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 Егорова Полина Александровна

Группа ИУ7 – 34Б

Преподаватель Барышникова Марина Юрьевна

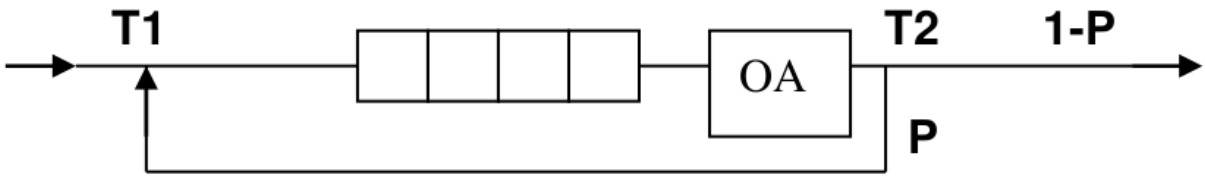
2021 г.

Описание условия задачи

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.

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.

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.

Входные данные:

Пункт меню (целое число от 0 до 4)

```
Меню:
 1 - моделирование очереди-массива из 1000 элементов
 2 - моделирование очереди-списка из 1000 элементов
 3 - вывод таблицы использованных адресов
 4 - вывод сравнения времени и памяти очереди-массива и очереди-списка
 0 - выход из программы
```

Выходные данные (в зависимости от выбранной команды):

1. Состояние очереди (списка/массива) и информация о моделировании:
    - Ожидаемое время моделирования
    - Полученное время моделирования
    - Погрешность
    - Количество вошедших заявок
    - Количество вышедших заявок
    - Среднее время в очереди
    - Время простоя аппарата
    - Количество срабатывания аппарата
2. Список использованных адресов.
3. Сравнительная характеристики времени работы ОА при реализации очереди в виде массива и списка и занимаемой ими памяти.

Обращение к программе:

Запуск через терминал (./app.exe)

Аварийные ситуации:

1. Некорректный ввод номера команды.

    На вход: число, большее чем 4 или меньшее, чем 0.

    На выход: *«Неверная команда.»*

2. Превышение предельного количества проходящих через ОА заявок.

   На вход: вероятность, равная 1

   На выход: *«Ни одна из 1000 заявок не вышла из очереди.»*

Описание структуры данных:

Структура для реализации очереди:

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

Структура для реализации очереди в виде списка:

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

Структура для реализации ОА:

```
// обслуживающий аппарат
struct oa
{
    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:

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

Константы:

```
#define TOTAL_NEED 1000 // суммарное число обработанных заявок
#define INTER_NEED 100 // число заявок, после которого нужно предоставить информацию о состоянии ОА

#define COMING_START 0 // мин время прихода
#define COMING_END 6 // макс время прихода
#define PROCESSING_START 0 // мин время обработки
#define PROCESSING_END 1 // макс время обработки

#define CHANCE 0.8 // вероятность
```

Теоретические расчеты:

Ожидаемое время моделирования (время прихода больше времени обработки) = (среднее время прихода заявки) * (количество) / 2

Время простоя аппарата = время моделирования – время прихода

Количество срабатываний = 1 / (1 - вероятность) * (количество заявок)

Пример работы:

Время поступления: 0 до 6 е.в.
Время обслуживания: 0 до 1 е.в.
Вероятность возвращения в «хвост»: 0.8

Теоретический результат моделирования:

Ожидаемое время моделирования: (6 – 0) * 1000 / 2 = 3000
Количество срабатываний: (1 / (1 - 0.8)) * 1000 = 5000
Время простоя ОА: (1 – 0) * 1000 / 2 = 500

Фактический результат моделирования:

```
Ожидаемое время моделирования: 3000.00
Полученное время моделирования: 3000.61
Погрешность: 0.02%

Количество вошедших заявок: 1006
Количество вышедших заявок: 1000
Среднее время в очереди: 8.04
Время простоя аппарата: 455.82
Количество срабатывания аппарата: 5187
```

Тесты:

Позитивные тесты

Входные данные	Действия программы	Выходные данные
Пункт 1 Моделирование очереди из 1000 элементов в виде массива	Вывод результата моделирования	Вывод меню, ожидание следующей команды
Пункт 2 Моделирование очереди из 1000 элементов в виде списка	Вывод результата моделирования	Вывод меню, ожидание следующей команды
Пункт 3 (до пункта 2)	Требование выполнить пункт 2	Вывод меню, ожидание следующей команды
Пункт 3 (после пункта 2)	Вывод массива первых 100 использованных адресов	Ожидание следующего ключа

Негативные тесты

Входные данные	Действия программы	Выходные данные
Пункт 1 При большом времени обработки или вероятности выхода из очереди == 0	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.

Сравнение эффективности:

Была произведена оценка эффективности работы модели (по ТЗ) при различном количестве вышедших из ОА заявок при времени прихода от 0 до 6:

Количество вышедших из ОА заявок	Время для списка (в тиках)	Время для массива (в тиках)	Память для списка (в байтах)	Память для массива (в байтах)
100	322392	204680	210	800
500	1194656	1008039	338	4000
1000	2182800	1593916	487	8000
2000	5227588	3062600	562	16000

Вывод:

Список выигрывает по памяти у массива от 4 до 25 раз, но проигрывает во времени примерно в 1,5 раза. Следовательно, целесообразно использовать массив, если важна скорость выполнения операции, и список, если важна затраченная на это память. В случае, когда очередь не предполагает больших размеров, целесообразнее использовать массив, но, когда идет речь об обработке большого количества заявок, которые могут привести к переполнению, в целях экономии времени стоит использовать односвязный список, поскольку объем в этом случае ограничен только объемом оперативной памяти. Также путем тестирования была выявлена фрагментация памяти.