|  |  |
| --- | --- |
| **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**

**«ОБРАБОТКА ОЧЕРЕДЕЙ»**

Студент Нисуев Нису Феликсович

Группа ИУ7 – 32Б

Преподаватель Барышникова Марина Юрьевна

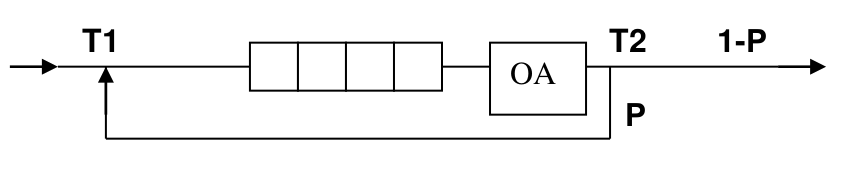
*2023г.*

**Описание условия задачи**

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 технического задания

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 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 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. Обеспечить по требованию пользователя выдачу на экран адресов элементов очереди при удалении и добавлении.

Входные данные:

1. Номер команды: целое число в диапазоне {-1} ∪ [1; 3].
2. Дополнения к таблице: строковое или целочисленное поле (в зависимости от команды)

Выходные данные:

1. Состояние очереди (списка/массива) и информация о моделировании:

* Ожидаемое время моделирования
* Полученное время моделирования
* Погрешность
* Количество вошедших заявок
* Количество вышедших заявок
* Среднее время в очереди
* Время простоя аппарата
* Количество срабатывания аппарата

1. Список использованных адресов.
2. Сравнительная характеристики времени работы ОА при реализации очереди в виде массива и списка и занимаемой ими памяти.

Обращение к программе:

Запуск через терминал (./target/app.exe)

Аварийные ситуации:

1. Неверная команда
2. Неверный пользовательский ввод
3. Переполнение очереди
4. Ошибка шанса (Элемент не может выйти из очереди)

# Набор тестов

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название теста** | **Пользовательский ввод** | **Вывод** |
| **Негативные тесты** | | | |
| 1 | Некорректный пункт меню | 99 | ERROR: Incorrect action |
| 2 | Некорректный ввод | 1  про | ERROR: Incorrect input |
| 3 | Некорректно введенное число выводимых адресов | 1  Y  -1 | ERROR: Incorrect input |
| **Позитивные тесты** | | | |
| 1 | Моделирование очереди  из 1000 элементов в  виде массива | 1 | {Информация об обработке заявок} |
| 2 | Моделирование очереди  из 1000 элементов в  виде списка | 2 | {Информация об обработке заявок} |
| 3 | Моделирование очереди  из 1000 элементов  Вывод адрессов 50 | 1  Y  50 | {Информация об обработке заявок} {Таблица адресов} |
| 4 | Невалидное выражение | 3  (( | [{Время проверки} mcs] Brackets are invalid |
| 5 | Валидное выражение | 3  (()) | [{Время проверки} mcs] Brackets are valid |

## Описание структуры данных:

|  |
| --- |
| #define REQUESTS\_CNT 1000        // Число обработанных заявок  #define REQUESTS\_INFO\_STEP 100   // Число заявок, после которого нужно предоставить информацию о состоянии ОА  #define T1\_RANGE(i) (!i ? 0 : 6) // Времена прихода  #define T2\_RANGE(i) (!i ? 0 : 1) // Времена обработка  #define CHANCE 0.8               // Вероятность  typedef struct {      size\_t cnt;                              // Количество длин очередей      size\_t sum;                              // Сумма длин очередей  } queue\_lens\_inf\_t;  typedef struct {      size\_t cnt;                    // Количество заявок вышедших из очереди      double sum;                  // Сумма времени заявок в очереди  } time\_in\_queue\_inf\_t;  /// @brief Структура Обрабатывающего аппарата  typedef struct {      double simple\_time;                      // Время простоя ОА      queue\_lens\_inf\_t requests;               // Информация об очереди      time\_in\_queue\_inf\_t requests\_in\_queue;   // Информация об времени заявки в очереди      size\_t in\_requests\_cnt;                  // Количество вошедших заявок      size\_t out\_requests\_cnt;                 // Количество вышедших заявок      size\_t service\_unit\_triggers\_cnt;        // Количество срабатываний ОА      double modeling\_time;                    // Время моделирования  } service\_info\_t;  /// @def QUEUE\_SIZE - максимальный размер стека  #define QUEUE\_SIZE 2000  /// @typedef aqtak\_t - Очередь на основе кольцевого статического массива  typedef struct {      double queue[QUEUE\_SIZE];      // Массив элементов очереди      size\_t els\_cnt;                // Кол-во элементов в очереди      double \*first;                 // Первый в очереди      double \*last;                  // Последний в очереди  } aqueue\_t;  /// @typedef node\_t - Очередь на основе списка  typedef struct node {      double el;              // Элемент      struct node \*next;      // Следующий элемент  } lqueue\_t; |

