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»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

«ОБРАБОТКА ОЧЕРЕДЕЙ»

по курсу «Типы и структуры данных»

Студент: Чепиго Дарья Станиславовна

Группа: ИУ7-34Б

Студент _____ Чепиго Д.С

подпись, дата *фамилия, и.о.*

Преподаватель _____ Барышникова М.Ю.

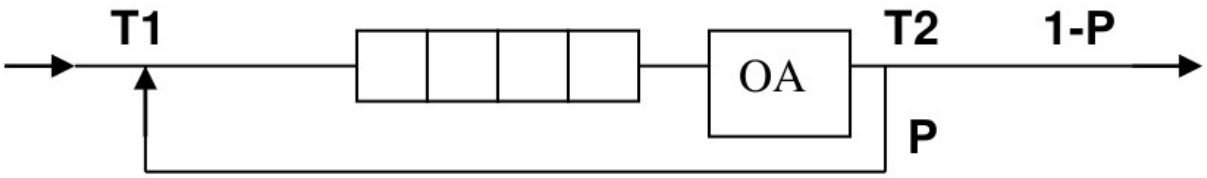
подпись, дата *фамилия, и.о.*

Оценка _____

Условие задачи

Вариант 5

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, Обеспечить по требованию пользователя выдачу на экран адресов элементов очереди при удалении и добавлении.

Техническое задание

Входные данные

- Целое число от 0 до 4 – пункт меню

Пункты меню:

1 - моделирование очереди из 1000 элементов в виде массива

2 - моделирование очереди из 1000 элементов в виде списка

3 - вывод таблицы когда-либо использованных адресов

4 - вывод сравнения времени и памяти для 1 и 2 пункта

0 - выход из программы

Выходные данные:

При выборе пункта меню, связанным с моделированием очереди выводится информация о промежуточных результатах обработки очереди, когда количество обработанных запросов кратно 100. Если во время обработки возникла ошибка переполнение, то на экран выводится сообщение о переполнении и количество заявок, которые успели обработаться. Иначе выводится подробная информация о результате моделирования очереди:

-Ожидаемое время моделирования

-Полученное время моделирования

-Погрешность

-Количество вошедших заявок

-Количество вышедших заявок

-Среднее время в очереди

-Время простоя аппарата

-Количество срабатывания аппарата

При выборе пункта меню 3 на экран выводятся последние n элементов списка, которые когда-то были использованы.

При выборе пункта меню 4 также выводится затраченное время и память для двух способов моделирования.

Действие программы:

Моделирование обработки очередей, которые реализованы с помощью списка и с помощью массива.

Обращение к программе:

Запускается командой ./app.exe через терминал, находясь в директории, содержащей программу. Для сборки программы существует makefile.

Описание алгоритма

1. Пользователь выбирает пункт меню

2. В зависимости от выбора пользователь может видеть информацию о моделировании очереди списком или массивом, а также сравнение этих методов.

3. Пользователь может увидеть массив когда-то занятых адресов, а также их текущее состояние.

4. Если пользователь захочет завершить программу, то для этого есть отдельный пункт меню.

Аварийные ситуации:

- Выбор несуществующего пункта меню

- Переполнение очереди, при реализации её массивом

- Просмотр массива адресов до выполнения реализации списком

Описание структур данных

Структура для реализации очереди списком:

```
struct queue_slot

{
	double arrival_time; //время прихода в очередь
	struct queue_slot *next; //указатель на след элемент
};
```

Структура для реализации очереди:
(переменная int max используется только в списке)

```
struct queue
{
	struct queue_slot *pin; //указатель на начало очереди
	struct queue_slot *pout; //указатель на конец очереди
	int len; //длина очереди
	int in_num; //число вошедших в очередь заявок
	int state; //переменная для вычисления средней длины
	int max; // переменная для подсчета очереди в списке
	double total_stay_time; //время нахождения заявок в очереди
};
```

