**МГТУ им. Н.Э.Баумана**

**Дисциплина “Типы и структуры данных”**

**Отчет по Лабораторной работе №5 по теме “** Обработка очередей**”.**

Работу выполнил:

Студент: Зейналов Зейнал

ИУ7-31Б

Варинат 14(7)

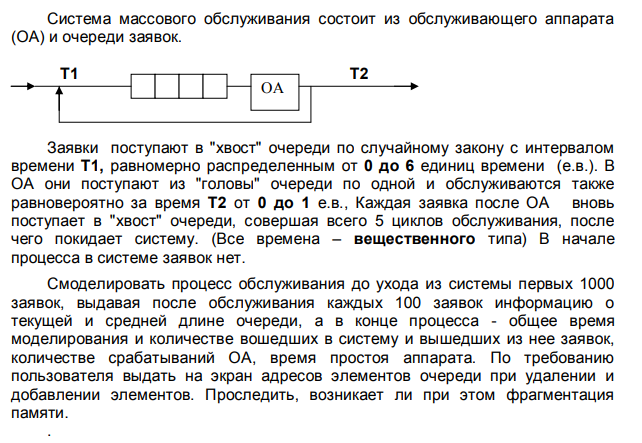
Москва, 2018.

Работа № 5

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.

Описание условия задачи



Описание ТЗ

Общая концепция системы

Программа выполняет операции над очередью, реализованной двумя способами. Очередь генерируется компьютером. Моделируется ситуация, описанная в условии задачи. Пользователь может выбирать интервалы времени  и способ реализации.

### Требования к функциональным характеристикам

Программа должна выполнять следующие функции:

* возможность изменения времен прихода и обработки заявок с экрана;
* наличие пояснений при выводе результата;
* вывод процента погрешности работы программы.
* Добавление элемента в очередь для двух реализаций
* удаление элемента из очереди для двух реализаций
* моделирование ситуации
* вывод времени работы программ
* вывод адресов памяти, задействованных при выполнении
* сравнение двух реализаций

На **вход** программа получает действие (номер пункта меню). При необходимости элементы типа int, для ограничения временных промежутков.

**Выход** должен быть представлен в виде набора данных. Также программа должна вывести данные о времени выполнения алгоритма.

Программа должна выдавать корректный ответ при вводе любых данных. Если произошла ошибка ввода, программа должна сообщить об этом.

## Способ обращения к программе

Программа представляет собой файл app.exe. Запускается в консоли. Для запуска достаточно команды ./app.exe. Если файл отсутствует можно собрать его с помощью утилиты make.

Аварийные ситуации:

* Ввод несуществующего пункта меню  
  Программа выведет информацию о неверном вводе пункта меню
* Неверный ввод интервалов моделирования (например (-1 10) или (10 1)  
  Программа выведет информацию о неверном вводе интервалов времени
* Переполнение очереди   
  Программа выведет информацию о том, что очередь переполнена

Описание структур данных

// очередь-список

*typedef* *struct* List List\_t;

*struct* List

{

int entry;

double time;

List\_t \*next;

};

// Очередь-массив

*typedef* *struct*

{

int entry;

double time;

}QUEUE;

double eps = 0.00001;

int code\_error = OK;

// Кол - во вхождений элемент в ОА

int count\_req = 0;

// Кол - во запросов к очередям

int req\_in = 0, req\_out = 0;

// Времена приходы и обработки, время ожидания и общее время работы соответсвенно

double t\_in = 0, t\_out = 0, t\_wait = 0, time = 0;

// Массив - очередь

QUEUE Queue[SIZE];

// Указатель на «вход» в очередь

QUEUE \*pin = &Queue[0];

// Указатель на «выход» из очереди

QUEUE \*pout = &Queue[0];

// Время выполнения 1000 операций в тиках

unsigned long long before, after;

// Указатели на голову и хвсот списка (вход и выход)

List\_t \*head = NULL;

List\_t \*tail = NULL;

// Время для сравнения и время работы

double t\_min, t\_work = 0;

double req\_show = 0;

// Массив свободных областей памяти

void \*adresses[SIZE];

