БГУИР

Кафедра ЭВМ

Лабораторная работа №6

Методы взаимодействия в сети передачи данных

Выполнил студент группы 950503 Проверил ассистент кафедры ЭВМ

Сякачёв П.В. Марцинкевич В.А.

Минск 2021

**1**. **Виртуальный канал.**

При формировании полномасштабной сети, то есть при объединении разрозненных физических сегментов в СПД той или иной сложности, возникает ряд специфических задач, направленных на оптимизацию взаимодействия между абонентами.

Упомянутые задачи решают на третьем и четвертом уровнях модели OSI.

Новые задачи обусловлены серьезными отличиями процессов передачи-приема пакета в пределах сегмента и между сегментами. Основные отличия заключаются в необходимости ретрансляций пакетов, а так же в возможном наличии альтернативных путей.

Одним из ключевых терминов транспортного уровня является термин соединение (connection). По сути дела, понятие соединения связано с понятием готовности. Если абоненты находятся в состоянии «нормальной готовности» передавать или принимать данные, то считают, что между ними установлено соединение. С учетом абстрагирования от более низких уровней модели OSI и инкапсуляции, соединение может быть выражено неявно.

При разговоре о соединениях невозможно обойти стороной вопрос о надежности.

Существуют два способа организации взаимодействия:

1. Без гарантированной доставки -- в СПД предпринимаются определенные усилия по доставке пакетов, но при этом ничего не гарантируется (при необходимости, соответствующий контроль возлагается на программы-абоненты).

2. С гарантированной доставкой -- алгоритм работы транспортной службы гарантирует доставку пакетов (программы-абоненты могут не контролировать наличие и очередность пакетов). Однако, соединение без гарантированной доставки практического смысла не имеет.

Поэтому наличие соединения как правило говорит о надежности.

**2**. **TCP.**

Классической реализацией оконного метода является оконный механизм протокола транспортного уровня TCP (Transmission Control Protocol) (основное RFC -- RFC 793). Протокол обеспечивает установление надежного соединения между сугубо пользовательскими или другими видами приложений, то есть доставка данных в правильном порядке гарантируется.

В стандарте TCP описано динамическое скользящее окно.

TCP соответствует клиент-серверной модели.

Сокет (socket) -- это «привязка» к виртуальному каналу, соединяющему между собой два взаимодействующих сетевых процесса, с точки зрения одного (любого) из этих процессов, причем с учетом всех трех уровней адресации.

Применительно к каждому TCP-соединению нужно выделять приложение, производящее или потребляющее сетевые данные, и TCP-процесс, предоставляющий коммуникационные услуги (например, специальный драйвер ОС).

Предназначенное для передачи сообщение разбивается на сегменты.

Минимальной учитываемой в окне единицей данных является октет, то есть байт. Все байты сообщения последовательно нумеруются так называемыми последовательными номерами -- SNs (Sequence Numbers). Нумерация начинается с некоторого начального последовательного номера -- ISN (Initial Sequence Number), который как правило не равен нулю, а генерируется реализациями случайно (например, на основе текущего времени) для того чтобы лучше управлять соединениями (например, после их ненормальных завершений). Принято, что сам ISN в нумерацию байтов не включается, то есть номер первого байта сообщения больше ISN на единицу.

Номером сегмента является SN первого байта данных в нем. По разным понятным причинам длина сегмента может варьировать, но она имеет ограничение. Поэтому важное значение имеет конфигурационный параметр MSS (Maximum Segment Size) -- максимальная длина сегмента (по умолчанию 536 байтов).



