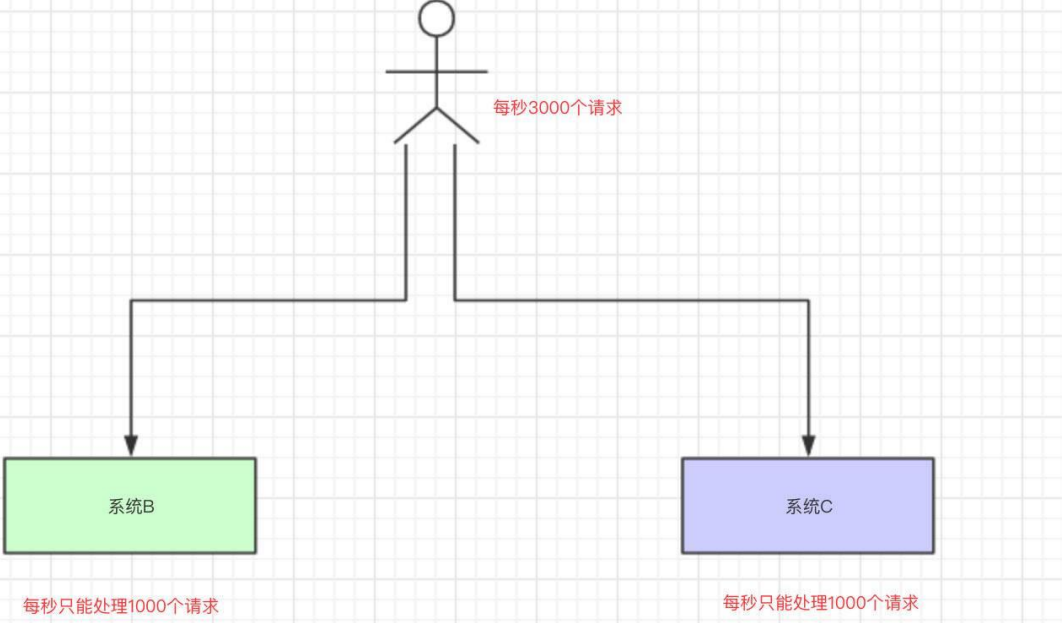


《数据结构》上机报告

2020 年 10 月 31 日

姓名： 王上游 学号： 1850767 班级： 19 计科 2 班 得分： _____

实验 题目	消息队列
实验 目的	1 、掌握队列的结构和基本操作； 2 、了解消息队列的使用场景（解耦、异步通信、限流消峰）
问题 描述	<p>某宝平台定期要搞一次大促，大促期间的并发请求可能会很多，比如每秒 3000 个请求。假设我们现在有两台机器处理请求，并且每台机器只能每次处理 1000 个请求。如图（1）所示，那多出来的 1000 个请求就被阻塞了（没有系统响应）。</p>  <p>请你实现一个消息队列，如图（2）所示。系统 B 和系统 C 根据自己能够处理的请求数去消息队列中拿数据，这样即便每秒有 8000 个请求，也只是把请求放在消息队列中。如何去拿消息队列中的消息由系统自己去控制，甚至可以临时调度更多的机器并发处理这些请求，这样就不会让整个系统崩溃了。</p>