// текущее кол-во элементов в массиве свободных областей

int count\_adress = 0;

// Кол-во срабатываний аппарата

int count\_OA = 0;

// Начальное кол-во вхождений

int entries = 0;

// Среднее расчетное время моделирования

double average = ((interval1.max - interval1.min) / 2.0) \* n;

Описание алгоритма

Пока не 1000 уникальных вышедших заявок

Если t\_in == 0 (время прихода)

Найти новое время

Если время обработки = 0

Найти новое время обработки

Если список/массив пустой

Добавить к времени ожидания время прихода

Иначе

К времени работы t\_work добавить время прихода

Удалить из очереди элемент и получить кол-во его вхождений

Если кол-во вхождений == 4  
 увеличиваем кол-во обработанных заявок

Иначе

Увеличиваем кол-во вхождений

Записываем эту заявку в конец очереди

Увеличить кол-во обработанных заявок

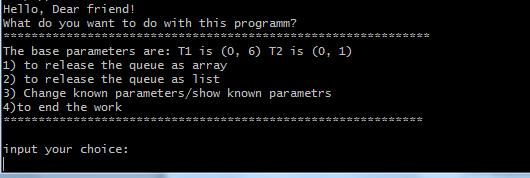
Выбрать минимальное время (обработки/прихода)

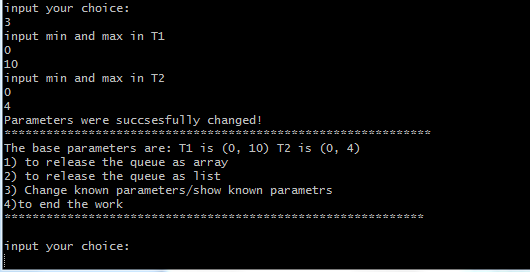
Если минимальное – время прихода

Создать новую заявку

Вычесть минимальное время из всех времен

Входные данные





Пользователь может провести моделирование на списке, на массиве,

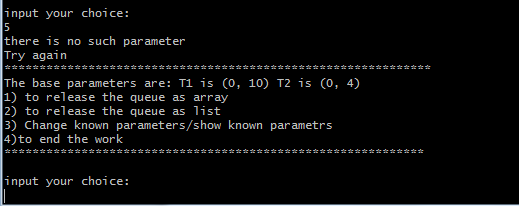
изменить интервалы прихода и обработки заявок,

просмотреть в ходе работы на время выполнения и на занимаемую память

1.Несуществующий пункт меню

Вход: 5

Вывод:

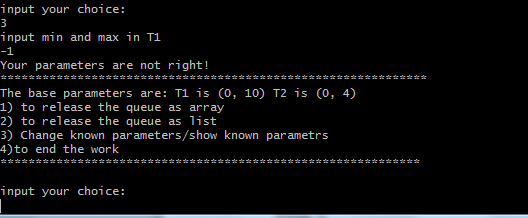


2. Некорректный ввод t (t < 0)

Вход:

n: -1

Вывод:



3. Переполнение массива

Вход:

Время прихода 0 4

Время обработки 0 1

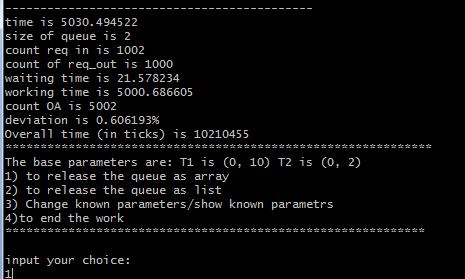
Вывод: Queue is already full

Теоретически рассчитываем время моделирования.

Время обработки заявки лежит в интервале от 0 до 2 е.в., значит среднее значение времени обработки одной заявки 1 е.в.

Так как каждая заявка обрабатывается 5 раза до выхода из системы, то среднее время обработки одной заявки будет: 1\*5=5 е.в., а общее время обработки 1000 заявок будет: 1000\*5 = 5000 е.в. Для прихода 1000 заявок, если каждая из них приходит в среднем за 5 е.в., потребуется 5000 е.в. Следовательно, ОА работает с небольшим простоем и время моделирования будет определяться временем прихода заявок, т.е. оно должно быть равно 5000 е.в. Время простоя будет равно разнице между временем обработки заявок и временем их обслуживания: 5000 – 5000 = 0000 е.в.

