Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____ИУ_____

КАФЕДРА _____ИУ7_____

Отчет

по лабораторной работе № 5

Дисциплина: Типы и Структуры Данных

Название лабораторной работы: Обработка очередей

Студент гр. **ИУ7-34Б Малышев Илья Николаевич**

(Подпись, дата) (И.О. Фамилия)

Преподаватель **Барышникова Марина Юрьевна**

(Подпись, дата) (И.О. Фамилия)

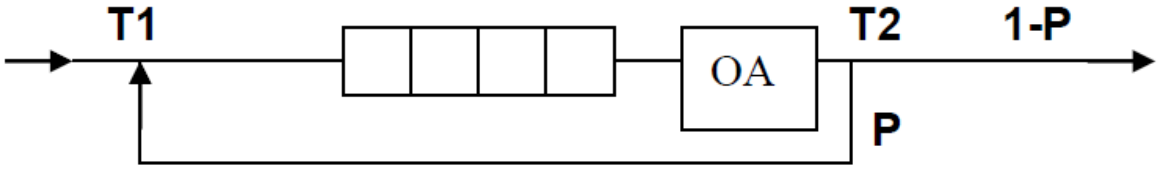
Москва, 2021

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.

Условия 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.

Техническое задание

Входные данные:

- Целое число, которое обозначает действие, которое должна выполнить программа (смотри “Задача программы”).
- Запрашиваемые данные, необходимые для выполнения некоторых действий;

Выходные данные:

- Промежуточное и финальное состояние системы моделирования;
- Содержание файла “statistics.txt”;
- Список освобождаемых во время моделирования адресов;
- Результаты замеров функций push и pop.

Задача программы:

Программа позволяет выполнить перечисленные ниже операции:

0. Выход;
1. [ОЧЕРЕДЬ-СПИСОК] Провести моделированиие;
2. [ОЧЕРЕДЬ-МАССИВ] Провести моделированиие;
3. Сгенерировать статистику по работе двух реализаций очереди;
4. Вывести готовую статистику из файла ('statistics.txt').

Способ обращения к программе:

Программа является консольной. После запуска исполняемого файла из консоли, программа выведет справку и приглашение на ввод.

Аварийные ситуации:

- Некорректный ввод номера команды;
- Отсутствие требуемых для работы программы текстовых файлов;

Структуры данных:

line_t - тип, представляющий статическую строку.

```
#define LINE_LEN 1024
typedef char line_t[LINE_LEN];
```

array_queue_t - структура, представляющая очередь, реализованную через массив.

```
typedef struct
{
  void **data;
  int len;
} array_queue_t;
```

queue_t – структура, представляющая очередь, реализованную через односвязный список.

```
typedef struct
{
  void *queue_head;
  int len;
} queue_t;
```

queue_note_t – структура, представляющая узел односвязного списка, через который реализовывалась очередь.

request_t - структура, представляющая «запрос», который обрабатывает ОА. Единственное поле, хранит информацию, о временной точке. Используется ситуативно.

```
typedef struct
{
  double time_point;
} request_t;
```

Тесты

#	Ошибка	Пример
1	**Неправильный ввод команды**	**q**
2	**Неправильный ввод команды**	**20**
3	**Неправильный ввод начальной длины**	**123123y**
4	**Отсутствие необходимых для работы программы системных файлов**	**statistics.txt**

Оценка эффективности

Измерения эффективности способов сложения будут производиться в единицах измерения – тактах процессора.

