Лабораторная работа №5. "Работа с очередью"

Студент Ларин Владимир - ИУ7-34Б

Описание условия задачи

Система массового обслуживания состоит из обслуживающего аппарата (ОА) и двух очередей заявок двух типов.

Заявки 1-го и 2-го типов поступают в "хвосты" своих очередей по случайному закону с интервалами времени `T1` и `T2`, равномерно распределенными от `1` до `5` и от `0` до `3` единиц времени (е.в.) соответственно. В ОА они поступают из "головы" очереди по одной и обслуживаются также равновероятно за времена `T3` и `T4`, распределенные от `0` до `4` е.в. и от `0` до `1` е.в. соответственно, после чего покидают систему. (Все времена – вещественного типа). В начале процесса в системе заявок нет.

Заявка 2-го типа может войти в ОА, если в системе нет заявок 1-го типа. Если в момент обслуживания заявки 2-го типа в пустую очередь входит заявка 1-го типа, то она ждет первого освобождения ОА и далее поступает на обслуживание (система с **относительным** приоритетом).

Смоделировать процесс обслуживания первых 1000 заявок 1-го типа. Выдать на экран после обслуживания каждых 100 заявок 1-го типа информацию о текущей и средней длине каждой очереди, количестве вошедших и вышедших заявок и о среднем времени пребывания заявок в очереди. В конце процесса выдать общее время моделирования и количество вошедших в систему и вышедших из нее заявок обоих типов. По требованию пользователя выдать на экран адреса 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 `6` включительно.
2. Командно-зависимые данные:
    - интервал времени прихода / интервал времени обработки
        - левая и правая границы интервала (два действительных неотрицательных числа через пробел таких, что второе не меньше первого)
    - Yеобходимое кол-во закрытых заявок для прекращения работы
        - новое значения количества (натуральное число)
    - Переключатель
        - `0` для выключения или `1` включения

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,
    - информация о длине очереди через каждые 100 обработанных заявок.
    - погрешность от эталлоного значения
2. Статистика по времени выполнения и объему занимаемой памяти при обработке очередей, реализованных односвязным списком и динамическим массивом.
3. Информация о текущих значениях параметров программы
4. Адреса высвобожденных элементов списка и только что добавленных - печатаются по запросу пользователя в файл.

Команды программы

- Моделирование
    - на массиве
    - на списке
- Параметры моделирования
    - Просмотреть параметры
    - Редактировать параметры
    - Сбросить параметры
- Анализ эффективности

Обращение к программе:

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

Аварийные ситуации:

1. Некорректный ввод номера команды.
    - **На входе**: число, большее, чем максмальный индекс команды или меньшее, чем минимальный.
    - **На выходе**: Сообщение об ошибке
2. Некорректный ввод интервала
    - **На входе**: ввод, отличный от указанного в ТЗ
    - **На выходе**: Сообщение об ошибке
3. Некорректный ввод необходимого числа заявок, вышедших из системы
    - **На входе**: ввод, отличный от указанного в ТЗ
    - **На выходе**: Сообщение об ошибке

Структуры данных

Элементом очереди `qtype_t` является заявка, представляющая собой собой структуру `request_t`

```
typedef struct {
    size_t id; // Номер заявки
    void *address; // её адрес
} request_t;

typedef request_t qtype_t;
```

Очередь, построенная на основе массива представляет собой структуру `queue_array_t`

```
typedef request_t qtype_t;
typedef struct {
    qtype_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
} queue_array_t;
```

Очередь, построенная на основе списка представляет собой структуру `queue_list_t`

```
typedef struct node node_t;
struct node {
    qtype_t data; // Данные узла
    node_t *next; // Указатель на следующий узел
};

typedef struct {
    node_t *head, *tail; // Голова и конец списка
    size_t length; // Длина списка
} queue_list_t;
```

Структура данных, объединяющия интерфейс взаимодействия с очередями разного типа `queue_list_t`

```
typedef enum {
    QUEUE_LIST,
    QUEUE_ARRAY
} queue_type_t;

typedef struct {
  queue_type_t type; // Тип очереди
  union {
    queue_list_t *list;
    queue_array_t *array;
  } queue; // Указатель на очередь
} queue_t;
```

