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 ПО ЛАБОРАТОРНОЙ РАБОТЕ №5
«Обработка Очередей»

Студент Щербина Михаил Александрович

Группа ИУ7 – 35Б

Приняла Никульшина Т.А.

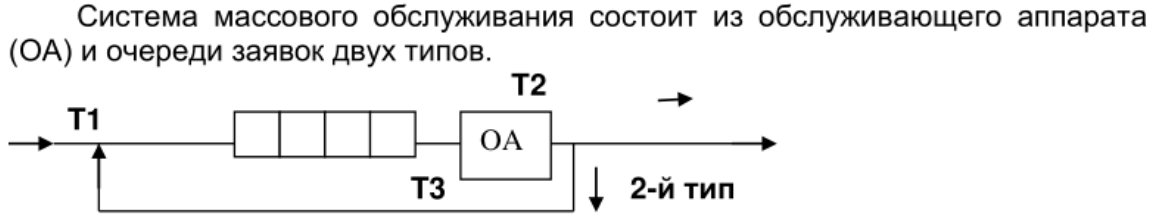
Описание условия задачи

Система массового обслуживания состоит из обслуживающих аппаратов (ОА) и очередей заявок двух типов, различающихся временем прихода и обработки. Заявки поступают в очереди по случайному закону с различными интервалами времени (в зависимости от варианта задания), равномерно распределенными от начального значения (иногда от нуля) до максимального количества единиц времени.В ОА заявки поступают из «головы» очереди по одной и обслуживаются за указанные в задании времена, распределенные равновероятно от минимального до максимального значений (все времена –вещественного типа).

Требуется смоделировать процесс обслуживания первых 1000 заявок первого типа, выдавая после обслуживания каждых 100 заявок первого типа информацию о текущей и средней длине каждой очереди и о среднем времени пребывания заявок каждого типа в очереди. В конце процесса необходимо выдать на экран общее время моделирования, время простоя ОА, количество вошедших в систему и вышедших из нее заявок первого и второго типов. Очередь необходимо представить в виде вектора и списка. Все операции должны быть оформлены подпрограммами. Алгоритм для реализации задачи один, независимо от формы представления очереди. Необходимо сравнить эффективность различного представления очереди по времени выполнения программы и по требуемой памяти. При реализации очереди списком нужно

проследить, каким образом происходит выделение и освобождение участков памяти, для чего по запросу пользователя необходимо выдать на экран адреса памяти, содержащие элементы очереди при добавлении или удалении очередного элемента.

Описание Технического задания



Заявки 1-го типа поступают в "хвост" очереди по случайному закону с интервалом времени T1, равномерно распределенным от 0 до 5 единиц
времени (е.в.). В ОА они поступают из "головы" очереди по одной и обслуживаются также равновероятно за время T2 от 0 до 4 е.в., после чего покидают систему. Единственная заявка 2-го типа постоянно обращается в системе, обслуживаясь в ОА равновероятно за время T3 от 0 до 4 е.в. и возвращаясь в очередь не далее 4-й позиции от "головы". В начале процесса заявка 2-го типа входит в ОА, оставляя пустую очередь (все времена – вещественного типа). Смоделировать процесс обслуживания первых 1000 заявок 1-го типа.

Входные данные:

1. Целое число, представляющее собой номер команды: целое число в диапазоне от 0 до 7
2. Времена прибытия / обработки заявок

Выходные данные:

1. Среднее время пребывания заявок в очереди. В конце процесса выдать общее время моделирования, время простоя аппарата, количество вошедших в систему и вышедших из нее заявок первого типа и количество обращений заявок второго типа.
2. Численная характеристика для очереди в виде списка/массива.

Функции программы:

1. Моделирование и характеристика для очереди в виде массива.
2. Моделирование и характеристика для очереди в виде массива.
3. Вывод адресов памяти удаленных элементов (для списка)
4. Изменения времен (прихода, обрабатывания)
5. Профилирование времени операций add/pop
6. Профилирование памяти
7. Профилирование времени симуляции
8. Выход из программы.

Обращение к программе:

Запускается через терминал.
Так же можно собрать программу используя cmake и make, и запустить ее (./lab_05).

