Оберган Татьяна ИУ7-35Б

Вариант 18

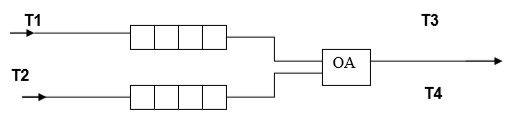
**Лабораторная работа №5**

Обработка очередей

**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.

**Условие задачи (вар 4):**

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 Т1 и Т2, 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 Т3 и Т4, 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 любого типа может войти в ОА, если:

* а) она вошла в пустую систему;
* б) перед ней обслуживалась заявка ее же типа;
* в) перед ней из ОА вышла заявка другого типа, оставив за собой пустую очередь (система с чередующимся приоритетом).

Смоделировать процесс обслуживания первых 1000 заявок 1-го типа, выдавая после обслуживания каждых 100 заявок информацию о текущей и средней длине каждой очереди, а в конце процесса - 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.

**Входные данные:**Интервалы времени, число заявок первого типа для обработки, через какой промежуток заявок сохранять лог.

**Выходные данные:**Лог, краткая сводка по процессу.

**Функция программы:**Реализация системы обработки заявок из двух очередей, с чередующимся приоритетом.

**Обращение к программе** осуществляется через консоль.

**Структуры данных:**

Очередь массивом:

**char** \*q1, \*q2;

Очередь списком:

node \*q1 = **NULL**, \*q2 = **NULL**;

**struct** node  
{  
 **char** inf;  
 node \*next;  
};

Дескриптор:

**struct** discriptor  
{  
 **void**\* low;  
 **void**\* up;  
 **void**\* p\_in;  
 **void**\* p\_out;  
 **int** max\_num;  
 **int** count\_request;  
 **int** sum\_size;  
 **int** curr\_size;  
 **int** out\_request;  
 **int** in\_request;  
};

Интервалы времени:

**struct** time\_range  
{  
 **double** min;  
 **double** max;  
};

**Функции:**

Функции ввода/вывода:  
**void** input\_values(**int** \*n, **int** \*interval, **int** \*log, time\_range \*T1, time\_range \*T2, time\_range \*T3, time\_range \*T4);  
**void** input\_range(time\_range \*T, **int** i);  
**void** print\_values(**int** n, **int** interval, **int** log, time\_range \*T1, time\_range \*T2, time\_range \*T3, time\_range \*T4);  
**void** print\_range(time\_range \*T, **int** i);

Функции для очереди массивом:

**void** arr\_push(discriptor\* d, **char**\* qu, **char** c);  
**char** arr\_pop(discriptor\* d, **char**\* qu);  
**void** arr\_print(discriptor\* d, **char**\* qu);  
  
**void** go\_array(**int** n, **int** interval, time\_range t1, time\_range t2, time\_range t3, time\_range t4, **int** log\_flag);

Функции для очереди списком:

node\* create\_node(**char** c);  
node\* add\_node(node \*head, node \*item);  
node\* pop\_node(node \*\*head);  
  
**void** free\_all(node \*head);  
  
  
node\* list\_push(node\* qu, **char** c, node\*\* used\_memory, **int** \*count\_used, node\*\* freed\_memory, **int**\* count\_freed, **int**\* second\_used);  
node\* list\_pop(node\*\* qu);  
**void** list\_print(node\* qu);  
  
**void** go\_list(**int** n, **int** interval, time\_range t1, time\_range t2, time\_range t3, time\_range t4, **int** log\_flag);

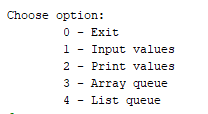
Функция генерации времени:

**double** rand\_time(**struct** time\_range t);

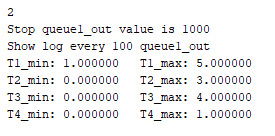
Функции взаимодействия с дескриптором:

**void** init\_discriptor(discriptor \*d);  
**void** init\_discriptor\_arr(discriptor \*d, **void** \*q);  
**void** upd\_dis\_del(discriptor \*d);  
**void** upd\_dis\_add(discriptor \*d);  
  
**int** queue\_is\_empty(discriptor d);

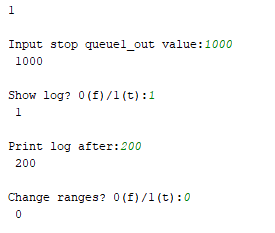
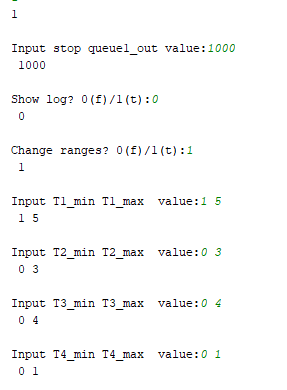
**Интерфейс:**

