Лабораторная работа №5. "Работа с очередью"

Студент Ивахненко Дмитрий - ИУ7-36Б

Описание условия задачи

Система массового обслуживания состоит из:

- обслуживающего аппарата (ОА)
- 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 длине очереди
- средней длине очереди.

В конце процесса выдать:

- общее время моделирования
- кол-во 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:

1. Номер команды - целое число в диапазоне от **`0`** **до** **`11`** включительно.
2. **Командно-зависимые данные:**
    - Изменить интервал времени прихода / интервал времени обработки
        - левая и правая границы интервала (два действительных неотрицательных числа через пробел таких, что второе не меньше первого)
    - Изменить вероятность **(P)** ухода заявки на еще один круг
        - новое значение вероятности (действительное число на интервале `[0.0; 1.0]`)
    - Изменить необходимое кол-во закрытых заявок для прекращения работы

- новое значения количества (натуральное число)

Выходные данные:

В зависимости от выбранного действия результатом работы программы могут являться:

1. Результат моделирования
    - общее время моделирования
    - кол-во вошедших в систему и вышедших из нее заявок,
    - среднее время пребывания заявки в очереди
    - время простоя аппарата,
    - количество срабатываний ОА.
    - информация о длине очереди через каждые 100 обработанных заявок.
2. Статистика по времени выполнения и объему занимаемой памяти при обработке очередей, реализованных односвязным списком и динамическим массивом.
3. Информация о текущих значениях основных констант программы
4. Адреса высвобожденных элементов списка и только что добавленных - печатаются по запросу пользователя в файл `adresses_statistic`.

Команды программы

```
1. Смоделировать очередь на массиве
2. Смоделировать очередь на списке
3. Изменить интервал времени прихода
4. Изменить инетрвал времени обработки
5. Изменить вероятность ухода заявки на второй круг
6. Изменить необходимое кол-во закрытых заявок
7. Сбросить значения констант до стандартных
8. Просмотреть значения констант
9. Провести анализ
10. Включить (выключить) печать адресов
11. Включить (выключить) вывод погрешности
```

Обращение к программе:

Запускается из терминала при помощи комманды `./bin/app`.

Аварийные ситуации:

1. Некорректный ввод номера команды.

   На входе: число, большее, чем максмальный индекс команды или меньшее, чем минимальный.

   На выходе: `[Ошибка] ::Некорректный ввод пункта меню::`

2. Некорректный ввод интервала

   На входе: ввод, отличный от указанного в ТЗ

На выходе: `[Ошибка] ::Некорректный ввод::`

3. Некорректный ввод вероятности

   На входе: ввод, отличный от указанного в ТЗ

   На выходе: `[Ошибка] ::Некорректный ввод::`

4. Некорректный ввод необходимого числа заявок, вышедших из системы

   На входе: ввод, отлчный от указанного в ТЗ

   На выходе: `[Ошибка] ::Некорректный ввод::`

5. Переполнение очереди, построенной на основе массива

   На входе: ситуация, при которой максимальная длина очереди во время моделирования превысит размер массива

   На выходе: `[Ошибка] ::Очередь переполнена::`

Структуры данных

Элементом очереди **`qtype_t`** является заявка, представляющая собой собой структуру **`Request`**

```
typedef struct {
    double handlT;      //  время, которое будет потрачено на обработку
    double deliverT;    //  время, в которое заявка стала в очередь
} Request;

typedef Request qtype_t;
```

Очередь, построенная на основе массива представляет собой структуру **`ArrQueue`**

```
typedef Request aqtype_t;
typedef struct {
    aqtype_t *buf;      // буффер с элементами очереди (динамический
массив)
    size_t capacity;    // размер буффера (максимальный размер очереди)
    size_t length;      // текущая длина очереди
    size_t head, tail;  // индексы "головы" и "хвоста" очереди
} ArrQueue;
```

Очередь, построенная на основе списка представляет собой структуру **`ListQueue`**

```
typedef Request lqtype_t;

/* Узел списка*/
typedef struct Node Node;
```

```
struct Node {
    lqtype_t data;      // элемент очереди (заявка)
    Node *next;         // указатель на следующий узел списка
};

/* Список */
typedef struct {
    Node *head, *tail;  // указатели на "голову" и "хвост" очереди
    size_t length;      // текущая длина очереди
} ListQueue;
```

