



Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана»

ФАКУЛЬТЕТ _____ Информатика и системы управления
КАФЕДРА _____ Программное обеспечение ЭВМ и информационные технологии

Отчет по дисциплине «Типы и структуры данных» Лабораторная работа №5

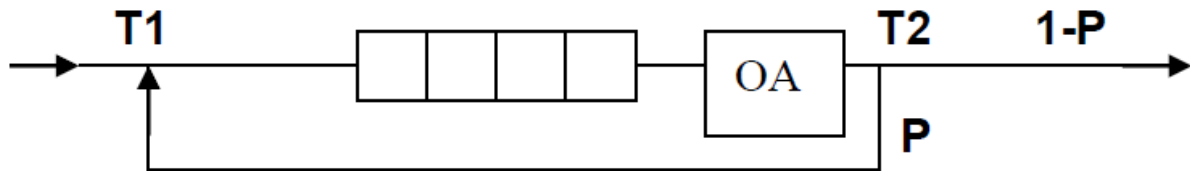
Вариант 3

Студент _____ Голикова С.М.
Группа _____ ИУ7-35Б

Москва, 2021

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 $T1$, 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 с вероятностью $P=0.8$ вновь поступает в "хвост" очереди, совершая новый цикл обслуживания, а с вероятностью $1-P$ 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, среднее время пребывания заявки в очереди, время простоя аппарата, количество срабатываний ОА. Обеспечить по требованию пользователя выдачу на экран адресов элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

2. Внешняя спецификация

а) Исходные данные и результаты

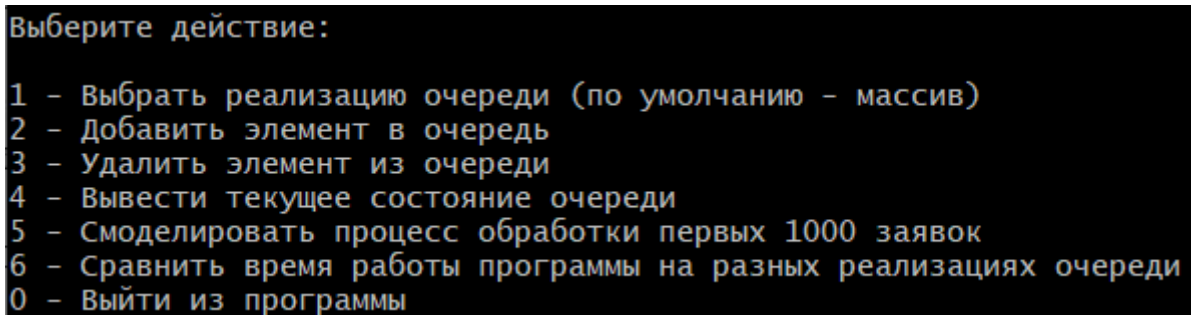
Входные данные:

- Целое число от 0 до 6 — номер пункта меню, который вызывает описанное в пункте действие
- Целое число — добавляемый элемент очереди

Допущения:

- 1) Очередь содержит только целые числа
- 2) Максимальный размер очереди — 1000

Выходные данные:



```
Выберите действие:
1 - Выбрать реализацию очереди (по умолчанию - массив)
2 - Добавить элемент в очередь
3 - Удалить элемент из очереди
4 - Вывести текущее состояние очереди
5 - Смоделировать процесс обработки первых 1000 заявок
6 - Сравнить время работы программы на разных реализациях очереди
0 - Выйти из программы
```

В зависимости от пункта меню (см.выше):

- Целые числа — элементы очереди.
- Адреса — адреса элементов очереди, реализованной в виде списка, и свободных областей.
- Результаты моделирования — общее время моделирования и количество вошедших в систему и вышедших из нее заявок, среднее время пребывания заявки в очереди, время простоя аппарата, количество срабатываний ОА.
- Результаты анализа эффективности различных реализаций очереди — время и объем памяти.

b) Задачи, реализуемые программой

- Операции работы с очередью — добавление элемента, удаление элемента, вывод текущего состояния и адресов элементов, вывод свободных областей.
- Моделирование процесса обслуживания 1000 заявок заданной системой.
- Сравнение эффективности различных реализаций очереди.

c) Способ обращения к программе

Программа запускается из терминала в директории с проектом при помощи команды «./app.exe».

d) Возможные аварийные ситуации и ошибки пользователя

Аварийные ситуации и ошибки:

- некорректный выбор пункта меню
- ввод не целого числа при добавлении в очередь
- переполнение очереди
- удаление элемента из пустой очереди
- вывод пустой очереди

В случае аварийной ситуации пользователю выдается сообщение об ошибке и происходит возвращение в меню.

