraphicsView>
#include <QGraphicsScene>
#include <QParallelAnimationGroup>
#include <unistd.h>
#include <QPropertyAnimation>
#include <ctime>

int number_rand()
{ qsrand(time(NULL));
    int n = qrand()%1000;
   return n;
}

QPropertyAnimation *widgetDrop(QObject *object,int n) {
    auto animation = new QPropertyAnimation(object, "geometry");
    animation->setDuration(100);//time delay 100ms
    animation->setKeyValueAt(0, QRect(0, 0, 0, 0));
    animation->setKeyValueAt(1, QRect(100, 400, n+100, n+100));
    animation->setEasingCurve(QEasingCurve::OutBounce);
    return animation;
}

int main(int argc, char **argv) {
    QApplication app(argc, argv);

    QMainWindow window;
    window.resize(700, 700);

    Circle *circles[200][];
    for (int i = 0; i < 200; ++i) { circles[i] = new Circle(&window); }

    auto animationGroup = new QSequentialAnimationGroup;
   int num ;
    for (int i = 0; i < 200; ++i)
    {
        num= number_rand();
        animationGroup->addAnimation(widgetDrop(circles[i],num));
    }
    animationGroup->start(QPropertyAnimation::DeleteWhenStopped);

    window.show();

    return QApplication::exec();
}
```

5:总体的代码

主函数分为3部分：信号处理函数$kill_process()$;进程函数$process()$;主函数$main()$;

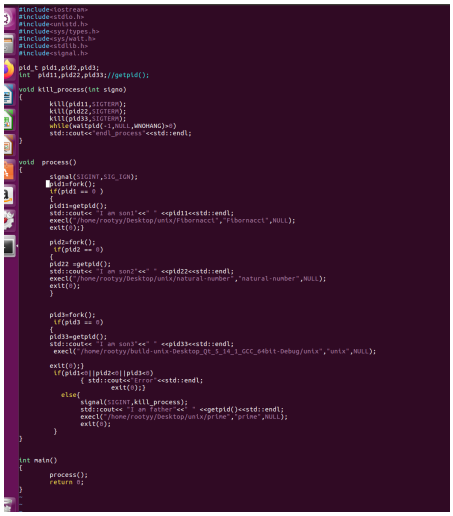
关于信号处理函数主要作用就是在父进程结束的时候，杀死子进程，并且回收子进程pcb.

$process()$函数就是创建子进程$pid1, pid2, pid3$.然后用$execl()$函数去调用进程程序。

为了方便输出，且不混乱，父进程，子进程1，2的结果都输出到文件里去，进程3就在终端演示。

信号处理函数$signal(SIGINT, SIG_IGN)$是保证父进程$ctrl + c$结束后，子进程不会结束.

信号处理函数$signal(SIGINT, kill_process)$是在父进程$ctrl + c$结束后，去执行信号处理函数，杀死并回收子进程。

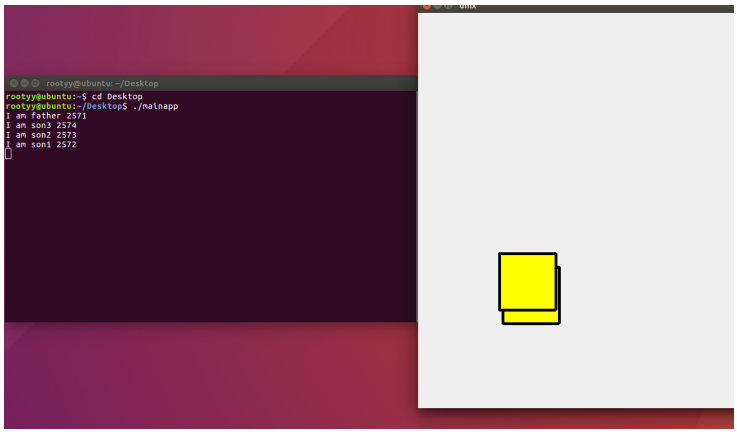


