|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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 «Работа с очередью» (ВАР. 7)**

Студент Смирнов Иван Владимирович

Группа ИУ7 – 32Б

Проверил Барышникова Марина Юрьевна

*2023 г.*

Оглавление

[Описание условия задачи 3](#_Toc151144494)

[Описание технического задания 3](#_Toc151144495)

[Описание внутренних структур данных 5](#_Toc151144496)

[Описание алгоритмов 6](#_Toc151144497)

[Сравнение времени работы и ресурсных затрат 6](#_Toc151144498)

[Оценка достоверности задания. Тестирование 8](#_Toc151144499)

[Тесты 10](#_Toc151144500)

[Ответы на контрольные вопросы 12](#_Toc151144501)

[Вывод 14](#_Toc151144502)

# Описание условия задачи

Программа работает со очередью, представленной как массивом, так и списком; выполняет операции добавления, удаления элементов. Смоделирована система массового обслуживания, которая состоит из обслуживающего аппарата (ОА) и очереди заявок, для обработки первых 1000 заявок (каждая заявка может быть обработана не один раз) и после каждых 100 вышедших заявок программа выдает информацию о текущей и средней длине очереди, а в конце - общее время моделирования и количестве вошедших в систему и вышедших из нее заявок, количестве срабатываний ОА, время простоя аппарата. По требованию пользователя выдать на экран адресов элементов очереди при удалении и добавлении элементов. Заявки поступают в "хвост" очереди по случайному закону с интервалом времени Т1, 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 Т2 от 0 до 1 е.в., Каждая заявка после ОА вновь поступает в "хвост" очереди, совершая всего 5 циклов обслуживания, после чего покидает систему. (Все времена – вещественного типа) В начале процесса в системе заявок нет. При желании пользователь может попросить программу вывести информацию об адресах памяти (для списка), чтобы проследить, наблюдается ли фрагментация в программе.

Описание технического задания

1. **Исходные данные и результат**

**Входные данные:**

Пользователь выбирает пункты меню с соответствующими действиями.

|  |
| --- |
| Выберите действие из списка, введя соответствующий номер:  1. МАССИВ: Ввести данные обслуживающего аппарата и вывести статистику работы.  2. МАССИВ: Вывести количественную характеристику выполнения операций над очередью.  3. СПИСОК: Ввести данные обслуживающего аппарата и вывести статистику работы.  4. СПИСОК: Вывести количественную характеристику выполнения операций над очередью.  0. Выход из программы. |

**Выходные данные:**

На экран выводятся результаты команд меню.

1. **Задачи, реализуемой программой**

**Программа требует от пользователя выбор типа работы с очередью (с помощью статического массива или с помощью списка). Программа умеет добавлять/удалять элемент из очереди (и вывести сравнительный анализ по этим операциям); запускает процесс отработки 1000 заявок (пользователь сам может задать максимальный размер очереди, интервал прибытия заявки, интервал обработки заявки). Также можно посмотреть адреса добавляемых и освобождаемых элементов очереди.**

**Способ обращения к программе**

**Программа запускается через терминал.**

1. **Возможные аварийные ситуации и ошибки пользователя**

1. Некорректный ввод номера команды.

На входе: число, большее чем 4 или меньшее, чем 0.

На выходе: сообщение «Введено недопустимое значение! Повторите

попытку»

2. Некорректный ввод количества элементов очереди.

На входе: отрицательное целое число, число, превышающее максимально

допустимое число для количества элементов стека или буква.

На выходе: сообщение «Введено недопустимое значение! Повторите

попытку.»

3. Некорректный ввод характеристик обслуживающего аппарата очереди.

На входе: буква или, любой другой нечисловой символ или отрицательное

число.

На выходе: сообщение «Введено недопустимое значение! Повторите

попытку.»

4. Некорректный ввод характеристик обслуживающего аппарата очереди.

На входе: правая граница интервала больше левой.

На выходе: сообщение «Правая граница должна быть больше левой!

Повторите попытку.»

5. Попытка создать новую очередь, при имеющейся в программе.

На входе: попытка создания новой очереди.

