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

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5**

Название предмета: Типы и структуры данных

Студент: Гриценко Алексей Михайлович

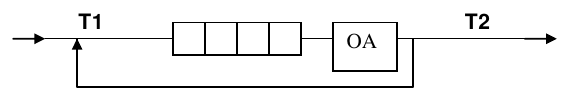
Группа: ИУ7-31Б

*2020г.*

1. **Описание условия задачи**

Смоделировать процесс обслуживания до ухода из системы первых 1000 заявок, выдавая после обслуживания каждых 100 заявок информацию о текущей и средней длине очереди, а в конце процесса - 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. Проследить, возникает ли при этом фрагментация памяти.

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 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.

1. **Техническое задание**
2. ***Описание исходных данных***
3. **Целое число, представляющее собой номер команды:** целое число в диапазоне от **0** до **4**.
4. **Командно-зависимые данные:**
   * целочисленные значения (количество элементов очереди, интервалы характеристик ОА).
5. ***Описание результата программы***
6. Результат моделирования работы ОА – данные о рабочем времени автомата, ожидаемое рабочее время автомата, погрешность, число вошедших заявок, число вышедших заявок, число необработанных заявок, число срабатываний автомата, время простоя автомата, количество переиспользованных адресов и адресов, взятых из новой памяти (в случае реализации на списке).
7. Количественная характеристика сравнения вариантов моделирования ОА очереди.

*.*

1. ***Задача, реализуемая программой***
2. Ввести данные обслуживающего аппарата и вывести статистику работы.
3. Вывести количественную характеристику выполнения операций над очередью.

п. 1-2 выполняют действия над очередью, реализованной на основе массива.

1. Ввести данные обслуживающего аппарата и вывести статистику работы.
2. Вывести количественную характеристику выполнения операций над очередью.

п. 3-4 выполняют действия над очередью, реализованной на основе связного списка.

1. ***Обращения к программе***

Способ обращения к программе происходит через консоль.

1. ***Описание возможных аварийных ситуаций и ошибок пользователя***
2. Некорректный ввод номера команды.  
   На входе: число, большее чем 4 или меньшее, чем 0.   
   На выходе: сообщение «Введено недопустимое значение! Повторите попытку»
3. 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.»
4. 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.»
5. Некорректный ввод характеристик обслуживающего аппарата очереди.  
   На входе: правая граница интервала больше левой.  
   На выходе: сообщение «Правая граница должна быть больше левой! Повторите попытку.»
6. Попытка создать новую очередь, при имеющейся в программе.  
   На входе: попытка создания новой очереди.  
   На выходе: сообщение «Ошибка! Очередь уже существует..»
7. **Описание внутренних структур данных**

Хранение заявки:

**typedef struct**

**{**

***double* time\_out;**

***int* num;**

**} task\_t;**

Поля структуры:

* ***double* time\_out –** время, затрачиваемое на обработку заявки;
* ***int* num *–***количество раз, сколько была обработана заявка;

Реализация очереди на основе кольцевого массива:

**typedef struct**

**{**

***unsigned* capacity, size, rear, front;**

***task\_t* \*arr;**

**} queuearr\_t;**

Поля структуры:

* ***unsigned* capacity, size, rear, front –** максимальный допустимый размер, текущий размер, хвост и голова очереди;
* ***task\_t* \*arr *–***указатель на массив заявок;

Реализация очереди на основе линейного односвязного списка:

**typedef struct**

**{**

***unsigned* capacity, size;**

***queuenode\_t* \*front, \*rear;**

**} queuelist\_t;**

Поля структуры:

* ***unsigned* capacity, size** **–** максимальный допустимый размер и текущий размер очереди;
* ***queuenode\_t* \*front, \*rear –** указатели на голову и хвост очереди;

Реализация элемента очереди:

**typedef struct queuenode**

**{**

***task\_t* task;**

***struct queuenode* \*next;**

**} queuenode\_t;**

Поля структуры:

* ***task\_t* task *–*** заявка ОА;
* ***struct queuenode* \*next *–***указатель на следующий элемент очереди;

Реализация массива свободных областей:

**typedef struct**

**{**

***size\_t* \*arr;**

***int* capacity;**

***int* ind;**

**} arr*\_t*;**

Поля структуры:

* ***size\_t* \*arr *–*** указатель на массив адресов;
* ***int* capacity *–***максимальная ёмкость массива;
* ***int* ind –** текущий незанятый элемент.