	<p>注：现实中互联网平台的每秒并发请求可以达到千万级。</p>	
基本要求	<p>定义顺序队列类型，使其具有如下功能：</p> <ol style="list-style-type: none"> (1) 建立一个空队列； (2) 释放队列空间，将队列销毁； (3) 将队列清空，变成空队列； (4) 判断队列是否为空； (5) 返回队列内的元素个数； (6) 将队头元素弹出队列（出队）； (7) 在队列中加入一个元素（入队）； (8) 从队头到队尾将队列中的元素依次输出。 <p>实验要求：</p> <ol style="list-style-type: none"> (1) 程序要添加适当的注释，程序的书写要采用缩进格式。 (2) 程序要具在一定的健壮性，即当输入数据非法时，程序也能适当地做出反应，如 插入删除时指定的位置不对 等等。 (3) 程序要做到界面友好，在程序运行时用户可以根据相应的提示信息进行操作。 (4) 根据实验报告模板详细书写实验报告，在实验报告中给出主要算法的复杂度分析。 	
选做要求	<p>无</p>	
	已完成选做内容（序号）	无

```
class QNode {
protected:
    QElemType data;    //数据域
    QNode* next; //指针域
public:
    friend ostream& operator<<(ostream& out, const QElemType& q);
    friend class LinkQueue;
    friend class ProcessSystem;
    //不定义任何函数，相当于struct LNode
};

class LinkQueue {
protected:
    QNode* front; //头指针
    QNode* rear;  //尾指针
public:
    /* P.59-60的抽象数据类型定义转换为实际的C++语言 */
    LinkQueue(); //构造函数，替代InitQueue
    ~LinkQueue(); //析构函数，替代DestroyQueue
    Status ClearQueue(); //清空队列
    Status QueueEmpty(); //判断队列是否为空
    int QueueLength(); //求队列长度
    Status GetHead(QElemType& e); //取队列头元素
    Status EnQueue(QElemType e); //入队
    Status DeQueue(QElemType& e); //出队
    Status QueueTraverse(Status(*visit)(QElemType e)); //遍历队列
    friend class ProcessSystem;
};

class ProcessSystem
{
private:
    string name[SUB_SYSTEM_NUM]; //各台处理器名称
    int speed[SUB_SYSTEM_NUM]; //各台处理器速度
    int count[SUB_SYSTEM_NUM]; //各台处理器已处理条数（1s内已处理）
public:
    ProcessSystem(const char* n[SUB_SYSTEM_NUM], const int
s[SUB_SYSTEM_NUM]);
    void clear();
    Status process(class LinkQueue& queue); //一秒钟内处理speed条消息
};
```

功能(函数)说明	<p>(1) LinkQueue 类成员函数</p> <pre> LinkQueue(); //构造函数, 替代InitQueue ~LinkQueue(); //析构函数, 替代DestroyQueue Status ClearQueue();//清空队列 Status QueueEmpty();//判断队列是否为空 int QueueLength();//求队列长度 Status GetHead(QElemType& e);//取队列头元素 Status EnQueue(QElemType e);//入队 Status DeQueue(QElemType& e);//出队 Status QueueTraverse(Status(*visit)(QElemType e));//遍历队列 </pre> <p>(2) ProcessSystem 成员函数</p> <pre> ProcessSystem(const char* n[SUB_SYSTEM_NUM],const int s[SUB_SYSTEM_NUM]); //构造函数 void clear();//清空重置 Status process(class LinkQueue& queue);//一秒钟内处理speed条消息 </pre>
界面设计和使用说明	<p>如图所示, 使用者仅需连续按下回车即可查看模拟的消息队列。 1 秒内并发的消息条数、各处理器每秒处理消息条数是代码中预设的常量。</p> <div data-bbox="255 943 1366 1319">  </div> <div data-bbox="255 1319 1315 2040">  </div>

