Учреждение образования

Белорусский государственный университет

информатики и радиоэлектроники

Кафедра электронных вычислительных машин

Лабораторная работа №5

**ДЕТЕРМИНИРОВАННЫЕ МЕТОДЫ ДОСТУПА К МОНОКАНАЛУ**

Выполнил: Проверил:

ст. гр. №950503 Марцинкевич В.А.

Гуринович А.В.

Минск 2021

**1**. **Моноканал.**

Концептуальная разница между случайными и детерминированными методами заключается в том, возникает ли случайность при «обращении» станции к моноканалу.

Кольцо можно рассматривать как своеобразный моноканал, один такт работы которого представляет собой полный либо частичный «обход» кадром всех станций. Более подробно типичную кольцевую топологию можно представить следующим образом.



Рисунок 1 - Цепи передатчика и приемника типичной кольцевой топологии

Применительно к приведенной топологии, при доступе к моноканалу никаких проблем казалось бы возникать не должно.

Действительно, физические коллизии для такой схемы невозможны, но проявляется то, что можно назвать особым видом логических коллизий. Если при некотором такте кольца какая-либо из станций имеет собственный кадр для передачи и при этом получила из кольца еще один кадр, который необходимо «продвигать» дальше, то появляется вопрос о том, какой из этих кадров передавать.

Частично противоречие может быть разрешено буферизацией кадров. Но возлагать на обычную пользовательскую станцию функции полноценного сетевого моста канального уровня крайне нецелесообразно. Кроме того, буферизация позволяет только «удерживать», то есть не терять кадры. Сугубо алгоритмический вопрос о том, какой же из кадров (имеющийся кадр для трансляции либо принятый кадр для ретрансляции) передавать раньше все-таки остается без ответа.

Единственным способом преодоления логических коллизий является введение приоритетов (priorities). В то время как все случайные методы «завязаны» на генератор случайных задержек, все детерминированные методы «завязаны» на систему приоритетов в том или ином виде. Возникает задача распределенного либо централизованного назначения приоритетов, причем ни одна из станций кольца заранее ничего «не знает» о других станциях.

При использовании механизма приоритетов не обойтись без так или иначе выраженного арбитра.

В качестве арбитра может выступать специальный служебный кадр, который в русскоязычной литературе обычно называют *маркером* (token).

Таким образом, основные критерии классификации детерминированных методов:

-- централизованное либо распределенное управление;

-- алгоритм назначения приоритетов;

-- топологические особенности.

На эффективность детерминированных методов наиболее существенное влияние оказывают те же факторы, что и в ситуациях со случайными методами:

-- количество взаимодействующих станций;

-- частота синхронизации;

-- длина кадра.

Если оценивать реализации, которые уже имеются на рынке, то все же детерминированные алгоритмы в среднем демонстрируют бо'льшую производительность. Однако оборудование в среднем более дорогостоящее.

**2**. **Token Ring.**

С точки зрения изучения детерминированных методов доступа к моноканалу наиболее наглядным примером является классический алгоритм, описанный в стандарте Token Ring (IEEE 802.5).

В Token Ring применяется централизованное управление. Закономерным следствием является необходимость включения в кольцо по крайней мере одной управляющей станции, наделенной особыми полномочиями и призванной инициализировать кольцо и следить за его работоспособностью. В терминологии Token Ring такую управляющую станцию обобщенно называют станцией-монитором (monitor station). Кроме единственной основной станции-монитора (active monitor) в состав кольца может входить некоторое количество резервных (standby monitors).

Функции станции-монитора:

1. Инициализировать подключившиеся к кольцу станции.

2. Тактировать работу кольца.

3. Контролировать наличие и валидность маркера.

4. Предотвращать зацикливания.

