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)

Отчет

по лабораторной работе №5
по теме
«Обработка очередей»
Вариант 2.

Дисциплина: Типы и структуры данных

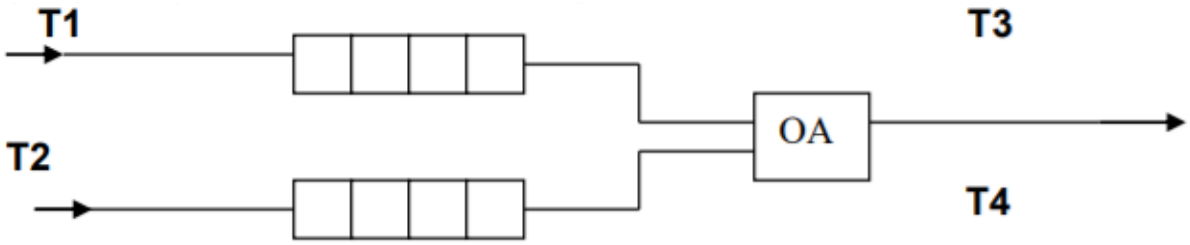
Студент ИУ7-31Б:
Косарев Алексей
Проверила:
Барышникова М.Ю.

Москва, 2021

1. Описание условия задачи

Смоделировать процесс обслуживания первых 1000 заявок 1-го типа. Выдать на экран после обслуживания каждых 100 заявок 1-го типа информацию о текущей и средней длине каждой очереди, количестве вошедших и вышедших заявок и о среднем времени пребывания заявок в очереди. В конце процесса выдать общее время моделирования и количество вошедших в систему и вышедших из нее заявок обоих типов. По требованию пользователя выдать на экран адреса элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

Система массового обслуживания состоит из обслуживающего аппарата (ОА) и двух очередей заявок двух типов.



Заявки 1-го и 2-го типов поступают в "хвосты" своих очередей по случайному закону с интервалами времени Т1 и Т2, равномерно распределенными от 1 до 5 и от 0 до 3 единиц времени (е.в.) соответственно. ОА ОА В ОА они поступают из "головы" очереди по одной и обслуживаются также равновероятно за времена Т3 и Т4, распределенные от 0 до 4 е.в. и от 0 до 1 е.в. соответственно, после чего покидают систему (все времена – вещественного типа). В начале процесса в системе заявок нет.

Заявка 2-го типа может войти в ОА, если в системе нет заявок 1-го типа. Если в момент обслуживания заявки 2-го типа в пустую очередь входит заявка 1-го типа, то она ждет первого освобождения ОА и далее поступает на обслуживание (система с относительным приоритетом).

2. Описание ТЗ

- Описание исходных данных

Исходными данными являются числовые данные, введенные пользователем, для выбора пунктов меню, а также времена поступления и обработки заявок.

Меню:

1. Моделирование с помощью массива
2. Моделирование с помощью списка
3. Настройка времен
4. Анализ времени

0. Выход из программы

- Описание выходных данных

- состояние очередей двух типов
- результаты моделирования
- адреса добавленных и удаленных элементов списка
- анализ работы с разными реализациями очереди по времени

- Пример работы и расчеты

1) Моделирование обработки заявок при изначальных параметрах времен:

```
--------- Modelling results ---------
Amount of in requests  (1 type): 1000
Amount of out requests (1 type): 1000
Amount of in requests  (2 type): 1976
Amount of out requests (2 type): 1950
Total modelling time: 2986.214057
Total stop time: 33.937284
Expected theoretical modelling time: 3000.000000
Defect: 0.459531
```

Проверим, действительно ли время моделирования должно быть ~3000 ед.вр. Так как среднее время поступления заявки 1 типа больше, чем время обработки этой же заявки, поэтому общее время моделирования будет определяться как “Среднее время поступления” * “Количество обработанных заявок” = 3 ед.вр. * 1000 = 3000 ед.вр. (в примере - 2986 ед.вр.).

