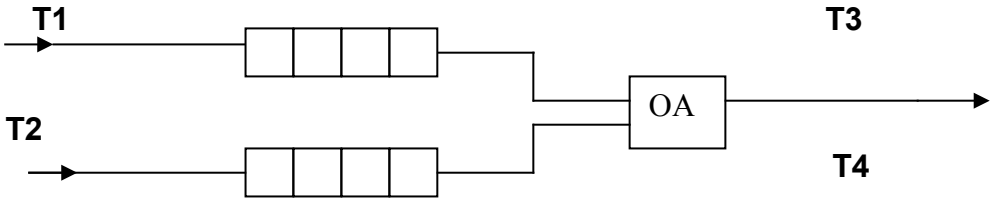
Лабораторная работа №4
Обработка очередей

Цель работы: отработка навыков работы с типом данных «очередь», представленным в виде одномерного массива и односвязного линейного списка. Сравнительный анализ реализации алгоритмов включения и исключения элементов из очереди при использовании двух указанных структур данных. Оценка эффективности программы (при различной реализации) по времени и по используемому объему памяти.

Задание (Вариант 4):

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



Заявки 1-го и 2-го типов поступают в "хвосты" своих очередей по случайному закону с интервалами времени **T1** и **T2**, 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 **T3** и **T4**, 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. По требованию пользователя выдать на экран адреса элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

Требования к входным данным:

В программе реализована возможность изменения входных параметров для обслуживающей системы: интервалы времени, кол-во заявок и интервал печати. Последние два параметра — целые, ненулевые числа. Интервалы времени задаются парами вещественных неотрицательных чисел, причем второе число должно быть не меньше первого у каждой пары.

Интерфейс программы:

```
****************************************************************
Выберите одно из следующих действий:
0: Изменить параметры/Просмотреть текущие
1: Моделирование для очереди на массиве
2: Моделирование для очереди на списке
3: Закончить работу
```

0. Программа печатает текущие параметры системы (интервалы времени, кол-во заявок и интервал печати) и предлагает пользователю их изменить (требование к вводимым изменениям см. требования к входным данным):

```
Кол-во заявок первого типа 1000
Интервал печати 100
T1 1 5
T2 0 3
T3 0 4
T4 0 1
Изменить параметры? (y-yes)
```

1. Моделирование работы ОА на очередях, реализованных на основе циклического массива фиксированной длины (START_SIZE 10000).

Вывод программы:

Для каждой итерации печатается кол-во обработанных заявок первого типа текущая длина очереди эл-тов первого типа, а так же ее средняя длина.

После выполнения моделирования печатается общее время моделирования, количество вошедших в систему и вышедших из нее заявок обоих типов.

Пример:

```
Обработано (заявки 1-го типа) 1000
Текущая длина очереди: 0
Средняя длина очереди: 6.92
Текущая длина очереди: 1
Средняя длина очереди: 29.4053
--------------------------------
Общее время моделирования 3024.94
Кол-во вошедших и вышедших (1 тип) 1000 1000
Кол-во вошедших и вышедших (2 тип) 1988 1987
Время работы 1145
************************************************************
```

2. Моделирование работы ОА на очередях, реализованных на основе односвязного списка. Список также имеет ограничение длины (реализовано в целях избежания использование всей доступной оперативной памяти при изменении пользователем параметров, равно START_SIZE). Вывод программы идентичен выводу из пункта 1.

После печати программа предложит вывести список незадействованных адресов памяти для обоих очередей.

3. Выход из программы

Обращение к программе:

Через консоль

Внутренние структура данных:

Структура списка:

```
typedef struct Node {
    request value;
    Node* next;
} my_list;
```

```
Функции для работы со списками:
my_list *create_node(request x);
my_list *add_to_list(my_list *tmp, my_list *head);
void free_all(my_list *head, my_list* memory, int m);
```

```
void print_adr(my_list *head);
Класс очереди (список)
class CQueue_list {
private:
    my_list *memory[MAX_MEMORY]; //массив указателей на свободные участки памяти
    int m;
    my_list *head; // при удалении эл-ты удаляются из головы списка
    my_list *tail; //при добавлении эл-ты попадают в хвост
    info_queue_request info;
public:
    CQueue_list();
    ~CQueue_list();
    void show();
    void show_adr();
    request PopFront();
    void PushBack(request x);
    bool Empty_CQueue();
    bool is_full();
};

Очередь с зацикленным массивом.

class CQueue_array {
    private:
        request* arr = NULL;
        int back;
        int front;
        int max_size;
        info_queue_request info;
        void increase();
    public:
        CQueue_array();
        CQueue_array(const CQueue_array &);
        ~CQueue_array();
        request PopFront();
        void PushBack(request x);
        bool Empty_CQueue();
        CQueue_array& operator= (const CQueue_array &obj);
        request * get_arr();
        void show();
        void show_adr() {}
        bool is_full();
};
```

Вычисление времени моделирования для начальных параметров:

Начальные параметры:

```
int n = 1000;
int interval = 100;
interval_time t1(1, 5);
interval_time t2(0, 3);
interval_time t3(0, 4);
interval_time t4(0, 1);
```

Среднее время обработки заявок 1-го типа: 2 е.в.
2-го типа: 0.5 е.в.
Среднее время прихода заявок 1-го типа: 3 е.в.
2-го типа: 1.5 е.в.
1000 заявок первого типа обработаются за 2 * 1000 = 2000 е.в.

