/\*

\*不同的浏览器对url的长度的限制是不同的，通过查询了解，大多数url的上限为2048B

\*1GB = 1024 \* 1024 \*1024 B

\*1GB 内存最多可以存储 512 \* 1024 = 524288 个url

\* 阻塞队列中数据数量限制就为524288

\* 设定url文件名为 url.txt

\* 最终输出的top100 url文件为 result.txt

\*/

#include<iostream>

#include<queue>

#include<pthread.h>

#include<stdio.h>

#include<unistd.h>

#include<fstream>

#define MAX\_NUM 524288

#define MAX\_THR 5 //线程数

Ofstream out;

pthread\_mutex\_t read\_mutex;

pthread\_mutex\_t write\_mutex;

bool flag = false; //标志文件是否读取完毕

vector<ofstream> v(100); // 100个小文件的写操作流

vector<ifstream >read(100); //100个小文件的读操作流

unordered\_map<string,int> mp;

class BlockQueue

{

private:

std::queue<string> \_queue;

int \_capacity;

pthread\_cond\_t \_cond\_pro;

pthread\_cond\_t \_cond\_con;

pthread\_mutex\_t mutex;

public:

BlockQueue(int que\_Maxcapacity)

:\_capacity(que\_Maxcapacity)

{

pthread\_mutex\_init(&mutex,NULL);

pthread\_cond\_init(&\_cond\_pro,NULL);

pthread\_cond\_init(&\_cond\_con,NULL);

}

~BlockQueue()

{

pthread\_mutex\_destroy(&mutex);

pthread\_cond\_destroy(&\_cond\_pro);

pthread\_cond\_destroy(&\_cond\_con);

}

//提供给生产者的接口（数据入队）

bool queuePush(string& data)

{

//queue是一个临界资源所以需要加锁保护

pthread\_mutex\_lock(&mutex);

//判断队列是否添加满了

while(\_queue.size() == \_capacity)

{

pthread\_cond\_wait(&\_cond\_pro,&mutex);

}

\_queue.push(data);

pthread\_mutex\_unlock(&mutex);

pthread\_cond\_signal(&\_cond\_con);

return true;

}

//提供给消费者的接口（数据出队）

bool queuePop(string& data)

{

pthread\_mutex\_lock(&mutex);

//判断队列是否为空

while(\_queue.empty()){

pthread\_cond\_wait(&\_cond\_con,&mutex);

}

data = \_queue.front();

\_queue.pop();

pthread\_mutex\_unlock(&mutex);

pthread\_cond\_signal(&\_cond\_pro);

return true;

}

};

void\* pro\_thr(void\* arg)

{

int i = 0;

BlockQueue\* queue = (BlockQueue\*)arg;

while(1){

pthread\_mutex\_lock(&read\_mutex);

String str;

If(getline(out,str))

{

pthread\_mutex\_lock(&read\_mutex);

flag = true;

break;

}

pthread\_mutex\_lock(&read\_mutex);

queue->queuePush(i);

}

return NULL;

}

void\* con\_thr(void\* arg)

{

string data;

BlockQueue\* queue = (BlockQueue\*)arg;

while(1){

If(queue.empty()&& flag == true){

Break;

}

queue->queuePop(data);

long num = Change(data);

pthread\_mutex\_lock(&write\_mutex);

v[num]<<data<<endl; //将url写入小文件

pthread\_mutex\_unlock(&write\_mutex);

}

return NULL;

}

//将url转化为数字并求其哈希映射,同时写入文件

long change(string url){

long num = 0;

for(int i = 0; i < url.size(); i++){

Num += url[i];

}

return num % 100;

}

//读取url文件，并且将url写入对应的小文件

void cut\_write(){

out.open(“url.txt”,ios:in); //以读的方式打开大文件

string url = “url”;

//以追加的方式打开100个小文件

for(int i = 0; i < 100; i++){

String tmp = url + to\_string(i) + “.txt”;

V[i].open(tmp.c\_str(),ios::app);

}

BlockQueue queue(MAX\_NUM);

pthread\_t pro\_tid[MAX\_THR];

pthread\_t con\_tid[MAX\_THR];

int i = 0;

int ret = 0;

//五个读取线程，五个写入线程

for(i = 0 ; i < MAX\_THR; i++)

{

pthread\_create(&pro\_tid[i],NULL,pro\_thr,(void\*)&queue);

if(ret != 0)

{

std::cerr << "pthread\_create pro\_thr error\n";

return -1;

}

}

for(i = 0; i < MAX\_THR; i++)

{

pthread\_create(&con\_tid[i],NULL,con\_thr,(void\*)&queue);

if(ret != 0)

{

std::cerr << "pthread\_create pro\_thr error\n";

return -1;

}

}

for(i = 0; i < MAX\_THR; i++)

{

pthread\_join(pro\_tid[i],NULL);

pthread\_join(con\_tid[i],NULL);

}

for(int i = 0; i < 100; i++){

String tmp = url + to\_string(i) + “.txt”;

v[i].close();

}

}

//统计每个url出现的次数

Void count\_size(){

//以读的方式打开文件，

String str = “url”;

for(int i = 0 ; i<100; i++){

String tmp = str + to\_string(i) + “.txt”;

read[i].open(“url.txt”,ios:in);)

String data ;

int count = 0;

While(getline(read[i],data)){

If(mp.find(data) != mp.end() )

mp[data]++;

else{

mp[data] = 1;

}

}

read[i].close();

}

}

struct cmp{

bool operator(pair<string,int>p1,pair<string,int>p1){

return p1.second > p2.secnd;

}

}

//获得top100的url

void get\_top(){

if(mp.size() >= 100){

//url的种数大于100，则构建元素个数为100的小堆

priority\_queue<pair<string,int>,vector<pair<string,int>>,cmp> pq(mp.begin(),mp.begin() + 100);

auto it = mp.begin() + 100;

While(it != mp.end()){

If(it->second > pq.top().second){

pq.pop();

pq.push(\*it);

}

}

//目前堆中的元素为top100，将其保存到文件中

ofstream file(“result.txt”,ios::out);

While(!pq.empty()){

file << pq.top().first<<endl;

pq.pop();

}

File.close();

}

else{

// 不存在top100，url种数小于 100

}

}

int main()

{

cut\_write(); // 分割文件

count\_size(); //统计所有小文件中 url 出现的次数

get\_top(); //获取top100的url保存至result文件中

return 0;

}