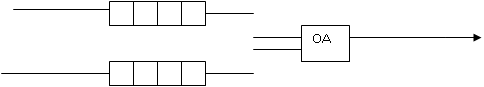
**Техническое задание**

**Задача**: Система массового обслуживания состоит из обслуживающего аппарата (ОА) и двух очередей заявок двух типов.

 Т1

Т3

Т2 Т4

Заявки 1-го и 2-го типов поступают в "хвосты" своих очередей по случайному закону с интервалами времени Т1 и Т2, равномерно распределенными от 1 до 5 и от 0 до 3 единиц времени (е.в.) соответственно. В ОА они поступают из "головы" очереди по одной и обслуживаются также равновероятно за времена Т3 и Т4, распределенные от 0 до 4 е.в. и от 0 до 1 е.в. соответственно, после чего покидают систему. (Все времена – вещественного типа)

В начале процесса в системе заявок нет.

Заявка любого типа может войти в ОА, если:

 а) она вошла в пустую систему;

 б) перед ней обслуживалась заявка ее же типа;

 в) перед ней из ОА вышла заявка другого типа, оставив за собой пустую очередь (система с чередующимся приоритетом).

Смоделировать процесс обслуживания первых 1000 заявок 1-го типа, выдавая после обслуживания каждых 100 заявок информацию о текущей и средней длине каждой очереди, а в конце процесса - общее время моделирования и количество вошедших в систему и вышедших из нее заявок обоих типов. Проследить, возникает ли при этом фрагментация памяти.

**Входные данные**: интервалы времен Т1, Т2, Т3, Т4; N - число заявок первого типа для обработки; P - каждые P обработанных заявок первого типа делать дамп информации.

**Выходные данные**: Дамп + краткая сводка по моделируемому процессу.

**Функция программы**: реализация системы обработки заявок из двух очередей, с чередующимся приоритетом.

**Замечание**: написаны две версии программы – с реализацией очереди через список, и через массив (фиксированной длины). При реализации очереди списком доступна функция проверки памяти на фрагментацию.

**Сколько вошло и сколько вышло  
количество вхождений  
не округлять  
в промежуточном сколько уже обработано**

**В конце добавить текущую длину 1ой и 2ой очереди**

**При проверке будет менять входные т1 т2 т3 т4**

**Используемые структуры**

В реализации очередью:

enum req\_type {*NONE*, *T1*, *T2*}; //типы заявок

typedef struct qNode { //заявка

struct qNode \*next; //указатель на заявку впереди

enum req\_type type; //тип заявки

float time; //сколько ждать следующую

} qNode;

typedef struct queue { //дескриптор очереди

struct qNode \*front, \*rear; //указатели на начало и конец

int elts; //текущее кол-во элементов

int ever\_elts;//сколько всего простояло заявок в этой очереди

} queue;

В реализации массивом:

enum req\_type {*NONE*, *T1*, *T2*}; //типы заявок

typedef struct queue {

float\* data\_store; //заявки

int front; //индекс конца

int capacity; //вместимость

int elts; //текущее кол-во элементов

int ever\_elts; //сколько всего простояло заявок в этой очереди

enum req\_type type; //тип заявок

} queue\_t;

**Функции и алгоритмы**

В реализации очередью:

Инициализация заявки. Заполнение типа и времени ожидания.

qNode \*getNewqNode(float , enum req\_type);

Создание новой очереди. Установка всех параметров на начальные пар-тры.

queue \* CreateQueue();

Добавление эл-та в очередь. Изменение конца очереди на данный элемент. Инкрементирование кол-ва текущих элементов и кол-ва всех элементов. Если очередь пуста, создается ее начало.

int Enqueue(queue \*, float , enum req\_type t);

Удаление элемента из очереди. Изменение начала очереди. Декрементирование кол-ва текущих элементов.

float Dequeue(queue \*);

Печать очереди.

void PrintQueue(queue \*);

Освобождение памяти для очереди.

void DestroyQueue(queue \*queueptr);

В реализации массивом:

Создание новой очереди. Установка всех параметров на начальные пар-тры.

queue\_t \*new\_queue(int capacity, enum req\_type t);

Добавление эл-та в очередь. Изменение конца очереди на данный элемент. Инкрементирование кол-ва текущих элементов и кол-ва всех элементов.

int Enqueue(queue\_t\* queue, float item, enum req\_type t);

Освобождение памяти для очереди.

void queue\_free(queue\_t\* queue);

Удаление элемента из очереди. Декрементирование кол-ва текущих элементов. Сдвиг всех элементов вперед.

float Dequeue(queue\_t\* queue);

Печать очереди.

void queue\_print(queue\_t\* q);

**Тесты**

Т1: 1.000 .. 5.000; mid = 3.0

Т3: 0.000 .. 4.000; mid = 2.0

Среднее время обработки заявок меньше среднего периода их добавления; моделирование производится по входу.

