Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

Отчет по лабораторной работе №5
«ОБРАБОТКА ОЧЕРЕДЕЙ»

Студент Козырнов Александр

Группа ИУ7 – 32Б

Преподаватель Барышникова М. Ю.

Вариант 6

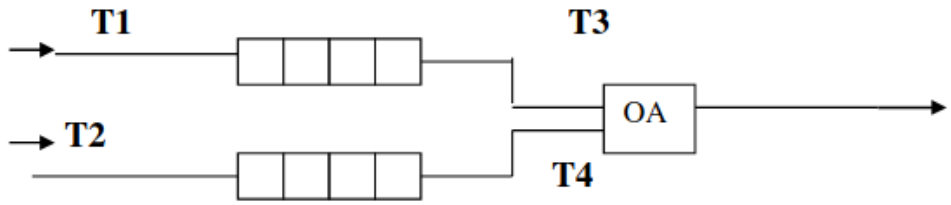
2023 г.

Оглавление

Описание условия задачи

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.

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 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 2-го типа может войти в ОА, если в системе нет заявок 1-го типа. Если в момент обслуживания заявки 2-го типа в пустую очередь входит заявка 1-го типа, то она ждет первого освобождения ОА и далее поступает на обслуживание (система с относительным приоритетом).

Смоделировать процесс обслуживания первых 1000 заявок 1-го типа. Выдать на экран после обслуживания каждых 100 заявок 1-го типа информацию о текущей и средней длине каждой очереди, количестве вошедших и вышедших заявок и о среднем времени пребывания заявок в

очереди. В конце процесса выдать 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.

ОПИСАНИЕ ТЕХНИЧЕСКОГО ЗАДАНИЯ

Входные данные:

Числовое значение номера пункта меню, выбор интервала обработки и его изменение.

Выходные данные:

Моделирование и характеристика для очередей, результат сравнения двух очередей.

Обращение к программе:

Запускается через терминал командой: ./app.exe.

Аварийные ситуации:

1. Ввод некорректного пункта меню.
2. Выбор некорректного интервала времени.
3. Некорректный ввод границ интервала обработки.

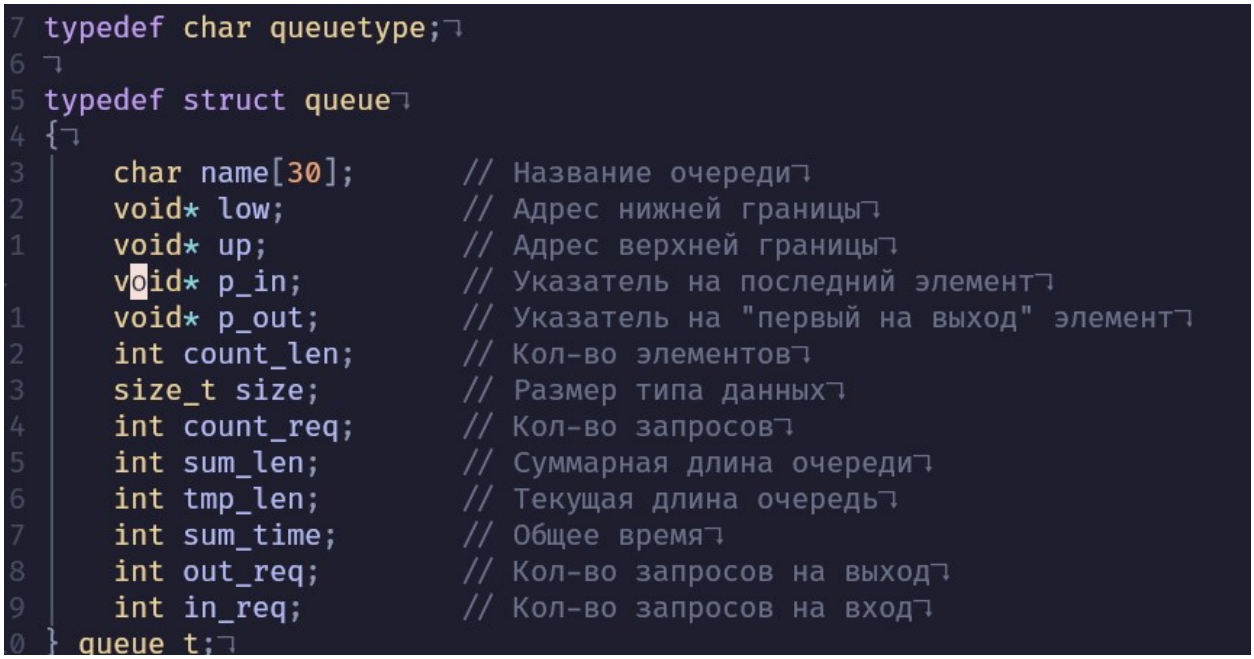
НАБОР ТЕСТОВ

№	Название теста	Пользовательский ввод	Вывод
1	Некорректный пункт меню	10	Введена некорректная