**На выходе: сообщение «Очередь уже существует. Выход из программы...»**

# **Описание внутренних структур данных**

Информация о заявках, очереди в виде массива и списка:

|  |
| --- |
| typedef struct  {  unsigned capacity, size, rear, front;  task\_t \*arr;  } queuearr\_t;  typedef struct queuenode  {  task\_t task;  struct queuenode \*next;  } queuenode\_t;  typedef struct  {  unsigned capacity, size;  queuenode\_t \*front, \*rear;  } queuelist\_t;  typedef struct  {  double time\_out;  int num;  } task\_t;  typedef struct  {  size\_t \*arr;  int capacity;  int ind;  } arr\_t; |

task\_t – структура для хранения заявки, где time\_out – время обработки заявки, num – сколько раз данную заявку уже обработали.   
queuearr\_t – очередь в виде статического массива, где capacity, size, rear, front – максимальный допустимый размер, текущий размер, хвост и голова очереди, а arr – массив заявок.

queuelist\_t – очередь в виде списка, в котором хранится указатели на голову и хвост типа queuenode\_t, а также capacity – максимальный размер и size – текущий размер.

queuenode\_t – узел для очереди в виде списка, в котором есть два поля: task – информация о заявке, \*next – указатель на следующий узел.

# Описание алгоритмов

1. Пользователь вводит номер команды из меню.

2. Пока пользователь не введет 0 (выход из программы), ему будет

предложено выполнять действия с двумя реализациями очереди – на

основе массива или на основе линейного односвязного списка.

3. При выборе команды вывода количественной характеристики выполнения

операций над очередью, выводится среднее значение

добавления/удаления элементов из очереди на основе 1000

добавлений/удалений.

4. При выборе команды вывода статистики ОА, выводится статистика ОА

после обработки каждых 100 заявок, а также общие данные, описанные в

секции “Выходные данные”.

# Сравнение времени работы и ресурсных затрат

В программе реализована функция, которая замеряет время в тактах процессора (метод rdtsc). При выборе пункта 2 или 4 программа выдает результат работы добавления/удаления элемента из очереди в виде массива и списка соответственно. В качестве результата выдается среднее из 100 замеров. Чтобы посмотреть информацию о том, сколько байт в памяти занимает очередь двух реализаций заданного размера, необходимо запустить работу аппарата и указать максимальное значение элементов очереди. Ниже приведена статистика для 10, 100 и 1000 элементов очереди.

|  |
| --- |
| Добавление элементов в очередь на основе массива: 35  Удаление элементов из очереди на основе массива: 46  Добавление элементов в очередь на основе списка: 65  Удаление элементов из очереди на основе списка: 58  Эффективность добавления у массива относительно списка: 46.15%  Эффективность добавления у массива относительно списка: 20.69%  10:  Bytes allocated for queue-array: 184б  Bytes allocated for queue-list: 264б  100:  Bytes allocated for queue-array: 1624б  Bytes allocated for queue-list: 2424б  1000:  Bytes allocated for queue-array: 16024б  Bytes allocated for queue-list: 24024б |

Из замерного эксперимента видно, что все операции при всех размерах сработали по времени быстрее у массива, нежели чем у списка. По памяти также массив занимает меньше байт, чем список. Из этого следует, что список проигрывает по времени и памяти массиву. Однако его следует использовать, когда заранее неизвестно количество элементов + память под список выделяется в ходе программы, а не сразу целиком, как в статическом массиве.

Сравним работу стека (л/р 4) и очереди (л/р 5) (аналогично в качестве результата выводится среднее из 100 замеров):

|  |
| --- |
| РАЗМЕР СТЕКА: 1000  Массив:  Время (push X1000): 10  Время (pop X1000): 8  Занимаемая память: 4004Б  Список:  Время (push X1000): 20  Время (pop X1000): 31  Занимаемая память: 20000Б  РАЗМЕР ОЧЕРЕДИ: 1000  Массив:  Добавление элементов в очередь на основе массива: 35  Удаление элементов из очереди на основе массива: 46  Bytes allocated for queue-array: 16024б  Список:  Добавление элементов в очередь на основе списка: 65  Удаление элементов из очереди на основе списка: 58  Bytes allocated for queue-list: 24024б |

Время измеряется в процессорных тиках, память в байтах. Во всех случаях стек работает быстрее (выигрыш во времени – примерно в 3 раза) и занимает меньше памяти.