1. **Описание алгоритма**
2. Пользователь вводит номер команды из меню.
3. Пока пользователь не введет 0, ему будет предложено выполнять действия с двумя реализациями очереди – на основе массива или на основе линейного односвязного списка.
4. При выборе команды вывода количественной характеристики выполнения операций над очередью, выводится среднее значение добавления/удаления элементов из очереди на основе 1000 добавлений/удалений.
5. При выборе команды вывода статистики ОА, выводится статистика ОА после обработки каждой 100 заявки, а также общие данные, описанные в секции “Выходные данные”. Сам алгоритм – в цикле, который продолжается, пока не будут обработаны 1000 заявок. Внутри мы добавляем в очередь элемент(в случае переполнения программа продолжит работу, выкинув заявку, при этом увеличится счетчик потерянных заявок), вместе с этим идет подсчет времени, после обработки каждой сотни заявок вывод, если очередь реализована на списке, занесение адреса памяти в массив адресов.

**Функции, использующиеся в программе**

arr\_t \*create\_array(const int capacity) - функция создания массива свободных областей, с известной максимальной емкостью

void output\_array(const arr\_t arr) – вывод массива свободных областей

void free\_array(arr\_t \*arr) – функция освобождения массива свободных областей

queuearr\_t \*create\_queuearr(const unsigned capacity) – функция создания очереди на основе массива

void enqueuearr(queuearr\_t \*const queue, const task\_t task) – функция добавления элемента в очередь

task\_t dequeuearr(queuearr\_t \*const queue) – функция получения элемента очереди

void freequeuearr(queuearr\_t \*queue) – функция очищения памяти очереди

queuenode\_t \*create\_queuenode(const task\_t task) – функция создания элемента очереди

queuelist\_t \*create\_queuelist(const unsigned capacity) – функция создания очереди на основе списка

void enqueuelist(queuelist\_t \*const queue, const task\_t task) – функция добавления элемента в очередь на основе списка

task\_t dequeuelist(queuelist\_t \*const queue, arr\_t \*const fmem) – функция забирания элемента очереди на основе списка

void freequeuelist(queuelist\_t \*queue, arr\_t \*const fmem) – функция чистки памяти для очереди на основе списка

void list\_model(queuelist\_t \*const queue, arr\_t \*const fmem, const int llim\_in, const int rlim\_in, const int llim\_out, const int rlim\_out, const int repeats) – создание модели на основе списка

void array\_model(queuearr\_t \*const queue, const int llim\_in, const int rlim\_in, const int llim\_out, const int rlim\_out, const int repeats) – создание модели на основе массива

1. **Тестирование**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Тест | Пользовательский ввод | Результат |
| 1 | Некорректный̆ ввод комманды | 5 | Введено недопустимое значение! Повторите попытку. |
| 2 | Некорректный̆ ввод количества элементов очереди | -1 | Введено недопустимое значение! Повторите попытку |
| 3 | Некорректный ввод количества элементов очереди | A | Введено недопустимое значение! Повторите попытку |
| 4 | Некорректный ввод характеристики ОА | Ввод интервала прихода:  A 8 | Введено недопустимое значение! Повторите попытку |
| 5 | Некорректный ввод характеристики ОА | Ввод интервала прихода:  0 B | Введено недопустимое значение! Повторите попытку |
| 6 | Некорректный ввод характеристики ОА | Ввод интервала прихода:  8 7 | Правая граница должна быть больше левой! Повторите попытку. |
| 7 | Некорректный ввод характеристик ОА | Ввод количества обслуживаний одной заявки:  -1 | Введено недопустимое значение! Повторите попытку |
| 8 | Попытка создания очереди при уже существующей | Попытка создания очереди | Ошибка! Очередь уже существует! |
| 9 | Корректный ввод характеристик ОА | 10000; 0 6; 0 1; 5 | Вывод информации ОА и количественная характеристика моделирования |
| 10 | Переполнение очереди | Переполнение очереди во время работы ОА | Очередь переполнена! 5Новый элемент не будет добавлен! |
| 11 | Вывод количественной характеристики выполнения операций над очередью | Выбор команды | Добавление элементов в очередь на основе **X**: **Время**  Удаление элементов из очереди на основе **X**: **Время** |

