Лабораторная работа №5.
Обработка очередей

Работу выполнил: Романов Алексей

группа ИУ7-33Б

МГТУ им. Баумана
2019

Условие задачи

Провести сравнительный анализ реализации алгоритмов включения и исключения элементов из очереди при использовании указанных структур данных, оценить эффективности программы по времени и по используемому объему памяти.

Техническое задание

Смоделировать процесс обслуживания до выхода из ОА2 первых 1000 заявок. Выдавать на экран после обслуживания в ОА2 каждых 100 заявок информацию о текущей и средней длине каждой очереди, а в конце процесса - общее время моделирования, время простоя ОА2, количество срабатываний ОА1, среднее времени пребывания заявок в очереди. Обеспечить по требованию пользователя выдачу на экран адресов элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

Входные данные

Вероятность выйти из первой очереди, время обработки в очередях, пункт выбора меню, элемент очереди.

Выходные данные

Текущее состояние очереди, временные замеры, количество занимаемой памяти.

Возможные аварийные ситуации

Некорректный ввод элемента очереди.

Структуры данных

Структура хранения очереди

```
typedef struct queue
{
   int size;
   queue_arr_t arr;
   queue_list_t list;
   double total_time;
   double avg_time;
} queue_t;
```

size — текущий размер очереди.
array — кольцевой массив, list — связанный список.
total_time — общее время обработки, avg_time — среднее время обработки.

Структура хранения связанного списка

```
typedef struct list
{
   node_t *list_head;
} queue_list_t;
```

list_head — указатель на начало списка.

```
typedef struct node
{
   double time_service;
   double total_time;
   struct node *next_node;
} node_t;
```

time_service — время последнего обслуживания.
total_time — общее время прибывания во всех очередях.

Структура хранения кольцевого массива

```
typedef struct queue_arr
{
   array_element_t *start;
   array_element_t *start_initial;
   array_element_t *end;
   array_element_t *end_initial;
} queue_arr_t;
```

start – текущее начало массива, start_inital – «настоящее» начало массива
end — текущий конец массива, end_inital – «настоящий» конец массива

```
typedef struct arr_elem
{
   double time_service;
   double total_time;
} array_element_t;
```

time_service — время последнего обслуживания.
total_time — общее время прибывания во всех очередях.

Алгоритм

Обрабатываются 100 заявок в первой очереди, подсчитывается время, которое для этого потребовалось. Далее, обрабатывается вторая очередь, в течении времени которое было затрачено для обработки первой очереди. Так продолжается, пока из второй очереди не выйдет 1000 заявок.

Тесты

Среднее время обработки в первой очереди: 1.5 секунды.

Вероятность выхода из первой очереди	Ожидаемое время моделирования	Реальное время моделирования
0.3	5000 секунд	4996 секунд
0.5	4500 секунд	4400 секунд
0.7	4500 секунд	4580 секунд

Среднее время обработки в первой очереди: 3 секунды.

Вероятность выхода из первой очереди	Ожидаемое время моделирования	Реальное время моделирования
0.3	10000 секунд	99500 секунд
0.5	6000 секунд	6075 секунд
0.7	4500 секунд	4580 секунд

Время

Добавление элементов

Количество элементов	Список	Массив
10	12100 тиков	2550 тиков
50	25500 тиков	11000 тиков
100	47500 тиков	22000 тиков

Удаление элементов

Количество элементов	Список	Массив
10	53000 тиков	2500 тиков
50	265000 тиков	13200 тиков
100	535000 тиков	27000 тиков

Память

Занимаемая память

Количество элементов	Список	Массив
10	240 байт	160 байт
100	2400 байт	1600 байт
1000	24000 байт	16000 байт

Выводы по проделанной работе

Использование связанных списков невыгодно при реализации очереди. Списки проигрывают как по памяти, так и по времени обработки. Но, когда заранее неизвестен максимальный размер очереди, то можно использовать связанный списки, так как в отличии от статического массива, списки ограничены в размерах только размером оперативной памяти. Стоит отметить, что при проведении тестов ни разу не наблюдалась фрагментация памяти (на моём ПК), но даже при этом, список проигрывает во времени обработки кольцевому массиву.

Контрольные вопросы

Что такое очередь?

Очередь — структура данных, для которой выполняется правило FIFO, то есть первым зашёл — первым вышел. Вход находится с одной стороны очереди, выход — с другой.

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

При хранении кольцевым массивом: кол-во элементов * размер одного элемента очереди. Память выделяется на стеке при компиляции, если массив статический. Либо память выделяется в куче, если массив динамический.

При хранении списком: кол-во элементов * (размер одного элемента очереди + указатель на следующий элемент). Память выделяется в куче для каждого элемента отдельно.

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

При хранении кольцевым массивом память не освобождается, а просто меняется указатель на конец очереди. При хранении списком, память под удаляемый кусок освобождается.

Что происходит с элементами очереди при ее просмотре?

Эти элементы удаляются из очереди.

Каким образом эффективнее реализовывать очередь. От чего это зависит?

Зная максимальный размер очереди, лучше всего использовать кольцевой статический массив. Не зная максимальный размер, стоит использовать связанный список, так как такую очередь можно будет переполнить только если закончится оперативная память.

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

При использовании кольцевого массива тратится больше времени на обработку операций с очередь, а так же может возникнуть фрагментация памяти. При реализации статическим кольцевым массивом, очередь всегда ограничена по размеру.

Что такое фрагментация памяти?

Фрагментация памяти — разбиение памяти на куски, которые лежат не рядом друг с другом. Можно сказать, что это чередование свободных и занятых кусков памяти.

На что необходимо обратить внимание при тестировании программы?

Корректное освобождение памяти.

Каким образом физически выделяется и освобождается память при динамических запросах?

При запросе памяти, ОС находит подходящий блок памяти и записывает его в «таблицу» занятой памяти. При освобождении, ОС удаляет этот блок памяти из «таблицы» занятой пользователем памяти.