Меню:

Вывод значений:

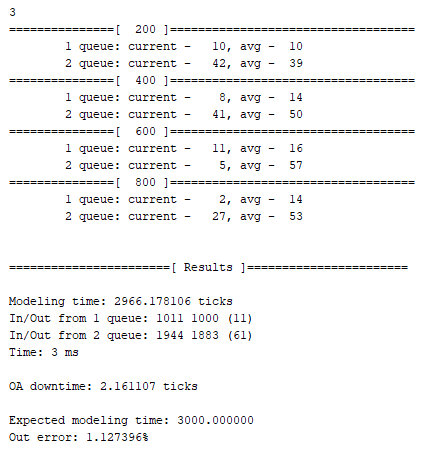


Изменение значений:

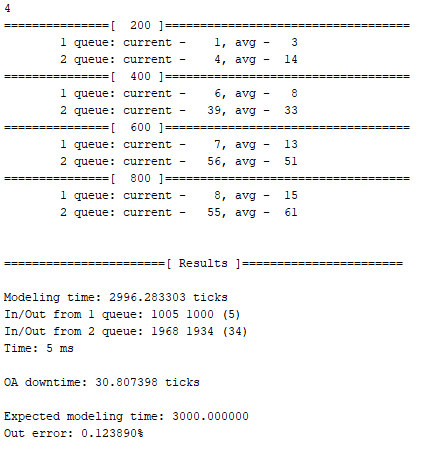


Работа программы:

(Очередь массивом)

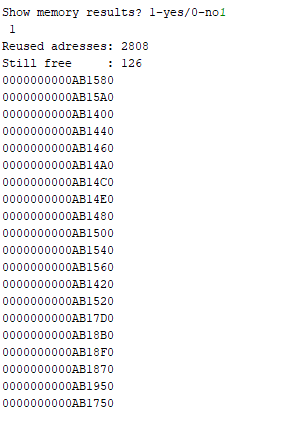


(Очередь списком)



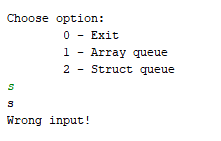
**Фрагментация памяти:**

Большая часть адресов используются повторно.



**Тесты:**

Некорректный ввод:



**Анализ эффективности (по памяти и времени):**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 000 | 10 000 | 100 000 |
| Массив | <1 (20 016b) | 4 (20 016b) | 36 (20 016b) |
| Список | 2 (800b) | 33 (337b) | 1164 (3808b) |

\*время в ms

Размер списка сильно варьируется.

**Контрольные вопросы:**

1. **Что такое очередь?**

Очередь – последовательной список переменной длины. Включение элементов идёт с «хвоста» списка, исключение – с «головы» списка. Принцип работы: первым вошел – первым вышел.

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

При реализации списком, память выделяется динамически по мере необходимости.

При реализации массивом, память выделяется сразу под все элементы, которые затем хранятся в массиве.

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

При реализации списком, считывается первый с головы (текущий) элемент, происходит смещение головы, а тот элемент освобождается.

При реализации очереди массивом, считывается текущий элемент, остальные элементы сдвигаются на 1 элемент в сторону текущего элемента.

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

При просмотре очереди текущий элемент из нее удаляется.

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

Выбор способа зависит от приоритетов: время или память.

При реализации списком легче добавить и удалить элемент, но при этом может возникнуть фрагментация памяти. При реализации массивом при удалении необходимо сдвигать все его элементы, что, при больших размерах, может быть очень затратно по времени.

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

При реализации очереди массивом не возникает фрагментации памяти, затрачивается время на сдвиг элементов. Чтобы этого избежать, можно использовать кольцевой массив, но будет сложнее реализовать операции работы с такой очередью. При реализации списком может возникнуть фрагментация памяти.

8. **Что такое фрагментация памяти?**

Фрагментация – чередование занятых и свободных участков памяти при последовательных запросах на добавление и удаление. Свободные участки могут быть слишком малы, чтобы хранить в них нужную информацию.

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

Необходимо обратить внимание на корректное освобождение памяти при удалении элемента из очереди.

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

Программа запрашивает блок памяти необходимого размера. ОС находит подходящий блок, записывает его адрес и размер в таблицу адресов, а затем возвращает данный адрес в программу. При запросе на освобождение указанного блока программы, ОС убирает его из таблицы адресов, адрес считается освобожденным. При попытке доступа к освобожденной памяти могут возникнуть ошибки.

**Вывод:** эффективнее использовать массив. Дает заметный выигрыш по времени при больших размерах очереди.