Очередь-список			
Начальное количество элементов	Время добавления – в тактах	Время извлечения – в тактах	Память, затраченная на хранение структуры (Без учета памяти на хранение данных) – в байтах
0	241	30	16
1	269	36	32
5	280	30	96
20	431	28	336
100	1122	29	1616
1000	9173	28	16016

Очередь-массив			
Начальное количество элементов	Время добавления – в тактах	Время извлечения – в тактах	Память, затраченная на хранение структуры (Без учета памяти на хранение данных) – в байтах
0	28	28	16
1	32	32	24
5	27	61	56
20	27	163	176
100	26	769	816
1000	29	8367	8016

Теоретические выкладки

- Время прихода: 0..6
- Время обработки: 0..1
- Вероятность выхода заявки из системы: 0.2

Результаты

- Время моделирования равно: 3000
- Время появления необходимого числа заявок: 1000 * 3
- Время простоя аппарата: 3000 – 500 * 5
- Число заявок: 1000
- Количество срабатываний: 1000 * 5

```
Общее время моделирования :                  3018.403
Количество вошедших в систему заявок :       1002
Количество вышедших из системы заявок :      1000
Среднее время пребывания заявки в очереди : 4.537
Время простоя аппарата :                     542.733
Количество срабатываний ОА :                 4904
```

Контрольные вопросы

1. **Что такое очередь?**

Очередь - структура данных, для которой выполняется правило FIFO, то есть первым зашёл - первым вышел. Вход находится с одной стороны очереди, выход - с другой.

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

При хранении кольцевым массивом: кол-во элементов * размер одного элемента очереди. Память выделяется на стеке при компиляции, если массив статический. Либо память выделяется в куче, если массив динамический. При хранении списком: кол-во элементов * (размер одного элемента очереди + указатель на следующий элемент). Память выделяется в куче для каждого элемента отдельно.

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

При хранении кольцевым массивом память не освобождается, а просто меняется указатель на конец очереди. При хранении списком, память под удаляемый кусок освобождается.

4. **Что происходит с элементами очереди при ее просмотре?**

Эти элементы удаляются из очереди.

5. **Каким образом эффективнее реализовывать очередь? От чего это зависит?**

Зная максимальный размер очереди, лучше всего использовать кольцевой статический массив. Не зная максимальный размер, стоит использовать связанный список, так как такую очередь можно будет переполнить только если закончится оперативная память.

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

При использовании линейного списка тратится больше времени на обработку операций с очередью, а так же может возникнуть фрагментация памяти. При реализации статическим кольцевым массивом, очередь всегда ограничена по размеру, но операции выполняются быстрее, нежели на списке.

7. **Что такое фрагментация памяти?**

Фрагментация памяти - разбиение памяти на куски, которые лежат не рядом друг с другом. Можно сказать, что это чередование свободных и занятых кусков памяти.

8. **На что необходимо обратить внимание при тестировании программы?**

При тестировании программы необходимо обратить внимание на эффективное выделение и корректное освобождение динамической памяти. Помимо этого стоит обратить внимание на корректность реализации кольцевого массива, чтобы не произошло записи в невыделенную область памяти. Еще стоит обратить внимание на возникновение фрагментации памяти.

9. **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 пользователями памяти.

Вывод

Из таблиц оценки эффективности видно, что память, необходимая для хранения одного «узла» очереди, организованной на односвязном списке, **в два раза превышает** память, необходимую для хранения одного узла «узла» очереди, организованной на массиве.

Из таблиц оценки эффективности видно, что у каждого из подходов есть своя «сильная сторона»: команда ***push*** для очереди-массива выполняется за фиксированной время O(const), в то время как команда ***pop*** обладает теми же свойствами, только по отношению к очереди-списку.

Из таблиц оценки эффективности видно, что зависимость времени выполнения «неэффективных» команд у двух подходов к реализации очереди от количества элементов в очереди фактически подчиняется линейному закону, однако в любой момент времени отклоняется от идеального значения. Это свидетельствует о наличии фрагментации.

Реализация очереди через массив «выигрывает» очередь-список по времени выполнения и по необходимой памяти, однако, как и в прошлой работе отличие составляет не более одного порядка. Это говорит о том, что практически разница неощутима. Таким образом, ввиду явления «фрагментация», правильнее использовать очередь-список.