Структура для реализации обслуживающего аппарата:

```
struct machine
{
	double time; //текущее время состояния аппарата
	double downtime; // время простоя аппарата
	int triggering; // количество срабатывания аппарата
	int processed_count; //кол-во обработанных из очереди заявок
};
```

Структура для реализации массива адресов:

```
struct mem_slot
{
	struct queue_slot *queue_slot; //указатель на участок памяти
	int busy; // состояние участка 1(занят) или 0
	struct mem_slot *next; //указатель на след элемент очереди
};
```

Теоретический расчёт

Время моделирования = max(среднее время прихода заявок, среднее время обработки заявок) * (количество)

Ожидаемое время обработки = (среднее время обработки заявки) * (количество) * 1 / (P -1)

Время простоя аппарата = (время моделирования) - (ожидаемое время обработки)

Количество срабатывания = 1 / (P -1) * (количество заявок)

При стандартных временных границах:
Время поступления : 0 до 6 единиц времени
Время обслуживания: 0 до 1 единиц времени
Вероятность возвращения в «хвост»: P = 0.8

Время моделирования: 3000 единиц времени
Ожидаемое время обработки: 2500 единиц времени
Количество вошедших заявок: 1000
Количество вышедших заявок: 1000
Время простоя аппарата: 500
Количество срабатываний ОА: 5000

```
Ожидаемое время моделирования: 3000.00
Полученное время моделирования: 3080.85
Погрешность: ~2.70%

Количество вошедших заявок: 1001
Количество вышедших заявок: 1000
Среднее время в очереди: 9.66
Время простоя аппарата: 559.67
Количество срабатывания аппарата: 5008
```

Таким образом погрешность в данном примере составила 2.70%

Сравнение эффективности

Время — в тактах, Память — в байтах

Для времени прихода от 0 до 6:

Количество вышедших из ОА заявок	Время для списка	Время для массива	Память для списка	Память для массива
100	520094	101887	160	800
500	2662338	812468	208	4000
1000	10704175	1980433	368	8000
2000	16773293	5198432	384	16000

Для времени прихода от 0 до 4:

Количество вышедших из ОА заявок	Время для списка	Время для массива	Память для списка	Память для массива
100	623520	130418	464	800
500	4746142	895874	2016	4000
1000	8002811	1801145	2864	8000
2000	28570601	7954010	8224	16000

Выводы из таблицы измерений:

Исходя из данных двух таблиц можно легко понять, что список проигрывает по скорости массиву примерно в 2-25 раза. Но список выигрывает по памяти, причем зависимость далеко не линейная, это видно из первой таблицы, так как на 100 элемента выигрыш составляет: 800/160=5 раз, а на 2000 : 16000/384 = 41. Таким образом можно сделать вывод, что массив использовать выгоднее, когда нам важна скорость, а список, когда нам важна

память. Но не стоит забывать, что это касается реализации на динамическом массиве. В отличии от статического массива, в динамическом массиве нам может понадобится перевыделять память,и на переписывание значений уйдет много времени. Из этого можно сделать вывод, что в случае когда очередь не предполагает больших размеров, целесообразнее использовать массив, но когда идет речь об обработке огромного количества заявок, которые могут привести к переполнению, в целях экономии времени стоит использовать односвязный список.

Тестирование

Позитивные тесты.

Входные данные	Действия программы	Выходные данные
Пункт 1 Моделирование *очереди из 1000 элементов в виде массива*	Корректная работа программы	Ожидание следующего ключа
Пункт 2 Моделирование *очереди из 1000 элементов в виде списка*	Корректная работа программы	Ожидание следующего ключа
Пункт 3 до пункта 2	Корректная работа программы Просьба выполнить пункт 2	Ожидание следующего ключа
Пункт 3 после пункта 2	Корректная работа программы Вывод массива адресов	Ожидание следующего ключа

Негативные тесты

Входные данные	Действия программы	Выходные данные
Ввод числа 10	Информация о неверном ключе меню	Завершение программы
Пункт 1 при большом времени обработки	Информация о переполнении массива	Ожидание нового ключа
Пункт 1 при большом времени ожидания	Информация о переполнении массива	Ожидание нового ключа

Контрольные вопросы

1. Что такое FIFO и LIFO?