# Оценка достоверности задания. Тестирование

Если попросить программу вывести на экран адреса памяти, то она выведет те адреса, в которых были динамически выделены/освобождены блоки памяти для каждого добавления/удаления элемента. В конце программа выводит освобожденные адреса памяти.

|  |
| --- |
| … Адреса:  562a19091ad0 562a19091b70 562a19091bf0 562a19091af0 562a19091b30 562a19091bd0 562a19091a50 562a19091ab0 562a19091bb0 562a19091a70 562a19091b90 562a19091a90 562a19091b50 562a19091b10 0 |

Фрагментация памяти возникает в очереди на основе списка – (в случае, когда среднее время прихода меньше упятеренного среднего времени обработки, в конце программы память из-под последней вышедшей заявки могла освободиться в любом месте выделенной памяти максимальный размер очереди, следовательно есть вероятность возникновения «дырки» в блоке очереди => фрагментация памяти на момент отработки процесса).  
При N = 1000; T1 = (0, 6); T2 = (0, 1); k = 5

Время работы аппарата очереди на массиве: макс(сред(Т1), сред(Т2)\*k)\*N = 3 \* 1000 = 3000.  
Число вошедших заявок: 1000  
Число вышедших заявок: 1000

Число срабатываний аппарата: (в данном случае) N\*k = 1000\*5 = 5000

Время работы аппарата очереди на списке определяется из макс(сред(Т1), сред(Т2)\*k). Если сред(Т1) больше, то итоговое время работы аппарата равно сред(Т1)\*N, иначе k \* сред(Т2) \* 1000.

При сред(Т1) < сред(Т2)\*k время простоя аппарата уменьшается, а общее время работы аппарата и число срабатываний аппарата увеличивается.

Подтверждение на двух случаях приведено ниже.

1. При N = 1000; T1 = (0, 6); T2 = (0, 1); k = 5

|  |
| --- |
| Рабочее время аппарата: 2973.098066 е.в. (ожидаемое рабочее время: 3000.000000 е.в., погрешность: 0.896731%)  Число вошедших заявок: 1000  Число вышедших заявок: 1000  Число необработанных заявок: 0  Число срабатываний аппарата: 5000  Время простоя аппарата: 2524.646256 е.в.  Bytes allocated for queue-list: 32024б  Время выполнения моделирования на основе списка: 505364 |

1. При N = 1000; T1 = (0, 6); T2 = (0, 10); k = 5

|  |
| --- |
| Рабочее время аппарата: 24899.250889 е.в. (ожидаемое рабочее время: 25000.000000 е.в., погрешность: 0.402996%)  Число вошедших заявок: 1999  Число вышедших заявок: 1000  Число необработанных заявок: 999  Число срабатываний аппарата: 7398 Время простоя аппарата: 5.041126 е.в.  Bytes allocated for queue-array: 16024б  Время выполнения моделирования на основе массива: 261955850 |

Предположения подтвердились.

# Тесты

В следующих тестах символ «>» обозначает ввод пользователем пункта меню, а символ Х – полученное значение в ходе программы.

1. **Позитивные тесты**

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Входные данные | Выходные данные | Что проверяется |
| pos\_01 | >1  (параметры) | Рабочее время аппарата: **Х** е.в. (ожидаемое рабочее время: **Х** е.в., погрешность: **Х**%)  Число вошедших заявок: **Х**  Число вышедших заявок: **Х**  Число необработанных заявок: **Х**  Число срабатываний аппарата: **Х**  Время простоя аппарата: **Х** е.в.  Bytes allocated for queue-array: **Х**б  Время выполнения моделирования на основе массива: **Х** | Умеет ли программа выполнять эксперимент с очередью на основе массива |
| pos\_02 | >2 | Добавление элементов в очередь на основе массива: **Х**  Удаление элементов из очереди на основе массива: **Х** | Умеет ли программа правильно вычислять время работы команд для очереди в виде массива |
| pos\_03 | >3  (параметры) | Рабочее время аппарата: **Х** е.в. (ожидаемое рабочее время: **Х** е.в., погрешность: **Х**%)  Число вошедших заявок: **Х**  Число вышедших заявок: **Х**  Число необработанных заявок: **Х**  Число срабатываний аппарата: **Х**  Время простоя аппарата: **Х** е.в.  Bytes allocated for queue-list: **Х**б  Время выполнения моделирования на основе списка: **Х** | Умеет ли программа выполнять эксперимент с очередью на основе списка |
| pos\_04 | >4  (параметры) | Добавление элементов в очередь на основе списка: **Х**  Удаление элементов из очереди на основе списка: **Х** | Умеет ли программа правильно вычислять время работы команд для очереди в виде списка |

В случае переполнения очереди в ходе работы программы процесс не останавливается, а новая заявка не добавляется в очередь.