В отличие от сегмента Ethernet, где все станции равноправны и действуют по одному и тому же алгоритму, в сегменте Token Ring предусмотрены станции несколько видов. Наряду с выделяемыми на канальном уровне станциями-мониторами, на более высоких уровнях рекомендуется выделять следующие станции:

1. System managers -- системные менеджеры (на них сосредоточены управляющие системой на основе Token Ring процессы).

2. Servers -- различные серверы (configuration report servers, ring error monitors, ring parameter servers).

3. Data stations -- информационные станции (обычные пользовательские станции).

Функциональное наполнение перечисленных видов станций выходит за рамки стандарта.

Не смотря на то, что теоретически кольцо предполагает некоторую возможность «распараллеливания» (то есть, одновременно по разным частям кольца могут циркулировать несколько кадров), очень обобщенно алгоритм Token Ring можно представить как «бесконечно» циркулирующий под управлением станции-монитора маркер, который анализируется всеми пользовательскими станциями и к которому при необходимости «цепляются» данные.

Для того чтобы понять заложенный в стандарт алгоритм, сначала необходимо рассмотреть форматы кадров Token Ring и назначение основных полей.

В стандарте предусмотрены четыре вида передаваемых последовательностей:

1. Token -- маркер.

2. Frame -- кадр.

3. Abort Sequence -- прерывающая последовательность.

4. Fill -- заполняющая последовательность.

Каждая из станций в любое время должна распознавать (и различать) маркеры, кадры и специальные последовательности.



Рисунок 2 - Формат маркера Token Ring



Рисунок 3 - Формат кадра Token Ring



Рисунок 4 - Формат прерывающей последовательности Token Ring

Поля:

1. SD (Starting Delimiter) -- начальный разделитель.

2. AC (Access Control) -- контроль доступа.

3. FC (Frame Control) -- контроль кадра.

4. DA (Destination Address) -- адрес назначения.

5. SA (Source Address) -- адрес источника.

6. RI (Routing Information) -- информация о маршрутизации (может отсутствовать).

7. INFO (Information) -- данные (могут отсутствовать).

8. FCS (Frame Check Sequence) -- контрольная сумма.

9. ED (Ending Delimiter) -- конечный разделитель.

10. FS (Frame Status) -- состояние кадра.

11. IFG (InterFrame Gap) -- межкадровый интервал.

SD и ED фактически являются флагами начала и конца кадра.

Между IFG и SD передается заполняющая последовательность.

С точки зрения алгоритма контроля доступа наибольший интерес представляет одноименное поле, а также поле состояния кадра.



Рисунок 5 - Формат поля контроля доступа

Где:

1. P (Priority bits) -- текущий уровень приоритета.

2. T (Token bit) -- идентификатор маркера: 0 -- маркер, 1 -- кадр.

3. M (Monitor bit) -- бит монитора.

4. R (Reservation bits) -- запрашиваемый уровень приоритета.



Рисунок 6 - Формат поля состояния кадра

Где:

1. A (Address-recognized bit) -- флаг распознания адреса (дублируется).

2. C (frame-Copied bit) -- флаг копирования кадра (дублируется).

3. r (reserved) -- зарезервировано.

Код программы:

**3. Marking**

class Marking {

private:

Marker value;

public:

constexpr Marking() : value() {}

void start();

void set\_source\_adress(const std::string &adress);

std::string get\_source\_adress() const;

void set\_destination\_address(const std::string &adress);

std::string get\_destination\_adress() const;

void set\_priority(const unsigned int &priority);

unsigned int get\_priority() const;

void set\_token(const unsigned int &token);

unsigned int get\_token() const;

void set\_reservation(const unsigned int &reserv);

unsigned int get\_reservation() const;

void set\_frame\_status(const unsigned int &frame);

unsigned int get\_frame\_status() const;

void set\_id(const unsigned int &id) noexcept;

unsigned int get\_id() const noexcept;

void set\_start\_delimiter(const unsigned int &delimiter);

unsigned int get\_start\_delimiter() const;

void set\_ending\_delimiter(const unsigned int &delimiter);

unsigned int get\_ending\_delimiter() const;

void set\_frame\_check\_sequence(const unsigned int &size);

size\_t get\_frame\_check\_sequence() const;

void set\_inter\_frame\_gap();

void set\_regime(const unsigned int &regime);

unsigned int get\_regime() const;

void set\_data(const std::string &data) {

memset(this->value.Data, '\0', sizeof(this->value.Data));

memcpy(this->value.Data, data.c\_str(), data.size());

this->value.FrameCheckSequence = data.size();

