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 «ОБРАБОТКА ОЧЕРЕДЕЙ»

Студент Дьяченко Артём Александрович

Группа ИУ7 – 33Б

Преподаватель Барышникова М. Ю.

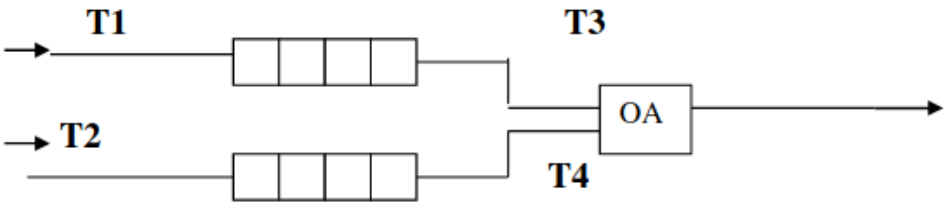
2022 г.

Оглавление

Описание условия задачи

Цель работы: отработка навыков работы с типом данных «очередь», представленным в виде одномерного массива и односвязного линейного списка. Сравнительный анализ реализации алгоритмов включения и исключения элементов из очереди при использовании двух указанных структур данных. Оценка эффективности программы (при различной реализации) по времени и по используемому объему 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. 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 немедленно поступает на обслуживание; обработка заявки 2-го типа прерывается и она возвращается в "хвост" своей очереди (система с абсолютным приоритетом и повторным обслуживанием).

Смоделировать процесс обслуживания первых 1000 заявок 1-го типа, выдавая после обслуживания каждых 100 заявок 1-го типа информацию о текущей и средней длине каждой очереди, а в конце процесса -общее время моделирования и количестве вошедших в

систему и вышедших из нее заявок обоих типов, среднем времени пребывания заявок в очереди, количестве «выброшенных» заявок второго типа. Обеспечить по требованию пользователя выдачу на экран адресов элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

Описание технического задания

Входные данные:

Числовое значение номера пункта меню, выбор интервала обработки и его изменение.

Выходные данные:

Моделирование и характеристика для очередей, результат сравнения двух очередей.

Обращение к программе:

Запускается через терминал командой: ./app.exe.

Аварийные ситуации:

1. Ввод некорректного пункта меню.
2. Выбор некорректного интервала времени.
3. Некорректный ввод границ интервала обработки.

НАБОР ТЕСТОВ

№	Название теста	Пользовательский ввод	Вывод
1	Некорректный пункт меню	10	Ошибка!!! Введена некорректная команда, попробуйте снова!!! Попробуйте ещё раз.
2	Некорректный пункт меню	abacaba	Ошибка ввода кода действия. Попробуйте ещё раз.
3	Некорректный выбор опции вывода информации о памяти	2 3	Ошибка!!! Некорректный номер введён!
4	Выбор некорректного интервала (не число)	3 a	Ошибка!!! Некорректный номер введён!
5	Ввод некорректных границ интервала	3 2 10 abac	Ошибка!!! Введён некорректный номер!
6	Выбор некорректного интервала (число не в промежутке 1...4)	3 5	Ошибка!!! Некорректный номер введён!
7	Корректный вызов пункта меню моделирования	1 или 2	Вывод временной характеристики про очередь в виде массива/списка
8	Выход из программы	0	Выход из программы

Описание структуры данных

Комбинированная структура очереди.

```
typedef struct queue
{
    char name[30];      // Название очереди
    void* low;          // Адрес нижней границы
    void* up;           // Адрес верхней границы
    void* p_in;         // Указатель на последний элемент
    void* p_out;        // Указатель на "первый на выход" элемент
    int count_len;      // Кол-во элементов
    size_t size;        // Размер типа данных
    int count_req;      // Кол-во запросов
    int sum_len;        // Суммарная длина очереди
    int tmp_len;        // Текущая длина очередь
    int sum_time;       // Общее время
    int out_req;        // Кол-во запросов на выход
    int in_req;         // Кол-во запросов на вход
} queue_r;
```

Структура узла очереди на списке.

```
typedef struct node
{
    char inf;           // Данные об элементе
    struct node *next;  // Указатель на следующий узел
} node_r;
```

Оценка эффективности

Реализация ОА на массиве

№	Кол-во заявок 1-го типа	Кол-во заявок 2-го типа	Время моделирования (условные е.в.)	Время моделирования (мкс)
1	1000	1959	2991	838
2	1000	2000	2977	703
3	1000	1972	3002	692
4	1000	1975	2984	732
5	1000	2018	3033	727
6	1000	1998	3013	710
7	1000	1947	2970	841
8	1001	2019	3026	823
9	1000	1966	2943	708
10	1001	1999	3015	985
Среднее	1000.2	1985.3	2995.4	775.9