1. **Оценка эффективности**

Измерения эффективности реализаций очереди будут производиться в единицах измерения – тактах процессора. При записи результатов использовалось среднее количество тактов, полученное по результатам 100 измерений.

**Время добавления элемента (в тактах процессора):**

|  |  |
| --- | --- |
| **Массив** | **Список** |
| 98 | 336 |

**Время удаления элемента (в тактах процессора):**

|  |  |
| --- | --- |
| **Массив** | **Список** |
| 55 | 182 |

**Время моделирования очереди (входные данные из условия) (в тактах процессора):**

|  |  |
| --- | --- |
| **Массив** | **Список** |
| 1602728 | 25339490 |

**Объём занимаемой памяти (в байтах):**

|  |  |  |
| --- | --- | --- |
| **Количество элементов** | **Массив** | **Список** |
| 10 | 176 | 248 |
| 100 | 1616 | 2408 |
| 1000 | 16016 | 24008 |

**Объём максимально занимаемой памяти (начальные условия) (в байтах):**

|  |  |
| --- | --- |
| **Массив** | **Список** |
| 16016 | 1736 |

1. **Выводы по проделанной работе**

Использование связанных списков не так выгодно при реализации очереди. Списки проигрывают как по памяти, так и по времени обработки. Но, когда заранее неизвестен максимальный размер очереди, то лучше использовать связанные списки, так как в отличии от статического массива, списки ограничены в размерах только размером оперативной памяти. Стоит отметить, что при проведении тестов у меня наблюдалась фрагментация памяти, в районе 1-2%)

1. **Ответы на вопросы**

**1. Что такое очередь?**

Очередь - структура данных, для которой выполняется правило FIFO, то есть первым зашёл - первым вышел. Вход находится с одной стороны очереди, выход - с другой.

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

При хранении кольцевым массивом: кол-во элементов \* размер одного элемента очереди. Память выделяется на стеке при компиляции, если массив статический. Либо память выделяется в куче, если массив динамический.

При хранении списком: кол-во элементов \* (размер одного элемента очереди + указатель на следующий элемент). Память выделяется в куче для каждого элемента отдельно.

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

При хранении кольцевым массивом память не освобождается, а просто меняется указатель на конец очереди. При хранении списком, память под удаляемый кусок освобождается.

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

Эти элементы удаляются из очереди.

**5. Каким образом эффективнее реализовывать очередь. От чего это зависит?** Зная максимальный размер очереди, лучше всего использовать кольцевой статический массив. Не зная максимальный размер, стоит использовать связанный список, так как такую очередь можно будет переполнить только если закончится оперативная память.

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

При использовании линейного списка тратится больше времени на обработку операций с очередью, а так же может возникнуть фрагментация памяти. При реализации статическим кольцевым массивом, очередь всегда ограничена по размеру, но операции выполняются быстрее, нежели на списке.

**7. Что такое фрагментация памяти?**

Фрагментация памяти - разбиение памяти на куски, которые лежат не рядом друг с другом. Можно сказать, что это чередование свободных и занятых кусков памяти.

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

При тестировании программы необходимо обратить внимание на эффективное выделение и корректное освобождение динамической памяти. Помимо этого стоит обратить внимание на корректность реализации кольцевого массива, чтобы не произошло записи в невыделенную область памяти. Еще стоит обратить внимание на возникновение фрагментации памяти.

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

При запросе памяти, ОС находит подходящий блок памяти и записывает его в «таблицу» занятой памяти. При освобождении, ОС удаляет этот блок памяти из «таблицы» занятой пользователями памяти.