Аварийные ситуации:

1. Некорректный ввод номера команды.
   На входе: число, большее чем 7 или меньшее, чем 0.
   На выходе: игнорирование ввода, вывод меню заново
2. Некорректный ввод номера команды.
   На входе: пустой ввод.
   На выходе: игнорирование ввода, вывод меню заново
3. Некорректный ввод времени

На входе: некорректный ввод

На выходе: игнорирование, прекращение ввода. Вывод сообщения об ошибке.

Описание Структур Данных

```
struct cons_t
{
	// those are stored together
	// preserving cache locality
	// | *next | *value |	value	|
	struct cons_t *next;
	void *value;
};
```

Нода связного списка. Владеет значением, значение хранится вместе с нодой.

```
typedef struct
{
	size_t el_size; // размер элемента
	size_t size; // кол-во элементов
	size_t capacity;
	cons_t *last;
	cons_t *first;
} queue_list_t;
```

Очередь в виде связного списка.

```
typedef struct
{
	size_t el_size; // размер элемента
```

```
        size_t capacity;
        size_t size;
        void *data; // used for malloc and free
        char *pin;
        char *pout;
} queue_vec_t;
```

Data - указатель на выделенную область памяти, используется для выделения / удаления памяти.

```
typedef struct
{
        union
        {
                queue_list_t list;
                queue_vec_t vec;
        } kind;
        int type;
} queue_t;
```

Queue_t обобщенный tagged union (тип-сумма).

Структуры для решения задания

```
typedef struct
{
        int type;
} task_t;
```

Описание Алгоритма Задания

1. Инициализировать очередь
2. Добавить в очередь заявку 2 типа
3. Время_работы := 0

4. Время_до_новой_заявки := 0
5. Время_до_конца_работы:= 0
6. Пока кол-во обработанных заявок 1 типа != 1000
    a. Если время_до_новой_заявки == 0
        i. Время_до_новой_заявки := случайное время прихода 1 заявки
        ii. Создать новую заявку
        iii. Если есть место в очереди,
            1. добавить её в очередь
            2. Кол-во обработанных заявок 1 типа += 1
    b. Если время_до_конца_работы == 0
        i. заявка := взять из очереди
        ii. Время_до_конца_работы := случайное время обрабатывания заявки данного типа
        iii. Время_работы += время_до_конца_работы
        iv. Если тип заявки первый
            1. Кол-во_обработанных_заявок_1_типа += 1
        v. Иначе
            1. очередь.вставить_не_дальше_4_места(заявка)
    c. Наименьшее_время = min(время_до_конца_работы, время_до_новой_заявки)
    d. Время_до_новой_заявки -= наименьшее_время
    e. Время_до_конца_работы -= наименьшее_время

Тесты

	Название	Ввод	Вывод
1	Некорректный ввод команды	1234	Игнорирование ввода, печать меню заново
2	Пустой ввод	Правда пустой	Игнорирование ввода, печать меню заново
3	Команда 1	0	Вывод результатов симуляции с использование м списка
4	Команда 2	1	Вывод результатов симуляции с использование м массива
5	Команда 3	Некорректный ввод промежутка времени	Прекращение ввода, вывод сообщения об ошибке, игнорирование ввода
6	Команда 3	Верный ввод	Обновление используемых времен
7	Команды 4, 5, 6, 7	Верный ввод команды	Отображение соответствующ ей информации
8			
9			

Расчет времени работы очереди

Теоретический расчет времени моделирования:
Время поступления заявки: 0-5 е.в.
Среднее время поступления = 2.5 е.в.
Вторая заявка постоянно крутится в автомате, добавляясь не дальше 4 позиции. Вероятность её выбрать (в стабилизировавшейся очереди, длиной больше 4) есть 1 / 4. Работать в данном случае автомат будет среднее время обработки заявок 1 типа. Аналогично для второго типа.
Имеем среднее время обработки = P(тип=1) * время обработки 1 + P(тип=2) * время обработки 2

(Ожидаемое) Кол-во заявок всех типов = (1 + P(тип=2)) * кол-во заявок 1 типа

Ожидаемое время обработки всех заявок = кол-во заявок всех типов * среднее время обработки

```
float prob_sampling_second = 1.0f / (float) (pos_from_front);
float prob_sampling_first = 1.0f - prob_sampling_second;
float mean_work_rate = prob_sampling_first * ts_mean(times.work_1) \
                     + prob_sampling_second * ts_mean(times.work_2);
float target_tasks_both_types = (float)target_tasks * (1 + prob_sampling_second);
float expected_time = mean_work_rate * (float) target_tasks_both_types;
float err = fabsf( x: work_time - expected_time) / expected_time * 100;
```

Время простоя всегда будет равно 0, т.к в очереди всегда будет крутится заявка 2 типа и она никогда не будет пустой.