Рисунок -- Пример сегментации TCP-сообщения

Код программы:

class Socket {

protected:

int sDescriptor;

int domain;

int type;

int protocol;

int port;

struct sockaddr\_in sAddr;

public:

constexpr Socket() : domain{AF\_INET}, protocol{SOCK\_STREAM}, type{0}, port(8080), sDescriptor{-1} {

memset(&sAddr, 0, sizeof(sockaddr));

}

constexpr Socket(const int domain, const int protocol,

const int type, const unsigned int port) : domain{domain}, protocol{protocol}, type{type}, port(port), sDescriptor{-1} {

memset(&sAddr, 0, sizeof(sockaddr));

}

virtual ~Socket() {

stop();

}

void change\_domain(const int domain) noexcept;

void change\_type(const int type) noexcept;

void change\_protocol(const int protocol) noexcept;

virtual void start();

virtual void stop();

virtual bool is\_open();

protected:

void create();

void init\_server\_sockaddr() noexcept;

void init\_client\_sockaddr() noexcept;

void bind\_name();

};

class Station : public Socket {

private:

int cDescriptor;

unsigned int queue;

public:

constexpr Station() : cDescriptor(-1), queue(20) {}

constexpr Station(const unsigned int queue) : cDescriptor(-1), queue(queue) {

}

~Station() {

stop();

}

void change\_queue(const unsigned int queue) noexcept;

void start() override;

void stop() override;

bool is\_open() override;

void send\_pack(Package &P);

Package& get\_pack();

void set\_queue\_connect(const unsigned int &queue);

void accept\_connection();

private:

unsigned int get\_status();

void send\_status(Status &S);

};

class Client : public Socket {

private:

IPv4 ip;

public:

constexpr Client() : ip{} {

memset(&sAddr, 0, sizeof(sAddr));

}

constexpr Client(const IPv4 &ip) : ip(ip) {

memset(&sAddr, 0, sizeof(sAddr));

}

void start() override;

void convert\_ip();

void send\_pack(const Package &pack);

Package& get\_pack();

void set\_ip(const IPv4 &ip);

private:

void connect\_to\_server();

unsigned int get\_status();

void send\_status(Status &S);

};

class IPv4 {

private:

std::array<unsigned char, 4> data;

public:

constexpr IPv4() : data{ {0} } {}

constexpr IPv4(unsigned char const a, unsigned char const b,

unsigned char const c, unsigned char const d)

: data{{a, b, c, d}} {}

constexpr IPv4(IPv4 const &other) noexcept : data(other.data) {}

IPv4& operator=(IPv4 const &other) noexcept;

std::string to\_string() const noexcept;

unsigned long to\_unlong() const noexcept;

void from\_unlong(const unsigned long data) noexcept;

friend std::ostream& operator<<(std::ostream &os, const IPv4 &a);

friend std::istream& operator>>(std::istream &is, IPv4 &a);

};

const size\_t MAX\_SIZE\_PACK = (sizeof(Pack) + 100) \* 2;

class Package {

private:

struct Pack value;

IPv4 sender;

IPv4 recipient;

public:

constexpr Package() : sender{}, recipient{}, value{} {}

constexpr Package(const IPv4 &sender, const IPv4 &recipient) : sender{sender}, recipient{recipient}, value{} {}

void start() noexcept;

char\* get\_data() noexcept;

void change\_sender(const IPv4 &sender) noexcept;

void change\_recipiend(const IPv4 &recipiend) noexcept;

template <typename Type>

void change\_data(Type &data) {

this->value.fother = typeid(data).hash\_code();

memset(this->value.data, '\0', MAX\_SIZE\_PACK\_DATA);

memcpy(this->value.data, data, sizeof(data) \* sizeof(\*data));

this->value.sizeData = sizeof(data);

}

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 std::ostream& operator<<(std::ostream &out, Package &P);

// friend std::istream& operator<<(std::istream &in, Package &P);

private:

void set\_flag() noexcept;

void set\_sender() noexcept;

void set\_recipiend() noexcept;

void set\_other\_flag() noexcept;

void set\_size\_pack() noexcept;

friend class boost::serialization::access;

private:

const unsigned long fstart = 02001;

};

Вывод: в ходе выполнения лабораторной работы был получен опыт работы с методами взаимодействия в сети передачи данных, а также был изучен оконный механизм протокола транспортного уровня TCP.