Отчёт по лабораторной работе №5 "Обработка очередей", Вариант 7

Выполнил: Калашков Павел ИУ7-36Б, Вариант 7 (по журналу)

Описание условия задачи

Цель работы: отработка навыков работы с типом данных «очередь», представленным в виде одномерного массива и односвязного линейного списка. 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.

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 вновь поступает в "хвост" очереди, совершая всего 5 циклов обслуживания, после чего покидает систему (все времена – **вещественного типа**). В начале процесса в системе заявок нет.

Смоделировать процесс обслуживания до ухода из системы первых 1000 заявок, выдавая после обслуживания каждых 100 заявок информацию о текущей и средней длине очереди, а в конце процесса - общее время моделирования и количестве вошедших в систему и вышедших из нее заявок, количестве срабатываний ОА, время простоя аппарата. По требованию пользователя выдать на экран адресов элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

Техническое задание

Исходные данные

Исходными данными для программы является данные, введённые с консоли (см. ниже);

Данные, введённые с консоли, связаны с меню программы:

```
Возможные действия:
 1 - Запустить очередь-массив
 2 - Запустить очередь-стек
 3 - Изменить интервалы прихода / обработки заявки
 4 - Сравнить очередь-стек и очередь-массив
 0 - Выйти из программы
Введите номер действия:
```

Поэтому введённые данные являются **строкой**, которая может содержать в себе номер действия, а также входные данные для конкретных действий (см. ниже)

Результаты

Выходными данными являются:

- текущее состояние очереди-массива
- текущее состояние очереди-списка
- результаты измерений времени работы программы

Задача, реализуемая программой

Задачей программы является моделироваие процесса обслуживания до ухода из системы первых 1000 заявок, с выдаванием после обслуживания каждых 100 заявок информации о текущей и средней длине очереди, а в конце процесса - общее время моделирования и количестве вошедших в систему и вышедших из нее заявок, количестве срабатываний ОА, время простоя аппарата.

При этом должна быть позможность по запросу пользователя выдавать на экран адресов элементов очереди при удалении и добавлении элементов.

Способ обращения к программе

Обращение к программе происходит посредством запуска исполняемого файла app.exe

Возможные аварийные ситуации и ошибки пользователя

Аварийной ситуацией является:

- несоблюдение формата входных данных (см. исходные данные);

Описание внутренних структур данных

Для моделирования обработки очередей были созданы 4 структуры:

```
typedef struct node node_t;

struct node
{
    int value;
    node_t *next;
};

typedef struct
{
    node_t *p_in;
    node_t *p_out;
} node_queue_t;

typedef struct
{
    int *p_in;
    int *p_out;
    int *p_start;
    int *p_end;
} array_queue_t;

typedef struct
{
    array_queue_t *array_queue;
    node_queue_t *node_queue;
    int interval_min;
    int interval_max;
    int process_min;
    int process_max;
} info;
```

Описанный алгоритм

Алгоритм делится на четыре главных части:

- 1 - вывод меню;
- 2 - запрос действия из меню;
- 3 - выполнение действия (если оно указано корректно);
- 4 - повтор, если действие - не выход из программы;

Вывод меню

Меню имеет формат:

```
Возможные действия:
 1 - Запустить очередь-массив
 2 - Запустить очередь-стек
 3 - Изменить интервалы прихода / обработки заявки
 4 - Сравнить очередь-стек и очередь-массив
 0 - Выйти из программы
Введите номер действия:
```

Запрос действия из меню

Программа запрашивает номер действия как строку, считывает и проверяет на корректность.

Если корректно, то переводит строку в число, иначе выводит сообщение об ошибке.

Выполнение действия

- 1 - Смоделировать при помощи очереди-массива процесс обслуживания до ухода из системы первых 1000 заявок, выдавая после обслуживания каждых 100 заявок информацию о состоянии очереди.
- 2 - Уточнить у пользователя, нужна ли дополнительная информация об адресах добавляемых и удаляемых элементов. Смоделировать при помощи очереди-списка процесс обслуживания до ухода из системы первых 1000 заявок, выдавая после обслуживания каждых 100 заявок информацию о состоянии очереди.
- 3 - Последовательно запросить у пользователя два интервала на положительном множестве чисел - эти интервалы будут задавать время поступления и обработки заявки.
- 4 - Измерить время работы и объём используемой памяти (средние) для операций добавления и удаления элементов в очередь-массив и очередь список, вывести результаты.

При исследовании работы программы в пункте 2.1 мы видим, что на Linux Ubuntu (на котором производилось написание и выполнение данной лабораторной работы) адреса выделяются с зависимостью от содержания списка свободных областей, **т.е. фрагментации не происходит**.

- 0 - Выйти из программы: вывести сообщение о завершении программы;

Анализ модели

Стандартный результат окончания работы модели (в данном - случае на примере очереди-массива):

```
 Общее время моделирования: 2947.025711
Количество вошедших заявок: 5000
Количество вышедших заявок: 1000
Количетсво срабатываний ОА: 5000
Время простоя аппарата: 77.007302 ед. вр.
```

При этом средняя длина очереди равна нулю.

