Лабораторная работа №4

1. Задание:

Система массового обслуживания состоит из обслуживающего аппарата (ОА) и очереди заявок. Заявки поступают в "хвост" очереди по случайному закону с интервалом времени T1, равномерно распределенным от 0 до 6 единиц времени (е.в.). В ОА они поступают из "головы" очереди по одной и обслуживаются также равновероятно за время T2 от 0 до 1 е.в., Каждая заявка после ОА с вероятностью P=0.8 вновь поступает в "хвост" очереди, совершая новый цикл обслуживания, а с вероятностью 1-P покидает систему. (Все времена – вещественного типа). В начале процесса в системе заявок нет. Смоделировать процесс обслуживания до ухода из системы первых 1000 заявок. Выдавать после обслуживания каждых 100 заявок информацию о текущей и средней длине очереди. В конце процесса выдать общее время моделирования и количество вошедших в систему и вышедших из нее заявок, среднее время пребывания заявки в очереди, время простоя аппарата, количество срабатываний ОА. Обеспечить по требованию пользователя выдачу на экран адресов элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

2. ТЗ:

Цель работы – приобрести навыки работы с очередями.

Входные данные.

На вход программа получает команду, запускающую процесс моделирования очереди (1 - запуск, 0 - выход).

Выходные данные.

Каждые 100 обработанных заявок программа выводит текущее состояние очереди (текущую и среднюю длину очереди). В результате работы, программа выводит общее время работы программы, количество срабатываний ОА и время, которое ОА простаивало.

Обработка ошибок.

В случае неправильного ввода команды старта, программа говорит пользователю об ошибке и предлагает ввести команду еще раз.

3. СД:

В программе были использованы специально созданные структуры для очереди через массив и через список соответственно.

Структура для массива:

```
typedef struct queue_m
{
int data[MAX_ELEM];
int pin;
int pout;
int full;
} queue_m;
```

Структуры для списка:

```
typedef struct node
{
int data;
struct node* next;
} node;


typedef struct queue_s
{
node* pin;
node* pout;
} queue_s;
```

4. Время работы функций:

Функция:	Описание функции:	Список (tick):	Массив (tick):	% (список от массива)
Inqueue	Добавление элемента в очередь	46	13	353 %
Dequeue	Удаление элемента из очереди	38	9	420 %

5. Выводы:

В результате работы была смоделирована очередь. Использовалось 2 типа реализации - очередь через массив и очередь через список. Реализация массивом занимает меньше времени на работу, зато список дает преимущество по памяти. Фрагментация памяти не возникает.

6. Контрольные вопросы:

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»). Принцип работы очереди: первым пришел – первым вышел, т. е. First In – First Out (FIFO).

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

- При реализации очереди массивом заранее выбирается максимальная длина очереди. Тогда во время выполнения программы будет сразу выделено N * int байт памяти. При реализации очереди списком память выделяется динамически, поэтому максимальное кол-во памяти, которое потребуется программа составит максимальное число элементов в очереди, умноженное на размер элемента.

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

- При реализации очереди массивом при удалении элемента элементы массива просто сдвигаются в сторону “головы” на 1 элемент. При реализации очереди списком так же происходит сдвиг элементов, но с последующем освобождением памяти со стороны “хвоста”.

4) Что происходит с элементами очереди при ее просмотре?

- Если реализация очереди не позволяет просмотреть элементы без их исключения, то элементы по очереди будут исключаться из “головы” и после просмотра поступать в “хвост”.

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

- Эффективность той или иной реализации зависит от условий использования и требований к используемым ресурсам. Если неважно кол-во используемой памяти, а важна скорость выполнения программы, то эффективней будет использовать реализацию очереди через массив. Если же необходимы минимальные затраты памяти, пусть и при более долгом выполнении программы, то оптимальнее будет использовать реализацию очереди через список.

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

- Если элементы очереди не повторяют свой цикл, то можно обойтись без использования указателей. Если же элементы очереди имеют несколько циклов (и тем более если их память гораздо больше памяти указателей), то лучше реализовать очередь через указатели, т.к. тогда сократится время на удаление/добавление одних и тех же элементов в очередь.

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

- Реализация через список с выделением динамической памяти имеет большее время выполнение программы, однако обеспечивает минимально возможное использованное кол-во памяти программой. Реализация через массив с разовым выделением необходимого кол-ва памяти имеет меньшее время выполнения программы, однако кол-во использованной памяти будет максимальным установленным, вне зависимости от того, использовалась она или нет.

8) Что такое фрагментация памяти?

- При динамическом выделении памяти, память из оперативной памяти выделяется “блоками”, причем не обязательно эти блоки находятся рядом. Таким образом может возникнуть случай, когда при запросе некоторого кол-ва памяти в оперативной памяти

не будет “блока” такого размера, который бы удовлетворил запрашиваемому кол-ву памяти.

9) На что необходимо обратить внимание при тестировании программы?

- При тестировании программы стоит обратить внимание на процесс выделения и освобождения 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.