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
Лабораторная работа №5
«Работа со очередью»

Группа ИУ7-34Б

Дисциплина Типы и структуры данных

Вариант 14

Студент Козлитин Максим Александрович

Преподаватель Силантьева Александра Васильевна
Рыбкин Юрий Алексеевич

2022г.

Цель работы

Отработка навыков работы с типом данных «очередь», представленным в виде одномерного массива и односвязного линейного списка. Оценка эффективности программы (при различной реализации) по времени и по используемому объему памяти.

Условие задачи

Система массового обслуживания состоит из обслуживающего аппарата (ОА) и очереди заявок.
Заявки поступают в "хвост" очереди по случайному закону с интервалом времени **T1,** 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 **T2** 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.
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.

Описание исходных данных

1. Выбор действия:

   Целое число от 0 до 15.

Способ обращения к программе

Работа с программой осуществляется с помощью консоли.

Программа запускается с помощью команды: **./app.exe**

Далее пользователь выполняет взаимодействие с помощью меню.
Меню выводится в начале и, далее, каждые 3 введенные команды.

Описания возможных аварийных ситуаций и ошибок пользователя

1. Ввод не совпадающий с форматом входных данных (описано в Описание исходных данных).
2. Ввод целых чисел превышающих максимальное допустимое значение типа данных int.

Описание внутренних структур данных

Очередь на массиве:

```
typedef struct
{
    size_t allocated; // количество элементов, на которые выделена память
    int *head; // указатель на голову
    int *tail; // указатель на хвост
    int *begin; // указатель на нижнюю границу массива
    int *end; // указатель на верхнюю границу массива
    int *data; // указатель область памяти массива
} queue_std_t;
```

size_t *используется для задания размеров наших массивов, так как является особым макросом, значение которого позволяет адресовать массив любой теоретически возможной длины для данной машины.*

int *- тип данных знакового целого числа.*

Очередь на связном списке:

```
struct node_t
{
    int data; // элемент очереди
    node_t *next; // указатель на следующий узел
}; // узел связного списка
```

```
typedef struct
{
    node_t *head; // указатель на голову
    node_t *tail; // указатель на хвост
} queue_linked_list_t;
```

size_t *используется для задания размеров наших массивов, так как является особым макросом, значение которого позволяет адресовать массив любой теоретически возможной длины для данной машины.*

int *- тип данных знакового целого числа.*

Система массового обслуживания:

```
typedef void* queue_t; // АТД очереди для СМО

typedef struct
{
    int runs; // количество запусков ОА
    double exec_time; // время работы ОА
} maintenance_unit_t; // обслуживающий аппарат (ОА)

typedef struct
{
    maintenance_unit_t unit; // обслуживающий аппарат
    queue_t q; // очердь
    size_t q_len; // длина очереди
    size_t cnt_req_in; // количество поступивших заявок
    size_t cnt_req_out; // количество ушедших заявок
    double exec_time; // время работы СМО
```

```
} queuing_system_t; // система массового обслуживания (СМО)
```

size_t *используется для задания размеров наших массивов, так как является особым макросом, значение которого позволяет адресовать массив любой теоретически возможной длины для данной машины.*

int *- тип данных знакового целого числа.*

Описание алгоритма

Симулия СМО - **qsystem_simulate**:

1. Выполняем дальнейшие действия пока не уйдет нужное количество заявок. (В нашем случае 1000)

2. Отправляем элемент в ОА и добавляем в сумму время его обслуживания.

3. Если сумма превышает время добавления заявки или очередь пуста то добавляем новую заявку.

   Если сумма превышает время добавления заявки, то сумме пристаем разницу во времени.

4. Возвращаемся к пункту 1.

Проверка на основе теоретического расчета

	Фактические данные	Теоретические данные	Погрешность
Кол-во вошедших заявок	1002	998	0,3992 %
Время моделирования	2994	3000	0.2004%

Для проверки правильности работы системы по входу общее время моделирования делим на время прихода одной заявки:

2994 / 3 = 998 заявок

Таким образом, определяем предполагаемое количество вошедших заявок. Фактически их пришло 1002, т.е. погрешность составляет

|100%*(1002-998)/1002| = 0,3992%

Проверим правильность работы системы по выходу.
За время моделирования аппарат простаивал – 488.3925 е.в..

Время моделирования должно было составить: 3000 е.в., а оно составило – 2994. Получаем погрешность:

$$100\%(3000 - 2994)/2994 = 0.2004\%.$$

Можем вычислить погрешность по-другому. Расчетное время работы равно 3000е.в., а фактическое − 2994 То есть, погрешность составит $100\%(3000\text{-}2994)/3000 = 0.2\%$.

Таким образом, проверка работы системы по входным и выходным данным показала, что погрешность работы системы составляет не более 1.1%, что удовлетворяет поставленному условию.

Оценка эффективности

Для оценки эффективности была проведена симуляция для двух реализация.

Алгоритм измерения времени:

1. Измеряется текущее время - начало.
2. Запускается выполнение функции.
3. Измеряется текущее время - конец.
4. Вычисляется количество наносекунд прошедших от начала и до конца.
5. Полученное значение добавляется в сумму.

6. Данное измерение производится 100 раз, вычисляя итоговую сумму всех запусков.
7. Ответ - среднее арифметическое.

Удаление и добавление:

	Добавление		Удаление	
	Массив	**Связный список**	**Массив**	**Связный список**
1	231	61	32	58
10	380	613	329	621
100	3503	6178	3275	6300
500	23629	31041	22070	31581
1000	36304	57477	35324	59278

Можем заметить, что время добавления и удаления элемента в очереди при реализации в виде массива происходит быстрее, нежели при альтернативной реализации. Это происходит из-за того, что для реализации на связном списке каждая операция выделяет или очищает память.