Описание основных функций

```
/* Создание и удаление очереди над массивом */
ArrQueue *arrQueueCreate(const size_t capacity);
void arrQueueReset(ArrQueue *queue);

/* Постановка элемента в очередь над массивом и удаление из нее */
bool enArrQueue(ArrQueue *queue, const aqtype_t *item);
bool deArrQueue(ArrQueue *queue, aqtype_t *item);

/* Проверка очереди над массивом на пустоту и заполненность */
bool arrQueueIsFull(const ArrQueue *queue);
bool arrQueueIsEmpty(const ArrQueue *queue);

/* Создание и удаление очереди над списком */
ListQueue *listQueueCreate();
void listQueueDelete(ListQueue *queue);

/* Постановка элемента в очередь над листом и удаление из нее */
bool enListQueue(ListQueue *queue, const lqtype_t *item);
bool deListQueue(ListQueue *queue, lqtype_t *item);

/* Проверка очереди над листом на пустоту */
bool listQueueIsEmpty(const ListQueue *queue);

/* Генерация новой заявки */
void requestGenerate(Request *request, double currTime);
```

Алгоритм

1. На экран пользователю выводится меню
2. Пользователь вводит номер команды
3. Выполняется действие согласно номеру команды

Моделирование очереди

0. В начальный момент очередь пустая, а время моделирования, время простоя и кол-во заявок, вышедших из системы, равны нулю.

1. Проверяется наличие заявок в очереди:

    - Если **очередь пустая**, значит ОА простаивает, поэтому увеличиваем суммарное время ожидания на интервал между постановкой ближайшей заявки в очередь и текущим временем. Текущее время моделирования становится равно времени прихода данной заявки.

    - Если **в очереди есть заявки**, то достаем первую заявку. Все заявки, которые придут во время обработки данной заявки, становятся в очередь. После окончания обработки текущее время моделирования увеличивается на время обработки заявки.

2. Определеям дальнейшую судьбу только что обработанной заявки:

    - Если заявка **уходит** из системы, то увеличиваем счетчик кол-ва вышедших из системы заявок.

    - Если заявка **не уходит**, то она становится в конец очереди.

3. Если из системы вышло менее 1000 заявок, переходим к пункту 1.

Генерация заявки:

Генерация заявки происходит на основе текущего времени моделирования `t`

1. Генерируется относительное время прихода заявки `delivT`, принадлежащее интервалу `[t; t + 2.0]`

2. Генерируется время, которое будет затрачено на обработку заявки `handlT`, приналлежащее интервалу `[0.0; 0.5]`

Тесты

Меню

```
1. Смоделировать очередь на массиве
2. Смоделировать очередь на списке
3. Изменить интервал времени прихода
4. Изменить инетрвал времени обработки
5. Изменить вероятность ухода заявки на второй круг
6. Изменить необходимое кол-во закрытых заявок
7. Сбросить значения констант до стандартных
8. Просмотреть значения констант
9. Провести анализ
10. Включить (выключить) печать адресов
11. Включить (выключить) вывод погрешности
```

# Описание теста	Вввод	Вывод
Неправильный номер команды	-1 <Enter>	Сообщение об ошибке
Неправильный интервал	3 <Enter> -1 9 <Enter>	Сообщение об ошибке
Неправильный ввод вероятности	4 <Enter> 5.0 <Enter>	Сообщение об ошибке
Неправильный ввод кол-ва	6 <Enter> -5000 <Enter>	Сообщение об ошибке
Провести сравнительный анализ	9 <Enter>	Таблица-результат анализа

Оценка эффективности

Оценка по времени

Для решения задачи были использованы очереди, реализованные в виде списка и массива. Учитывая особенности каждой структуры, можно сделать вывод, что массив по времени окажется эффективнее списка, поскольку для добавления/удаления элемента в массив не нужно тратить время на дорогостоящую по времени операцию выделения памяти.

Оценка по памяти

Оценка по памяти во многом зависит от исходных данных задачи.

- Если максимальная длина очереди будет сравнима с размером массива, то последний окажется эффективнее списка, поскольку не хранит для каждого элемента указатель на следующий.
- В противном случе, список, благодаря своей гибкости, окажется эффектинвее массива, так как не будет занимать неиспользуемую память.

Практический результат

Размер массива	Средняя длина очереди	Время [массив] (μs)	Время [список] (μs)	Память [массив] (фактическая) (байт)	Память [массив] (всего) (байт)	Память [список] (байт)
2048	2.7	574.971	1048.369	832	32048	1208
2048	1512.6	667.687	1399.585	28304	32048	42416

Вывод

Выбор структуры данных для релизации очереди зависит от двух основных факторов:

- известна ли заранее максимальная длина очереди,
- какие требования к скорости работы программы.