Среднее время прихода заявки = 3 ед. вр.

Среднее время обработки заявки = 0.5 ед. вр.

Т.к. время прихода больше времени обработки, то:

Время моделирования = время моделирования по входу = среднее время прихода заявки * количество вошедших заявок = 3000 ед. вр.

Среднее настоящее время моделирования равняется 3000.206396 ед. вр.

Расхождение настоящего времени моделирования и расчётного составляет (3000.206396 - 3000) * 100 % / 3000 = 0.006531 %

Сравнение операций

Сравним время работы и объём используемой памяти для используемых операций добавления и удаления элемента в очередь-массив и очередь-список, усреднённого по N = 1000

Для очереди-массива:

Время добавления, очередь-массив, нс	Время удаления, очередь-массив, нс	Память на 1 элемент, очередь-массив, байты
13	6	4

Для стека-списка:

Время добавления, стек-список, нс	Время удаления, стек-список, нс	Память на N элементов, стек-список, байты
279	10	16

Видим, что и реализация очереди при помощи статического массива является эффективнее как по времени (в 22 раза для добавления и в 1.5 раза для удаления), так и по памяти (в 4 раза), т.к. использование статического массива позволяет однократно выделить память, в то время как при использовании списка необходимо каждый раз аллоцировать память (при добавлении) и освобождать (при дополнении).

Теоретическая часть

1. Что такое FIFO и LIFO?

Это приципы добавления и удаления элементов для структур данных:

- FIFO - First In First Out - первым пришёл - первым ушёл (верно для, например, очереди)
- LIFO - Last In First Out - последним пришёл - первым ушёл (верно, например, для стека)

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

При реализации в виде статического массива память выделяется один раз размером

```
sizeof(int) * queue_length
```

При реализации в виде динамического массива память будет выделяться несколько раз (точнее говоря, первый раз выделяться и несколько раз после этого перевыделяться). Объём выделяемой памяти зависит от способа перевыделения (можно увеличивать длину массива на константу, можно увеличивать в несколько раз и т.д.)

При реализации в виде связного списка память будет выделяться каждый раз при добавлении элемента. Объём суммарно занятой памяти будет равен

```
sizeof(node_t) * queue_length
```

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

При реализации статическим массивом память будет освобождаться один раз при окончании работы программы.

При реализации динамическим массивом память будет перевыделяться при расширении массива (соответственно, первично занятая область будет освобождаться), а также окончательно освобождаться в конце. При удалении элемента из очереди освобождения памяти не происходит.

При реализации в виде связного списка память будет особождаться каждый раз при удалении элемента.

4. Что происходит с элементами очереди при ее просмотре?

Элемент исключается из очереди (удаляется из очереди).

5. От чего зависит эффективность физической реализации очереди?

В основном, от способа реализации очереди и от её длины. Так, при известной (и не очень большой) длине эффективнее всего будет использовать статический массив, в то время как при известной и большой - динамический, а при крайне большой и неизвестной - список.

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

Реализация массивом является эффективнее, однако зависит от памяти (неэффективно при больших длинах).

7. Что такое фрагментация памяти, и в какой части ОП она возникает?

Фрагментация - выделение памяти не последовательными блоками, вледствие чего получаются "дыры", внешние и внутренние (внутренние - неиспользуемая часть выделенного блока, внешние - свободный блок, но он мал для использования). Возникает в динамической памяти (куче).

8. Для чего нужен алгоритм «близнецов».

Алгоритм "близнецов" необходим для ускорения работы программы (для увеличения эффективности работы). Алгоритм "близнецов" значительно уменьшает фрагментацию памяти и резко ускоряет поиск блоков памяти.

9. Какие дисциплины выделения памяти вы знаете?

Самый подходящий (best fit): выделяется тот свободный участок, размер которого равен запрошенному или превышает его на минимальную величину.

Первый подходящий (first fit): выделяется первый же найденный свободный участок, размер которого не меньше запрошенного.

10. На что необходимо обратить внимание при тестировании программы?

При тестировании программы необходимо проверять правильность работы программы при различном заполнении очередей, т.е. когда время моделирования определяется временем обработки заявок / временем прихода заявок.

Также необходимо отслеживать переполнение очереди, если очередь в программе ограничена.

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

Для выделения памяти нужно знать объём выделяемой памяти (т.е. объём, занимаемый одним элементом и количество элементов).

При освобождении памяти адреса освобождаются и возвращаются в кучу, становятся доступны для повторного выделения. При представлении на битовой карте сбрасываются в 0 биты, соответствующие освобождённым кадрам.

Выводы по проделанной работе

В ходе лабораторной работы я научился создавать, использовать и удалять очереди, а также освоил два способа их реализации: при помощи массива и при помощи списка. Также я выяснил, что зачастую хранение очереди в виде массива является более эффективным способом, чем хранение в виде списка (более, чем на 50%), однако только если массив статический. Эффективность использования динамического массива можно повысить, зная пределы возможных размеров или оптимизируя алгоритм выделения памяти (например, не увеличивать в два раза, а добавлять 100 элементов), а при неизвестном и достаточно большом диапазоне размерности очереди необходимо использовать реализацию при помощи списка.