Способы организации данных. Первым пришел — первым вышел, т. е. First In – First Out (FIFO) – так реализуются очереди. Первым пришел — последним вышел, т. е. Last In – First Out (LIFO)- так реализуются линейные односвязные списки.

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

При реализации массивом единожды выделяется память для хранения только самих элементов. При реализации в виде списка для каждого элемента дополнительно выделяется память для хранения адреса следующего элемента.

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

При реализации очереди списком указателю Pout присваивается значение следующего элемента списка, а память из-под удаляемого элемента освобождается.

При реализации очереди массивом память из-под удаляемого элемента не освобождается, так как изначально она была выделена под весь массив. Освобождение будет происходить аналогично – единожды для всего массива.

4. Что происходит с элементами очереди при ее просмотре?

При просмотре очереди элементы удаляются и происходит очистка очереди, влекущая за собой освобождение памяти в случае реализации ее списком

5. От чего зависит эффективность физической реализации очереди?

Выяснилось, что эффективнее и по памяти, и по времени реализовывать очередь массивом. Это зависит от самой структуры элемента в случае памяти, и от запросов в оперативную память в случае времени.

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

В случае массива недостатком является фиксированный размер очереди.

В случае односвязного списка недостатками являются затраты по скорости и фрагментация.

7. Что такое фрагментация памяти?

Фрагментация — возникновение участков памяти, которые не могут быть использованы. Фрагментация может быть внутренней — при выделении памяти блоками остается не задействованная часть, может быть внешней — свободный блок, слишком малый для удовлетворения запроса

8. Для чего нужен алгоритм «близнецов»?

Идея этого алгоритма состоит в том, что организуются списки свободных блоков отдельно для каждого размера 2^k, 0 <= k <= m. Вся область памяти кучи состоит из 2^m элементов, которые, можно считать, имеют адреса с 0 по $2^m - 1$. Первоначально свободным является весь блок из 2^m элементов. Далее, когда требуется блок из 2^k элементов, а свободных блоков такого размера нет, расщепляется на две равные части блок большего размера; в результате появится блок размера 2^k (т.е. все блоки имеют длину, кратную 2). Когда один блок расщепляется на два (каждый из которых равен половине первоначального), эти два блока называются ***близнецами***. Позднее, когда оба близнеца освобождаются, они опять объединяются в один блок.

9. Какие дисциплины выделения памяти вы знаете?

Две основные дисциплины сводятся к принципам "самый подходящий" и "первый подходящий". По дисциплине "самый подходящий" выделяется тот свободный участок, размер которого равен запрошенному или превышает его на минимальную величину. По дисциплине "первый подходящий" выделяется первый же найденный свободный участок, размер которого не меньше запрошенного.

10. На что необходимо обратить внимание при тестировании программы?

При тестировании программы необходимо обратить внимание на корректность выводимых данных, проследить за выделением и освобождением выделяемой динамически памяти, предотвратить возможные аварийные ситуации.

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

В оперативную память поступает запрос, содержащий необходимый размер выделяемой памяти. Выше нижней границы свободной кучи осуществляется поиск блока памяти подходящего размера. В случае если такой найден, в вызываемую функцию возвращается указатель на эту область и внутри кучи она помечается как занятая. Если же найдена область, большая необходимого размера, то блок делится на две части, указатель на одну возвращается в вызываемую функцию и помечается как занятый, указатель на другую остается в списке свободных областей. В случае если области памяти необходимого размера не было найдено, в функцию возвращается NULL. При освобождении памяти происходит обратный процесс. Указатель на освобождаемую область поступает в оперативную память, если это возможно объединяется с соседними свободными блоками, и помечается свободными.