1. **Максимальный размер очереди заранее неизвествен** - список.
   В таком случае массив может оказаться очень неэффективным по памяти, поскольку часть выделенной памяти под массив использована не будет. Кроме того, в таком случае возникает риск переполнения очереди, так как длина массива ограничена сверху.
   Конкретно, насколько список окажется эффективнее, зависит от выбранного размера массива и длины очереди. (В моем случае, при средней длине очереди `2.7` и размере массива `2048` элементов, список эффективнее в `26.5` раз или на `2550%`! Кроме того, замечу, что в случае массива использовалось лишь `2.6%` выделенной памяти.)

2. **Максимальный размер очереди известен заранее** - массив.
   В этом таком случае, при грамотном выборе размера массива, практически вся память, выделенная под массив будет использована, и при этом не придется хранить указатель на следующий элемент. Конкретно, насколько массив окажется эффективнее, зависит от выбранного размера массива и длины очереди. (В моем случае, при средней длине очереди `1512.6` и размере массива `2048` элементов, массив эффективнее в `1.32` раза или на `32%`! Кроме того, замечу, что на этот раз использовалось `88%` выделенной памяти под массив)

3. **Скорость програмы приоритетнее эффективности по памяти** - массив.
   Массив в лююбом случае эффективнее списка по времени, так как не требует выделения памяти при каждой постановке элемента в очередь и освобождения памяти при каждом выходе элемента из очереди.

Фрагментация памяти при использовании списка

Проанализирую часть вывода адресов списка.

```
-> (Адрес) - добавленный элемент, (Адрес) -> - высвобожденный элемент
-> (0x556efeb32a30)
-> (0x556efeb32a50)
-> (0x556efeb32a70)
-> (0x556efeb32a90)
(0x556efeb32a30) ->
-> (0x556efeb32a30)
-> (0x556efeb32ab0)
-> (0x556efeb32ad0)
-> (0x556efeb32af0)
(0x556efeb32a50) ->
-> (0x556efeb32a50)
-> (0x556efeb32b10)
-> (0x556efeb32b30)
-> (0x556efeb32b50)
(0x556efeb32a70) ->
-> (0x556efeb32a70)
-> (0x556efeb32b70)
-> (0x556efeb32b90)
(0x556efeb32a90) ->
-> (0x556efeb32a90)
```

Легко заметить, что происходит переиспользование адресов, откуда делаю вывод, что на моей машине не происходит фрагментация.

Контрольные вопросы

1. Что такое очередь?

- Очередь – это последовательный список переменной длины, включение элементов в который идет с одной стороны (с «хвоста»), а исключение – с другой стороны (с «головы»).

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

- При хранении списком память выделяется по одному элементу (при добавлении, вставке). При хранении массивом память выделяется один раз на максимальное количество элементов в очереди.

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

- При хранении списком память освобождается по одному элементу (при удалении). При хранении массивом память освобождается один раз после завершения работы с очередью.

4. Что происходит с элементами очереди при ее просмотре?

- При просмотре очереди элементы последовательно удаляются.

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

- См. «Вывод».

6. В каком случае лучше реализовать очередь посредством указателей, а в каком – массивом?

- См. «Вывод».

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

`Недостатки:`

- при реализации обычным массивом (не кольцевой структуры) при удалении элемента необходимо сдвигать весь массив
- при реализации массивом кольцевой структуры у очереди есть верхняя граница (то есть возможно переполнение)
- при реализации списком затрачивается дополнительная память на указатели и время на выделение памяти под новый элемент, освобождение его.

`Достоинства:`

- при реализации массивом кольцевой структуры добавление и удаление элементов очень быстрое (просто меняются указатели на начало или конец)
- при реализации списком затрачиваемая память теоретически ограничена лишь объёмом оперативной памяти

8. Что такое фрагментация памяти?

- Фрагментация – процесс появления незанятых участков в памяти (как оперативной, так и виртуальной и на магнитных носителях). Вызвана наличием в каждом виде памяти деления на мелкие единицы фиксированного размера, в то время как объём информации не обязательно кратен этому делению.

9. На что необходимо обратить внимание при тестировании программы?

- На переполнение очереди при реализации массивом кольцевой структуры и на процент расхождения расчётного времени и общего времени моделирования (он должен быть в пределах 2-3 процентов).

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

Выделение памяти происходит блоками — непрерывными фрагментами оперативной памяти (таким образом, каждый блок — это несколько идущих подряд байт). В какой-то момент в куче может не оказаться блока подходящего размера и, даже если свободная память достаточна для размещения объекта, операция выделения памяти окончится неудачей.