3. Внутренние структуры данных

```
#define MAX_QUEUE_SIZE 1000    // максимальный размер очереди

// очередь в виде массива

typedef struct
{
    int data[MAX_QUEUE_SIZE]; // массив элементов
    int head_idx;             // индекс "головы" очереди
    int tail_idx;             // индекс "хвоста" очереди
} array_queue_t;

// узел списка с информацией об элементе

typedef struct node
{
    int data;                  // значение элемента
    struct node *next;         // указатель на следующий
                               // элемент очереди
} node_t;

// очередь в виде списка

typedef struct list_queue_t
{
    node_t *head;              // указатель на "голову" очереди
    node_t *tail;              // указатель на "хвост" очереди
} list_queue_t;

// массив свободных областей

typedef struct
{
    size_t address[MAX_QUEUE_SIZE]; // массив областей
    int size;                        // размер массива областей
} free_areas_t;

#endif
```

4. Описание алгоритма

Алгоритм моделирования процесса обслуживания 1000 заявок:

Все действия выполняются, пока 1000 заявок не покинут обслуживающий аппарат.

Случайным образом генерируются время прихода новой заявки и время обслуживания заявки из очереди. Выбирается меньшее время и выполняется соответствующее действие (добавление заявки в очередь или извлечение из очереди). Это время добавляется к общему времени моделирования, а второе время уменьшается на величину первого. Первое время снова генерируется случайным образом.

После извлечения заявки из очереди случайным образом генерируется вероятность ее возврата, и заявка соответственно либо возвращается в очередь, либо покидает систему (тогда количество ушедших заявок увеличивается на 1).

Если очередь пуста во время работы ОА, увеличивается время простоя.

5. Основные функции, используемые в программе

Имя	<code>int choose_action(void);</code>
Функция	Выбор действия в меню
Имя	<code>int array_input_element(array_queue_t *queue);</code> <code>int list_input_element(list_queue_t *queue, free_areas_t *areas);</code>
Функция	Ввод элементов с добавлением их в очередь (1 - массив, 2 - список)
Имя	<code>int array_push(array_queue_t *queue, int element);</code> <code>int list_push(list_queue_t *queue, int element);</code>
Функция	Добавление одного элемента в очередь (1 - массив, 2 - список)
Имя	<code>int array_delete_element(array_queue_t *queue);</code> <code>int list_delete_element(list_queue_t *queue, free_areas_t *areas);</code>
Функция	Удаление последнего элемента из очереди (1 - массив, 2 - список)
Имя	<code>int array_pop(array_queue_t *queue);</code> <code>int list_pop(list_queue_t *queue, size_t *address);</code>
Функция	Извлечение последнего элемента из очереди (1 - массив, 2 - список)
Имя	<code>void array_print(array_queue_t *queue);</code> <code>void list_print(list_queue_t *queue, free_areas_t *areas);</code>
Функция	Вывод текущего состояния очереди (1 - массив, 2 - список)
Имя	<code>void array_queue_simulation(void);</code> <code>void list_queue_simulation(void);</code>
Функция	Моделирование процесса обслуживания (1 - массив, 2 - список)

Имя	<code>void print_free_areas(free_areas_t *areas);</code> <code>void exclude_addresses(free_areas_t *areas, size_t addr);</code>
Функция	Работа с освобождаемыми областями памяти
Имя	<code>void free_list(list_queue_t *queue);</code>
Функция	Освобождение списка

6. Тесты

№ теста	Ввод	Действие	Вывод
1	7	Ввод неверного пункта меню	Ошибка ввода: необходимо ввести номер одного из предложенных вариантов
2	2 abc	Попытка добавить в очередь не целое число	Ошибка ввода: введено не целое число
3	2 1	Добавление (n + 1)-го элемента, где n - максимальный размер очереди	Невозможно добавить элемент: очередь заполнена
4	3	Удаление элемента из пустой очереди	Невозможно удалить элемент: очередь пуста
5	4	Вывод пустой очереди	Текущее состояние очереди (представление - массив): Очередь пуста
6	2 1	Валидное добавление элемента в очередь	Элемент 1 успешно добавлен в очередь
7	3	Валидное удаление элемента	Элемент 1 успешно удален
8	4	Вывод очереди в виде массива	Текущее состояние очереди (представление - массив):

