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 ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

Группа ИУ7-35Б

Тема работы **Обработка очередей**

Студент Баранов Николай Алексеевич

Преподаватель Барышникова Марина Юрьевна

2022 г.

Цель работы: Приобрести навыки работы с типом данных «очередь», представленным в виде одномерного массива и односвязного линейного списка, провести сравнительный анализ реализации алгоритмов включения и исключения элементов из очереди при использовании указанных структур данных, оценить эффективность программы по времени и по используемому объёму памяти.

1. Описание условия задачи

Система массового обслуживания состоит из обслуживающего аппарата и двух очередей заявок двух типов. Заявки 1-го и 2-го типов поступают в хвосты своих очередей по случайному закону с интервалами времени T1 и T2, в ОА они поступают из головы очереди по одной и обслуживаются также равновероятно за времена T3 и T4, после чего покидают систему (все времена – вещественного типа). В начале процесса заявок в системе нет.

Заявка может войти в ОА, если или она вошла в пустую систему, или перед ней обслуживалась заявка того же типа, или если обслуживалась заявка другого типа, но заявок другого типа не осталось (система с чередующимся приоритетом).

Смоделировать процесс обслуживания первых 1000 заявок 1-го типа, выдавая после обслуживания каждых 100 заявок информацию о текущей и средней длине очереди, а в конце процесса – общее время моделирования и количество вошедших в систему и вышедших из неё заявок всех типов.

2. Описание ТЗ

2.1. Описание исходных данных и результатов.

При чтении интервала времени программа считывает 2 числа с плавающей точкой – начало и конец интервала. Начало интервала всегда меньше его конца, числа не могут быть отрицательными. Среднее время интервала обработки для второй очереди должно быть меньше среднего времени интервала поступления в хвост очереди, так как иначе есть риск, что приоритет системы никогда не изменится, и 1000 заявок 1 типа никогда не будут обработаны.

Выбор действия происходит в меню, где вводится число – номер действия.

После обслуживания каждых 100 заявок выводится информация о текущей и средней длине очереди для каждой из них, в конце процесса выводится также время работы ОА, время простоя ОА, общее время работы, ожидаемое время работы и погрешность при измерении.

При сравнении эффективности для каждого способа выводится среднее время работы и максимальная затраченная память на хранение элементов очередей.

2.2. Описание задачи, реализуемой программой.

Программа позволяет смоделировать процесс обработки заявок, выводить информацию о процессе и задавать интервалы.

2.3. Способ обращения к программе.

Запуск программы:

./app.exe

В этом случае программа начнёт выполнять свою задачу. Ввод осуществляется вручную с клавиатуры.

2.4. Описание возможных аварийных ситуаций и ошибок пользователя.

2.4.1. Переполнение буфера. Возникает, если строка при чтении целого числа оказалась длиннее чем 1 символ.

2.4.2. Неожиданный символ. Возникает, если в строке присутствуют символы, не предусмотренные программой, или из-за неправильного формата входных данных (например, лишний пробел, лишняя точка в числе, пробел посреди числа и т. д.).

2.4.3. Число не входит в диапазон. Возникает, если пользователь ввёл число, которое не входит в диапазон допустимых значений.

2.4.4. Ошибка при выделении памяти. Практически не зависит от действий пользователя. Возникает только при неудачной попытке выделить память на куче, поэтому вероятность возникновения крайне мала.

2.4.5. Переполнение очереди. Возникает, если в процессе моделирования одна из очередей переполнилась.

2.4.6. Среднее значение выше допустимого. Возникает, если введенное среднее время обработки больше или равно среднему времени поступления в очередь.

3. Описание внутренних структур данных.