调试分析	<p>调试分析验证了程序的正确性，健壮性。</p> <pre> 淘宝4号服务器分系统处理了一条消息，消息编号：No. 06486 wukhffhpbzkw 淘宝5号服务器分系统处理了一条消息，消息编号：No. 06487 zltilicsncv 淘宝4号服务器分系统处理了一条消息，消息编号：No. 06488 fvorrivzsxb 淘宝5号服务器分系统处理了一条消息，消息编号：No. 06489 rjoyyenicub 淘宝4号服务器分系统处理了一条消息，消息编号：No. 06490 rjdoecorjao 淘宝5号服务器分系统处理了一条消息，消息编号：No. 06491 rjhuwuagcaz 淘宝4号服务器分系统处理了一条消息，消息编号：No. 06492 uwffowhhdvg 淘宝5号服务器分系统处理了一条消息，消息编号：No. 06493 pagpnltcmhl 淘宝4号服务器分系统处理了一条消息，消息编号：No. 06494 nznxpkrdanu 淘宝5号服务器分系统处理了一条消息，消息编号：No. 06495 kbuaikkvgut 淘宝4号服务器分系统处理了一条消息，消息编号：No. 06496 uzqbofjbajm 淘宝5号服务器分系统处理了一条消息，消息编号：No. 06497 hvuforsjtum 淘宝4号服务器分系统处理了一条消息，消息编号：No. 06498 yovdlebutlj 淘宝5号服务器分系统处理了一条消息，消息编号：No. 06499 dpochvkadwm 第1秒~第2秒演示开始，按回车键继续 </pre>
心得体会	<p>程序通过建立 LinkQueue 链队列类的友元类 ProcessSystem，模拟大型网站平台的并发消息处理。考虑到需要做频繁插入删除，因此选择了链式结构。建立空队列、判断队列是否为空、入队、出队、取队首元素的算法复杂度为 $O(1)$，清空、销毁、遍历队列和求队列长度的算法复杂度为 $O(n)$。</p> <p>对消息队列的应用场景略作了解。</p> <p>解耦，生产端和消费端不需要相互依赖</p> <p>异步，生产端不需要等待消费端响应，直接返回，提高了响应时间和吞吐量</p> <p>削峰，打平高峰期的流量，消费端可以以自己的速度处理，同时也无需在高峰期增加太多资源，提高资源利用率</p> <p>提高消费端性能。消费端可以利用 buffer 等机制，做批量处理，提高效率。</p>
代码实现	<pre> #include <iostream> #include <cstring> #include <iomanip> #include <cstdlib> //malloc/realloc函数 #include <conio.h> using namespace std; /* P.10 的预定义常量和类型 */ #define TRUE 1 #define FALSE 0 #define OK 1 #define ERROR 0 #define INFEASIBLE -1 #define LOVERFLOW -2 //避免与<math.h>中的定义冲突 #define SUB_SYSTEM_NUM 5 //子系统数量 #define TOTAL_MSG 8000 //消息总数 typedef int Status; #define textlen 12 </pre>

```

/* P.61 形式定义 */
typedef struct msg {
    int No;
    char text[textlen];
}QElemType; //可根据需要修改元素的类型

class LinkQueue; //提前声明，因为定义友元要用到
class ProcessSystem;

class QNode {
protected:
    QElemType data;    //数据域
    QNode* next; //指针域
public:
    friend ostream& operator<<(ostream& out, const QElemType& q);
    friend class LinkQueue;
    friend class ProcessSystem;
    //不定义任何函数，相当于struct LNode
};

ostream& operator<<(ostream& out, const QElemType& q)
{
    out << "消息编号: No." << setw(5) << setfill('0') << setiosflags(ios::right)
    << q.No << "    " << q.text;
    return out;
}

class LinkQueue {
protected:
    QNode* front; //头指针
    QNode* rear;  //尾指针
public:
    /* P.59-60的抽象数据类型定义转换为实际的C++语言 */
    LinkQueue(); //构造函数，替代InitQueue
    ~LinkQueue(); //析构函数，替代DestroyQueue
    Status ClearQueue();//清空队列
    Status QueueEmpty();//判断队列是否为空
    int QueueLength();//求队列长度
    Status GetHead(QElemType& e);//取队列头元素
    Status EnQueue(QElemType e);//入队
    Status DeQueue(QElemType& e);//出队
    Status QueueTraverse(Status(*visit)(QElemType e));//遍历队列
    friend class ProcessSystem;

```



```

};

/* 初始化队列，替代InitQueue */
LinkQueue::LinkQueue()
{
    /* 申请头结点空间，赋值给头/尾指针 */
    rear = front = new QNode;
    if (front == NULL)
        exit(LOVERFLOW);

    front->next = NULL; //头结点的next域
}

/* 销毁队列，替代DestroyQueue */
LinkQueue::~~LinkQueue()
{
    /* 整个链表(含头结点)依次释放
       没有像链表、栈等借助 QNode *p, *q; 而直接借用了front和rear */
    while (front) { //若链表为空，则循环不执行
        rear = front->next; //抓住链表的下一个结点
        delete front;
        front = rear;
    }

    rear = front = NULL;
}

/* 清空队列（保留头结点） */
Status LinkQueue::ClearQueue()
{
    QNode* q, *p = front->next; //指向首元

    /* 从首元结点开始依次释放 */
    while (p) {
        q = p->next; //抓住链表的下一个结点
        delete p;
        p = q;
    }

    front->next = NULL; //头结点的next域置NULL
    rear = front;      //必须要，否则尾指针指向不正确
    return OK;
}

```

```

/* 判断是否为空队列 */
Status LinkQueue::QueueEmpty()
{
    #if 1
        /* 判断front和rear指针是否相等 */
        if (front == rear)
            return TRUE;
        else
            return FALSE;
    #else
        /* 判断头结点的next域即可 */
        if (front->next == NULL)
            return TRUE;
        else
            return FALSE;
    #endif
}

/* 求队列的长度 */
int LinkQueue::QueueLength()
{
    int len = 0;
    QNode* p = front->next; //指向首元

    while (p) {
        len++;
        p = p->next;
    }
    return len;
}

/* 取队头元素 */
Status LinkQueue::GetHead(QElemType& e)
{
    /* 空队列则返回 */
    if (front == rear) //用front->next==NULL也可以
        return ERROR;

    /* 取首元结点的值 */
    //e = front->next->data;
    memcpy(&e, &(front->next->data), sizeof(QElemType));
    return OK;
}