			1 2 3 4 5
9	4	Вывод очереди в виде односвязного списка	<p>Текущее состояние очереди (представление - список):</p> <p>head -> 1 : 00000217b09c81a0 2 : 00000217b09c8260 3 : 00000217b09c8240 4 : 00000217b09c8200 tail -> 5 : 00000217b09c8160</p> <p>Массив освободившихся областей:</p> <p>Массив освободившихся областей пуст</p>
10	4	Вывод очереди в виде односвязного списка с наличием освободившихся областей	<p>Текущее состояние очереди (представление - список):</p> <p>head -> 3 : 00000217b09c8240 4 : 00000217b09c8200 tail -> 5 : 00000217b09c8160</p> <p>Массив освободившихся областей:</p> <p>217b09c81a0 217b09c8260</p>

7. Теоретический расчет времени моделирования и сравнение его с полученными на практике результатами

Время прихода заявки лежит в интервале от 0 до 6 е.в. Для прихода 1000 заявок, если каждая из них приходит в среднем за 3 е.в., потребуется *3000 е.в.*

Время обработки заявки лежит в интервале от 0 до 1 е.в., значит, среднее значение времени обработки одной заявки 0.5 е.в. Вероятность возврата заявки в очередь 0.8, а вероятность выхода, соответственно, 0.2. Тогда для обработки заявки необходимо 5 проходов. Среднее время обработки одной заявки будет: $0.5 * 5 = 2.5$ е.в., а общее время обработки 1000 заявок будет: $1000 * 2.5 = 2500$ е.в.

Следовательно, ОА работает с простоем и время моделирования будет определяться временем прихода заявок, т.е. оно должно быть равно *3000 е.в.*

Время простоя будет равно разнице между временем обработки заявок и временем их обслуживания: $3000 - 2500 = 500$ е.в.

На практике при выполнении программы получены следующие результаты:

```
=====
Ожидаемое время моделирования: 3000 е.в.
Полученное время моделирования: 3057.994385 е.в.

Ожидаемое время простоя: 500 е.в
Полученное время простоя: 510.070129 е.в.

Количество вошедших заявок: 5008, из них повторно возвращенных заявок: 4008
Количество срабатываний ОА: 5008, из них успешно обработанных заявок: 1000

Время среднего пребывания заявки в очереди: 1.201864 е.в.

Проверка работы системы по входным и выходным данным:
Погрешность по входу: 1.896486%
Погрешность по выходу: 1.933146%
=====
```

Проверка работы системы по входу:

Общее время моделирования делим на время прихода одной заявки:

$3057.99 / 3 = 1019.33$ заявок - предполагаемое количество вошедших заявок

Фактически их пришло $5008 - 4008 = 1000$ (количество вошедших минус количество возвращенных в очередь), т.е. погрешность составляет

$$|100\% * (1000 - 1019.33) / 1019.33| = 1.896\%$$

Проверка работы системы по выходу:

Расчетное время работы равно 3000 е.в., а фактическое – 3057.99. То есть, погрешность составит

$$|100\% * (3057.99 - 3000) / 3000| = 1.933\%$$

Таким образом, проверка работы системы по входным и выходным данным показала, что погрешность работы системы составляет не более 2%, что удовлетворяет поставленному условию.

8. Анализ эффективности

Время добавления элемента в очередь (такты):

Количество элементов	Массив	Список
50	2090	26073
100	4552	38242
500	21635	122029

Время удаления элемента из очереди (такты):

Количество элементов	Массив	Список
50	1814	19535
100	3973	29321
500	12951	100409

Объем памяти (байты)

Количество элементов	Массив	Список
50	4008	800
100	4008	1600
500	4008	8000

9. Выводы

Очередь, реализованная при помощи кольцевого статического массива, в несколько раз эффективнее по времени, чем очередь, реализованная при помощи односвязного списка, так как при работе с массивом нужно работать только с указателями, а в случае односвязного списка необходимо работать и с указателями, и с памятью (выделением и освобождением).