Для очередей используются 4 структуры:

```
typedef struct my_queue my_queue;

typedef int push_f(void*, QUEUE_TYPE);
typedef int pop_f(void*, QUEUE_TYPE*);
typedef void destroy_f(my_queue*);
typedef size_t size_f(void*);
typedef size_t memsize_f(void*);
typedef bool is_empty_f(void*);
typedef void clear_f(void*);

struct my_queue
{
    void *queue;
    push_f *push;
    pop_f *pop;
    destroy_f *destroy;
    size_f *size;
    memsize_f *memsize;
    is_empty_f *is_empty;
    clear_f *clear;
};

typedef struct _aqueue _aqueue;

struct _aqueue
{
    size_t size;
    size_t begin;
```

```
    size_t end;
    QUEUE_TYPE *arr;
};

typedef struct _lqueue _lqueue;

typedef struct _leaf _leaf;

struct _leaf
{
    _leaf *next;
    QUEUE_TYPE value;
};

struct _lqueue
{
    size_t size;
    _leaf *head;
    _leaf *tail;
};
```

my_queue – обертка для очереди. Поле queue хранит указатель на реализацию очереди (без информации о типе), остальные поля хранят указатели на функции обработки: вставка, удаление, деструктор, получение количества элементов, получение размера используемой памяти, проверка на пустоту и очищение очереди.

_aqueue – очередь на массиве.

Её поля:

size – целое число. Хранит размер очереди.

begin, end – целые числа. Хранят индексы начала и конца очереди в массиве.

arr – указатель на тип, который зависит от значения QUEUE_TYPE. Хранит элементы очереди.

_leaf – листья списка.

Её поля:

value – тип зависит от значения QUEUE_TYPE. Хранит значение.

next – указатель на _leaf. Хранит указатель на следующий элемент.

_lqueue – очередь на списке.

Её поля:

size – целое число. Хранит размер очереди.

head, tail – указатели на _leaf. Хранят начало и конец очереди.

4. Описание алгоритма.

Сначала создаются очереди, после чего программа заходит в цикл, из которого выходит либо после обслуживания первых 1000 заявок 1 типа, либо в случае ошибки. В цикле идёт проверка состояний стеков и ОА. Если пришло время добавления заявки в один из стеков, то в него добавляется заявка и генерируется время следующего прихода заявки (на первой итерации добавление не происходит). Если ОА завершил обработку, то в зависимости от пустоты

стеков либо начнёт обработку следующей заявки по вышеуказанным правилам, либо будет простаивать.

5. Сравнительный анализ разных способов хранения и перемножения.

Для замера времени и памяти программа запускала 1000 итераций, на каждой из которых моделировалась работа ОА для каждой из очередей.

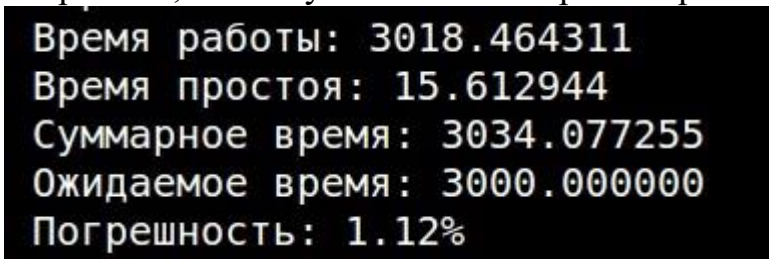
В таблицах ниже приведены результаты работы программы для разных входных диапазонов.

T1	T2	T3	T4	Время работы, мкс (массив)	Используемая память, байт (массив)	Время работы, мкс (список)	Используемая память, байт (список)
1..5	0..3	0..4	0..1	701	160064	841	2336
0..5	0..5	0..4	0..4	737	160064	933	36896
0..3	0..4	0..2	0..3	592	160064	737	24800

Время работы во всех случаях для списка оказалось примерно на 20-25% дольше, чем для массива. При этом для массива используемая память в несколько раз больше (даже в случае с равными диапазонами память для массива превысила память для очереди на списке более чем в 4 раза). Чтобы избежать переполнения, статический массив пришлось сделать достаточно большим, а также пришлось его закольцевать, чтобы не перевыделять память, когда начло очереди сдвинется ближе к концу массива. Можно сделать динамический массив, однако в данном случае придётся постоянно следить, чтобы при очередном перевыделении памяти индексы начала и конца очереди оставались корректными. Поэтому, на мой взгляд, несмотря на то, что список работает немного дольше, для данной задачи очередь на списке предпочтительнее, чем очередь на статическом массиве.