```



```

/* 元素入队列 */
Status LinkQueue::EnQueue(QElemType e)
{
    QNode* p;

    p = new QNode;
    if (p == NULL)
        return LOVERFLOW;

    //p->data = e;
    memcpy(&(p->data), &e, sizeof(QElemType));
    p->next = NULL; //新结点的next必为NULL
    rear->next = p; //接在当前队尾的后面
    rear = p;      //指向新的队尾

    return OK;
}

/* 元素出队列 */
Status LinkQueue::DeQueue(QElemType& e)
{
    QNode* p;

    /* 空队列则返回 */
    if (front == rear) //用front->next==NULL也可以
        return ERROR;

    p = front->next; //指向首元
    front->next = p->next; //front指向新首元, 可能为NULL
    /* 如果只有一个结点, 则必须修改尾指针 */
    if (rear == p)
        rear = front;

    /* 返回数据并释放结点 */
    //e = p->data;
    memcpy(&e, &(p->data), sizeof(QElemType));
    delete p;

    return OK;
}

/* 遍历队列 */
Status LinkQueue::QueueTraverse(Status(*visit)(QElemType e))

```

```

{
    //extern int line_count; //在main中定义的打印换行计数器（与算法无关）
    QNode* p = front->next;

    //line_count = 0;    //计数器恢复初始值（与算法无关）
    while (p && (*visit)(p->data) == TRUE)
        p = p->next;

    if (p)
        return ERROR;

    cout << endl; //最后打印一个换行，只是为了好看，与算法无关
    return OK;
}

Status Myvisit(QElemType e)
{
    cout << e;
    return OK;
}

class ProcessSystem
{
private:
    string name[SUB_SYSTEM_NUM]; //各台处理器名称
    int speed[SUB_SYSTEM_NUM]; //各台处理器速度
    int count[SUB_SYSTEM_NUM]; //各台处理器已处理条数（1s内已处理）
public:
    ProcessSystem(const char* n[SUB_SYSTEM_NUM], const int
s[SUB_SYSTEM_NUM])
    {
        for (int i = 0; i < SUB_SYSTEM_NUM; i++)
        {
            name[i] = n[i];
            speed[i] = s[i];
            count[i] = 0;
        }
    };
    void clear()
    {
        memset(count, 0, sizeof(int) * SUB_SYSTEM_NUM);
    };
    Status process(class LinkQueue& queue) //一秒钟内处理speed条消息
    {

```

```

        bool flag = false;
        for (int i = 0; !queue.QueueEmpty(); i = (i + 1) %
SUB_SYSTEM_NUM)
        {
            if (!i)
                flag = false;
            if (i == SUB_SYSTEM_NUM - 1 && !flag)
            {
                clear();
                return ERROR;
            }
            if (count[i] >= speed[i])
                continue;
            flag = true;
            QElemType q;
            queue.DeQueue(q);
            cout << name[i] << "分系统处理了一条消息，" << q << endl;
            count[i]++;
        }
        clear();
        return OK;
    }
};

const char* sysname[SUB_SYSTEM_NUM] = {
    "淘宝1号服务器",
    "淘宝2号服务器",
    "淘宝3号服务器",
    "淘宝4号服务器",
    "淘宝5号服务器",
};

const int sysspeed[SUB_SYSTEM_NUM] = { 1000,1200,1300,1500,1800 };

ProcessSystem PS(sysname, sysspeed);
LinkQueue RequestQueue;

int main()
{
    srand((unsigned int)(time(0)));
    for(int i = 0; i < TOTAL_MSG; i++)
    {
        QElemType q;
        q.No = i;
    }
}

```

```

        for (int j = 0; j < textlen - 1; j++)
            q.text[j] = 'a' + rand() % 26;
        q.text[textlen - 1] = '\0';
        RequestQueue.Enqueue(q);
    }
    cout << "已经随机生成" << TOTAL_MSG << "条消息进入消息队列" <<
endl;
    cout << "按回车键开始演示消息队列将" << TOTAL_MSG << "条消息分发给"
    << SUB_SYSTEM_NUM << "个子系统完成" << endl;
    while (getchar() != '\n')
        ;
    int sectime = 0;
    while (!RequestQueue.QueueEmpty())
    {
        cout << "第" << sectime++ << "秒~第" << sectime << "秒演示开始，"
        按回车键继续" << endl;
        while (getchar() != '\n')
            ;
        PS.process(RequestQueue);
    }
    return 0;
}

```