1. **Негативные тесты**

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Входные данные | Выходные данные | Что проверяется |
| neg\_01 |  | Сообщение: « Введено недопустимое значение! Повторите попытку.» | Пустой ввод |
| neg\_02 | >-1 | Сообщение: « Введено недопустимое значение! Повторите попытку.» | Правильность ввода пункта меню |
| neg\_03 | >100 | Сообщение: « Введено недопустимое значение! Повторите попытку.» | Правильность ввода пункта меню |
| neg\_04 | >А | Сообщение: « Введено недопустимое значение! Повторите попытку.» | Правильность ввода пункта меню |
| neg\_05 | >1  (ввод буквы) | Сообщение: « Введено недопустимое значение! Повторите попытку.» | Правильность ввода параметров |
| neg\_06 | >1  (ввод любого из двух интервалов) 8 2 | Сообщение: «Правая граница должна  быть больше левой!  Повторите попытку.» | Попытка ввести неправильно интервал |
| neg\_07 | >3(1)  >3(1) | Сообщение:  «Очередь уже существует!» | Попытка создать вторую очередь |

# Ответы на контрольные вопросы

1. **Что такое FIFO и LIFO?**

FIFO (First-In, First-Out) и LIFO (Last-In, First-Out) - это два основных метода управления данными в структурах данных. В FIFO новые элементы добавляются в конец очереди, а удаление происходит из начала очереди, что соответствует принципу "первым пришел - первым вышел". В LIFO новые элементы добавляются в начало структуры, а удаление происходит также из начала, что соответствует принципу "последним пришел - первым вышел".

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

**Объем памяти, выделяемый под хранение очереди, зависит от ее реализации. В случае статической реализации, где размер очереди фиксирован заранее, выделяется фиксированный объем памяти, достаточный для хранения максимального количества элементов. В случае динамической реализации, память выделяется по мере добавления элементов в очередь.**

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

**При удалении элемента из очереди в различных реализациях освобождение памяти может происходить по-разному. В статической реализации память не освобождается, а только индексы начала и конца очереди обновляются. В динамической реализации память, занимаемая удаленным элементом, может быть освобождена и использована для хранения новых элементов.**

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

**При просмотре элементов очереди значение «головы» очереди передается и при этом она удаляется (указатель на голову смещается).**

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

**Эффективность физической реализации очереди зависит от нескольких факторов, включая алгоритмы, используемые для добавления и удаления элементов, объем доступной памяти, скорость доступа к памяти и требования к производительности.**

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

**Различные реализации очереди имеют свои достоинства и недостатки в зависимости от выполняемых операций. Например, статическая реализация может быть эффективной, если известно максимальное количество элементов, но она не позволяет динамически изменять размер очереди. Динамическая реализация позволяет гибко изменять размер очереди, но может потребовать больше памяти и иметь более сложные алгоритмы добавления и удаления элементов.**

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

**Фрагментация памяти - это явление, при котором свободное пространство в оперативной памяти разбивается на маленькие фрагменты, которые не могут быть использованы для размещения больших блоков данных. Фрагментация может возникать в физической памяти, например, из-за последовательности выделений и освобождений динамической памяти, в ходе которой оставшиеся блоки памяти «разбросаны» в разных участках (адресах) ОП.**

1. **Для чего нужен алгоритм «близнецов».**

**Алгоритм "близнецов" используется для управления памятью в операционных системах. Он позволяет эффективно использовать физическую память путем разделения ее на страницы и отображения виртуальной памяти на эти страницы (таким образом физическая память экономится).**

1. **Какие дисциплины выделения памяти вы знаете?**

**Статическое на стеке, динамическое на куче.**

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

**При тестировании программы необходимо обратить внимание на несколько аспектов, включая правильность работы программы в различных сценариях, обработку ошибок и исключительных ситуаций, производительность программы, а также соответствие программы требованиям и спецификациям. Для динамического выделения памяти (например, список) необходимо проконтролировать правильное выделение и освобождение памяти.**

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

**При динамических запросах физическая память выделяется и освобождается с помощью аллокаторов памяти. Аллокаторы памяти управляют выделением и освобождением блоков памяти в зависимости от запросов программы. При выделении памяти аллокатор ищет свободный блок памяти достаточного размера и отмечает его как занятый. При освобождении памяти аллокатор помечает блок как свободный и может объединять соседние свободные блоки для уменьшения фрагментации памяти.**

# Вывод

В рамках задания были реализованы разные представления очереди (в виде массива и связанного списка), операции добавления/удаления. В программе предусмотрена команда для вывода адресов памяти элементов очереди, а также вывод свободных ячеек памяти. Работа с аппаратом и обработка 1000 заявок у очереди в виде списка занимает больше времени и памяти, чем представления в виде массива.