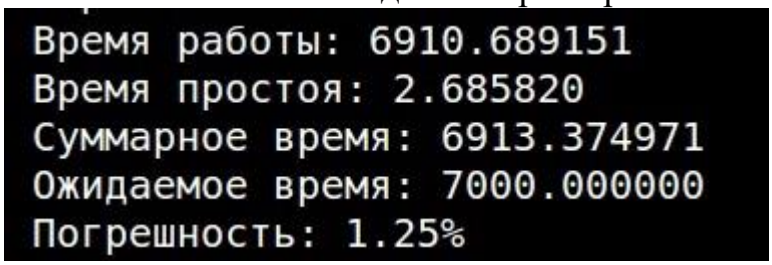
6. Расчёт времени моделирования.

Сначала рассмотрим систему с интервалами 1..5, 0..3, 0..4, 0..1. Для такого интервала теоретическое время работы системы будет равно <T1> * 1000, где <T1> – среднее значение интервала прихода заявки первого типа. Получится 3000. В данном интервале заявки второго типа генерируются быстрее, но при этом они очень быстро обрабатываются, из-за чего они почти никак не влияют на время работы, поэтому точность измерения времени высокая.

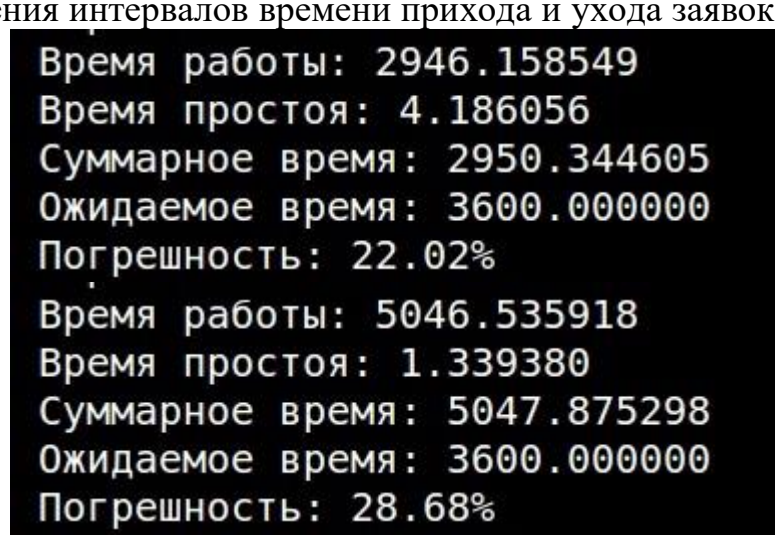


Также высокая точность измерений будет, если <T3> > <T1>, где <T3> - среднее время обработки заявки 1 типа(при условии, что заявки второго типа

создаются быстрее и ещё быстрее обрабатываются). Но здесь время работы будет вычисляться уже по формуле <T3> * 1000. Возьмём предыдущие интервалы, но поменяем <T3> на 4..10. Ожидаемое время работы получится 7000.



Для остальных случаев теоретический расчёт времени работы не может быть вычислен с подходящей точностью. Самый точный результат получается из рекуррентного соотношения, результат для которого не вычисляется за O(1), а из-за чередующегося приоритета очереди время работы может стать крайне непредсказуемым. Особенно это заметно для системы, в которой интервалы для разных типов заявок совпадают. Для примера приведу систему, в которой заявки всех типов приходят в интервале 0..5, а уходят в интервале 0..4. Вычислять время буду по формуле <T3> * (1 + <T2> / <T4>) * 1000, где <T2> и <T4> - средние значения интервалов времени прихода и ухода заявок 2 типа.