			команда, попробуйте снова.
2	Некорректный пункт меню	abacaba	Введена некорректная команда, попробуйте снова.
3	Некорректный выбор опции вывода информации о памяти	2 3	Некорректный выбор!
4	Выбор некорректного интервала (не число)	3 a	Введена некорректная команда, попробуйте снова.
5	Ввод некорректных границ интервала	3 2 10 abac	Ошибка!!! Введён некорректный номер!
6	Выбор некорректного интервала (число не в промежутке 1...4)	3 5	Введем некорректный номер!
7	Корректный вызов пункта меню моделирования	1 или 2	Вывод временной характеристики очереди, реализованной массивом/списком
8	Выход из программы	0	Выход из программы

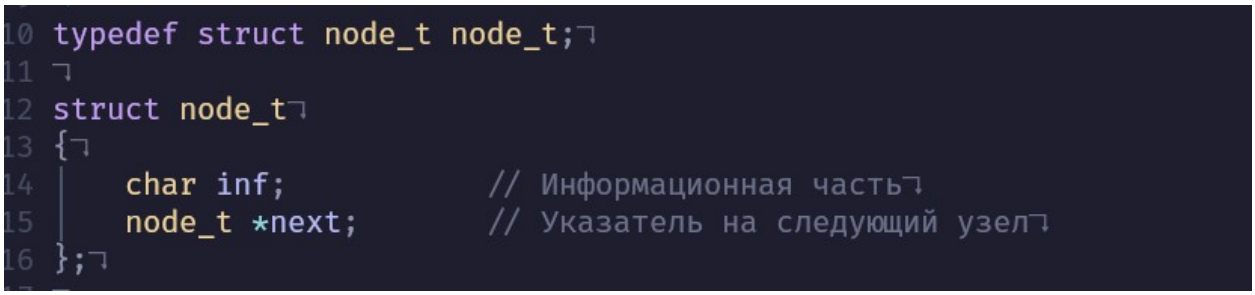
Описание структуры данных

Комбинированная структура очереди. Позволяет пользоваться как списковой структурой, так и массивом



```c
typedef char queuetype;

typedef struct queue
{
    char name[30];      // Название очереди
    void* low;          // Адрес нижней границы
    void* up;           // Адрес верхней границы
    void* p_in;         // Указатель на последний элемент
    void* p_out;        // Указатель на "первый на выход" элемент
    int count_len;      // Кол-во элементов
    size_t size;        // Размер типа данных
    int count_req;      // Кол-во запросов
    int sum_len;        // Суммарная длина очереди
    int tmp_len;        // Текущая длина очередь
    int sum_time;       // Общее время
    int out_req;        // Кол-во запросов на выход
    int in_req;         // Кол-во запросов на вход
} queue_t;
```

Структура узла очереди на списке.



```c
typedef struct node_t node_t;

struct node_t
{
    char inf;           // Информационная часть
    node_t *next;       // Указатель на следующий узел
};
```

Оценка эффективности

Реализация ОА на массиве

№	Кол-во заявок 1-го типа	Кол-во заявок 2-го типа	Время моделирования (условные е.в.)	Время моделирования (мкс)
1	1000	2019	3027.63	714
2	1000	2028	2971.85	646
3	1000	1991	2940.79	643
4	1000	1993	3037.32	690
5	1000	2014	3014.47	645
6	1001	2016	3026.30	681
7	1000	2043	3037.55	631
8	1000	1983	2986.37	684
9	1000	1950	2978.20	632
10	1002	2007	3030.99	695
Среднее	1000.3	2004.4	3005	666.1

Реализация ОА на списке

№	Кол-во заявок 1-го типа	Кол-во заявок 2-го типа	Время моделирования (условные е.в.)	Время моделирования (мкс)
1	1000	1970	3034.24	2762
2	1000	2011	3009.42	2734
3	1001	2028	3002.10	2859
4	1000	1959	2980.58	2772
5	1001	2019	3034.10	2401
6	1000	2027	3026.44	2105
7	1000	2062	3036.96	2764
8	1000	1955	2953.25	2713
9	1000	1989	2987.46	2830
10	1000	2033	2982.03	2524
Среднее	1000.2	2005.3	3004.3	2646.4

Операции над очередью

	Массив	Список
Добавление	231	1491
Удаление	189	315

Память (байт)

Кол-во элементов	Массив	Список
10	114	264
100	204	1704
1000	1104	16104
n	N + 104	16 * n + 104

Тестирование задания

Теоретический расчет времени моделирования очереди = *max*(среднее время прихода заявки 1-го типа, среднее время обработки заявки 1-го типа) * количество.