Реализация ОА на списке

№	Кол-во заявок 1-го типа	Кол-во заявок 2-го типа	Время моделирования (условные е.в.)	Время моделирования (мкс)
1	1000	1985	3016	2294
2	1000	1987	3012	1827

3	1001	2062	3053	1205
4	1000	2000	3000	1559
5	1000	1980	2971	2432
6	1000	2008	3013	1600
7	1000	1992	3049	639
8	1000	2009	2967	2419
9	1001	1921	1921	2167
10	1000	2007	2991	2107
Среднее	1000.2	1995.1	2899.3	1824.9

Операции над очередью

	Массив	Список
Добавление	231	1491
Удаление	189	315

Память (байт)

Кол-во элементов	Массив	Список
10	10	90
100	100	900
1000	1000	9000
n	n	9 * n

Тестирование задания

Теоретический расчет времени моделирования очереди на массиве = *max*(среднее время прихода заявки 1-го типа, среднее время обработки заявки 1-го типа) * количество.
1-й тип – так как у него абсолютный приоритет.

Время моделирования заявок:
T1: 1...5
T2: 0...3
T3: 0...4
T4: 0...1

Число заявок 1 типа, вошедших: 1000, вышедших = 1000

Число заявок 2 типа, вошедших: 2000,
Вышедших = (время моделирования / max(приход, обработки)) = 3000 / max(1.5, 0.5) = 2000

Время моделировния: 3000

Практические результаты:

```
Общее время моделирования:       2964.903790
Погрешность работы ОА:   1.169874%

Среднее время обработки заявки 1 очереди:        3.000000
Среднее время обработки заявки 2 очереди:        1.500000
Число вошедших в 1 очередь:      1001
Число вышедших из 1 очереди:     1000
Число вошедших во 2 очередь:     1981
Число вышедших из 2 очереди:     1564
Число выброшенных заявок из 2 очереди: 1209
Время работы (мкс): 763

Погрешность ввода 1 очереди:     1.284905%
Погрешность ввода 2 очереди:     0.222476%
Время простоя ОА:        6.588884
```

Теоретический расчет времени моделирования очереди на списке = (среднее время обработки заявки 1-го типа) * количество.

Время моделирования заявок:
T1: 1...5
T2: 0...3
T3: 0...4
T4: 0...1

Число заявок 1 типа, вошедших: 1000, вышедших = 1000

Число заявок 2 типа, вошедших: 2000,
Вышедших = (время моделирования / max(приход, обработки)) = 3000 / max(1500, 500) = 2000

Время моделировния: 3000

Практические результаты:

```
Общее время моделирования:       2978.049355
Погрешность времени моделирования:      0.731688%

Число вошедших в 1 очередь:      1001
Число вышедших из 1 очереди:     1000
Число вошедших во 2 очередь:     2023
Число вышедших из 2 очереди:     1739
Число выброшенных заявок из 2 очереди: 577
Время работы (мкс): 1775
Среднее время обработки заявки 1 очереди:       3.000000
Среднее время обработки заявки 2 очереди:       1.500000
Погрешность ввода 1 очереди:     0.837818%
Погрешность ввода 2 очереди:     1.895558%
Время простоя ОА (в усл. ед. в.):       26.313083

Количество повторно используемых адресов: 0
Количество неиспользуемых адресов: 3316
```

Видно, что в случае реализации ОА на списке возникает фрагментация – «дырок» выходит больше кол-ва всех поступивших в очреедь элементов.

Описание алгоритма

1. После запуска программы пользователю предлагается ввести пункт меню.
2. Пользователь вводит вещественные или целые данные, в зависимости от пункта меню.
3. Программа продолжает работу до момента, пока пользователь на запрос ввода пункта меню не введёт 0, или пока не возникнет аварийная ситуация – в таком случае программа тоже завершится.

Ответы на контрольные вопросы

1. Что такое очередь?
Очередь – это последовательный список переменной длины, включение элементов в который идет с «хвоста», а исключение – с «головы». Принцип работы очереди: первым пришел – первым вышел, т.е. First In – First Out (FIFO).

