

*Отчет по дисциплине:*

**Типы и структуры данных**

*Лабораторная работа:*

**"Работа с очередью"**

Выполнил студент: Сироткина П.Ю.

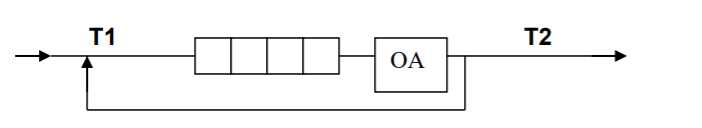
Группа: ИУ7-36Б

Преподаватель: Силантьева А.В.

**Москва, 2020 г.**

1. **Условие задачи**

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. В начале процесса в системе заявок нет.

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.

1. **Описание ТЗ**

*А) Входные и выходные данные:*

Входные данные:

Целое число от 1 до 4, представляющее собой пункт меню программы. В зависимости от выбора пункта может потребоваться выбрать дополнительный подпункт, который также является целым числом, лежащим в некотором интервале.

Выходные данные:

Командно-зависимо: печать информации о результате моделирования работы обслуживающего аппарата, в случае реализации очереди динамическим односвязным списком также возможна печать адресов памяти при удалении/добавлении элемента из очереди, печать сравнительного анализа для операций добавления и удаления при различных реализациях (статическим массивом или динамическим односвязным списком), а также сравнительный анализ занимаемой памяти при указанных реализациях и различных размерах очереди.

*Б) Задачи, реализуемые программой:*

Программа способна решать следующие задачи, которые запускаются посредством выбора соответствующего пункта меню:

**1) Запустить процесс моделирования на основе односвязного линейного списка** - программа запускает процесс моделирования, затем выводит информацию о моделировании на экран, затем производит подсчет погрешности в процентном соотношении, и результат также выводится на экран. Очередь реализуется посредством динамического односвязного списка. По предпочтению пользователя выводится информация об адресах памяти при удалении/добавлении элемента из очереди (для наглядности это реализовано для первых 2 заявок, иначе вывод был бы слишком громоздким при ожидаемых 5000 срабатываниях ОА).

**2) Запустить процесс моделирования на основе статического массива** - программа запускает процесс моделирования, затем выводит информацию о моделировании на экран, затем производит подсчет погрешности в процентном соотношении, и результат также выводится на экран. Очередь реализуется посредством статического массива.

**3) Вывести анализ эффективности операций для статического массива и связного списка -** печать сравнительного анализа для операций добавления и удаления при различных реализациях (статическим массивом или динамическим односвязным списком), а также печать сравнительного анализа занимаемой памяти при указанных реализациях и различных размерах очереди.

**4) Выход -** завершение работы программы.

*В) Способы обращения к программе:*

Пользователь работает с программой посредством меню, команды которого описаны выше.

1. **Описание внутренних структур данных**

Структура для хранения заявки:

**typedef struct**

**{**

**int num\_of\_passes;**

**} task\_t;**

Поле *num\_of\_passes* - количество прохождений заявки через обслуживающий аппарат.

Структура для хранения очереди, реализованной на основе статического массива:

**typedef struct**

**{**

**task\_t tasks[MAX\_ARR\_SIZE];**

**task\_t \*head;**

**task\_t \*tail;**

**int count;**

**} arr\_queue\_t;**

Поле *tasks* - статический массив заявок, где MAX\_ARR\_SIZE – максимальная вместительность статического массива, объявленная ранее.

Поле *head -* указатель на заявку, которая определяет начало очереди.

Поле *tail -* указатель на заявку, которая определяет конец очереди.

Поле *count* - количество элементов в очереди.

Структура для хранения очереди на основе динамического односвязного списка:

**typedef struct**

**{**

**int count;**

**list\_node\_t \*head;**

**} list\_queue\_t;**

Поле count - количество элементов в очереди.

Поле *head* - указатель на узел списка, который начинает очередь.