}

const char\* get\_data() const noexcept;

private:

template<class Archive>

void save(Archive &ar, const unsigned int version) const {

ar & this->value;

}

template<class Archive>

void load(Archive &ar, const unsigned int version) {

ar & this->value;

}

BOOST\_SERIALIZATION\_SPLIT\_MEMBER()

friend class boost::serialization::access;

friend std::ostream& operator<<(std::ostream &out, const Marking &M) {

out << M.value;

return out;

}};

**4. Marker**

const size\_t ADRESS\_SIZE = 13;

const size\_t MAX\_SIZE\_MSG\_DATA = 150;

struct Marker {

unsigned int StartingDelimiter;

Access AccessControl;

char DestinationAddress[ADRESS\_SIZE];

char SourceAddress[ADRESS\_SIZE];

char Data[MAX\_SIZE\_MSG\_DATA];

size\_t FrameCheckSequence;

unsigned int EndingDelimiter;

FrameState FrameStatus;

unsigned int InterFrameGap;

unsigned int Id;

unsigned int Regime;

template<class Archive>

void save(Archive &ar, const unsigned int version) const {

ar & this->StartingDelimiter;

ar & this->AccessControl;

ar & this->DestinationAddress;

ar & this->SourceAddress;

ar & this->Data;

ar & this->FrameCheckSequence;

ar & this->InterFrameGap;

ar & this->Id;

ar & this->Regime;

}

template<class Archive>

void load(Archive &ar, const unsigned int version) {

ar & this->StartingDelimiter;

ar & this->AccessControl;

ar & this->DestinationAddress;

ar & this->SourceAddress;

ar & this->Data;

ar & this->FrameCheckSequence;

ar & this->InterFrameGap;

ar & this->Id;

ar & this->Regime;

}

void set\_frame\_status(const unsigned int &flag) {

this->FrameStatus.set\_frame\_copied(flag);

}

unsigned int get\_frame\_status() const {

return this->FrameStatus.get\_frame\_copied();

}

void set\_priority\_bit(const unsigned int &priority) {

this->AccessControl.set\_priority\_bit(priority);

}

void set\_token\_bit(const unsigned int &token) {

this->AccessControl.set\_token\_bit(token);

}

void set\_reservation\_bit(const unsigned int &reserv) {

this->AccessControl.set\_reservation\_bit(reserv);

}

unsigned int get\_priority\_bit() const {

return this->AccessControl.get\_priority\_bit();

}

unsigned int get\_token\_bit() const {

return this->AccessControl.get\_token\_bit();

}

unsigned int get\_reservation\_bit() const {

return this->AccessControl.get\_reservation\_bit();

}

BOOST\_SERIALIZATION\_SPLIT\_MEMBER()

friend class boost::serialization::access;

friend std::ostream& operator<<(std::ostream &out, const struct Marker &M) {

out << "\n\t---Marker---\n";

out << "Starting Delimiter: " << M.StartingDelimiter << "\n";

out << "\tAccess Control\n" << M.AccessControl << "\t----------\n";

out << "Destination Address: " << M.DestinationAddress << "\n";

out << "Source Address: " << M.SourceAddress << "\n";

out << "Data: " << M.Data << "\n";

out << "Frame Check Sequence: " << M.FrameCheckSequence << "\n";

out << "Ending Delimiter: " << M.EndingDelimiter << "\n";

out << "\tFrame Status\n" << M.FrameStatus << "\t----------\n";

out << "Inter Frame Gap: " << M.InterFrameGap << "\n";

out << "\t---END MARKER---\n";

return out;