	Массив	Связный список
Время, наносекунды	94867	297449
Разница Массив - Связный список		202582
Отношение Массив / Связный список		32

Таким образом можем заметить, реализация на основе массива оказалась быстрее на 68%. Это может происходить так как в симуляции выполняется множество операций удаления и добавления, по итогам сравнения реализаций для удаления и добавления - массив превосходит альтернативную реализацию, соотвественно и здесь прослеживается данная зависимость.

Память

Реализация	Структура	Запись	Структура + 1 запись	Полный объем
Массив	48	4	52	4048
Связный список	16	16	32	64
Разница Массив - Связный список	32	-12	20	3984
Отношение Массив / Связный список	300	25	162,5	6325

Для полного объема при вычислении для массива было взято значение - 1000 записей, так как наполненность не влияет на итоговый объем памяти.
Для реализации на связном списке было взято значение - 3, так как в ходе проделанной работы было выяснено, что средняя длина очереди равна именно этому значению.

Можем заметить, что связный список дает огромный выигрыш по памяти - реализация на массиве составляет 6325% от реализации на связном списке.

Это происходит потому что случае с реализацией на массиве объем полезных данных не влияет на итоговый объем занятой памяти.
Таким образом наполненность полезными данными будем сильно влиять, но мы можем заметить, что очередь составляет малый объем полезных данных, и в этом случае реализация на связном списке гораздо выгоднее по памяти.

Вывод

В ходе проделанной работы, я приобрел навыки работы с типом данных «очередь», представленным в виде одномерного массива и односвязного линейного списка. Произвел оценку эффективности программы (при различной реализации) по времени и по используемому объему памяти.

По результатам, проведенной оценки эффективности, было выявлено, что реализация на массиве быстрее альтернативной реализации на 68%.
Но по объему занимаемой памяти реализация на связном списке превосходит реализацию на массиве. Было установлено, что объем полезных данных в очереди крайне мал по отношению к ее размеру, данным фактом обусловлены показатели при сравнении объема занимаемой памяти.

Среди адресов, выведенных в файл addrs.txt были отфильтрованы уникальные элементы, в результате, при 1000 заявках появляется около 60 уникальных адресов. То есть происходит фрагментация. Другими словами, благодаря просмотру адресов элементов связного списка, была выявлена фрагментация памяти при удалении и добавлении элементов в очередь. То есть при запросах на выделение и освобождение памяти под очередной элемент выделяется не та память, которая была только что освобождена при удалении элемента.

Подробные результаты и методы оценки эффективности программы можно изучить в пункте «Оценка эффективности».

Контрольные вопросы

1. **Что такое FIFO и LIFO?**

   First In – First Out (FIFO) или первым пришел – первым вышел.
   Используется в структуре данных очередь.
   Last In – First Out (FIFO) или последним пришел – первым вышел.
   Используется в структуре данных стек.

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

   При моделировании простейшей линейной очереди на основе одномерного массива выделяется последовательная область памяти из m мест по L байт, где L – размер поля данных для одного элемента размещаемого типа. В каждый текущий момент времени выделенная память может быть вся свободна, занята частично или занята полностью.

При физической реализации очереди в виде односвязного линейного списка дескриптор будет отличаться отсутствием верхней границы очереди. В этом случае объем очереди ограничивается только объемом доступной оперативной памяти. Память выделяется под каждый элемент очереди отдельно при надобности.

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

   Исключается элемент, находящийся по адресу «головы». Указатель на «голову» при этом переместится на предыдущий элемент (вперед). При этом стоит следить, что нужно сдвигать в начало массива указатель, если он перешел верхнюю границу.

   Исключается элемент, находящийся по адресу «головы». Указатель на «голову» при этом переместится на предыдущий элемент. При исключении элемента необходимо следить за опустошением очереди, а также не забывать освобождать память под удаляемый элемент.

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

   Элементы из очереди удаляются, и присмотр происходит в порядке добавления, т.е сохраняя исходный порядок.

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

   Эффективность реализации очереди зависит от эффективности ее базовых операций: удаления и добавления.

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

   Реализация на массиве более эффективна по скорости выполнения за счет, того, что не нужно выделять память при каждой операции удаления и добавления. При этом по памяти она будет не эффективна, так как не учитывает объем полезных данных.

   Реализация на связном списке менее эффективна по скорости за счет того, что каждая операция добавления и удаления требует работы с динамической памятью. Зато объем требуемой памяти будет прямо пропорционален объему полезных данных.

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

В процессе моделирования очереди может оказаться, что при последовательных запросах на выделение и освобождении памяти под очередной элемент выделяется не та память, которая была только что освобождена при удалении элемента. То есть может возникнуть фрагментация памяти.

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

   Схема выделения памяти, которая совмещает в себе слияние буферов. То есть буффер делится на несколько маленьких буферов, которые потом сливаются в итоговый.

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

   Выбирается первая первый попавшийся свободный блок памяти. Более эффективный по времени.

   Выбирается наиболее подходящий блок памяти. Менее эффективный по времени, но исключает возможную фрагментацию памяти.

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

    При реализации кольцевой очереди стоит обратить внимание на переходы через границы массива, обработку пустой очереди и переполнение.

    Также на освобождение и выделение памяти.

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

    При конструировании объекта указывается размер запрашиваемой под объект памяти, и, в случае успеха, выделенная область памяти, условно говоря, «изымается» из «кучи», становясь недоступной при последующих операциях выделения памяти.
    Противоположная по смыслу операция — освобождение занятой ранее под какой-либо объект памяти: освобождаемая память, также условно говоря, возвращается в «кучу» и становится доступной при дальнейших операциях выделения памяти.