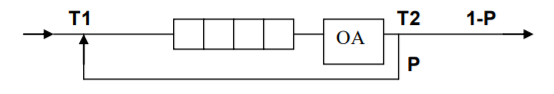
**Лабораторная работа №4**

**Условие**

Система массового обслуживания состоит из обслуживающего аппарата (ОА) и очереди заявок.



Заявки поступают в "хвост" очереди по случайному закону с интервалом времени Т1, равномерно распределенным от 0 до 6 единиц времени (е.в.). В OA они поступают из "головы" очереди по одной и обслуживаются также равновероятно за время Т2 от 0 до 1 е.в., Каждая заявка после ОА с вероятностью Р=0.8 вновь поступает в "хвост" очереди, совершая новый цикл обслуживания, а с вероятностью 1-Р 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.

**Сроки исполнения**

2 недели, начиная с 12.10.2018

**Основание для разработки**

Заказчик – Силантьева Александра Васильевна

Исполнитель – Ильясов Идрис Магомет-Салиевич, группа ИУ7-33Б

Основание для разработки – лабораторная работа №5

**Назначение разработки**

Общая концепция системы – консольное приложение.

Описание функционала системы:

Ввод и вывод производится пользователем вручную через консоль с помощью меню.

В меню пользователь может выбрать работу с очередью в виде массива или списка, проверить наличие или отсутствие фрагментации при работе со списком и сравнить время работы и используемую память при работе с массивом и списком.

Изначально стеки пустые. Максимальный размер стека – 10 элементов. Максимальный размер стека изменяется вручную в файлах.

**Требования к программе**

Программа написана полностью на языке Си.

При неправильном вводе пользователем какого-либо значения программа завершает свою работу, выдавая сообщение об ошибке.

**Входные данные**

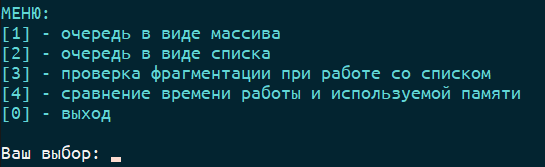
Вводится целое число – номер выбранного пользователем пункта меню. Вводится через консоль при запросе «Ваш выбор: ». Позже мной был замечен недостаток, так как не был реализован ввод времен и процента возврата.

**Выходные данные**

Данная программа выводит на экран:

* Таблицу с графами «Обработано заявок», «Текущая длина очереди», «Средняя длина очереди» после обслуживания каждых 100 заявок.
* Общее время моделирования, количество срабатываний обслуживающего аппарата, время простоя аппарата, количество вошедших и вышедших заявок.
* Процент расхождения расчетного времени и получившегося при работе программы для входа и выхода.
* При проверке пользователем фрагментации памяти при работе со списками выводится таблица с графами «Заявка» и «Адрес памяти» каждого элемента.
* Время затраченное на приход определенного количества заявок и их уход из очереди при реализации очереди массивом и списком, а также используемая память.

**Работа с программой**

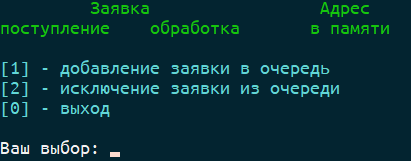


Работа с программой осуществляется в консоли с помощью меню:

[1] – очередь в виде массива. При выборе данного пункта меню программа моделирует процесс обслуживания 1000 заявок. В данном случае очередь реализована массивом. При этом выводится таблица после выхода из очереди каждых 100 заявок. Также выводится общее время моделирования, количество вошедших и вышедших заявок, количество срабатываний обслуживающего аппарата.

[2] – очередь в виде списка. При выборе данного пункта меню программа моделирует процесс обслуживания 1000 заявок. В данном случае очередь реализована линейным односвязным списком. При этом выводится таблица после выхода из очереди каждых 100 заявок. Также выводится общее время моделирования, количество вошедших и вышедших заявок, количество срабатываний обслуживающего аппарата.

[3] – проверка фрагментации при работе со списком. Открывается меню для проверки наличия или отсутствия фрагментации.



[1] – создает заявку и добавляет ее в очередь, реализованную списком. При этом выводится на экран адреса памяти, по которому заявки расположены.

[2] – исключает заявку из очереди, реализованной списком. При этом выводится на экран таблицу оставшихся в очереди заявок.

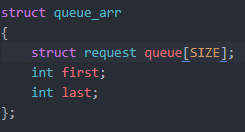
[3] – завершение работы с проверкой фрагментации при работе со списком.

[4] – сравнение времени работы и используемой памяти. При выборе данного пункта меню выводится на экран информация о времени прихода заданного количества заявок (вводится пользователем), времени ухода и используемую память при реализации очереди списком и массивом.

[0] – завершение работы с программой.

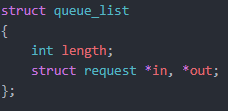
**Структуры данных и функции**

Очередь-массив представляет собой структуру:



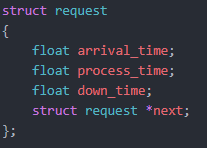
queue[SIZE] – массив запросов, first – расположение первой заявки в очереди, last – расположение последней заявки в очереди.

Очередь-список представляет собой структуру:



length – длина очереди, in – последняя заявка в очереди, out – первая заявка в очереди.

Заявка представляет собой структуру:



arrival\_time – время прихода заявки, process\_time – время обработки заявки, down\_time – время ожидания, next – следующая заявка в списке.

