|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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 **ПРОГРАММНАЯ ИНЖЕНЕРИЯ (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04** Программная инженерия

**Отчет**

|  |  |
| --- | --- |
| **По лабораторной работе №** | 5 |

Моделирование очереди

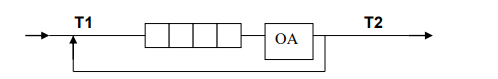
**Дисциплина:** Типы и структуры данных

|  |  |  |  |
| --- | --- | --- | --- |
| Студент | ИУ7-35Б |  | А. В. Толмачев |
|  | (Группа) |  | (И.О. Фамилия) |
|  |  |  |  |

Москва, 2022

**Описание условия 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 вновь поступает в "хвост" очереди, совершая всего 5 циклов обслуживания, после чего покидает систему (все времена – вещественного типа). В начале процесса в системе заявок нет.

Смоделировать процесс обслуживания до ухода из системы первых 1000 заявок, выдавая после обслуживания каждых 100 заявок информацию о текущей и средней длине очереди, а в конце процесса - общее время моделирования и количестве вошедших в систему и вышедших из нее заявок, количестве срабатываний ОА, время простоя аппарата. По требованию пользователя выдать на экран адресов элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

**Описание ТЗ**

1. **Описание исходных данных**

Исходными данными является интервал времени поступления заявок и интервал времени обработок заявок, пункты меню, данные заявки – при работе с отдельной очередью.

Ограничение исходных данных:

1. Ввод значений интервалов времени осуществляется через пробел, в одну строку.
2. Минимальное значение нижней границы интервала – 0
3. Не допустим ввод интервала [0;0]
4. Границы интервалов – целые или вещественные значения
5. Допустимо моделирование при совпадении верхней и нижней границ интервалов, если они отличны от 0
6. Ввод любых символов, кроме чисел в указанном формате, является некорректным.
7. Пункт меню задается целым числом. Возможен выбор одновременно только 1 пункта меню.
8. Ввод отсутствующего пункта меню является некорректным вводом.
9. Данными заявки является количество циклов в очереди и время до конца обработки.
10. Данные заявки вводятся через пробел и являются целыми числами.
11. Любой другой ввод данных заявки является некорректным.
12. **Описание задачи, реализуемой программой**

* Программа позволяет осуществить моделирование работы системы массового обслуживания. Возможен выбор типа используемой при моделировании очереди.
* Программа позволяет выбрать реализацию очереди – очередь на основе списка и очередь на основе массива.
* При работе с очередями доступны следующие операции:
  + Добавить задачу в очередь
  + Удалить задачу из очереди
  + Посмотреть содержимое очереди
  + Очистить очередь
  + Посмотреть список адресов

При работе с очередью-списком также возможен просмотр адресов.

* Программа позволяет просмотреть эффективность операций добавления и извлечения из очереди для реализаций очереди на основе массива и списка.

1. **Выходные данные**

* Сравнение эффективности использования очереди на основе списка и очереди на основе массива
* Содержимое очереди
* Список адресов элементов очереди и список адресов освобожденных элементов
* Значение удаляемого элемента очереди
* Результаты моделирования

1. **Способ обращения к программе**

Для запуска программы необходимо из консоли вызвать исполняемый файл ./app.exe.

1. **Описание возможных аварийных ситуаций и ошибок пользователя**

Аварийные ситуации:

1. Ошибка выделения памяти

Ошибки пользователя:

1. Ввод данных, не удовлетворяющих ограничениям исходных данных, описанных ранее
2. Попытка извлечь элемент из пустой очереди
3. Попытка посмотреть содержимое пустой очереди
4. Выбор несуществующего пункта меню

**Описание внутренних структур данных**

Очередь, реализованная на основе списка:

**struct** queue\_type\_list

{

**struct** node \*start;

**struct** node \*end;

**size\_t** length;

};

Узел списка:

**struct** node

{

**task\_t** \*task;

**struct** node \*next;

};

Список, реализованный на основе массива:

**struct** queue\_type\_array

{

**task\_t** \*\*content;

**size\_t** start;

**size\_t** end;

**size\_t** size;

};

Структура для хранения задачи для аппарата обслуживания:

**typedef** **struct** **task\_t**

{

**size\_t** count\_cycles\_complete;

**float** time\_until\_end\_process;

} **task\_t**;

Структура для хранения статистики моделирования:

**typedef** **struct** **stat\_t**

{

**size\_t** count\_in;

**size\_t** count\_out;

**size\_t** count\_app\_trigger;

} **stat\_t**;

Для хранения освобожденных адресов использовался массив

**Набор тестов**

1. Позитивные тесты
2. Ввод корректных данных для моделирования
3. Корректный выбор пункта меню
4. Извлечение элемента из непустой очереди
5. Добавление элемента в непустую очередь
6. Просмотр содержимого непустой очереди
7. Негативные тесты

Тесты с входными данными, не удовлетворяющими ограничениям исходных данных, записанным ранее.