Практические результаты:

[Service time 2479.725586, (expected: 2500.000000, err: 0.81 %)]

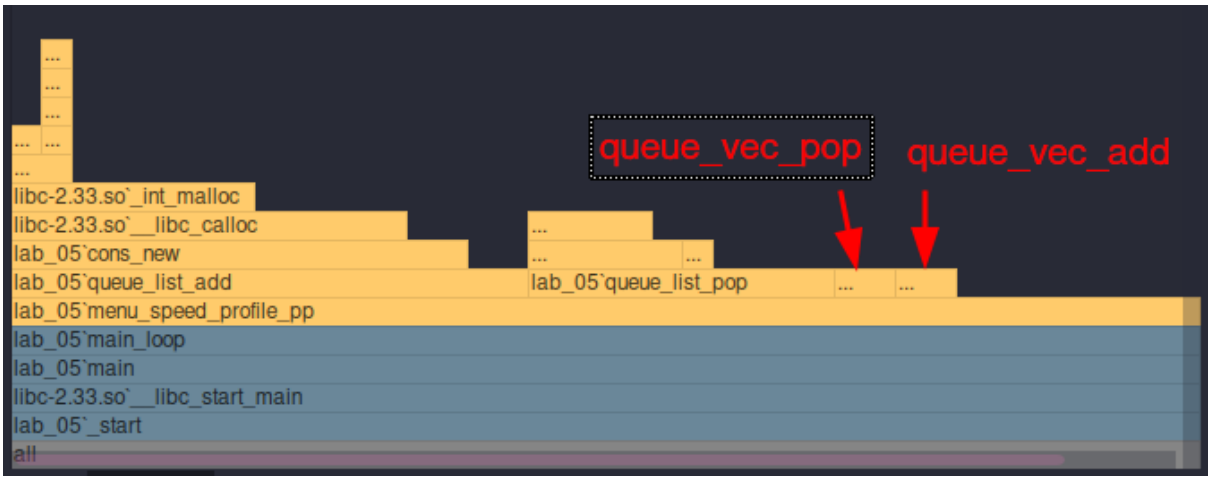
Tasks in: 1000
Tasks out: 1021
Tasks failed: 0
Calls: 1021
Stall time: 0.000

Видно, что погрешность в пределах допустимого задачей.

Оценка Эффективности

Скорость добавления / удаления элемента

```
Speed profile (push, pop):
|                |                          vec|                          list|
|           size|                add|            pop|                add|            pop|
|             10|                340|            350|               2662|            934|
|             50|               1496|           1598|              10957|           4151|
|            100|               2964|           3155|              19491|           8154|
|            500|              14728|          12815|              56825|          18409|
|           1000|              13379|          14273|              89779|          37043|
|           5000|              67598|          71322|             490999|         187245|
|          10000|             136257|         145291|             872081|         377444|
```



Скорость симуляции (для 1000 заявок 1 типа)

```
Running simulation...
List takes 502140 on average
Vector takes 265964 on average
```

Используемая память (в структуре храняться 32 битные int)

```
Memory profile (holding int):
Queue of 10000 elements
Vector queue size in bytes: 40048
Elements: 0, size list: 40, size of vector: 40048
Elements: 500, size list: 10040, size of vector: 40048
Elements: 1000, size list: 20040, size of vector: 40048
Elements: 1500, size list: 30040, size of vector: 40048
Elements: 2000, size list: 40040, size of vector: 40048
Broken at 2001, where list_size = 40060
Full 20.010000 %
```

Ответы на контрольные вопросы

1. Что такое FIFO и LIFO?