На практике при выполнении программы получены следующие результаты: время моделирования –



Таким образом, проверка работы системы по входным и выходным данным показала, что погрешность работы системы составляет не более 0.6%, что удовлетворяет поставленному условию.

Оценка эффективности

|  |  |  |
| --- | --- | --- |
| Выполнение модуляции  Массив  t, такты | Выполнение модуляции  Список  t, такты | % |
| 9784913 | 14731450 | 33 |
| Память  Массив  байты | Память  Список  байты | % |
| 8000 | 816 | 80% |
| Память на 1 эл очереди  Массив  байты | Память на 1 эл очереди  Список  байты | % |
| 16 | 20 | 20% |

Выводы по проделанной работе

Была реализована программа, моделирующая ситуацию с очередью, каждая заявка которой может обрабатывается по 5 раз, до тех пор, пока не обработалось 1000 уникальных заявок.

Были реализованы функции добавления и удаления элементов для двух способов реализации очередей: списком и массивом.

Реализация Списком проигрывает по времени реализации массивом на 33%. Однако выигрывает по памяти на 80%, это обусловлено тем, что память под массив выделяется статически, а под очередь списком - динамически. Таким образом, реализация списком проигрывает реализации массивом по времени. Но с массивом сложнее отслеживать конец очереди .

# Ответы на 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, память выделяется только под элементы массива.

Она выделяется единовременно под весь массив фиксированной длины.

При реализации списком память выделяется под элемент и его адрес. Она выделяется динамически, каждый раз при добавлении элемента.

3. Каким образом освобождается память при удалении элемента из очереди при

ее различной реализации?

При реализации массива память не освобождается до конца работы с очередью.

При реализации списком память освобождается при каждом удалении элемента из очереди.

4. Что происходит с элементами очереди при ее просмотре?

При просмотре очереди ее элементы удаляются.

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

Эффективнее реализовывать очередь массивом. Но, если количество заявок, которые необходимо обработать много меньше длины массива, то эффективнее реализовывать очередь списком.

6. В каком случае лучше реализовать очередь посредством указателей, а в

каком – массивом?

Очередь списком лучше реализовывать в случае, если не важно время работы или невозможно сказать какое количество памяти понадобится для хранения очереди. Массивом лучше реализовывать в том случае, если важно время обработки.

7. Каковы достоинства и недостатки различных реализаций очереди в

зависимости от выполняемых над ней операций?

Массив:

* Плюсы:
  + высокая скорость работы (при реализации циклического массива)
  + хранится только сам элемент
* Минусы:
  + необходимость сдвига данных при каждом удалении элемента(если  массив не циклический)
  + необходимость отслеживать переполнение массива

Список:

* плюсы:
  + память ограничена только памятью компьютера
* минусы:
  + скорость работы существенно ниже из-за динамического выделения памяти
  + хранится не только сам элемент, но и указатель на следующий элемент списка
  + необходимость отслеживания правильности работы с памятью

8. Что такое фрагментация памяти?

Фрагментация памяти - это ситуация, когда каждый блок памяти выделяется в новом месте, а не единым большим блоком.

9. На что необходимо обратить внимание при тестировании программы?

При тестировании программы необходимо:  
o проверить правильность работы программы при различном заполнении  
очередей, т.е., когда время моделирования определяется временем обработки  
заявок и когда определяется временем прихода заявок;  
o отследить переполнение очереди, если очередь в программе ограничена.  
  
При реализации очереди списком необходимо тщательно следить за  
освобождением памяти при удалении элемента из очереди. При этом по запросу  
пользователя должны выдаваться на экран адреса памяти, содержащие элементы  
очереди при добавлении или удалении элемента очереди. Следует проверять,  
происходит ли при этом фрагментация памяти (выделение непоследовательных  
адресов память).

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

Память выделяется с помощью функции Malloc. Эта функция выбирает доступный участок памяти нужного размера и помещает туда элемент.

Память освобождается функцией free.