|  |  |
| --- | --- |
|  | **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 К ЛАБОРАТОРНОЙ РАБОТЕ №5 ПО «ТИПАМ И СТРУКТУРАМ ДАННЫХ»**

**Тема: «Обработка очередей»**

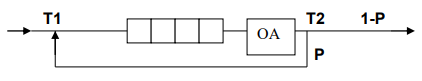
Группа: ИУ7-32Б

|  |  |  |
| --- | --- | --- |
| Студент: |  | Сидоров Максим Михайлович |

Преподаватель: Никульшина Татьяна Александровна

**Цель работы:** научиться работать с очередью и смоделировать ее обработку.

**Условие задачи:** Система массового обслуживания состоит из обслуживающего аппарата (ОА) и очереди заявок.



Заявки поступают в "хвост" очереди по случайному закону с интервалом времени Т1, 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 Т2 от 0 до 1 е.в., Каждая заявка после ОА с вероятностью Р=0.8 вновь поступает в "хвост" очереди, совершая новый цикл обслуживания, а с вероятностью 1-Р 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, 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.

**Входные данные:** номер команды из меню, терминальный ввод.

Команды:

1. Моделирование на основе массива
2. Моделирование на основе списка
3. Сравнение списка и массива
4. Сравнение операций
5. Выйти

**Выходные данные:**

- время моделирования

- погрешности

- сравнение списка и массива

**Аварийные ситуации:**

- ввод неправильной команды

**Описание СД:**

1. Для хранения элемента очереди-списка было выбрано следующее представление:

typedef struct queue\_cell queue\_cell;

struct queue\_cell

{

int a;

queue\_cell \*prev;

};

*Листинг 1. Представление элемента очереди-списка.*

1. Для хранения очереди-списка было выбрано следующее представление:

typedef struct

{

queue\_cell \*head;

queue\_cell \*tail;

int len;

int in\_num;

int out\_num;

int max\_len;

} queue\_list;

*Листинг 2. Представление очереди-списка.*

1. Для хранения очереди-массива было выбрало следующее представление:

#define QUEUE\_SIZE 1000

typedef struct

{

int arr[QUEUE\_SIZE];

int head;

int tail;

int len;

int in\_num;

int out\_num;

} queue\_arr;

*Листинг 3. Представление очереди-массива.*

**Алгоритмы:**

**Удаление:** чтобы удалить элемент из очереди, нужно освободить память из-под элемента, на который указывает указатель на голову очереди, а указатель на голову очереди переместить на элемент, на который указывал освобожденный элемент. Необходима проверка на пустоту очереди.

**Добавление:** чтобы добавить элемент в конец очереди, нужно создать новый элемент списка, указатель этого элемента поставить на текущий хвост очереди, а указатель на хвост очереди переместить на только что созданный элемент.

**Функции:**

bool arr\_is\_empty(queue\_arr \*a) – проверка очереди-массива на пустоту

bool arr\_is\_full(queue\_arr \*a) – проверка очереди-массива на переполнение

bool push\_arr\_new(queue\_arr \*a) – добавление в очередь-массив нового элемента

void push\_arr\_old(queue\_arr \*a) – добавление последнего элемента очереди-массива в начало

void pop\_arr(queue\_arr \*a) – удаление элемента из очереди-массива

void print\_arr\_addresses(queue\_arr \*a) – вывод адресов очереди-массива

void simulate\_arr(data \*data, int code, bool flag) – моделирование очереди-массива

void simulate\_arr\_and\_print() – моделирование очереди-массива и вывод времени и погрешностей

void print\_menu() – вывод меню

double gen\_time(int start, int end) – генерирование время из промежутка

void compare() – сравнить очередь-массив и очередь-список

bool list\_is\_empty(queue\_list \*a) – проверка очереди-списка на пустоту

bool list\_is\_full(queue\_list \*a) – проверка очереди-списка на переполнение

bool push\_list\_new(queue\_list \*a) – добавление в очередь-список нового элемента

void push\_list\_old(queue\_list \*a) – добавление в начало очереди-списка элемента с конца списка

void pop\_list(queue\_list \*a) – удаление элемента очереди-списка

void free\_list(queue\_list \*a) – очищение очереди-списка

void print\_list\_addresses(queue\_list \*a) – вывод адреса очереди-списка

void simulate\_list(data \*data, int code, bool flag) – моделирование очереди-списка

void simulate\_list\_and\_print() – моделирование очереди-списка и вывод времени и погрешностей

**Тестовые данные:**

|  |  |  |
| --- | --- | --- |
| **Описание** | **Вход** | **Выход** |
| Некорректная команда | А, -1, 15 | Некорректная команда |
| Моделирование очереди-списка | - | Время и погрешности |
| Моделирование очереди-массива | - | Время и погрешности |
| Сравнение алгоритмов | - | Объем памяти и время |