Это структуры данных, описывающие очередь и стек соответственно. Очередь – это последовательный список переменной длины, включение элементов в который идет с «хвоста», а исключение – с

«головы». Принцип работы очереди: первым пришел – первым вышел, т.е. First In – First Out (FIFO). Принцип работы стека: первым пришел, последним ушел.

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

При реализации списком, под каждый новый элемент выделяется память (при необходимости) размером sizeof(ноды) + размер элемента, которым владеет нода, для каждого элемента отдельно.
При реализации массивом, (кол-во элементов) * sizeof(элемента). Память выделяется сразу.

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

При удалении элемента из очереди в виде массива, перемещается указатель, память не освобождается. Память освобождается при удалении очереди, с помощью функции free().
При удалении элемента из очереди в виде списка, сразу освобождается память из данного элемента

4. Что происходит с элементами очереди при ее просмотре?

При просмотре очереди, голова удаляется, и указатель смещается. Таким образом при просмотре очереди ее элементы удаляются.

5. От чего зависит эффективность физической реализации очереди?

Какой размер очереди, постоянство размераочереди, важность времени добавления, критичность фрагментации памяти
Реализуя очередь в виде массива фрагментации не возникает. Быстрее работают операции добавления и удаления элементов. Также необходимо знать размер типа данных тип данных.

При реализации в виде списка — легче писать код для удаления и добавления элементов, однако может возникнуть фрагментация памяти.

Если изначально знать размер очереди то лучше воспользоваться массивом. Не зная размер — списком. Способ реализации зависит от того, в чем мы больше ограниченны, в памяти или во времени.

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

Если важна скорость выполнения, то лучше использовать массив, так как все операции с массивом выполняются быстрее, но очередь ограничена по памяти (так как массив статический).
Но если неизвестно количество элементов в очереди — то лучше использовать список. Список он ограничен только оперативной памятью, но может возникнуть фрагментация памяти.

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

Фрагментация – чередование участков памяти при последовательных запросах на выделение и освобождение памяти. «Занятые» участки чередуются со «свободными» - однако последние могут быть недостаточно большими для того, чтобы сохранить в них нужные данные. Фрагментация возникает в куче.

8. Для чего нужен алгоритм близнецов?

Алгоритм близнецов нужен, чтобы бороться с фрагментацией памяти.

9. Какие дисциплины выделения памяти вы знаете?

Дисциплины выделения памяти решают вопрос: какой из свободных участков должен быть выделен по запросу. По дисциплине "самый подходящий" выделяется тот свободный участок, размер которого равен запрошенному или превышает его на минимальную величину. По дисциплине "первый подходящий" выделяется первый же найденный свободный участок, размер которого не меньше запрошенного. Практически во всех случаях дисциплина "первый подходящий" эффективнее дисциплины "самый подходящий".

10. На что необходимо обратить внимание при тестировании программы?

Список: За памятью в ноде при добавлении / удалении. И за указателями в очереди (*first, *last).

При реализации очереди в виде массива (кольцевого) надо обратить внимание на правильность записи в выделенную область памяти (чтобы была не выше / ниже границ)

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

Malloc относится к системным вызовам. Выделяется виртуальная память.ОС находит свободный блок памяти и возвращает указатель на него программе.
При запросе на освобождение указанного блока программы, ОС убирает его из таблицы адресов, однако указатель на этот блок может остаться в программе. Обращение к этому адресу и попытка считать данные из этого блока может приводит в segfault.

Вывод

Когда мы создаем очередь в виде списка, память под все элементы выделяется не сразу, что эффективно (по памяти). Но память каждый раз выделяется динамически, что неэффективно по времени. При работе в очередями-списками может возникнуть фрагментация памяти (возникнет, с большой вероятностью).
К преимуществам списка можно отнести тот факт, что очередь-список может быть ограничена объемом оперативной памяти компьютера. А также операции удаления и добавления элемента в очередь легче реализовать, чем с очередью-массивом. При выполнении этих операций выполняется выделение или освобождение памяти, что стоит много времени и может привести к ошибке.
К недостаткам очереди в виде массива память выделяется сразу же большим объемом и память не убавляется. Очереди-массив более эффективнее по скорости чем очередь-список, его операции удаления и добавления элемента выполняются быстрее.