```
#include<iostream>
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<stdlib.h>
#include<signal.h>

pid_t pid1,pid2,pid3;
int  pid11,pid22,pid33;//getpid();

void kill_process(int signo)
{
        kill(pid11,SIGTERM);
        kill(pid22,SIGTERM);
        kill(pid33,SIGTERM);
        while(waitpid(-1,NULL,WNOHANG)>0)
        std::cout<<"endl_process"<<std::endl;
}


void  process()
{
        signal(SIGINT,SIG_IGN);
        pid1=fork();
        if(pid1 == 0 )
        {
        pid11=getpid();
        std::cout<< " I am son1"<<" " <<pid11<<std::endl;
        execl("/home/rootyy/Desktop/unix/Fibornacci","Fibornacci",NULL);
        exit(0);}

        pid2=fork();
         if(pid2 == 0)
         {
         pid22 =getpid();
         std::cout<< " I am son2"<<" " <<pid22<<std::endl;
         execl("/home/rootyy/Desktop/unix/natural-number","natural-number",NULL);
         exit(0);
         }


         pid3=fork();
          if(pid3 == 0)
          {
          pid33=getpid();
          std::cout<< " I am son3"<<" " <<pid33<<std::endl;
          execl("/home/rootyy/build-unix-Desktop_Qt_5_14_1_GCC_64bit-Debug/unix","unix",NULL);

          exit(0);}
          if(pid1<0||pid2<0||pid3<0)
                { std::cout<<"Error"<<std::endl;
                        exit(0);}
            else{
                signal(SIGINT,kill_process);
                std::cout<< " I am father"<<" " <<getpid()<<std::endl;
                execl("/home/rootyy/Desktop/unix/prime","prime",NULL);
                exit(0);
            }
}

int main()
{
        process();
        return 0;
}
```

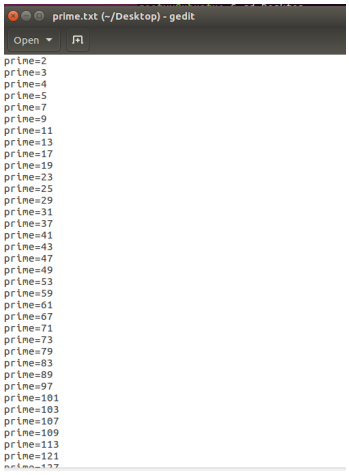
三：结果展示

1：终端展示

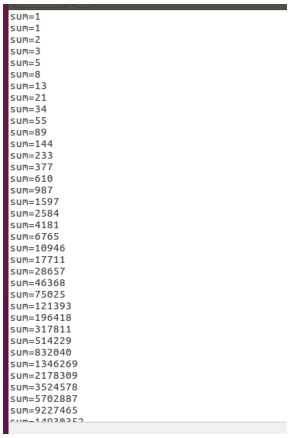
由于进程3我是弄了一个动画输出，所以会发一个$demo$过去。进程3的效果就是有不停的正方形输出然后重合到一个正方体中，有节奏的跳着输出。。



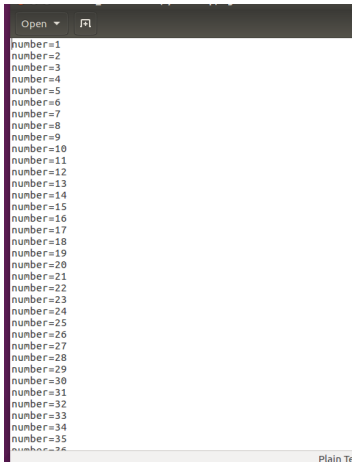
2:父进程输出



3：进程1的输出

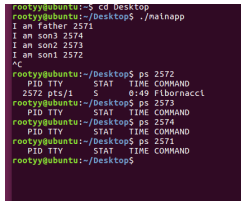


4：进程2的输出



四：问题总结

有一个问题就是$ctrl + c$后杀死了父进程，然后子进程没有杀死，即信号处理函数没有作用，我测试其他的简单的代码是可以的，但是放在这个程序里就没有作用了，我目前还没找到解决办法，希望老师能提点提点。



查看了所有进程的状态，$ctrl + c$之后，子进程1并没有被杀死，进程3可以在图框上点X就可以结束，进程2是运行完了（输出了100000个数）所以才结束的.

这个是这个实验的不足之处。