**Министерство образования и науки**

**Российской Федерации**

**Федеральное государственное бюджетное образовательное**

**учреждение высшего профессионального образования**

**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)**

Кафедра автоматизированных систем управления(АСУ)

РЕАЛИЗАЦИЯ МНОГОПОТОЧНОЙ ОБРАБОТКИ ДАННЫХ

**Отчет по лабораторной работе №2**

по дисциплине «Теория вычислительных процессов»

Вариант № 17

Выполнил:

студент ФДО ТУСУР

специальность: 230105

группа з-439-f

Туданов Станислав Александрович

Проверил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

Томск— 2016

Оглавление

[1. Задание 2](#_Toc453339162)

[2. Анализ задачи 4](#_Toc453339163)

[3. Чтение параметров 5](#_Toc453339164)

[4. Структуры данных в проекте 6](#_Toc453339165)

[5. Линейный анализ данных 7](#_Toc453339166)

[6. Выбор следующего элемента для анализа 8](#_Toc453339167)

[7. Простой многопоточный алгоритм 9](#_Toc453339168)

[8. Сеть Петри 10](#_Toc453339169)

[9. Многопоточная обработка с помощью сети Петри 12](#_Toc453339170)

[10. Реализация сети Петри 13](#_Toc453339171)

[11. Многопоточная обработка с помощью системного пула потоков 14](#_Toc453339172)

[12. Многопоточная обработка сетью Петри и пулом потоков 15](#_Toc453339173)

[13. Пример работы программы 16](#_Toc453339174)

[Выводы 17](#_Toc453339175)

[Список литературы 18](#_Toc453339176)

[Приложение А (обязательное Исходный код 19](#_Toc453339177)

# Задание

Цель выполнения лабораторной работы № 2 — освоить реализацию многопоточной обработки данных, а также пула потоков и механизма асинхронного ввода/вывода.

В работе необходимо реализовать многопоточную обработку массива структур данных (из N элементов) четырьмя способами:

1. При помощи массива из M потоков (M ≤ N), используя для синхронизации объект ядра — семафор.
2. При помощи массива из M потоков (M ≤ N), используя для синхронизации сеть Петри, моделирующую семафор.
3. При помощи пула из M потоков (M ≤ N), используя системный пул потоков или асинхронные потоки ввода/вывода.
4. При помощи пула из M потоков (M ≤ N), моделируя его при помощи сети Петри.

При обработке массива данных массивом потоков каждый поток либо заранее выбирает диапазон элементов массива данных, которые он будет обрабатывать, либо просто берет первый необработанный элемент. Завершив обработку одного элемента, поток переходит к обработке следующего.

При обработке массива данных пулом потоков, завершив обработку одного элемента массива данных, освободившийся в пуле поток переходит к обработке следующего необработанного элемента.

Чтобы не требовалось создавать слишком большие массивы (для которых эффект от параллельной обработки будет наиболее очевидным), можно имитировать ситуацию, когда обработка одного элемента массива требует больше процессорного времени, чем в действительности. Для этого после обработки очередного элемента массива поток может делать паузу на указанное количество миллисекунд.

Входной файл должен иметь имя «input.txt» или «input.xml».

Формат входного файла для лабораторной работы

|  |  |
| --- | --- |
| Поле | Значение |
| PA | Выбранный способ обработки массива |
| N | Размер массива структур данных (значения полей каждой структуры генерируются программой случайным образом) |
| M | Количество параллельных потоков (если 0, то принимается равным числу процессорных ядер в системе) |
| PT | Пауза после обработки каждого элемента массива (мс) |
| … | Дополнительные входные данные (зависят от варианта) |

Выходной файл должен иметь имя «output.txt». Формат вы- ходного файла представлен в таблице

|  |  |
| --- | --- |
| Поле | Значение |
| T1 | Время, требующееся на линейную обработку массива (без распараллеливания вычислений) |
| TP | Время, требующееся на параллельную обработку массива выбранным способом |
| … | Результаты обработки (зависят от варианта) |

Для ввода и вывода данных допускается использование в программе визуального интерфейса вместо файлового ввода/вывода.

Вариант задания

Структура содержит сведения о вещах в багаже пассажиров (ФИО пассажира, название вещи, количество единиц, вес одной единицы). Требуется найти общий вес вещей с названием X. В выходной файл необходимо вывести список пассажиров, имеющих в багаже вещь X, а также найденный вес.

# Анализ задачи

В данной задаче есть несколько частей, которые можно реализовывать по очереди.