}};

**5. Station Monitor**

class StationMonitor : public Pseudoterminal {

private:

unsigned int id;

unsigned int priority;

unsigned int regime;

public:

StationMonitor();

~StationMonitor();

bool start(const std::string &adress);

void stop();

void send\_marker();

void set\_priority\_mode(const unsigned int &priority);

void set\_early\_release\_regime(const unsigned int &flag);

bool get\_priority\_mode();

bool get\_early\_release\_regime();

private:

void create\_marker();

void check\_marker();

void check\_early\_release\_regime();

void thread\_start() override;

void thread\_read() override;

void send\_status(Status &S);

};

**6. Thread Control**

#define THREAD\_RUN 1

#define THREAD\_STOP 2

#define THREAD\_WAIT 3

#define THREAD\_SET\_FLAG(mut, condition, flag, status) { \

std::lock\_guard<std::mutex> lq(mut); \

flag = status; \

condition.notify\_one(); \

}

**7. Access**

#define TOKEN\_BIT\_MARKER 0

#define TOKEN\_BIT\_FRAME 1

#define PRIORITY\_BITS\_OFF 0

struct Access {

unsigned short PriorityBits;

unsigned short TokenBit;

unsigned short MonitorBit;

unsigned short ReservationBits;

template<class Archive>

void save(Archive &ar, const unsigned int version) const {

ar & this->PriorityBits;

ar & this->TokenBit;

ar & this->MonitorBit;

ar & this->ReservationBits;

}

template<class Archive>

void load(Archive &ar, const unsigned int version) {

ar & this->PriorityBits;

ar & this->TokenBit;

ar & this->MonitorBit;

ar & this->ReservationBits;

}

void set\_priority\_bit(const unsigned int &priority) {

this->PriorityBits = priority;

}

void set\_token\_bit(const unsigned int &token) {

this->TokenBit = token;

}

void set\_reservation\_bit(const unsigned int &reserv) {

this->ReservationBits = reserv;

}

unsigned int get\_priority\_bit() const {

return this->PriorityBits;

}

unsigned int get\_token\_bit() const {

return this->TokenBit;

}

unsigned int get\_reservation\_bit() const {

return this->ReservationBits;

}

BOOST\_SERIALIZATION\_SPLIT\_MEMBER()

friend class boost::serialization::access;

friend std::ostream& operator<<(std::ostream &out, const struct Access &A) {

out << "Priority Bits: " << A.PriorityBits << "\n";

out << "Token Bit: " << A.TokenBit << "\n";

out << "Monitor Bit: " << A.MonitorBit << "\n";

out << "Reservation Bits: " << A.ReservationBits << "\n";

return out;

}

};

**8. Frame State**

#define FRAME\_COPIED\_OFF 0

#define FRAME\_COPIED\_ON 1

struct FrameState {

unsigned short AddressRecog;

unsigned short FrameCopied;

unsigned short Reserved;

template<class Archive>

void save(Archive &ar, const unsigned int version) const {

ar & this->AddressRecog;

ar & this->FrameCopied;

ar & this->Reserved;

}

template<class Archive>

void load(Archive &ar, const unsigned int version) {

ar & this->AddressRecog;

ar & this->FrameCopied;

ar & this->Reserved;

}

void set\_frame\_copied(const unsigned int &flag) {

this->FrameCopied = flag;

}

unsigned int get\_frame\_copied() const {

return this->FrameCopied;

}

BOOST\_SERIALIZATION\_SPLIT\_MEMBER()

friend class boost::serialization::access;

friend std::ostream& operator<<(std::ostream &out, const struct FrameState &FS) {

out << "Address Recog: " << FS.AddressRecog << "\n";

out << "Frame Copied: " << FS.FrameCopied << "\n";

out << "Reserved: " << FS.Reserved << "\n";

return out;

}

};