Министерство образования и науки 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

По дисциплине «Типы и структуре данных»

Название **«Обработка очередей»**

Студент Дубов Андрей Игоревич
фамилия, имя, отчество
Группа ИУ7-33Б

Вариант 5

Тип лабораторной работы Учебная

Студент _____ Дубов А. И.
подпись, дата *фамилия, и.о.*

Преподаватель Рыбкин Ю. А.
_____ Силантьева А. В.
подпись, дата *фамилия, и.о.*

2022 г.

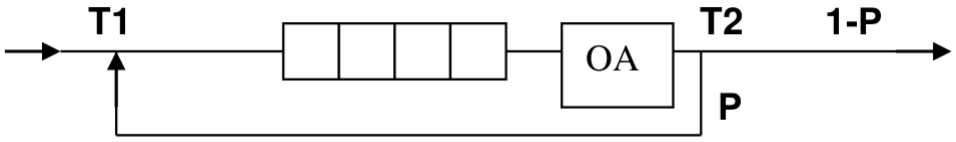
ОГлавление

Условие задачи

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.

Описание технического задания

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.

Входные данные:

1. Команда для вызова функции
2. Данные запрашиваемые программой

Выходные данные:

1. Уведомление о выполнение команды
2. Результаты моделирования

Аварийные ситуации:

1. Некорректный ввод номера команды.
2. Некорректный ввод времени обработки.

Описание структуры данных

Структура очереди в виде массива

```
typedef struct queue
{
    int items[SIZE];
    int rear, front;
} queue_t;
```

items – массив элементов
rear – указатель на новый элемент
front – указатель на удаляемый элемент

Структура узла очереди

```
typedef struct node
{
    int data;
    struct node *next;
} node_t;
```

data – элемент
next – указатель на следующий элемент
front – указатель на удаляемый элемент

Структура очереди в виде списка

```
typedef struct node_q
{
    node_t *rear, *front;
} node_q;
```

rear – указатель на новый элемент
front – указатель на удаляемый элемент

Описание алгоритма

1. Пользователю предлагается режим очереди-массива
2. Пользователь моделирует либо управляет выделенной очередью
3. Пока пользователь не введет 0 (выход из программы), ему будет предложено вводить номера команд и выполнять действия по выбору.

Набор тестов

	Название теста	Пользователь вводит	Вывод
1	Некорректный ввод команды	45	No such option or wrong input
2	Пустой ввод	Пустой ввод.	No such option or wrong input

3	Команда 1 1	1	Вывод временной характеристики про очередь в виде массива
4	Команда 2 1	2	Вывод временной характеристики про очередь в виде односвязного списка
5	(неверный ввод границл)	D 4 3 в 0 0	Incorrect input
6	Команда 1\2 2	Число	Добавлен элемент
7	Команда 1\2 3		Удалён элемент
8	Команда 1\2 4		Вывод очереди
9	Команда 0	0	Выход из программы

Оценка эффективности

На массиве

	Число заявок	***Время моделирования (ус.е.в.)***
1	1049	3148.270264
2	1024	3074.983398
3	1004	3014.630859
Сред.	1025.6	3 079.29

На списке

	Число заявок	***Время моделирования (ус.е.в.)***
1	984	2953.099854
2	997	2993.080322
3	996	2988.004639
Сред.	992.3	2 978.06

Погрешности для массива

Ввод 0 6 0 1

```
OA workes 5281 times
Modeling time: 3157.599121
OA downtime 500.393372
Count of in elemnts: 1052
Count of out elements: 1000
```

время моделирования определяется временем входа
рассчетное время моделирования: (6 – 0) / 2 * 1052 = 3000
погрешность: 0.04%

Ввод 0 1 0 6

```
OA workes 4881 times
Modeling time: 14556.975586
```

```
OA downtime 0.000000
Count of in elemnts: 29113
Count of out elements: 1000
```

время моделирования определяется временем обслуживания
рассчетное время моделирования: (1 – 0) / 2 * 29113 = 14556.5
погрешность: 0.003%

Ввод 0 2 0 2

```
OA workes 4876 times
Modeling time: 2435.348389
OA downtime 0.000000
Count of in elemnts: 2435
Count of out elements: 1000
```

время моделирования определяется временем входа
рассчетное время моделирования: (2 – 0) / 2 * 2435 = 2435
погрешность: 0.014%

Вывод

К недостаткам очереди в виде списка можно отнести тот факт, 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 привести 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.

Ответы на контрольные вопросы

1. Что такое FIFO и LIFO?

Для очереди выполняется правило FIFO, то есть первым зашёл - первым вышел. Вход находится с одной стороны очереди, выход - с другой. На стек действует правило LIFO — последним пришел, первым вышел.

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(element) + 8 байт (для указателя) в куче, для каждого элемента отдельно.

При реализации массивом (кольцевым), (кол-во элементов) * sizeof(элемента).

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

При удалении элемента из очереди в виде массива, перемещается указатель, память не освобождается. Память освобождается в конце программы. Если массив статически, 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 ее просмотре?

При просмотре очереди, головной элемент («голова») удаляется, и указатель смещается. То есть при просмотре очереди ее элементы удаляются.

5. От чего зависит эффективность физической реализации очереди?

При реализации очереди в виде массива (кольцевого статического), может возникнуть переполнение памяти, фрагментации не возникает. Быстрее работают операции добавления и удаления элементов. 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.

Также способ реализации зависит от того, в чем мы больше ограниченны, в памяти или во времени.

6. 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). Сдвигов нет, если использовать кольцевой статический массив, но усложняется реализация алгоритмов добавления и удаления элементов.

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

7. 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.

8. Для чего нужен алгоритм «близнецов»?

Метод близнецов (или buddy system) является схемой выделения памяти, сочетающей в себе возможность слияния буферов и распределитель по степени числа 2. В основе метода лежит создание буферов малого размера путем деления пополам больших буферов и слияния смежных буферов по мере возможности. При разделении буфера на два каждая половина называется близнецом (buddy) второй.

Использование алгоритма «близнецов» для ускорение работы программы, то есть увеличение эффективности, при этом он уменьшает вероятность фрагментации памяти и улучшает поиск блок памяти для выделения.

9. Какие дисциплины выделения памяти вы знаете?

- Первый подходящий:
  Выбирается первая найденная подходящая память для выделения под данный элемент. При этом является более эффективным по поиску выделение памяти.

- Самый подходящий:
  Выбирается из всей возможной памяти самая «лучшая» подходящая память для выделения, но при этом вероятность нахождения такой памяти мала, поэтому бывают случай когда выделяется память, так что образуются пустые места в памяти, то есть выделение памяти, больше чем ожидалось.

10. 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 (кольцевого) 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.