Теперь посчитаем, сколько за это моделирование ОА смог бы обработать заявок 2 типа. Так как заявок 1 типа было обработано 1000 штук, значит ОА потратил на это “Количество обработанных заявок” * “Среднее время обработки” = 1000 * 2 ед.вр. = ~2000 ед.вр. Это значит, что у него было еще “Общее время моделирования” - “Время обработки заявок 1 типа” = 3000 ед.вр. - 2000 ед.вр. = 1000 ед.вр. на обработку заявок 2 типа. Следовательно, за это время он смог обработать “Время обработки заявок 2 типа” \ “Среднее время обработки заявки 2 типа” = 1000 ед.вр. \ 0.5 ед.вр. = 2000 заявок 2 типа (в примере - 1950 заявок 2 типа).

2) Моделирование обработки заявок со следующими временами: T1 = 0 - 2, T2 = 0 - 4, T3 = 2 - 5, T4 = 0 - 1:

```
--------- Modelling results ---------
Amount of in requests  (1 type): 3492
Amount of out requests (1 type): 1000
Amount of in requests  (2 type): 1715
Amount of out requests (2 type): 0
Total modelling time: 3465.564776
Total stop time: 0.000000
Expected theoretical modelling time: 3500.000000
Defect: 0.983864
```

В данном примере общее время моделирования определяется временем обработки 1000 заявок 1 типа, так как среднее время обработки заявки 1 типа больше, чем среднее время поступления заявки этого же типа. Отсюда же следует, что ОА не будет простаивать (что и видно в результатах моделирования), так как очередь с заявками 1 типа всегда не пуста к моменту окончания обработки очередной заявки. И именно поэтому по окончании моделирования не была обработана ни одна заявка 2 типа (потому что приоритет у заявки 1 типа).

3. Описание задачи, реализуемой в программе

Программа моделирует процесс обработки заявок, которые поступают из очередей. Очереди реализованы двумя способами (массивом и списком). При моделировании с использованием списков создается файл, в который записываются адреса вошедших и вышедших из очередей заявок. Также можно вывести время моделирования для разных реализаций очередей.

4. Способ обращения к программе

Обращение к программе происходит через консоль, путем запуска файла с расширением .exe (./main.exe).

5. Описание возможных аварийных ситуаций и ошибок пользователя

```
#define OK 0                        // Нет ошибок
#define QUEUE_OVERFLOW -5           // Переполнение очереди
#define QUEUE_EMPTY -2              // Пустая очередь
```

Также при некорректном вводе (например, вводе буквы вместо времени или вводе несуществующего пункта меню) программа будет запрашивать ввод у пользователя, пока тот не введет корректное значение.

6. Описание внутренних структур данных

- Структура очереди с помощью односвязного списка (1 элемент):

```
typedef struct list_queue_tt list_queue_t;

// Очередь в виде списка
struct list_queue_tt
{
   char type;
   list_queue_t *next;
};
```

- Структура очереди с помощью массива:

```
#define MAX_QUEUE_LEN 5000              // Максимальная длина очереди

// Очередь в виде массива
typedef struct
{
   char data[MAX_QUEUE_LEN];
   char *pointer;
   char *start;
   char *end;
```

```
} array_queue_t;
```

- Структура для сохранения данных об очереди:

```
typedef struct
{
   int cur_queue_len;      // Текущая длина очереди
   int avg_queue_len;      // Средняя длина очереди
   int in_amount;          // Количество вошедших заявок
   int out_amount;         // Количество вышедших заявок
   double avg_queue_time;  // Среднее время пребывания заявок в очереди
} queue_t;
```

- Структура данных моделирования:

```
typedef struct
{
   double modeling_time;   // Общее время моделирования
   double stop_time;       // Время простоя
   int in_amount_type_1;   // Количество вошедших в систему заявок 1
типа
   int in_amount_type_2;   // Количество вошедших в систему заявок 2
типа
   int out_amount_type_1;  // Количество вышедших из системы заявок 1
типа
   int out_amount_type_2;  // Количество вышедших из системы заявок 2
типа
} model_t;
```

7. Алгоритмы

Алгоритм моделирования обработки заявок из двух очередей:

1. Смотрим, закончилось ли время ожидания заявки какого-либо типа. Если закончилось, то в соответствующую очередь добавляем новый элемент и получаем новое время ожидания для этого типа очереди.
2. Смотрим, закончилось ли время обработки заявки в ОА. Если да, то вносим в ОА заявку 1 типа (если очередь 1 типа не пуста), иначе вносим заявку 2 типа и получаем новое время обработки для соответствующего типа заявки.
3. Если обработалась очередная сотня заявок 1 типа, то выводим информацию о состоянии двух очередей
4. Если обе очереди оказались пусты, значит увеличиваем время простоя на минимальное из двух времен ожидания заявок.
5. Находим минимальное из трех времен не равное нулю(ожидание заявки 1 типа, ожидание заявки 2 типа, обработка заявки) и уменьшаем каждое из этих времен на значение минимального.
6. Пункты 1-4 повторяются пока в ОА не обработается первая 1000 заявок первого типа.

Алгоритм добавления элемента в очередь:

1. Выделяем память под новый элемент

2. Добавляем значение (тип заявки) нового элемента в выделенную память
3. В указатель нового элемента добавляем указатель на элемент, который находился в конце очереди

Алгоритм удаления элемента из очереди:

1. Переносим указатель очереди на предыдущий элемент
2. Освобождаем память из под удаляемого элемента

8. Тесты

Положительные тесты:

1. Моделирование с изначальными настройками времени.
2. Моделирование с настройками времени, с которыми не будет простоя в ОА.
3. Моделирование с настройками времени, при которых общее время моделирования вычисляется по времени работы ОА.
4. Моделирование с настройками времени, при которых общее время моделирования вычисляется по времени поступления заявок 1 типа в очередь.

Негативные тесты:

1. Некорректный выбор пункта меню (ввод буквы).
2. Некорректный выбор пункта меню (ввод несуществующего номера пункта).
3. Некорректный ввод значения времени (ввод буквы).
4. Некорректный ввод значения времени (ввод отрицательного значения).
5. Переполнение очереди-массива (например, при таких настройках времени, когда заявки 2 типа вообще не попадают в ОА ни разу).

9. Временная эффективность и затраты памяти

Время моделирования для изначальных настроек времени:

T1, T2	modelling time, ms	
T3, T4	list	array
1.000 - 5.000, 0.000 - 3.000	472	198
0.000 - 4.000, 0.000 - 1.000		

Время моделирования для следующих времен: T1 = 0 - 2, T2 = 0 - 4, T3 = 2 - 4, T4 = 0 - 1:

T1, T2 / T3, T4	modelling time, ms	
	list	array
0.000 - 2.000, 0.000 - 4.000	13573	166
2.000 - 4.000, 0.000 - 1.000		

Время моделирования для следующих времен: T1 = 2 - 4, T2 = 4 - 6, T3 = 0 - 2, T4 = 0 - 1:

T1, T2 / T3, T4	modelling time, ms	
	list	array
2.000 - 4.000, 4.000 - 6.000	199	98
0.000 - 2.000, 0.000 - 1.000		

Из результатов анализа использования разных типов представления очереди (в виде массива и в виде списка) видно, что по времени очередь в виде массива быстрее в любом случае, чем очередь в виде односвязного списка.

```
Memory:
Struct for list element (bytes): 16
Struct for array queue (bytes): 5024 (for 5000 elements)
Required memory for 5000 elements using list: 80000
```

По памяти выгодней использовать односвязный список, только если длина очереди намного меньше длины массива, выделенного под очередь, иначе - выгодней массив.

Из анализа видно, что структура одного элемента односвязного списка занимает 16 байт, в то время как структура с массивом на 5000 элементов занимает 5024 байта. Значит, используя список для создания очереди на те же 5000 элементов, мы потратим в 80000 \ 5024 = ~16 раз больше памяти, чем используя массив.

Посчитаем, до какого процента заполненности массива выгодней использовать список:

Используя 5024 байта можно получить очередь-список длиною 5024 \ 16 = 314 элементов. 314 элементов это 314 \ 5000 = 0.0628 (6.3%) от общей длины массива. Следовательно в данном примере эффективней по памяти использовать список, только когда количество элементов в очереди не превышает 6.3% от заданной длины массива.