Ожидаемое время моделирования: 1000 \* 3 = 3000 е.в.

*Тесты при реализации очереди списком:*

|  |  |  |  |
| --- | --- | --- | --- |
| # | Число вошедших заявок 1 типа | Время моделирования по входу (е.в.) | Используемая память (байт) |
| 1 | 1015 | 3058.835 | 36816 |
| 2 | 1051 | 3078.876 | 37476 |
| 3 | 1025 | 3051.980 | 36672 |
| 4 | 1004 | 3030.083 | 35808 |
| 5 | 1028 | 3045.987 | 36732 |
| Сред. | 1024.6 | 3053.1522 | 36700 |

Расхождение: (3000 – 3053.1522) / 3000 = 1.7%

*Тесты при реализации очереди массивом:*

|  |  |  |  |
| --- | --- | --- | --- |
| # | Число вошедших заявок 1 типа | Время моделирования по входу (е.в.) | Используемая память (байт) |
| 1 | 1001 | 3029.920 | 32080 |
| 2 | 1004 | 3004.449 | 32080 |
| 3 | 1002 | 3070.628 | 32080 |
| 4 | 1000 | 3017.204 | 32080 |
| 5 | 1005 | 2989.001 | 32080 |
| Сред. | 1002,4 | 3022,22 | 32080 |

Расхождение: (3000 – 3022.22) / 3000 = 0.7%

Контрольные вопросы

*1. Что такое очередь?*

Очередь – последовательной список переменной длины. Включение элементов идёт с «хвоста» списка, исключение – с «головы» списка. Принцип работы: первым пришёл – первым вышел, First In First Out.

*2. Каким образом, и какой объем памяти выделяется под хранение очереди при различной ее реализации?*

При реализации очереди списком, под каждый новый элемент выделяется память из кучи, элементы связываются указателями.

При реализации очереди массивом, выделяется блок памяти из N\*sizeof(element) байт, где N – максимальное количество элементов в очереди, элементы следуют друг за другом последовательно.

*3. Каким образом освобождается память при удалении элемента из очереди при ее различной реализации?*

При реализации очереди списком, головной элемент считывается, указатель на «голову» очереди переходит на следующий элемент, считанный элемент удаляется.

При реализации очереди массивом, головной элемент считывается, остальные элементы массива сдвигаются на 1 – длина очереди уменьшается на 1, элемент [1] массива «затирает» головной элемент [0].

*4. Что происходит с элементами очереди при ее просмотре?*

При просмотре очереди, головной элемент из неё удаляется. Остальные элементы сдвигаются (массив), либо указатель на начало передвигается на следующий элемент (список).

*5. Каким образом эффективнее реализовывать очередь. От чего это зависит?*

При реализации очереди списком, проще всего добавлять и удалять из неё элементы, однако может возникнуть фрагментация памяти. При реализации очереди массивом дефрагментации не возникает, однако может возникнуть переполнение памяти, а добавление и удаление элементов сложнее. Способ реализации зависит от того, в чем мы сильнее ограничены – в памяти или во времени выполнения операций.

*6. В каком случае лучше реализовать очередь посредством указателей, а в каком – массивом?*

Очередь лучше реализовывать с помощью указателей, если новые элементы в среднем появляются реже, чем происходит полное очищение очереди – в общем случае фрагментация не возникает. Реализация с помощью указателей применима если требуется строгий контроль фрагментации.

*7. Каковы достоинства и недостатки различных реализаций очереди в зависимости от выполняемых над ней операций?*

При реализации очереди массивом не возникает фрагментации памяти, однако может произойти переполнение очереди, а также затрачивается дополнительное время на сдвиг элементов. Сдвига можно избежать, если использовать кольцевой массив, однако при этом усложняются операции добавления и удаления элементов. Наконец, при реализации списком проще всего реализуются алгоритмы добавления и удаления элементов, но может возникнуть фрагментация.

*8. Что такое фрагментация памяти?*

Фрагментация – чередование участков памяти при последовательных запросах на выделение и освобождение памяти. «Занятые» участки чередуются со «свободными» - однако последние могут быть недостаточно большими для того, чтобы сохранить в них нужное данное.

*9. На что необходимо обратить внимание при тестировании программы?*

При реализации очереди списком необходимо следить за освобождением памятти при удалении элемента из очереди. Если новые элементы приходят чаще, чем удаляются старые, очередь растёт и может происходить фрагментации памяти.

*10. Каким образом физически выделяется и освобождается память при динамических запросах?*

Программа даёт запрос ОС на выделение блока памяти необходимого размера. ОС находит подходящий блок, записывает его адрес и размер в таблицу адресов, а затем возвращает данный адрес в программу.

При запросе на освобождение указанного блока программы, ОС убирает его из таблицы адресов, однако указатель на этот блок может остаться в программе. Попытка считать данные из этого блока может привести к непредвиденным последствиям, поскольку они могут быть уже изменены.