Структура для хранения узла динамического односвязного списка:

**typedef struct list\_node**

**{**

**task\_t task;**

**struct list\_node *\**next;**

**} list\_node\_t;**

Поле *task* - заявка, которая хранится в конкретном узле списка.

Поле *next* - указатель на следующий по очереди узел списка.

1. **Описание алгоритма:**

Моделирование работы системы реализуется следующим образом.

Возможны 2 состояния работы обслуживающего аппарата:

А) Автомат не работает. Тогда возможны ситуации:

* Если очередь не пуста, необходимо начать обработку первой в очереди заявки
* Если очередь пуста, необходимо подождать, пока в очередь придет новая заявка (это время - время простоя), затем добавить заявку в очередь.

Б) Автомат работает. Тогда возможны ситуации:

* Новая заявка успеет добавиться в очередь раньше, чем аппарат закончит работу. В таком случае необходимо вычесть время, которое потребуется заявке прийти в очередь, из времени, которое осталось до завершения работы аппарата, добавить заявку в очередь, а затем сгенерировать время для ожидания новой заявки.
* Новая заявка не успеет добавиться в очередь до окончания обработки текущей заявки аппаратом. В таком случае необходимо вычесть время, которое потребуется аппарату для завершения работы, из времени, которое потребуется заявке для того, чтобы прийти в очередь, затем работа аппарата завершается и цикл запускается снова.

Таким образом, 4 возможными ситуациями покрыты все возможные события, и работа аппарата разбивается на множество малых событий, идущих друг за другом.

Добавление элемента в очередь на основе статического массива:

Если размер очереди достиг максимального значения, добавить элемент невозможно. Иначе, если хвост очереди доходит до физической границы конца массива, то указатель “сбрасывается” в начало, иначе сдвигается на одну ячейку влево стандартным способом. Количество элементов в очереди увеличивается.

Удаление элемента из очереди на основе статического массива:

Если очередь пуста, удалить элемент невозможно. Иначе, если голова очередь доходит до физической границы конца массива, то указатель “сбрасывается” в начало, иначе сдвигается на одну ячейку вправо обычным способом. Количество элементов в очереди уменьшается.

Добавление элемента в очередь на основе динамического односвязного списка:

Достигается конец списка посредством связей. Затем после последнего элемента выделяется память под новый узел, он связывается с бывшим последним узлом очереди посредством указателя next.

Удаление элемента из очереди на основе динамического односвязного списка:

Если очередь пуста, удалить элемент невозможно. Указатель головы очереди смещается к следующему элементу, указатель на бывшую голову освобождается.

1. **Тестирование программы**

|  |  |
| --- | --- |
| *Действие пользователя* | *Действие программы* |
| Некорректный ввод пункта меню программы:  -4 | Сообщение: “Ошибка: команда должна быть одним из чисел, предложенных в меню.”  Завершение работы программы. |
| Некорректный ввод пункта меню программы:  sdfjsjfd | Сообщение: “Ошибка: команда должна быть одним из чисел, предложенных в меню.”  Завершение работы программы. |
| Некорректный ввод пункта меню программы:  5 | Сообщение: “Ошибка: команда должна быть одним из чисел, предложенных в меню.”  Завершение работы программы. |
| Переполнение очереди при добавлении элемента в статический массив | Возврат кода ошибки ERR\_FULL\_ARR (1).  Завершение работы программы. |
| Удаление элемента из пустого массива | Возврат кода ошибки ERR\_EMPTY\_ARR (2) или ERR\_EMPTY\_LIST (3) соответственно для каждой реализации.  Завершение работы программы. |
| Ошибка выделения памяти при добавлении элемента в динамический односвязный список | Возврат кода ошибки ERR\_MEM (4).  Завершение работы программы. |
| Выбор пункта 1 или 2 | Печать результатов моделирования, расчет расхождения от теоретически ожидаемого времени в процентном соотношении. |
| Выбор пункта 1.1 | Печать результатов моделирования, расчет расхождения от теоретически ожидаемого времени в процентном соотношении, также печатаются адреса при удалении/добавлении. Можно наглядно проследить, что чаще всего фрагментация отсутствует. |
| Выбор пункта 3 | Печать сравнительной характеристики: |

1. **Оценка эффективности**



Для каждого замера по времени проводилось 1000 опытов.