время прихода этих заявок: 3*1000 е.в.
1000 е.в. будет потрачено на обработку заявок 2-го типа =>
кол-во заявок равно 1000/0.5 = 2000
Среднее время моделирования 3000 е.в.

Сравнение эффективности реализаций:

Выполнение программы:

При тестировании очереди, реализованной на списке, наблюдалась фрагментация памяти. Одной из причин возникновения фрагментации является существование двух очередей, которые последовательно заполняют определенные участки памяти, и поэтому представлены в памяти разрежено.

Массив:

№	Число заявок 1-го типа	Число заявок 2-го типа	Время моделирования (е.в.)	Время работы, мкс
1	1000	2033	3107.44	1061
2	1000	2013	2998.43	638
3	1000	2033	3065.86	625
4	1000	2002	3044.03	562
5	1000	1882	2988.4	663
6	1000	2052	3056.37	850
среднее	1000	2002.5	3043.42	733,17

Список:

№	Число заявок 1-го типа	Число заявок 2-го типа	Время моделирования (е.в.)	Время работы, мкс
1	1000	2025	3048.84	1588
2	1000	1979	3023.37	2073
3	1000	2030	3029.63	1401
4	1000	2000	3014.88	2247
5	1000	2454	3097.57	2454
6	1000	2011	3060.91	1643
среднее	1000	2083,16	3045,86	1901

Погрешность времени моделирования: 1,5%

Сравнение эффективности реализаций:

Эффективность реализации работы со списком линейно зависит от кол-ва операций добавления и извлечения эл-та из очереди. Недостатком очереди-списка явл-тся то, что при совершении этих операций он обязательно выделяет или освобождает память.

В тоже время, очередь-массив производит операции выделения/освобождения память один раз.

Также при представлении очереди в виде списка используется большее кол-во памяти для хранения указателя.

К недостаткам очереди-списка также можно отнести неравномерность распределения памяти — возникновение внутренних и внешних дыр (фрагментация памяти).

Также имеет значение, известно ли заранее максимальное кол-во эл-тов в очереди. Очередь-список позволяет воспользоваться памятью, ограниченной лишь объемом оперативной памяти компьютера. При переполнении очереди-массива, необходимо совершать дополнительно копирование и расширять размер массива.

Теоретическая часть:

1. Что такое очередь?

Очередь – это последовательный список переменной длины, включение элементов в который идет с одной стороны (с «хвоста»), а исключение – с другой стороны (с «головы»).

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

Массив: При моделировании простейшей линейной очереди на основе одномерного массива выделяется последовательная область памяти константного размера. Выделение памяти происходит в начале работы программы. В каждый текущий момент времени выделенная память может быть вся свободна, занята частично или занята полностью.

Список: Память по эл-т очереди выделяется непосредственно в процессе его добавления. Объем памяти, который занимает очередь изменяется в процессе выполнения программы и напрямую зависит от кол-ва эл-тов в очереди в каждый момент времени.

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

Массив: освобождение памяти происходит в конце работы программы (или при удалении очереди). При удалении эл-та из очереди происходит только смещение указателя head.

Список: При удалении эл-та из очереди происходит освобождение памяти, которая была выделена под этот эл-т.

4. Что происходит с элементами очереди при ее просмотре?

При просмотре очереди, хвостовой элемент из нее удаляется. При этом у очереди-списка освобождается память, которую занимал удаляемый элемент. У очереди массива перемещается указатель на следующи эл-т.

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

Способ реализации зависит от того, какие требование предъявляются программе. Если более приоритетным является быстродействие, то следует реализовывать очередь с помощью списка. Если необходимо избежать фрагментации памяти, то следует использовать очередь на списке. Также выбор способа реализации зависит от того, известно ли заранее количество эл-тов в очереди и насколько оно велико. При реализации очереди с помощью массива необходимо отслеживать его переполнение и дополнительно выделять эл-ты.

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. С другой стороны, на реализацию очереди списка затрачивается большее кол-во времени (на добавления)

8. Что такое фрагментация памяти?

В процессе моделирования очереди может оказаться, что при последовательных запросах на выделение и освобождении памяти под очередной элемент выделяется не та память, которая была только что освобождена при удалении элемента. Участки свободной и занятой памяти могут чередоваться, т.е. может возникнуть фрагментация памяти.

9. На что необходимо обратить внимание при тестировании программы?

При тестировании программы необходимо обратить внимание на фрагментацию памяти при реализации очереди списком, переполнение очереди для очереди-массива, переполнение очередей при вводе некорректных параметров работы ОА, которое приводит к увеличению скорости заполнения очередей.

10. Каким образом физически выделяется и освобождается память при динамических

запросах?

Программа дает 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.