1. Чтение параметров.
2. Выделение памяти и создание данных.
3. Расчет алгоритма при простом линейном способе обхода данных.
   1. Простой многопоточный алгоритм, синхронизация с помощью семафора
   2. Многопоточный, с синхронизацией с помощью сети Петри
   3. Системный пул потоков
   4. Системный пул потоков с синхронизацией с помощью сети Петри.
4. Выбор одного из 4-х способов многопоточной обработки данных.
5. Итоговый анализ данных, полученных в п.3
6. Вывод итога.

Рассмотрим каждый из этих пунктов.

# Чтение параметров

В данном пункте просто открывается файл и из него читаются числа-параметры. Все это реализуется в функции

void read\_params()

# Структуры данных в проекте

Для начала зададим структуру данных:

typedef struct tagBagaj

{

char \*fio, \*name;

int weight, cnt;

} TBagaj;

Поля в структуре обозначают следующее:

* fio – Фамилия пассажира
* name – название груза
* cnt - количество
* weight - вес

Опишем переменные для данных в памяти:

// Данные

TBagaj \*data;

Выделим память и случайным образом заполним поля данных для дальнейшего анализа.

void create\_random\_bagaj()

Напишем процедуру обработки одного элемента. Данные обработки хранятся в глобальном поле, к которому будет реализован экслюзивный доступ с помощью семафоров или сети Петри

Для обработки одного элемента данных используется следующая процедура

// Обрабатываем данные

void Obrabotat( TBagaj \*d )

Также она в ходе работы обеспечивает задержку для более реалистичной демонстрации пользы многозадачности.

# Линейный анализ данных

Для сравнительного анализа данные в начале обсчитываются с помощью линейного анализа. Это происходит следующим образом – в цикле в процедуру обработки передаются все данные.

В конце работы заполняется поле l\_time – время линейной обработки.

Также во всех алгоритмах многопоточной обработки аналогичным образом засекается время и заполняется поле t\_time

# Выбор следующего элемента для анализа

Все алгоритмы выбирают следующий элемент данных для работки с помощью функции get\_last.

Эта функция не потокобезопасна, вызывающая функция сама должна обеспечить эксклюзивный доступ к данным.

# Простой многопоточный алгоритм

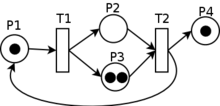
В данном варианте работы создается нужное количество цепочек, каждая из которых выбирает элементы, пока они есть. После завершения свободных элементов каждая цепочка просто завершается.

Для блокировки используется семафор, внутри выбирается следующий свободный элемент, после чего семафор освобождается.

Функция создания цепочек просто создает нужное количество цепочек, после чего ждет завершения всех цепочек. После завершения происходит суммирование результатов через функцию агрегации и вывод результата.

# Сеть Петри

Сети Петри — математический аппарат для моделирования динамических дискретных систем. Впервые описаны Карлом Петри в 1962 году.

Сеть Петри представляет собой двудольный ориентированный мультиграф, состоящий из вершин двух типов — позиций и переходов, соединённых между собой дугами. Вершины одного типа не могут быть соединены непосредственно. В позициях могут размещаться метки (маркеры), способные перемещаться по сети.

Событием называют срабатывание перехода, при котором метки из входных позиций этого перехода перемещаются в выходные позиции. События происходят мгновенно, либо разновременно, при выполнении некоторых условий.

Схема сети Петри

Для реализации семафора секции для второй части задания создана следующая схема:

Она работает следующим образом – при работе цепочка пытается заблокировать критическую секцию и перейти из левого узла в правый. Это получается одновременно только у одной.

При завершении работы цепочка совершает переход сверху вниз (фишка тут-же возвращается) и в предпоследнем узле накапливаются фишки. После запука всех цепочек основная цепочка пытается запустить конечный переход, но это получается только при окончании работы всех остальных цепочек.

Вторая сеть Петри для четвертой части работает следующим образом:

Каждая цепочка пытается забрать фишку у первой, но получается только у M из них. После работы фишка возвращается и наполняет как первый узел, так и конечный. Начальная цепочка ждет, пока все узлы не будут обработаны и только потом совершается конечный переход и она завершает работу

Узлы цепочек

# Многопоточная обработка с помощью сети Петри

Основная проблема многопоточной обработки – блокировка доступа к критическим секциям. Для этого в данной части работы используется сеть Петри со следующим графом:

1. Узел с номером p\_lock\_node – общий ущел блокировки. Вначале содержит фишку для блокировки.
2. Узлы p\_priv\_nodes – несколько узлов, каждый для отдельной цепочки. При работе пытается получить фишку из первого пункта, таким образом блокируя доступ.
3. Узлы p\_end\_nodes – несколько узлов по одной для каждой цепочки. В конце работы цепочки передает фишку в узел p\_end\_node.
4. Узел p\_end\_node – конечный узел. Когда в нем накопится достаточное количество фишек из узлов p\_end\_nodes, происходит оконечный переход и завершение основной цепочки

В программе строится граф Петри таким образом, что узлы p\_lock\_node и p\_end\_node общие для всех цепочкиек и каждая из них имеет свою часть. При попытке выбора следующих данных фишка уходит к одну из цепочек и не может быть получена другими до тех пор, пока эта цепочка не вернет ее. Таким образом реализуется исключающая блокировка симулирующая мьютекс из предыдущего примера.

# Реализация сети Петри

Сеть Петри реализована в программе с помощью массива

int tokens[20]; // узлы

Каждый элемент которого описывает количество фишек в узле сети.

Также есть структура

typedef struct tagPLink

{

int from, to, n;

void set( int \_from, int \_to, int \_n = 1 )

{

from = \_from;

to = \_to;

n = \_n;

}

} TPLink;

Которая описывает отдельную связь в сети. Для описания всех связей создан массив:

TPetryLink links[30];

В нем присутствуют следующие связи:

3 связи для каждой из N цепочек. Первая и вторая связь описывает кольцо до нулевого узла. Третья связь связывает «лишнюю» фишку и конечный узел.

В конце есть отдельная связь кратность N, которая вызывается из центральной цепочки после запуска всех остальных цепочек. Она передвигается только в одном случае – когда все N цепочек положат свои фишки в конечный узел. После передвижения программа завершается.

Для перемещения фишек по графу создана функция

void move\_petry( int link )

Получающая на вход номер связи и возвращающая управлению после того, как перемещение фишек успешно завершено.

# Многопоточная обработка с помощью системного пула потоков

В данном режиме основная цепочка потока работает в обычном режиме, а в моменты ее засыпания (Sleep или иное ожидание) цепочка может обрабатывать данные в дополнительной функции. Для этого используется функция QueueUserAPC, которая запускает асинхронную функцию в пределах другой цепочки.

В программе запускается нужное количество цепочек, каждая из которых просто спит, пока цепочке не будет требоваться завершение. После запуска всех цепочек в каждую из них запускается асинхронная функция, которая и делает в задаче основную работу. По завершению всех функций вызывается завершение всех цепочек и они прекращают работу.

# Многопоточная обработка сетью Петри и пулом потоков

Это комбинация двух предыдущих способов – потоки выполнения находятся в пользовательских асинхронных функциях (основная функция потока спит), а логика синхронизации находится в сети Петри, а не в семафоре при вызове функции get\_last.

Если входной параметр m равен 0, то по условию задачи необходимо определить количество ядер процессора и записать это число в параметр количества цепочек. Для этого используется функция:

void check\_proc\_count()

{

if (m == 0)

{

SYSTEM\_INFO sysinfo;

GetSystemInfo( &sysinfo );

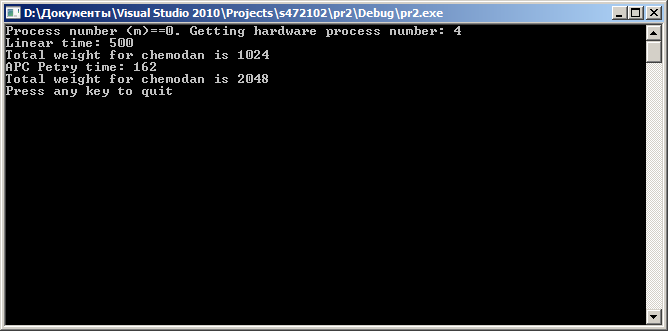
m = sysinfo.dwNumberOfProcessors;

printf( "Process number (m)==0. Getting hardware process number: %d\n", m );

}

}

# Пример работы программы



# Выводы

В данной работе мы научились писать многопоточные приложения, которые совместно используют данные, а для нормальной работы используют блокировки. Также были изучены сети Петри и способы замены семафоров на вышеуказанные сети.

# Список литературы

1. Теория сетей Петри и моделирование систем , Питерсон Дж.
2. C++ для профессионалов, Николас А. Солтер, Скотт Дж. Клепер
3. Освой самостоятельно C++ за 21 день, Либерти Д.

# *Приложение А (обязательное)* *Исходный код*