**Временная и пространственная эффективность**

Время добавления элементов в очередь

|  |  |  |
| --- | --- | --- |
| Количество элементов | Очередь-массив, микросекунд | Очередь-список, микросекунд |
| 100 | 0.004600 | 0.020600 |
| 1000 | 3.232300 | 6.964000 |
| 10000 | 37.321400 | 82.683400 |
| 100000 | 381.720300 | 893.522100 |

Время извлечения элементов из очереди

|  |  |  |
| --- | --- | --- |
| Количество элементов | Очередь-м­ассив, микросекунд | Очередь-список, микросекунд |
| 100 | 0.003000 | 0.014100 |
| 1000 | 3.193600 | 6.173200 |
| 10000 | 39.162600 | 70.923100 |
| 100000 | 399.133200 | 759.398600 |

Время моделирования

|  |  |  |
| --- | --- | --- |
| Временные интервалы моделирования | Очередь-м­ассив, микросекунд | Очередь-список, микросекунд |
| 0 6 0 1 | 1953 | 2397 |
| 0 2 0 2 | 16810 | 19424 |
| 0 1 0 6 | 4482368 | 4429808 |
| 0 6 0 3 | 6198 | 8386 |

Затраты по памяти

|  |  |  |
| --- | --- | --- |
| Количество элементов | Очередь-м­ассив, байт | Очередь-список, байт |
| 100 | 2912 | 3208 |
| 1000 | 26272 | 32008 |
| 10000 | 241952 | 320008 |
|  |  |  |

**Вывод**

При реализации программы память под приходящие заявки выделялась динамически. Поэтому при реализации очереди на основе массива терялась возможная эффективность от выделения единого блока памяти, т. к. под каждую задачу все равно выделялся маленький отдельный блок.

Это явно видно из результатов - моделирование на очереди-массиве практически не выигрывает по времени. При этом также выделяется цельный огромный блок памяти, т. е. теряются все преимущества.

Выбор структуры данных для физической реализации очереди зависит от требований к программе.

Если заранее неизвестен размер очереди, а также известно, что размер очереди может изменяться в широком диапазоне, то реализации очереди на основе массива будет менее эффективна. В этом случае будут лишние затраты по памяти – так как память из под очереди освобождается только при завершении работы программы, а выделяется под массив единым блоком, то может выйти ситуация, когда выделен огромный блок памяти, а вы очереди всего несколько элементов. Также динамическое расширение массива, при заполнении очереди, занимает дополнительно время – нужно найти новый блок памяти походящего размера, перенести туда данные, после чего очистить старую область.

Соответственно, при неизвестном размере очереди будет более эффективным реализация очереди на основе списка, т.к. память выделяется и освобождается для каждого элемента отдельно. Однако в этом случае постоянное выделение и освобождение небольших блоков памяти требует дополнительного времени, и может приводить к фрагментации памяти.

Также требуется хранить указатель на следующий элемент => дополнительные затраты по памяти.

На моей машине при реализации очереди на основе списка фрагментации не возникло.

Эффективность реализации FIFO и LIFO в л/р 4 и 5 примерно одинакова. Добавление/удаление элемента в стек занимает такое же время, как добавление/удаление элемента в очередь. Затраты по памяти в случае FIFO чуть больше, т.к. нужно также хранить указатель на конец очереди.

**Ответы на контрольные вопросы**

1. Что такое FIFO и LIFO?

FIFO – first in first out – очередь, первым извлекается элемент, который первым вошел

LIFO – last in first out – стек, первым извлекается элемент, который последним вошел

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

При реализации очереди на основе массива память выделяется единым блоком под некоторое, заданное изначально, число элементов. При необходимости этот блок памяти расширяется. Объем памяти: sizeof(element) \* start\_count\_elemnts;

При реализации очереди на основе списка память выделяется под каждый элемент очереди по отдельности, и очищается при извлечении элемента из очереди.

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

При реализации очереди на основе массива память освобождается только при окончании работы программы, что влечет излишние расходы по памяти, если в очереди количество элементов может сильно меняться в пределах большого диапазона, и если это число не известно заранее.

При реализации очереди на основе списка память выделяется под каждый элемент очереди по отдельности, и очищается при извлечении элемента из очереди.

1. Что происходит с элементами очереди при ее просмотре?

При просмотре элемент очереди извлекается из очереди.

1. От чего зависит эффективность физической реализации очереди?

Эффективность зависит от поставленной задаче.

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

Реализация очереди на основе массива эффективна, если заранее известно примерное число элементов в очереди, и так же это число остается примерно постоянным при выполнении программы.

В этом случае память выделится единым блоком, не будет расширяться и будет почти полностью использоваться.

Если же число элементов не известно заранее, придется динамически расширять выделенную область памяти, что повлечет излишние затраты по времени – в худшем случае нужно найти свободную область памяти, скопировать туда элементы из старой области памяти, и очистить старую область памяти.

Также если число элементов изменяется в довольно широком диапазоне, например, от 1e1 до 1e5, то большую часть времени будет использоваться меньшая часть выделенной памяти, что неэффективно.

При использовании очереди на основе списка требуется дополнительное время для выделения памяти под каждый элемент и очистку этой памяти. Выделить малый блок памяти будет легче, чем найти память под огромный блок, однако постоянно выделения малых блоков памяти может привести к фрагментации памяти. Также необходимо хранить указатель на следующий элемент, что влечет дополнительные затраты по памяти.

Однако если число элементов не известно заранее, и размеры очереди могут изменяться в довольно широком диапазоне, такая реализация будет более эффективной.

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

Фрагментация памяти – дробление памяти на мелкие и несмежные свободные области маленького размера.

1. Для чего нужен алгоритм «близнецов».

Метод близнецов – схема выделения памяти — позволяет быстрее найти свободный блок памяти и уменьшить фрагментацию.

9. Какие дисциплины выделения памяти вы знаете?

Дисциплина самый подходящий — в этом случае выделяется наиболее подходящий участок памяти — равный требуемому или минимально превышающий его.

Дициплина первый подходящий — при таком подходе выделяется первый

участок памяти, равный или больше запрашиваемому по объему.

Алгоритм близнецов.

10. На что необходимо обратить внимание при тестировании программы?

1) Проверить правильность работы программы при различном заполнении очередей

2) Проверить обработку некорректного ввода пользователя

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

При динамическом запросе памяти операционная система находит подходящий блок, используя определенную дисциплину выделения памяти, после чего помечает блок занятым. После освобождения памяти в добавляет блок в массив свободных адресов.