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)

Лабораторная Работа №5 «Обработка очередей»

Вариант №2

Студент **Шахнович Дмитрий Сергеевич**

Группа **ИУ7-22Б**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

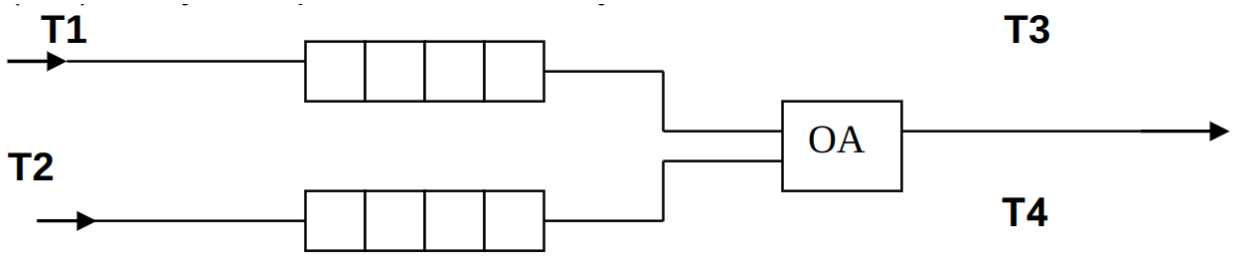
Студент **Шахнович Д.С.**

Оценка _____

2023 г.

Описание условия задачи

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.

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). 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.

Техническое задание

Исходные данные:

Для симуляции задаются граничные значения интервалов создания и выполнения заявок, а также необходимое количество заявок 1-го типа для завершения симуляции.

В меню действия заданы числами 0-4, которые включают запуск симуляция со статическим массивом и списком, а также их сравнение.

Выходные данные:

Программа может выдавать данные о симуляции, для очереди-списка вывод информации об адресах заявок, а также результаты сравнения двух симуляций.

Описание задания:

Проведение симуляций очередей заявок.

Способы обращения к программе:

Запуск программы через терминал, затем управление программой с помощью меню. Пункты меню:

1 — Провести симуляцию на статическом массиве.

2 — Провести симуляцию на списке.

3 — Изменить параметры симуляции.

4 — Сравнить реализации симуляций.

0 — Выход.

Аварийные ситуации:

1) Ввод несуществующей команды в меню;

   Сообщение: «Ошибка: Некорректная команда.»

2) Ошибка ввода/вывода;

   Сообщение: «Ошибка функций ввода/вывода.»

3) Переполнение статического очереди;

   Сообщение: «Ошибка: очередь переполнена.»

4) Неудачная попытка выделения памяти;

   Сообщение: «Ошибка выделения памяти.»

5) Ввод литералов или чисел вне запрашиваемого диапазона;

   Сообщение: «Ошибка: Некорректный формат ввода.»

Описание структур данных

```
/// Структура заявки
struct request_t
{
   double exec_time; /// Время, требуемое для выполнения заявки
   double crt_time; /// Время создания заявки
};
```

```
/// Очередь заявок в виде статического массива. Массив закольцован.
struct static_queue
{
   request_t arr[STATIC_QUEUE_SIZE]; /// Статический массив заявок.
   request_t *end; /// Указатель на физический конец очереди.
   request_t *pin; /// Указатель на входную часть массива.
   request_t *pout; /// Указатель на выходную часть массива.
   int full; /// Флаг заполненности очереди.
};
```

```
/// Узел очереди-списка заявок
struct qnode
{
   request_t data; /// Часть узла с данными в виде заявок.
   qnode* next; /// Указатель на следующий узел.
};

/// Очередь-список
struct list_queue
{
   qnode *pin; /// Указатель на входную часть списка.
   qnode *pout; /// Указатель на выходную часть списка.
};
```

```
/// Узел обычного списка.
struct node_t
{
   void *data; /// Указатель на данные узла.
   node_t *next; /// Указатель на следующий узел.
};
```

Описание алгоритма

1. Вывести пользователю меню и ожидать ввода номера команды;
2. В зависимости от выбора пункта провести симуляцию, сравнить реализации или изменить параметры симуляции.