10. Вывод

В процессе выполнения данной лабораторной работы я изучил два разных варианта реализации очереди (с помощью массива и списка).

Проведя анализ обработки двух реализаций очередей, я понял, что по времени массив абсолютно эффективнее для реализации очереди, чем односвязный список.

Вопрос выбора между массивом и списком для эффективного использования памяти зависит от количества элементов в очередях. Если элементов в очередях будет намного меньше, чем выделенная длина массива, а именно ~6.3% от длины, то выгодней использовать односвязный список. Если же количество элементов в очередях будет большим, то лучше пользоваться реализацией массивом, так как это значительно экономит память (при использовании списка требуется в 16 раз больше памяти, чем при использовании массива).

Если в задаче заранее известно количество элементов очереди, то лучше использовать массив для его реализации. Если же не понятно, сколько в очереди будет элементов, то рациональней использовать список.

11. Ответы на контрольные вопросы

1. Что такое FIFO и LIFO?
   LIFO (Last In – First Out) - принцип работы стека, последним пришел – первым ушел.
   FIFO (First In – First Out) - принцип работы очереди, первым пришел - первым ушел.

2. Каким образом, и какой объем памяти выделяется под хранение очереди при различной ее реализации?
   Если очередь реализована в виде статического или динамического массива (вектора), то для ее хранения обычно отводится непрерывная область памяти ограниченного размера, имеющая нижнюю и верхнюю границу, которая выделяется в начале программы сразу под все элементы массива.
   Если очередь реализована в виде односвязного линейного списка, то для его хранения отводится указатель на структуру, содержащую указатель на такую же структуру и само значение элемента (в моем случае char type). При каждом добавлении элемента выделяется новая область памяти, адрес которой записывается в указатель на конец очереди (вход в очередь).

3. Каким образом освобождается память при удалении элемента из очереди при ее различной реализации?
   При реализации очереди с помощью статического массива память выделяется при компиляции и не меняется во время работы программы. В конце работы программы освобождается память из под массива.
   При реализации очереди с помощью линейного односвязного списка при удалении элемента сначала по указателю на начало очереди считывается информация об исключаемом элементе, а затем указатель смещается к предыдущему элементу (то есть второму элементу в очереди). После чего освобождается память, выделенная под элемент.

4. Что происходит с элементами очереди при ее просмотре?
   Если очередь реализуется в классическом виде, то при ее просмотре элементы удаляются.

5. От чего зависит эффективность физической реализации очереди?
   Если очередь реализована массивом, то будет меньше затрат по времени и памяти, но массив ограничен в размере. А односвязный список наоборот, медленнее по времени и требует больше памяти, однако в некоторых случаях список все же экономит память.

6. Каковы достоинства и недостатки различных реализаций очереди в зависимости от выполняемых над ней операций?
   При реализации очереди в виде массива проще обращаться, например, к предыдущему элементу. В очереди-списке сделать это сложнее, так как каждый элемент списка указывает на следующий элемент в очереди, а не предыдущий.

7. Что такое фрагментация памяти, и в какой части ОП она возникает?
   Фрагментация это выделение непоследовательных адресов в памяти.

8. Для чего нужен алгоритм «близнецов».
   Алгоритм близнецов значительно снижает фрагментацию памяти и резко ускоряет поиск блоков. Наиболее важным преимуществом этого подхода является то, что даже в наихудшем случае время поиска не превышает O(log(Smax)- log(Smin)), где Smax Smin - соответственно максимальный и минимальный размеры используемых блоков.
9. Какие дисциплины выделения памяти вы знаете?
   Две дисциплины: “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 (эффективнее).

10. На что необходимо обратить внимание при тестировании программы?
    - Проверить правильность расчета общего времени моделирования,т.е., когда время моделирования определяется временем обработки заявок или временем прихода заявок.
    - Отслеживать переполнение очереди (для реализации очереди массивом).

11. Каким образом физически выделяется и освобождается память при динамических запросах?
    Память представляет собой бинарную кучу с признаком — занятость - незанятость ячейки. Динамический запрос меняет признак ячейки.