**Сравнение эффективности:**

Рассмотрим время моделирования.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **T1, е.в.** | **Т2, е.в.** | **Время работы модели для очереди-списка, мкс** | **Время работы модели для очереди-массива, мкс** | **Процент эффективности массива** |
| **0..6** | **0..1** | 187 | 162 | 15.4% |
| **0..6** | **0..2** | 193 | 168 | 14.9% |
| **0..4** | **0..1** | 169 | 144 | 17.3% |
| **0..6** | **0..6** | 345 | Переполнение | - |
| **0..1** | **0..12** | 2604 | Переполнение | - |

Реализация через очередь-список в среднем медленнее на 16% из-за того, что для каждого элемента списка нужно выделять или освобождать память, а для массива достаточно переместить указатель на хвост или голову.

Рассмотрим затрачиваемую память при моделировании.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **T1, е.в.** | **Т2, е.в.** | **Память для очереди-списка, байты** | **Память для очереди-массива, байты** | **Процент эффективности массива** |
| **0..6** | **0..1** | 256 | 4020 | -93.6% |
| **0..6** | **0..2** | 11576 | 4020 | 188% |
| **0..4** | **0..1** | 4176 | 4020 | 3.9% |
| **0..6** | **0..6** | 64784 | 4020 | - |
| **0..1** | **0..12** | 974752 | 4020 | - |

Размер затрачиваемой памяти напрямую связан со временем прихода заявок и временем их обработки. Чем дольше время прихода и быстрее время обработки заявки, тем меньше длина очереди, поэтому список выигрывает по памяти статический массив.

Для списка на 1000 элементов:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Добавление, тики** | **Удаление, тики** | **Память, байты** |
| **FIFO** | 42 | 48 | 24020 |
| **LIFO** | 40 | 52 | 24008 |

Для массива на 1000 элементов:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Добавление, тики** | **Удаление, тики** | **Память, байты** |
| **FIFO** | 24 | 14 | 4020 |
| **LIFO** | 22 | 12 | 4008 |

Очередь и стек работают примерно за одинаковое время и затрачивают примерно одинаковое количество памяти.

Эффективнее использовать стек и очередь через массив, так как не нужно выделять динамически память под новый элемент и не нужно переприсваивать указатели на элементы. Использовать список нужно тогда, когда количество элементов неизвестно или невозможно выделить один большой блок памяти под массив.

**Вывод:** очередь и стек, реализованные с помощью массива работают быстрее, чем реализованные на списке. А размер затрачиваемой памяти зависит от времени прихода заявок и времени их обработки в системе.

При удалении массив быстрее списка примерно в 4 раза, а при добавлении примерно в 2 раза. При реализации с помощью списка нужно следить за фрагментацией памяти.

**Ответы на вопросы:**

1. **Что такое FIFO и LIFO?**

FIFO – очередь, структура данных, для которой выполняется правило First In First Out, добавление с одной стороны, удаление – с другой. LIFO – стек, структура данных, для которой выполняется правило Last In First Out, удаление и добавление с одной стороны.

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

При реализации массивом объем памяти: (размер массива) \* sizeof(элемента). Если массив статический, то память выделяется на стеке, а если динамический – в куче.

При реализации списком: (количество элементов) \* (sizeof(элемент) + sizeof(указатель)). Память выделяется в куче для каждого элемента отдельно.

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

При использовании массива память не освобождается, сдвигается указатель на конечный элемент.

При использовании списка освобождается память из-под удаляемого элемента.

1. **Что происходит с элементами очереди при ее просмотре?**

Рассматриваемый элемент удаляется из очереди.

1. **От чего зависит эффективность физической реализации очереди?**

Эффективность реализации очереди зависит от способа реализации: на массива или на списке. Если можно выделить один большой кусок памяти, то эффективнее массив, иначе список.

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

Достоинства массива: скорость. Недостатки: ограниченность размера.

Достоинства списка: память ограничена размером оперативной памяти. Недостатки: низкая скорость.

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

Фрагментация памяти – разбиение памяти на куски, которые лежат не рядом друг с другом. Происходит в виртуальной памяти.

1. **Для чего нужен алгоритм «близнецов».**

Данный алгоритм снижает фрагментацию памяти и ускоряет поиск блоков.

1. **Какие дисциплины выделения памяти вы знаете?**

Наиболее подходящий блок – наименьший по объему, удовлетворяющий запросу.

Первый подходящий – первый блок, удовлетворяющий запросу.

1. **На что необходимо обратить внимание при тестировании программы?**

Для списка: на корректное освобождение памяти и на фрагментацию памяти.

Для массива: на переполнение.

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

При выделении резервируются блоки памяти под определенный данные. При освобождении памяти эти блоки перестают быть зарезервированными и происходит уплотнение памяти.