Министерство образования и науки 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 (ИУ)

КАФЕДРА: Программное обеспечение ЭВМ и информационные технологии (ИУ7)

ОТЧЕТ ПО ЛАБРАТОРНОЙ РАБОТЕ №5

Название предмета: Типы и структуры данных

Студент: Кашима Ахмед

Группа: ИУ7-33Б

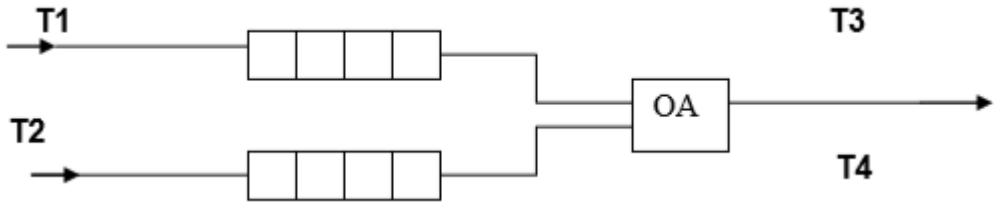
2022 г.

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 , 25

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:

Исполняемый файл app.exe создается путем автоматической сборки проекта с помощью файла makefile. Для выполнения работы следует запустить данный исполняемый файл без каких-либо аргументов.

Структуры данных:

```
typedef struct data - структура для хранения элемента очереди
{
      double i;
      data_t *next;
} data_t;

double mas[N] = {-1}; - массив с очередью 1
double mas1[N] = {-1}; - массив с очередью 2
double *pin = mas, *pout = mas; - начало и конец очереди 1
double *pin1 = mas1, *pout1 = mas1; - начало и конец очереди 2

data_t *empty[2*N] = {NULL}; - массив пустых адресов
data_t *full[2*N] = {NULL}; - массив полных адресов
data_t *pin = NULL, *pout = NULL; - начало и конец 1 очереди
```

```
data_t *empty1[2*N] = {NULL}; - массив пустых
data_t *full1[2*N] = {NULL}; - массив полных
data_t *pin1 = NULL, *pout1 = NULL; - начало и конец 2 очереди

long long int begin_time, end_time; - время в тиках
double t1 = 0, t2 = 0, toa = 0, tmin = 0, tstop = 0, t_all = 0;
   1. время добавления эл1, эл2, время обработки, минимальное время,время
      простоя, общее время выполнения
int type = 1; - тип элемента в ОА
int count_in1 = 0, count_out1 = 0, count_in2 = 0, count_out2 = 0;
   1) кол-во эл1 на вход, на выход, кол-во эл2 на вход, на выход
int count1 = 0, count2 = 0, count_sr1 = 0, count_sr2 = 0, iter;
   • колво эл-тов в 1 очереди, кол во элементов во 2 очереди, среднее первой
     очереди, во второй
double time_in1 = 0, time_in2 = 0, t1_sr = 0, t2_sr = 0, t3_sr = 0, t4_sr =
0; - средние генерирующиеся времена
```

Тесты:

1.Несуществующий пункт меню

Вход: 10

Вывод: Input error

2. Некорректный ввод t (t < 0)

Вход:

n: -1

Вывод: Input error

3. Переполнение массива

Вход: t1 0 1

t2 0 3

t3 0 10

t4 0 10

Вывод: FULL Array

4. Невозможно завершить выполнение

Вход: t1 0 1

t2 0 1

t3 0 5

t4 0 5

Вывод: can’t stop process

Теоретически рассчитываем время моделирования. Время обработки заявки 1 типа лежит в интервале от 0 до 4 е.в., значит среднее значение времени обработки одной заявки 2 е.в. Значит, общее время обработки **1000** заявок будет: 1000*2 = **2000 е.в**.

Для прихода 1000 заявок, если каждая из них приходит в среднем за 3 е.в., потребуется **3000 е.в.** Следовательно, время моделирования будет определяться временем прихода заявок, т.е. оно должно быть равно **3000 е.в.**

В то же время, за 3000 е.в. успеет прийти **2000** заявок второго типа, если их среднее время прихода равно 1.5 е.в.

В свободное от обработки заявок 1ого типа время, ОА будет обрабатывать заявки 2ого типа. На обработку 2000 заявок 2 типа, если среднее время обработки равно 0.5 е.в. потребуется **1000 е.в.**

Время простоя будет равно разнице между временем обработки заявок и временем их обслуживания: 5000 – 4000 = 1000 е.в. Но, с учетом того, что это время будет использовано для обработки заявок 2 типа, время простоя = 1000 - 1000 = **0 е.в.**

На практике получаем следующие результаты:

используя списка:

process time = 3014.737043

Wait time = -11.126792

in 1 type = 1015

in 2 type = 2018

out 1 type = 1000

out 2 type = 2012

used memory 48528

percent is 1.500000

используя массива:

process time = 3032.905085

Wait time = -32.457498

in 1 type = 1016

in 2 type = 2001

out 1 type = 1000

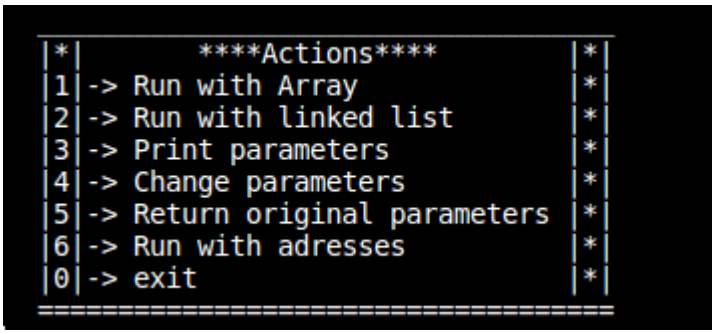
out 2 type = 1975

percent is 1.600000

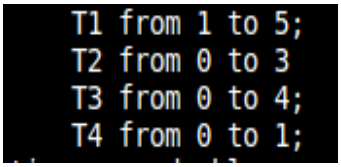
used memory 8000

Интерфейс:

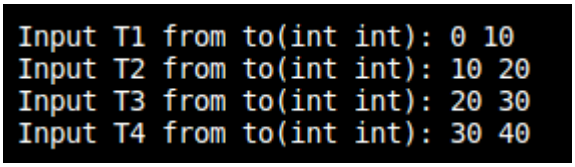
Меню:



Вывод значений:

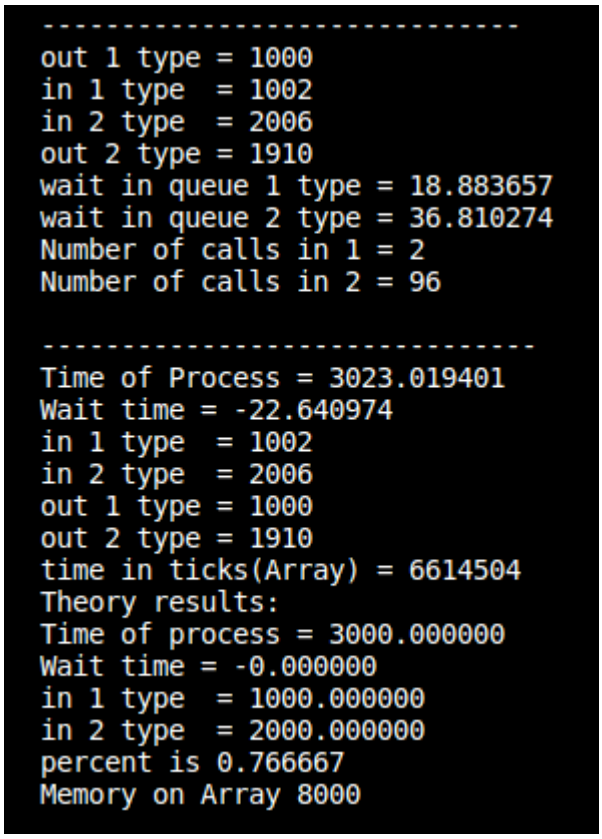


Изменение значений:

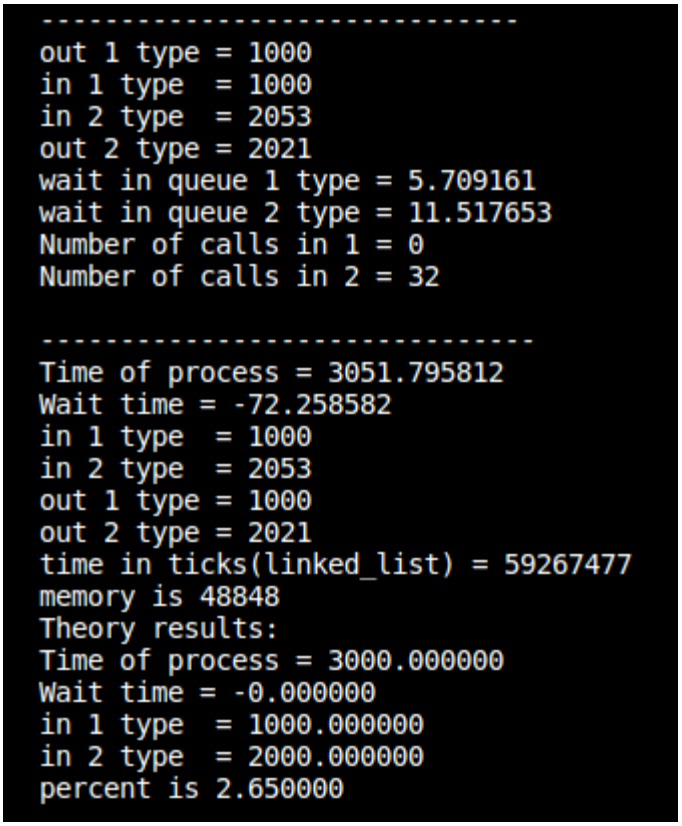


Работа программы:

(Очередь массивом)



(Очередь списком)



Фрагментация памяти:

Большая часть адресов используются повторно.

Анализ эффективности (по памяти и времени):

Выполнение модуляции Массив t, такты	Выполнение модуляции Список t, такты	%
984892	48229875	97

Память Массив t, такты	Память Список t, такты	%
8000	48000	60%
Память на 1 эл очереди Массив	Память на 1 эл очереди Список	%
8	16	50%

Размер списка сильно варьируется.

Выводы по проделанной работе

Была реализована программа, моделирующая ситуацию с 2мя очередями с чередующимся приоритетом.
Были реализованы функции добавления и удаления элементов для двух способов реализации очередей: списком и массивом.

Реализация Списком проигрывает по времени реализации массивом на 97%. По памяти на ~60%. Таким образом, реализация списком проигрывает реализации массивом. Но с массивом сложнее отслеживать конец очереди и необходимо задуматься о возможности переполнения очереди.

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.