По памяти односвязный список эффективнее для очередей с размером примерно до 25% заполненности. Кроме того, список удобнее использовать в тех случаях, когда максимальный размер очереди неизвестен.

Фрагментация при работе с очередью, реализованной в виде односвязного списка, возникала в некоторых случаях.

10. Контрольные вопросы

1) Что такое FIFO и LIFO?

FIFO (First In – First Out) - принцип работы очереди: первым пришел – первым вышел. Очередь – это последовательный список переменной длины, включение элементов в который идет с одной стороны (с «хвоста»), а исключение – с другой стороны (с «головы»).

LIFO (Last In – First Out) - принцип работы стека: последним пришел – первым ушел. Стек – это последовательный список с переменной длиной, в котором включение и исключение элементов происходит только с одной стороны – с его вершины.

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

Реализация очереди:

- Связный список

Память выделяется по мере необходимости:

$(\text{sizeof}(\text{type}) + \text{sizeof}(\text{type_t*})) * \text{count}$,

где count — число элементов, type — тип элементов, type_t — тип узла.

- Статический кольцевой массив

Память выделяется в начале работы программы:

$\text{sizeof}(\text{type}) * \text{count}$,

где count — число элементов, type — тип элементов.

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

Реализация очереди:

- Связный список

Освобождается память, ранее занимаемая первым элементом (“головой” очереди), и смещается указатель, который указывает на “голову” очереди. Так, память освобождается по мере необходимости.

- Статический кольцевой массив

Меняется указатель на “голову” очереди. Память освобождается при завершении работы программы.

4) Что происходит с элементами очереди при ее просмотре?

Элементы извлекаются из очереди (уничтожаются).

5) От чего зависит эффективность физической реализации очереди?

При известном максимальном размере очереди эффективнее использовать кольцевой статический массив, так как по времени обработки очереди он выигрывает у связного списка.

Если максимальный размер очереди не известен, выгоднее использовать связный список, так как его размер ограничен лишь объемом оперативной памяти.

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

Реализация очереди:

- Связный список:

- 1) Достоинства: размер очереди ограничен лишь объемом оперативной памяти.
- 2) Недостатки: может возникнуть фрагментация памяти, а также такой способ реализации является проигрышным по времени.

- Статический кольцевой массив:

- 1) Достоинства: операции при такой реализации выполняются гораздо быстрее.
- 2) Недостатки: очередь всегда будет ограничена по размеру.

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

Фрагментация памяти - разбиение памяти на куски, которые лежат не рядом друг с другом. Так появляются фрагменты памяти, которые нельзя использовать (например, становится невозможно выделить блок памяти большого размера). Возникает в куче.

8) Для чего нужен алгоритм «близнецов»?

Алгоритм «близнецов» используется для выделения участков памяти по запросу.

Идея этого алгоритма состоит в том, что организуются списки свободных блоков отдельно для каждого размера $2k$, $0 \leq k \leq m$. Вся область памяти кучи состоит из $2m$ слов, которые, можно считать, имеют адреса с 0 по $2m - 1$. Первоначально свободным является весь блок из $2m$ слов. Далее, когда требуется блок из $2k$ слов, 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. Преимуществом этого алгоритма является скорость, но его реализация усложняется за счет необходимости вести систему списков свободных блоков.

9) Какие дисциплины выделения памяти вы знаете?

Дисциплины выделения памяти решают вопрос: какой из свободных участков должен быть выделен по запросу.

- «Самый подходящий»

Выделяется тот свободный участок, размер которого равен запрошенному или превышает его на минимальную величину.

- «Первый подходящий»

Выделяется первый найденный свободный участок, размер которого не меньше запрошенного.

Практически во всех случаях дисциплина "первый подходящий" эффективнее дисциплины "самый подходящий". Это объясняется, во-первых, тем, что при поиске первого подходящего не требуется просмотр всего списка или карты до конца, во-вторых, тем, что при выборе всякий раз "самого подходящего" остается больше свободных блоков маленького размера - внешних дыр.

10) На что необходимо обратить внимание при тестировании программы?

На аварийные ситуации (переполнение очереди при ее реализации массивом, пустая очередь) и на корректное освобождение памяти.

11) 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 пользователем памяти, тем самым делая его снова свободным.