LAB3_REPORT

解析命令行：

首先是gettime函数，调用clock()即可,时间单位是s：

```
static double gettime() {
  uint64_t t=clock();
  return ((double)t)/CLOCKS_PER_SEC;
}
```

在main函数中使用字符串比较函数判断即可，代码如下：

```
int main(int argc, char **argv) {
  int rounds=1;
  for(int i=0;i<argc;i++){
    if(strcmp(argv[i],"perf")==0){
      i++;
      if(strcmp(argv[i],"-r")==0){
        i+=1;
        rounds=atoi(argv[i]);
        i+=1;
        strcpy(fn,argv[i]);
        break;
      }
      else{
        strcpy(fn,argv[i]);
        break;
      }
    }
  }
  void (*func)() = lookup(fn);
  run(func, rounds);
  return 0;
}
```

经过测试，能成功解析(包括-r参数)。main函数将会调用run来将func运行rounds次，故在run函数中进行数据的处理：

```
  FILE *fp=fopen("./data.txt","w+");
  fprintf(fp,"%s\n",fn);
  for(int i=0;i<rounds;i++){
    fprintf(fp,"%f ",elapsed[i]);
  }
  fclose(fp);
  //使用shell运行test.py文件
  const char cmdstr[20]="python3 test.py";
  system(cmdstr);
```

其中fn是文件名的字符串，将时间数据写入data.txt文件中，然后使用shell来运行test.py文件进行数据的分析处理然后在命令行打印出分析信息和图表。
在impl.c中添加lab1中的三个函数,将函数类型改成void并且不设参数，在impl.c中定义全局变量a,b,m,并在main.c中声明，在main.c中还定义了一个宏来生成随机数：

```
#define gen_rand a=rand();\
    b=rand();\
    m=rand();\
    FILE *fp=fopen("./multimod_num.txt","a");\
    fprintf(fp,"%ld %ld %ld\n",a,b,m);\
    fclose(fp);
```

将每一次的a,b,m写入multimod_num.txt文件中(注意在命令行输入运行前需要先删去原来的这个文件不然就是在原文件基础上写入)。
在run函数中，修改使其运行multimod_p前生成随机数并写入文件。

```
for (int round = 0; round < rounds; round++) {
    if(strcmp(fn,"multimod_p1")==0||strcmp(fn,"multimod_p2")==0 ||
strcmp(fn,"multimod_p3")==0){
      gen_rand;
    }
    double st = gettime();
    func();
    double ed = gettime();
    elapsed[round] = ed - st;
  }
```

接下来是test.py文件：

```
import matplotlib.pyplot as plt
import numpy as np
import sys
from scipy.stats import kstest
func_mul=['multimod_p1','multimod_p2','multimod_p3']
attr=['norm','uniform','expon']

f=open("./data.txt","r")
filename=f.readline().strip()
print('func is '+filename)
data=[]
for line in f.readlines():
    for d in line.split():
        data.append(float(d))
data=np.array(data)
index=np.array(range(len(data)))
plt.ylim(min(data),max(data))
plt.plot(index,data)
plt.xlabel("index")
plt.ylabel("run time(ms)")
plt.title("run time of "+filename)
plt.show()

dom=np.zeros(50)
inter=float(max(data))/50
for i in range(len(dom)):
```

```
    n=int(float(data[i])/inter)
    dom[n]=dom[n]+1
plt.bar(range(50),dom)
plt.xlabel("index")
plt.ylabel("count")
plt.title("run time count of "+filename)
plt.show()

result=[0,0]
str=""
for i in range(len(attr)):
    tmp=kstest(dom,attr[i])
    if(i==0):
        result=tmp
        str=attr[i]
        continue
    if(result[1]<tmp[1]):
        result=tmp
        str=attr[i]
print("平均数是",np.mean(data))
print("标准差是",np.std(data))
print("分布检验结果为"+str)
print(result)
```

data数组存储了时间数据，此时绘制的图是每一次函数运行时间的图。dom存储了data的分布，表示data在每个区间内的数据多少。使用scipy.stats.kstest来验证分布，在此比较三种分布：正态分布(norm)，均匀分布(uniform)，指数分布(expon)，kstest验证的返回值是个二元列表，分别代表统计数和置信度，取置信度最高的那个分布(multimod_p2和multimod_p3由于数据的本身的一些特性导致它们并不服从这些分布，因此置信度都接近0)。并计算了平均数和标准差。
以下为multimod的运行结果(因为我的multimod_p1实在太慢所以只运行了100次，其余运行了1000000次)：

时间/秒	平均数	标准差
multimod_p1	12.5709	7.8049
multimod_p2	1.719111x10^(-6)	7.53657×10^(-6)
multimod_p3	5.42437×10^(-7)	6.23404×10^(-7)

p1中的运行时间有相比很快的也有相比很慢的，方差极大，原因应该是数据的问题，输入数越小时时间越快，因为它是O(N)的。p2的运行时间趋势大致是一段时间快一段时间慢，后期有大"尖峰"，原因不明，猜测可能是运行时电脑抽风。p3也是一快一慢，相比较快的时间和相比较慢的时间都比较均匀。总体来说运行时间上p1>>p2>p3，且p2和p3相比差距较小,其标准差也相差不大。
然后是运行时间的区间分布，我将其分成了50个区间，每个区间大小是data中的最大值/50，以下为三个multimod的运行时间分布：

可以看出p2和p3都仅集中在几个区间,他们并不服从正态分布，p1比较分散，但也不服从正态分布，较服从均匀分布。

multimod实现的性能差异：

a,b,m的大小：

由于之前已经将所有随机生成的a,b,m都写入了文件multimod_num.txt中，故在test.py中将其读出：

```
if(filename in func_mul):
    a=[]
    b=[]
    m=[]
    fp=open("./multimod_num.txt","r")
    for line in fp.readlines():
        temp=line.split()
        a.append(int(temp[0]))
        b.append(int(temp[1]))
        m.append(int(temp[2]))
    plt.ylim(min(data),max(data))
    plt.scatter(a,data,label="a")
    plt.scatter(b,data,label="b")
    plt.scatter(m,data,label="m")
    plt.xlabel("number size")
    plt.ylabel("run time(ms)")
    plt.legend(loc="upper left")
    plt.title("number size and run time of "+filename)
    plt.show()
```

绘制图像如下：

对于p1来说，运行时间和b的大小呈线性关系，因为它是需要要循环b次。看出对于p2和p3，a,b,m的大小对运行时间的影响并不是很明显，因为p2和p3仅需常数时间。

a,b,m二进制中1的个数：

```
    num_a=[]
    num_b=[]
    num_m=[]
    for i in range(len(a)):
        num_a.append(bin(a[i]).count('1'))
        num_b.append(bin(b[i]).count('1'))
        num_m.append(bin(m[i]).count('1'))
    plt.ylim(min(data),max(data))
    plt.scatter(num_a,data,label="a")
    plt.scatter(num_b,data,label="b")
    plt.scatter(num_m,data,label="m")
    plt.xlabel("number of binary 1")
    plt.ylabel("run time(ms)")
    plt.legend(loc="upper left")
    plt.title("num of binary 1 and run time of "+filename)
    plt.show()
```

使用bin函数来计算二进制中1的个数，绘图如下:

对于p1来说，二进制中1的个数并无太大影响，它的运行时间主要取决于整体的b的大小，而1的个数虽然在一定程度上能反映b的大小，但是这种相关性是比较小的，可能b很大(比如2^20)但它二进制1的个数很少。对于p3来说，二进制中1的个数并无太大影响，对于p2来说，二进制1的个数集中在中间区域时所花时间较长。