2. Каким образом, и какой объем памяти выделяется под хранение очереди при различной ее реализации?
При реализации списком, под каждый новый элемент выделяется память размером sizeof(элемента) + 8 байт (для указателя) в куче, для каждого элемента отдельно. При реализации массивом, (кол-во элементов) * sizeof(элемента). Если массив статический, то память выделяется в стеке, если массив динамический, то - в куче.

3. Каким образом освобождается память при удалении элемента из очереди при её различной реализации?
При удалении элемента из очереди в виде массива, перемещается указатель, память не освобождается. Память освобождается в конце программы. Если массив статический, то после завершении программы, если динамический — с помощью функции free().
При удалении элемента из очереди в виде списка, освобождается память из данного элемента сразу. (Указатель на «голову» переходит на следующий элемент, считанный элемент удаляется, память освобождается)

4. Что происходит с элементами очереди при её просмотре?
При просмотре очереди, головной элемент («голова») удаляется, и указатель смещается. То есть при просмотре очереди ее элементы удаляются.

5. Каким образом эффективнее реализовывать очередь? От чего это зависит?
При реализации очереди в виде массива, может возникнуть переполнение памяти, фрагментации не возникает. Быстрее работают операции добавления и удаления элементов (при кольцевой реализации), хотя при обычной – удаление в массиве будет работать медленнее. Также необходимо знать тип данных.

При реализации в виде списка — легче удалять и добавлять элементы, переполнение памяти может возникнуть только если закончится оперативная память, однако может возникнуть фрагментация памяти.

Если изначально знать размер очереди и тип данных, то лучше воспользоваться массивом. Не зная размер — списком.

6. В каком случае лучше реализовать очередь посредством указателей, а в каком – массивом?
Если важна скорость выполнения, то лучше использовать массив, так как все операции с массивом выполняются быстрее, но очередь ограничена по памяти (так как массив статический).

Но если неизвестно сколько будет элементов в очереди — то лучше использовать список, так как он ограничен только оперативной памятью, но может возникнуть фрагментация памяти.

7. Каковы достоинства и недостатки различных реализаций очереди в зависимости от выполняемых над ней операций?
При реализации очереди в виде массива не возникает фрагментация памяти, так же может возникнуть переполнение очереди, и тратиться дополнительное время на сдвиги элементов (классический массив).

При реализации очереди в виде списка, проще выполнять операции добавления и удаления элементов, но может возникнуть фрагментация памяти.

8. Что такое фрагментация памяти?
Фрагментация – чередование участков памяти при последовательных запросах на выделение и освобождение памяти. «Занятые» участки чередуются со «свободными» - однако последние могут быть недостаточно большими для того, чтобы сохранить в них нужное данное.

9. На что необходимо обратить внимание при тестировании программы?
При реализации очереди в виде списка необходимо следить за освобождением памяти при удалении элемента из очереди. Если новые элементы приходят быстрее, чем уходят старые, то может возникнуть фрагментация памяти.

При реализации очереди в виде массива надо обратить внимания на корректную работу с ним, чтобы не произошло записи в невыделенную память.

10. Каким образом физически выделяется и освобождается память при динамических запросах?
Программа дает запрос ОС на выделение блока памяти необходимого размера. ОС находит подходящий блок, записывает его адрес и размер в таблицу адресов, а затем возвращает данный адрес в программу.

При запросе на освобождение указного блока программы, ОС убирает его из таблицы адресов, однако указатель на этот блок может остаться в программе. Обращение к этому адресу и попытка считать данные из этого блока может привести к неопределенному поведению, так как данные могут быть уже изменены.

Вывод

К недостаткам очереди в виде списка можно отнести то, что используется большее количество памяти, так как помимо самих

элементов необходимо хранить указатели. Также при работе в очередями-списками может возникнуть фрагментация памяти. К преимуществам можно отнести тот факт, что очередь-список позволяет воспользоваться памятью, ограниченной лишь объёмом оперативной памяти компьютера, а также операции удаления и добавления элемента в очередь легче реализовать, чем с очередью-массивом, но при выполнении этих операций выполняется выделение или освобождение памяти, что может перевести к ошибке.

К недостаткам очереди в виде массива можно отнести то, что такая очередь будет ограничена по памяти и может возникнуть переполнение. Преимущество очереди-массива над очередью-списком — операции удаления и добавления элемента выполняются намного быстрее. Следует учитывать, что в данной реализации использовался кольцевой массив. Если использовать обычный – операция удаления элемента будет проигрывать по скорости списку, т.к. придётся «двигать» все элементы этого массива.