Как видим, добавление элемента в очередь на основе статического массива происходит в среднем в 12 раз быстрее, а удаление - в 7.

Объем занимаемой памяти для очереди на основе статического массива в среднем в 2-3 раза меньше, чем объем для очереди на основе динамического односвязного списка.

**Вывод:** практически всегда эффективнее использовать очередь на основе массива, особенно статического. Он выигрывает по всем параметрам с большим отрывом (в 2-3 раза по памяти и в 7-12 раз по времени в случае статической реализации). Это происходит за счет того, что для статического массива память выделяется на стеке, а для списка на куче. Доступ к стеку происходит гораздо быстрее, чем к куче.

Однако память для хранения очереди на основе массива ограничена размером стека, а память очереди на основе списка ограничена лишь размером доступной оперативной памяти, т.к. память выделяется в куче, поэтому, когда размер очереди достигает больших размеров, необходимо использовать очередь на основе списка.

1. **Ответы на контрольные 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 на основе массива выделяется:

(вместительность статического массива) \* (размер одной заявки) + 2 \* (размер указателя на одну заявку (голова и хвост)) + (размер переменной для количества заявок).

При реализации очереди на основе динамического массива:

(количество элементов очереди) \* (размер одной заявки + размер указателя на узел списка) + (размер переменной для количества заявок).

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

При реализации очереди на основе массива хвост очереди лишь смещается на одну ячейку назад по кольцу, как такового самого удаления не происходит. При реализации на основе списка память освобождается менеджером памяти.

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

При классической реализации элементы очереди удаляется при ее просмотре.

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

Практически всегда эффективнее использовать очередь на основе массива, особенно статического. Он выигрывает по всем параметрам с большим отрывом (в 2-3 раза по памяти и в 7-12 раз по времени в случае статической реализации). Это происходит за счет того, что для статического массива память выделяется на стеке, а для списка на куче. Доступ к стеку происходит гораздо быстрее, чем к куче.

Однако память для хранения очереди на основе массива ограничена размером стека, а память очереди на основе списка ограничена лишь размером доступной оперативной памяти, т.к. память выделяется в куче, поэтому, когда размер очереди достигает больших размеров, необходимо использовать очередь на основе списка, поэтому если размер очереди предполагается большим, нужно использовать список.

*6. В каком случае лучше реализовать очередь посредством указателей, а в каком – массивом?*

Посредством указателей лучше реализовывать, когда размер предполагаемой очереди относительно небольшой. Иначе - лучше реализовывать списком.

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

При кольцевой структуре сложнее реализовать алгоритмы включения и исключения элементов, однако статической памятью легче управлять, чем динамической.

*8. Что такое фрагментация памяти?*

Фрагментация памяти - явление, при котором информация записывается в несмежные блоки памяти, таким образом она находится в разных местах доступной памяти.

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

При тестировании программы необходимо обратить внимание на корректное выделение и освобождение памяти менеджером памяти. Также необходимо проверить правильность реализации операций для кольцевой очереди.

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

На компьютере имеется некоторая память, которая доступна для использования программами. При запуске программы операционная система загружает программу в некоторую часть этой памяти. Память, используемая программой, разделена на несколько частей, каждая из которых выполняет определенную задачу. При запросе памяти операционная система, если может, выделяет кусок памяти и “отдает” ее для использования программой. Освобождение памяти - задача вызывающей стороны. Когда память освобождается, операционная система снова может распределять ее.