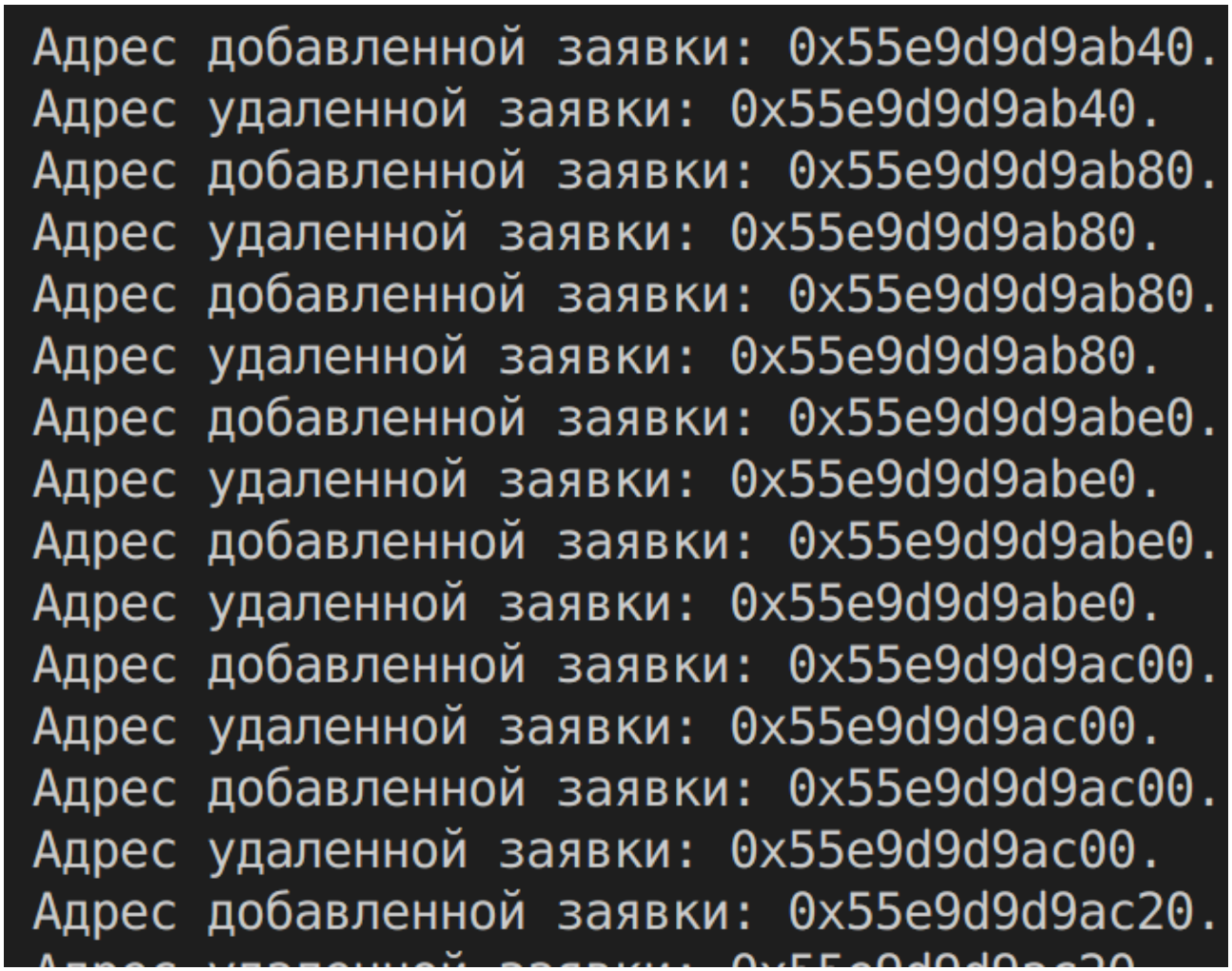
При возникновении ошибок или завершения пункта вернуться к выбору пункта меню.

Тестовые данные

Негативные тесты			
1	Ввести некорректный код в меню.	12	Ошибка: Некорректная команда.
2	При запросе числа ввести литерал.	sda	Ошибка: Некорректный формат ввода.
3	Ввести отрицательное число для времени или кол-ва заявок.	-123	Ошибка: Некорректный формат ввода.
4	Ввести некорректный диапазон времен(не по возрастанию числа).	12 1	Ошибка: Некорректный формат ввода.

Фрагментация памяти.

При реализации очереди списком может возникать фрагментация памяти:



Как видно, в списке присутствуют моменты, когда после удаления заявки следующая помещается не в освобожденную, а дальше.