# Теоретические расчеты:

Ожидаемое время моделирования (время прихода больше времени обработки) = (среднее время прихода заявки) \* (количество) / 2

Время простоя аппарата = время моделирования – время прихода

Количество срабатываний = 1 / (1 - вероятность) \* (количество заявок)

**Пример работы №1:**

Время поступления: 0 до 6 е.в.

Время обслуживания: 0 до 1 е.в.

Вероятность возвращения в «хвост»: 0.8

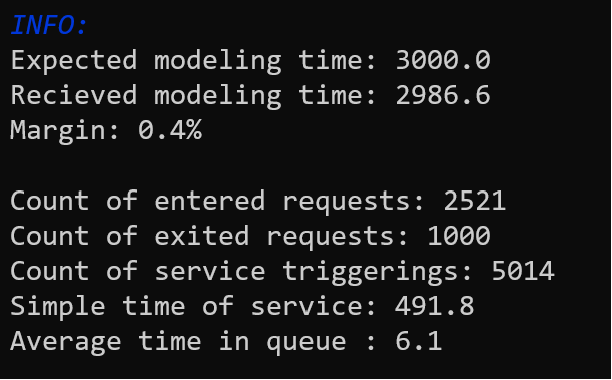
**Теоретический результат моделирования:**

Ожидаемое время моделирования: (6 – 0) \* 1000 / 2 = 3000

Количество срабатываний: (1 / (1 - 0.8)) \* 1000 = 5000

Время простоя ОА: (1 – 0) \* 1000 / 2 = 500

**Фактический результат моделирования:**



**Пример работы №2:**

Время поступления: 0 до 6 е.в.

Время обслуживания: 0 до 10 е.в.

Вероятность возвращения в «хвост»: 0.8

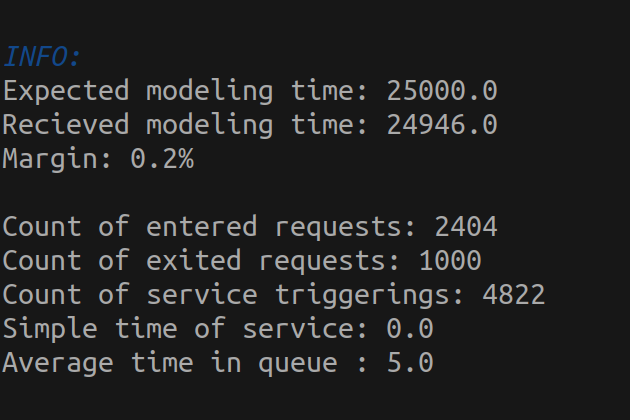
**Теоретический результат моделирования:**

Ожидаемое время моделирования: ((10 – 0) \* 1000 / 2) \* (1 / (1 – 0.8)) = 25000

Количество срабатываний: (1 / (1 - 0.8)) \* 1000 = 5000

Время простоя ОА: 0

**Фактический результат моделирования:**



**Контрольные вопросы**

**1. Что такое FIFO и LIFO?**

Способы организации данных. Первым пришел — первым вышел, т. е. First In – First Out (FIFO) – так реализуются очереди. Первым пришел — последним вышел, т. е. Last In – First Out (LIFO)- так реализуются линейные односвязные списки.