В программе есть следующие функции:

**Инициализация очереди:**

void init\_queue\_arr(struct queue\_arr \*queue);

void init\_queue\_list(struct queue\_list \*queue);

**Добавление заявки в очередь и ее удаление:**

int qin\_arr(struct queue\_arr \*queue, struct request new);

struct request qout\_arr(struct queue\_arr \*queue);

void qin\_list(struct queue\_list \*queue, struct request \*new);

struct request qout\_list(struct queue\_list \*queue);

**Проверка очереди на пустоту:**

int is\_empty\_queue\_arr(struct queue\_arr \*queue);

int is\_empty\_queue\_list(struct queue\_list \*queue);

**Создание новой заявки:**

struct request create\_new\_request\_arr(float t1, float t2, float t3, float t4);

void create\_new\_request\_list(struct request \*new, float t1, float t2, float t3, float t4);

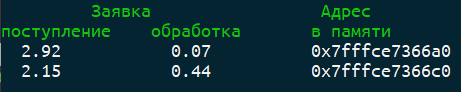
**Замер времени нахождения заявки в очереди:**

void waiting\_in\_queue\_arr(struct queue\_arr \*que, float t);

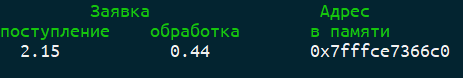
void waiting\_in\_queue\_list(struct queue\_list \*queue, float t);

**Проверка на фрагментацию памяти:**

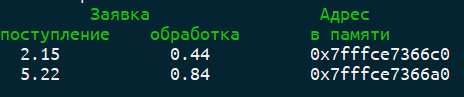
Чтобы проверить наличие или отсутствие фрагментации при работе со списками, добавим 2 заявки в очередь с адресами 0x7fffce7366a0, 0x7fffce7366c0.



Удалим одну заявку с адресом 0x7fffce7366a0.

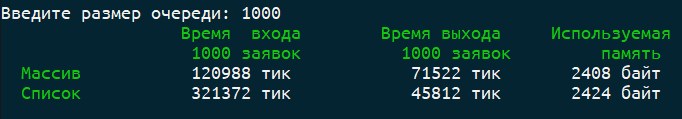
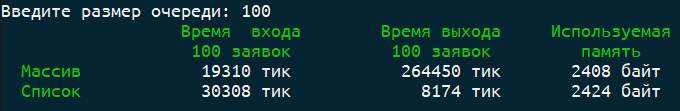
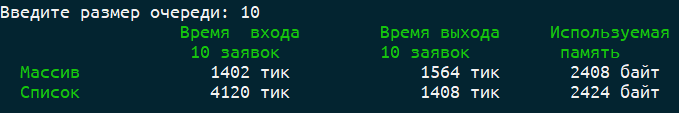


При добавлении новой заявки она будет находится по адресу 0x7fffce7366a0.

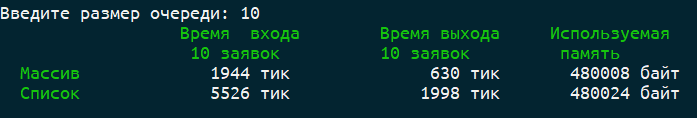


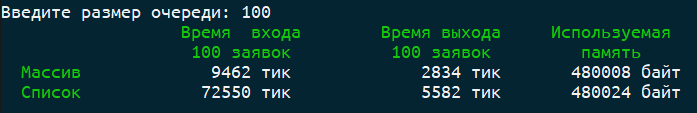
Как мы видим, фрагментации памяти при работе с очередью, реализованной списком не наблюдается.

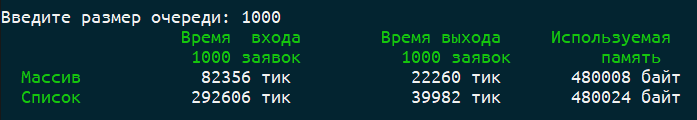
**Сравнение времени и памяти**



Мной был реализован неэффективный алгоритм выхода заявки из очереди (при удалении сдвигались все элементы массива). Эффективнее в данном случае реализовать простой перенос первого элемента на элемент вперед (не сдвигая при этом сами элементы).







При реализации очереди массивом и списком размер используемой памяти не сильно отличается.

При реализации массивом размер очереди ограничен до SIZE (размер массива). Значит, возможно переполнение очереди. При реализации очереди списком размер очереди ограничен размером оперативной памяти. При работе с программой (реализация списком) возможна фрагментация памяти, но на тестируемом ПК таковой не наблюдалось.

При первом методе удаления элемента, время на удаления в массиве затрачивалось больше, чем на список. Во втором способе удаления, элементы из массива удаляются быстрее.

**Модули и тесты**

**Функциональные тесты**

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Вывод** | **Класс эквивалентности** |
| Меню: y | Сообщение об ошибке, завершение программы | Некорректный ввод в меню |
| Меню: 3 | Успешный ввод | Корректный ввод в меню |

**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 (FIFO).

*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]. При реализации более эффективного метода, индекс первого элемента переносится вперед.

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

При просмотре очереди, головной элемент из неё удаляется (вытаскивается). 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 фрагментации не возникает, однако может возникнуть переполнение памяти, а добавление и удаление элементов сложнее. Способ реализации зависит от того, в чем мы больше 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 нужные данные.

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

При реализации очереди списком необходимо следить за освобождением памяти 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.