Теоретические расчеты

Рассмотрим исходное условие:

Время прихода заявки 1-го типа от 1 до 5 у.е., среднее время — 3 у.е

Время выполнения заявки 1-го типа от 0 до 4 у.е., среднее время — 2 у.е.

Время прихода заявки 2-го типа от 0 до 3 у.е., среднее время — 1.5 у.е

Время выполнения заявки от 2-го типа от 0 до 1 у.е. среднее время 0.5 у.е.

Среднее время прихода заявок 1-го типа > времени выполнения, это значит что время симуляции определяется временем прихода заявок.

Значит общее время работы: 3 * 1000 = 3000 у.е

При этом за время прихода заявок 1-го типа приходят 2 заявки 2-го типа и их выполнение ровно влезает в разницу между выполнением и приходом заявок 1-го типа.

За время работы придет 3000/1.5 = 2000 заявок 2-го типа. По соображениям, расписанным выше, почти все заявки 2-го типа должны быть выполнены.

Значит количество выполненных заявок 2-го типа — 2000

Ожидаемое время простоя равно нулю, так как выполнение заявки 1-го типа и 2-х заявок 2-го типа равно времени прихода заявок 1-го типа.

Рассмотрим симуляцию в программе:

```
Количество выполненных запросов 1: 1000
Количество выполненных запросов 2: 1933
Количество вошедших запросов 1: 1001
Количество вошедших запросов 2: 1962
Среднее время выполнения заявок 1: 2.01
Среднее время выполнения заявок 2: 0.49
Общее время: 2965.67
Время работы:2961.89
Время простоя: 3.78

Ожидаемое количество выполненных запросов 1: 1000.00
Ожидаемое количество выполненных запросов 2: 2000.00
Ожидаемое количество вошедших запросов 1: 1000.00
Ожидаемое количество вошедших запросов 2: 2000.00
Ожидаемое среднее время выполнения заявок 1: 2.00
Ожидаемое среднее время выполнения заявок 2: 0.50
Ожидаемое общее время: 3000.00
Ожидаемое время простоя: 0.00
Ожидаемое время работы: 3000.00

Погрешность общего времени: 1.14 %
Погрешность времени работы: 1.27 %
Погрешность вошедших заявок 1: 0.10 %
Погрешность вошедших заявок 2: 1.90 %
```

Как видно все данные в пределах погрешности 2%.

Рассмотрим ситуацию, когда время выполнения определяет время работы программы. Заменим интервал выполнения 1-ой заявки на интервал от 0 до 10 у.е. Среднее время — 5 у.е.

Так как оно больше времени прихода, то именно оно определяет время работы.

Общее время работы — 5 * 1000 = 5000 у.е

За это время придет 5000 / 3 = 1667 Заявок 1-го типа, то есть они будут накапливаться. За это время придет 5000 / 1.5 = 3333 заявок 2-го типа, но ни одна из них не выполнится, так как ОА все время будет занят заявками 1-го типа.

Рассмотрим симуляцию:

```
Количество выполненных запросов 1: 1000
Количество выполненных запросов 2: 2
Количество вошедших запросов 1: 1693
Количество вошедших запросов 2: 3336
Среднее время выполнения заявок 1: 5.02
Среднее время выполнения заявок 2: 0.58
Общее время: 5023.85
Время работы:5023.54
Время простоя: 0.31

Ожидаемое количество выполненных запросов 1: 1000.00
Ожидаемое количество выполненных запросов 2: 0.00
Ожидаемое количество вошедших запросов 1: 1666.67
Ожидаемое количество вошедших запросов 2: 3333.33
Ожидаемое среднее время выполнения заявок 1: 5.00
Ожидаемое среднее время выполнения заявок 2: 0.50
Ожидаемое общее время: 5000.00
Ожидаемое время простоя: 0.00
Ожидаемое время работы: 5000.00

Погрешность общего времени: 0.48 %
Погрешность времени работы: 0.47 %
Погрешность вошедших заявок 1: 1.58 %
Погрешность вошедших заявок 2: 0.08 %
```

Как видно, все погрешности снова в пределе 2%.