Описание основных функций

- `queue_t *create_queue(queue_type_t type);` - создание очереди нужного типа
- `void free_queue(queue_t *queue);` - Уничтожение очереди
- `void empty_queue(queue_t *queue);` - Удаление всех элементов очереди
- `bool is_empty_queue(const queue_t *queue);` - Проверка не пуста ли очередь

- `bool push_queue(queue_t *queue, const qtype_t *item);` - Добавление элемента в очередь
- `bool pop_queue(queue_t *queue, qtype_t *item);` - Получение элемента из очереди
- `size_t length_queue(const queue_t *queue);` - Получение длинны очерели
- `size_t get_queue_memory_size(const size_t size, queue_type_t type);` - Рассчет объёма памяти занимаемой очередью

Алгоритм

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

Моделирование очереди

0. Создаются пустые две очереди
1. Инициализируются переменная текущего времени, равная нулю
2. Генерирутся случайные время прибытия, двух заявок каждого типа
3. Пока колличество обработанных заявок больше 1000
    1. Проверяю пришло ли время добавить заявку в одну из очередей
    2. Если пришло, то
        - добавляю заявку в очередь
        - указываю новое случайное время прибытие новой заявки
    3. Проверяю свободен ли апарат
    4. Если свободен, то
        - Выбираю заявку
        - Изменяю время, когда освободится апарат
    5. Изменяю текущее время, на время ближайшего события

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 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 случае, список, благодаря своей гибкости, окажется эффективнее массива, так как не будет занимать память "прозапас".

В таблице приведено среднее для 1000 повторений

Пример 1

Суммарная длина очередей - 600

Очередь на массиве:

- время работы: 236 тиков
- объём занимаемой памяти: 16608 Байт
- объём используемой памяти: 9696 Байт

Очередь на списке:

- время работы: 261 тиков
- объём занимаемой памяти: 14464 Байт

Пример 2

Суммарная длина очередей - 1000

Очередь на массиве:

- время работы: 392 тиков
- объём занимаемой памяти: 16608 Байт
- объём используемой памяти: 16096 Байт

Очередь на списке:

- время работы: 428 тиков
- объём занимаемой памяти: 24064 Байт

Фрагментация памяти при использовании списка

Проанализирую часть вывода адресов списка.

```
Created: 0x558effabfd20 // 1
Closed:  0x558effabfd20 // 1
Created: 0x558effabfd20 // 1
Created: 0x558effac0d50 // 2
Created: 0x558effac0d70 // 3
Closed:  0x558effac0d70 // 3
Created: 0x558effac0d70 // 3
Closed:  0x558effabfd20 // 1
Closed:  0x558effac0d50 // 2
Created: 0x558effac0d50 // 2
Closed:  0x558effac0d50 // 2
Closed:  0x558effac0d70 // 3
Created: 0x558effac0d70 // 3
....
Created: 0x558effac0d90
Created: 0x558effabfd20 // 1
Created: 0x558effac0db0
```

Легко заметить, что происходит переиспользование адресов, откуда делаю вывод, что на моей машине не происходит фрагментация. Первый выданный адрес использовался до самого конца моделирования.

Контрольные вопросы

1. Что такое очередь?

- Очередь — абстрактный тип данных с дисциплиной доступа к элементам "первый пришёл — первый вышел" (FIFO, англ. first in, first out).

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 при инициализации структуры "прозапас" и расширяется по мере необходимости.

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?

- Смотрите "Вывод".

6. В каком случае лучше реализовать очередь посредством указателей, а в каком – массивом? Смотрите "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 (в том числе кольцевой структурой) у очереди есть верхняя граница, т.е. необходимо со временем расширять массив
    - 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 используется память без запаса

8. Что такое фрагментация памяти?

- Фрагментация – процесс появления незанятых участков в памяти.

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?

- Программа запрашивает у ОС страницу виртуальной памяти. Далее стандартная библиотека в этой виртуальной памяти, по запросу пользователя резервирует блок памяти нужного размера. Способ резервирования и освобождения зависит от реализизации аллокатора.

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.

Конкретно, насколько список окажется эффективнее, зависит от выбранного размера массива и длины очереди.

2. **Максимальный размер очереди извествен 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.

3. **Скорость програмы приоритетнее эффективности по памяти** - массив.

Массив в любом 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.