Чтобы найти время моделирования, мы должны найти максимальное время между обработкой и приходом заявки первого типа (потому что у нее относительный приоритет). В данном случае время прихода больше времени обработке, и среднее время прихода равно 3 единиц условного времени. Так как обрабатывается заявка первого типа быстрее, чем приходит, то у нас почти каждая заявка первого типа будет обработана.

Время моделирования будет равно 1000 * (ср. Время прихода первой заявки) = 3000 у. е. в. За это время успеет прийти 2000 заявок второго типа. Это находится из выражения ВремяМоделирования / max(время обр, время прих). В данном случае время прихода в срежнем больше времени обработки для второй заявки и равно 1.5. То есть 2000 заявок = 3000 / 1.5. Так как они обрабатываются быстрее, чем приходят, то почти все заявки второго типа будут обработаны во время простоя ОА. Простой ОА гарантирован, так как заявки первого типа обрабатываются быстрее, чем приходят.

Время моделирования заявок: (Расчет)
T1: 1...5
T2: 0...3

T3: 0...4
T4: 0...1

Число заявок 1 типа, вошедших: 1000, вышедших = 1000

Число заявок 2 типа, вошедших: 2000,
Вышедших = (время моделирования / max(приход, обработки)) = 3000 / max(1.5, 0.5) = 2000

Время моделировния: 3000

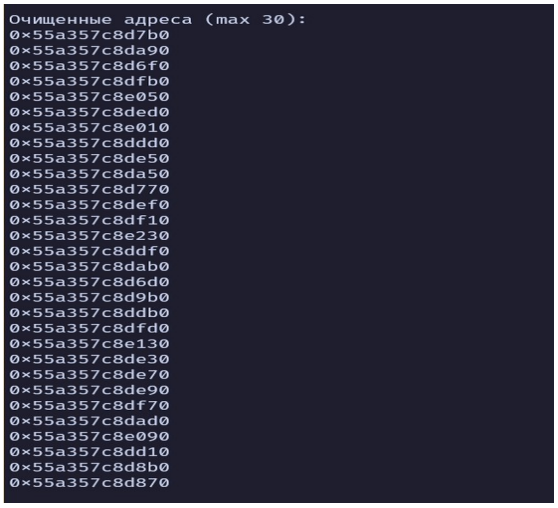
Практические результаты на массиве:

```
Общее время моделирования: 2978.404501
Погрешность работы ОА: 0.719850%

Среднее время обработки заявки 1 очереди: 3.000000
Среднее время обработки заявки 2 очереди: 1.500000
Число вошедших в 1 очередь: 1000
Число вышедших из 1 очереди: 1000
Число вошедших во 2 очередь: 1966
Число вышедших из 2 очереди: 1930
Время работы (мкс): 836

Погрешность ввода 1 очереди: 0.725069%
Погрешность ввода 2 очереди: 0.987257%
Время простоя ОА: 14.310019
```

Практические результаты на листе:

```
Общее время моделирования: 3019.180037
Погрешность времени моделирования: 0.639335%

Число вошедших в 1 очередь: 1001
Число вышедших из 1 очереди: 1000
Число вошедших во 2 очередь: 2027
Число вышедших из 2 очереди: 1986
Время работы (мкс): 2500
Среднее время обработки заявки 1 очереди: 3.000000
Среднее время обработки заявки 2 очереди: 1.500000
Погрешность ввода 1 очереди: 0.535908%
Погрешность ввода 2 очереди: 0.706151%
Время простоя ОА (в усл. ед. в.): 10.787760
```

В случае реализации ОА на списке возникает фрагментация – «дырок» выходит больше кол-ва всех поступивших в очередь элементов.



Описание алгоритма

1. После запуска программы пользователю предлагается ввести пункт меню.

1. Пользователь вводит вещественные или целые данные, в зависимости от пункта меню.

2. Программа продолжает работу до момента, пока пользователь на запрос ввода пункта меню не введёт 0.

Ответы на контрольные вопросы

1. Что такое FIFO и LIFO?