Оба случая были получены при одном запуске программы для очереди на массиве для вышеуказанного интервала. В первом случае время моделирования оказалось меньше рассчитанного, а во втором – больше. При этом погрешность в каждом случае больше 20%, что даёт понять, что точно рассчитать время моделирования невозможно.

7. Набор тестов.

Тесты описаны только для ввода значений. Отсутствие выходных данных означает в данном случае что тест позитивный.

В меню проверяется только то что после каждого действия запускается нужная функция.

Входные данные	Выходные данные	Что и где проверялось
0		Ввод минимального целого числа
3		Ввод целого числа не на границе
4		Ввод целого числа максимального размера
0 4		Ввод диапазона.
	Ошибка. Пустой ввод.	Ввод пустой строки при вводе числа
what	Ошибка. Встречен некорректный символ.	Ввод не числа, если нужно целое число
12	Ошибка. Переполнение буфера при вводе.	Переполнение буфера (при чтении целого числа)
1 -1	Ошибка. Начало интервала должно быть меньше конца интервала.	Начало интервала больше его конца
-1 1	Ошибка. Границы интервала не могут быть меньше 0.	Начало интервала меньше 0
0 5	Ошибка. Введённые значения превышают допустимые.	Среднее время интервала времени обработки больше или равно среднему времени интервала времени поступления заявки (интервал поступления заявки считать 0..5)

8. Выводы по проделанной работе.

В ходе выполнения работы я изучил операции с очередью и реализовал 2 очереди: на статическом массиве и на списке, а также реализовал моделирование работы обслуживающего аппарата с переменным приоритетом, для которого сравнил эффективность работы при использовании вышеуказанных очередей.

9. Ответы на вопросы.

9.1. Что такое FIFO и LIFO?

Принципы работы очереди (первый пришёл – первый ушёл) и стека (последний пришёл – первый ушёл).

9.2. Каким образом, и какой объём памяти выделяется под хранение очередей при различной её реализации?

Для статического массива – при создании, для динамического также при полном заполнении всей используемой под массив памяти, для списка – при добавлении каждого элемента.

9.3. Каким образом освобождается память при удалении элемента из очереди при различной её реализации?

Для массива – при уничтожении (для динамического массива можно также при сильном снижении количества используемой памяти, но из-за плавающего начала и конца очереди это будет проблематично), для списка – при удалении каждого элемента.

9.4. Что происходит с элементами очереди при её просмотре?

В общем случае смотреть можно только добавленный раньше остальных элемент и только когда его достают из очереди.

9.5. От чего зависит эффективность физической реализации очереди?

От поставленной задачи.

9.6. Каковы достоинства и недостатки различных реализаций очереди в зависимости от выполняемых на ней операций?

Статический массив эффективнее, когда нам заранее известен максимальный размер очереди.

Если размер очереди заранее неизвестен, то лучше использовать очередь на списке, однако он может легко создать фрагментацию памяти.

Динамический массив тоже эффективен при неизвестных размерах, однако под него сложнее выделить память, чем под список.

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

Это выделение маленьких кусков памяти в разных местах на куче, что в дальнейшем может помешать выделить память большего размера.

9.8. Для чего нужен алгоритм «близнецов»?

Для снижения фрагментации памяти.

9.9. Какие дисциплины выделения памяти вы знаете?

Первый подходящий, наиболее подходящий, алгоритм близнецов.

9.10. На что необходимо обратить внимание при тестировании программы?

Крайние случаи, случаи получения и обработки некорректных данных, переполнение очереди.

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

При выделении ищется подходящий блок, после чего он помечается занятым.

При освобождении этот блок добавляется в список свободных адресов.