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

При реализации массивом единожды выделяется память для хранения только самих элементов. При реализации в виде списка для каждого элемента дополнительно выделяется память для хранения адреса следующего элемента.

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

При реализации очереди списком указателю голове списка присваивается значение следующего элемента списка, а память из-под удаляемого элемента освобождается.

При реализации очереди массивом память из-под удаляемого элемента не освобождается, так как изначально она была выделена под весь массив. Освобождение будет происходить аналогично – единожды для всего массива.

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

При просмотре очереди элементы удаляются и происходит очистка очереди, влекущая за собой освобождение памяти в случае реализации ее списком

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

Выяснилось, что эффективнее и по памяти, и по времени реализовывать очередь массивом. Это зависит от самой структуры элемента в случае памяти, и от запросов в оперативную память в случае времени.

**6. Каковы достоинства и недостатки различных реализаций очереди в**

**зависимости от выполняемых над ней операций?**

В случае массива недостатком является фиксированный размер очереди.

В случае односвязного списка недостатками являются затраты по скорости и фрагментация.

**7. Что такое фрагментация памяти?**

Фрагментация — возникновение участков памяти, которые не могут быть использованы. Фрагментация может быть внутренней — при выделении памяти блоками остается не задействованная часть, может быть внешней — свободный блок, слишком малый для удовлетворения запроса

**8. Для чего нужен алгоритм «близнецов»?**

Идея этого алгоритма состоит в том, что организуются списки свободных блоков отдельно для каждого размера 2k, 0 <= k <= m. Вся область памяти кучи состоит из 2m элементов, которые, можно считать, имеют адреса с 0 по 2m – 1. Первоначально свободным является весь блок из 2m элементов. Далее, когда требуется блок из 2k элементов, а свободных блоков такого размера нет, расщепляется на две равные части блок большего размера; в результате появится блок размера 2k (т.е. все блоки имеют длину, кратную 2). Когда один блок расщепляется на два (каждый из которых равен половине первоначального), эти два блока называются ***близнецами***. Позднее, когда оба близнеца освобождаются, они опять объединяются в один блок.

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

Две основные дисциплины сводятся к принципам "самый подходящий" и "первый подходящий". По дисциплине "самый подходящий" выделяется тот свободный участок, размер которого равен запрошенному или превышает его на минимальную величину. По дисциплине "первый подходящий" выделяется первый же найденный свободный участок, размер которого не меньше запрошенного.

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

При тестировании программы необходимо обратить внимание на корректность выводимых данных, проследить за выделением и освобождением выделяемой динамически памяти, предотвратить возможные аварийные ситуации.

**11. Каким образом физически выделяется и освобождается память при**

**динамических запросах?**

В оперативную память поступает запрос, содержащий необходимый размер выделяемой памяти. Выше нижней границы свободной кучи осуществляется поиск блока памяти подходящего размера. В случае если такой найден, в вызываемую функцию возвращается указатель на эту область и внутри кучи она помечается как занятая. Если же найдена область, большая необходимого размера, то блок делится на две части, указатель на одну возвращается в вызываемую функцию и помечается как занятый, указатель на другую остается в списке свободных областей. В случае если области памяти необходимого размера не было найдено, в функцию возвращается NULL. При освобождении памяти происходит обратный процесс. Указатель на освобождаемую область поступает в оперативную память, если это возможно объединяется с соседними свободными блоками, и помечается свободными.

**Сравнение эффективности:**

Была произведена оценка эффективности работы модели (по ТЗ) при 1000 вышедших из ОА заявок при времени прихода от 0 до 6:

# 

# Вывод:

Очередь основанная на циклическом массиве работает быстрее и использует меньше памяти чем очередь на односвязном списке. Но если надо будет выделять новую память то в циклическом массиве это будет сделать труднее.