FIFO расшифровывается как First In First Out (Первым Зашел, Первым вышел). Означает, что при добавлении в хранилище этот элемент обязательно выйдет первым. Используется в структуре «Очередь».

LIFO расшифровывается как Last In First Out (Последним зашел, Первым вышел). Означает, что при добавлении в хранилище этот элемент выйдет последним (если зашел первым), или выйдет первым, если зашел последним. Ассоциируется со стопкой книг (самая нижняя книга будет взята последней, когда самая верхняя — первой). Используется в структуре «Стек».

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

При реализации списком, под каждый новый элемент выделяется память размером sizeof(элемента) + 8 байт (для указателя) в куче, для каждого элемента отдельно. При реализации массивом, (кол-во элементов) * sizeof(элемента). Если массив статический, то память выделяется в стеке, если массив динамический, то - в куче.

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

При удалении элемента из очереди в виде массива, перемещается указатель, память не освобождается. Память освобождается в конце программы. Если массив статический, то после завершении программы, если динамический — с помощью функции free().

При удалении элемента из очереди в виде списка, освобождается память из данного элемента сразу. (Указатель на «голову» переходит на следующий элемент, считанный элемент удаляется, память освобождается)

4. Что происходит с элементами очереди при её просмотре?

При просмотре очереди, головной элемент («голова») удаляется, и указатель смещается. То есть при просмотре очереди ее элементы удаляются.

5. **От чего зависит эффективность физической реализации очереди?**

Различные типы очередей (например, очередь на основе массива или связанного списка) могут иметь разную эффективность в зависимости от операций, которые вы часто выполняете. Например, массивы обычно обеспечивают быстрый доступ к элементам по индексу, но могут быть неэффективными для операций вставки и удаления в середине очереди.

Эффективность может зависеть от размера очереди и ее емкости. Если очередь часто достигает своей максимальной емкости и требует динамического изменения размера, это может повлиять на производительность.

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

При реализации очереди в виде массива не возникает фрагментация памяти, так же может возникнуть переполнение очереди, и тратиться дополнительное время на сдвиги элементов (классический массив).

При реализации очереди в виде списка, проще выполнять операции добавления и удаления элементов, но может возникнуть фрагментация памяти.

7. Что такое фрагментация памяти, и в какой части ОП она возникает?

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.

Это происходит внутри физической оперативной памяти компьютера. Оперативная память делится на непрерывные блоки, и фрагментация может возникнуть, когда эти блоки становятся разбросанными по всей памяти. Это может быть вызвано, например, разными процессами, которые динамически

выделяют и освобождают память, что приводит к разрывам между занятыми и свободными блоками памяти.

8. Для чего нужен алгоритм «близнецов».

Нужен для ускорения выделения блока динамической памяти.

9. Какие дисциплины выделения памяти вы знаете?

Статическое выделение памяти (автоматическое на стеке), динамическое выделение памяти (на куче).

10. На что необходимо обратить внимание при тестировании программы?

При реализации очереди в виде списка необходимо следить за освобождением памяти при удалении элемента из очереди. Если новые элементы приходят быстрее, чем уходят старые, то может возникнуть фрагментация памяти.

При реализации очереди в виде массива надо обратить внимание на корректную работу с ним, чтобы не произошло записи в невыделенную память.

11. 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 указного блока программы, ОС убирает его из таблицы адресов, однако указатель на этот блок может остаться в программе. Обращение к этому адресу и попытка считать данные из этого блока может привести к неопределенному поведению, так как данные могут быть уже изменены.

Вывод

К недостаткам очереди в виде списка можно отнести то, что используется большее количество памяти, так как помимо самих элементов

необходимо хранить указатели. Также при работе с очередями (на списках) может возникнуть фрагментация памяти. К преимуществам можно отнести тот факт, что очередь (на списке) позволяет воспользоваться памятью, ограниченной лишь объёмом оперативной памяти компьютера, а также операции удаления и добавления элемента в очередь легче реализовать, чем с очередью (на массиве), но при выполнении этих операций выполняется выделение или освобождение памяти, что может перевести к ошибке.

К недостаткам очереди в виде массива можно отнести то, что такая очередь будет ограничена по памяти и может возникнуть переполнение. Преимущество очереди на массиве над очередью на списке — операции удаления и добавления элемента выполняются намного быстрее. Следует учитывать, что в данной реализации использовался кольцевой массив. Если использовать обычный – операция удаления элемента будет проигрывать по скорости списку, т.к. придётся «двигать» все элементы этого массива.