Замеры эффективности

Замеры операций проводились следующим образом: Для каждого количества заявок 1-го типа 1000 раз проводилась симуляция с замером времени и максимального количества заявок в очередях. По всем прогонам эти значения усреднялись, получая результирующие значения в таблицах:

Время выполнения вычислений

Кол-во заявок 1-го типа	Время в статической очереди, мкс	Время в списке-очереди, мкс	Отношение времени списка к статическому
100	20	14	0.7
500	68	73	1.07
1000	150	153	1.02

2000	252	293	1.16
3000	371	436	1.17
5000	595	726	1.22
10000	1186	1460	1.23
20000	2361	2933	1.24

Объемы памяти

Кол-во заявок 1-го типа	Объем статической очереди, байт	Объем списка-очереди,байт	Отношения объема списка к статическому
100	588	856	1.45
500	1356	1984	1.46
1000	1932	2824	1.46
2000	2668	4048	1.51
3000	3276	4984	1.52
5000	4236	6400	1.51
10000	6028	9040	1.5
20000	8540	13024	1.52

Как видно из таблиц использования очереди в виде списка увеличивает время симуляции в ~1.2 раза, хотя при количестве заявок ~ список незначительно выигрывает по скорости. По памяти же, список всегда проигрывает: список занимает ~ в 1.5 раз больше памяти.

Ответы на вопросы

1. Что такое FIFO и LIFO?

LIFO и FIFO — дисциплины организации элементов в структуре данных. В LIFO первым выходит тот, элемент, который зашел последним, а В FIFO — первым. Яркими представилемя являются — Стек- LIFO, Очередь — FIFO.

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

В случае реализации с помощью списка память выделяется динамически под каждый узел очереди. В случае реализации статическим массивом память выделяется на стеке единым куском под все элементы. Как показал эксперимент, для хранения очереди списком требуется больше памяти чем для хранения статическим массивом.

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

При реализации статическим массивом память из-под элемента не освобождается, так как она выделена на стеке. В такой реализации просто сдвигается указатель выхода очереди на следующий элемент. При реализации списком при удалении память освобождается из-под узла элемента.

4. Что происходит с элементами очереди при ее просмотре?

В классической реализации при просмотре очереди ее элементы из нее удаляются.

5. От чего зависит эффективность физической реализации очереди?

Эффективность реализации зависит от вида выделения памяти, так как динамическое выделение памяти как правило замедляет программу, а также от размера дополнительных данных, таких как указатель на следующий элемент при реализации списком.

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

Особенностями реализации очереди статическим массивом является большая скорость выполнения операций добавления и удаления и эффективность по памяти, однако при данной реализации может возникнуть переполнение массива и такая реализация сложнее для написания, из-за кольцевой структуры. Однако если структура статического массива не кольцевая, то удаление элемента может быть сильно замедлено из-за сдвига.

Реализация списком, как правило, работает медленнее из динамического выделения памяти и занимает больше памяти, однако размер такой очереди зависит только от объема оперативной памяти.

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

Фрагментация памяти - явление, при котором данные некоторой структуры данных разбросаны по разным участкам памяти, а не лежат друг за другом. Такое может возникать в куче при динамическом выделении/освобождении памяти.

8. Для чего нужен алгоритм «близнецов».

Алгоритм близнецов используется для быстрого(O(logn)) выделения памяти по запросу. Суть алгоритма в том, чтобы при запросе памяти разбивать её на блоки кратные степени двойки, доходя до минимального размера, необходимого по запросу.

9. Какие дисциплины выделения памяти вы знаете?

Выделения памяти может быть статическим или динамическим.

Статическое выделение памяти происходит для данных, определенных до начала программы, то есть литералы.

Динамически выделяются все остальные, при этом память для объектов может быть зарезервирована, если их размер известен до начала программы или выделятся во время выполнения.

10. На что необходимо обратить внимание при тестировании программы?

При реализации списком важно следить, что все узлы при удалении освобождаются, и при завершении программы все ресурсы освобождены.

При реализации массивом важно следить за тем, чтобы данные не начали записываться в области вне массива.

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

При динамическом запросе программа запрашивает у ОС блок памяти необходимого размера. ОС находит блок памяти такого размера и заносит его адрес и размер в таблицу адресов и затем передает адрес в программу.

Выводы

Во многих системах может использоваться тип данных очередь и при её использовании разработчик должен решить какую её реализацию стоит использовать в зависимости от требований к системе: для скорости лучше использовать статический массив, но может возникнуть переполнение. Если важнее удобство и неограниченный размер, то лучше использовать реализацию списком.