Packet System

상속과 디자인 패턴을 활용한 Packet System

목차

Design : Packet system 1 / 3

- 아이디어

  스위치 구문을 사용하여 패킷 종류에 따라 case 문을 주고 이를 통해 패킷 처리과정을 분기

- 문제점

  1. 패킷 종류가 많은 경우 스위치 문이 지나치게 커질 우려
  2. 스위치 구문으로 패킷을 처리하기 때문에, 새로운 패킷 종류를 추가하는 경우 소스 코드 안의 case 문을 수정해야 함

Design : Packet system 2 / 3

- 아이디어

  Strategic pattern을 사용, 스위치 구문 제거.

  모든 패킷을 패킷 별로 각각 패킷 클래스로 구현. 각 패킷 별로 서로 다른 패킷 처리 과정을 가지므로, 이러한 처리 과정을 패킷 별 고유한 행위라 가정, 패킷의 처리 과정을 해당 패킷 클래스의 멤버 메소드로 구현하고 패킷 처리 시에는 그 메소드를 호출.

  패킷을 받으면 패킷의 종류에 따라 패킷 클래스 별로 구현된 패킷 프로세스 알고리즘이 작동

- 문제점

  패킷 종류가 늘어나자 패킷 종류 별로 중복되는 부분이 발생. 중복되는 작업을 개별적으로 구현하면 시간이 많이 걸릴 뿐만 아니라 구현 과정에서 실수가 발생할 확률이 커짐

Design : Packet system 3 / 3

- 아이디어
  1. Inheritancy 를 사용해 클래스별 공통 부분 분리.

     패킷마다 공통적으로 들어가는 메소드와 필드를 찾아내 모든 패킷 클래스들의 base class 정의하고 Packet class들이 base class를 상속.

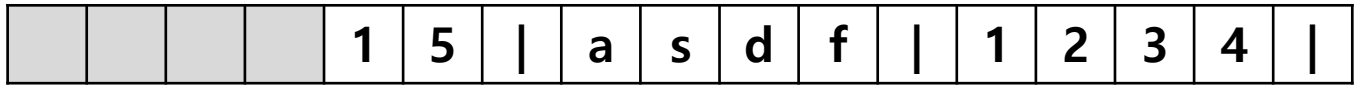
  2. Template pattern을 사용해 클래스 메소드의 공통 부분과 개별적인 처리가 필요한 부분을 구분

     (de)serialize 과정과 패킷 처리 과정에 template pattern을 적용해 공통되는 부분을 묶고, 각 패킷 별로 처리가 달라져야 하는 부분은 derived class에서 virtual method 로 구현하도록 base class에 인터페이스를 정의

Packet Protocol

- 구조
  - 패킷은 header 부분과 buffer 부분으로 나뉘며, header에는 이어서 받아야할 buffe의 길이가 들어감.
  - Header의 길이는 sizeof(size_t)로 고정하고, Buffer의 길이는 임의의 길이가 가능.
  - 각 Packet class 별로 serialize() 메소드가 각각 정의되며, Buffer 부분은 이 메소드에 정의 된 규칙에 따라 serialize 됨. Deserialize 과정도 serialize 와 동일한 방식이 적용됨.
  - Packet class의 각 필드 값을 구분하기 위해 구분자로 '|' 문자를 사용. 구분자와 겹치지 않기 위해 입력 값 중에 같은 문자가 있는 경우 이를 특정한 다른 문자(space 또는 '_' 등)로 변경하여 저장.
- 프로토콜 예시
  - PK_CS_LOGIN_REQUEST : packet id(15), id("asdf"), password("1234")

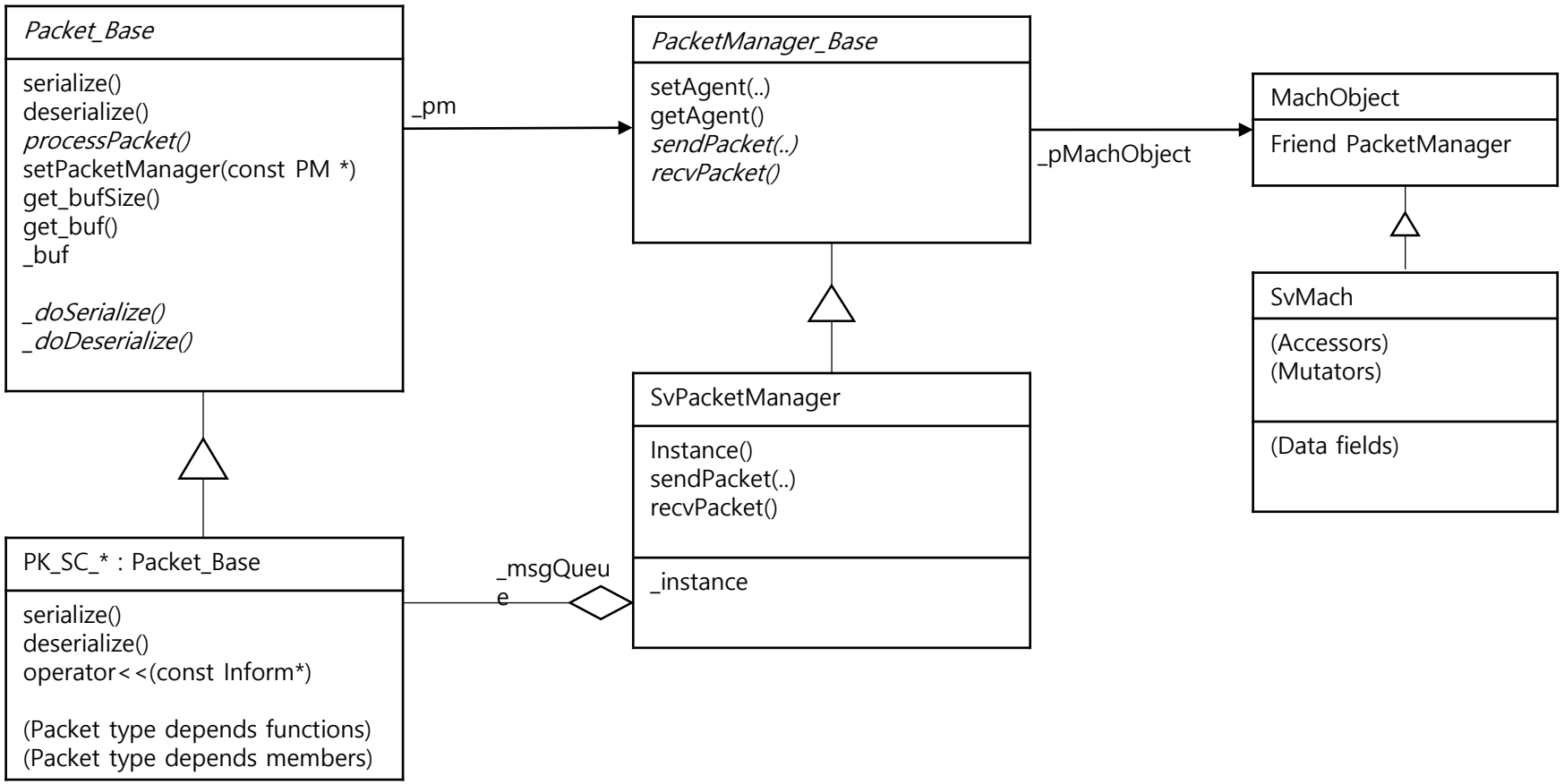
    header space(13)



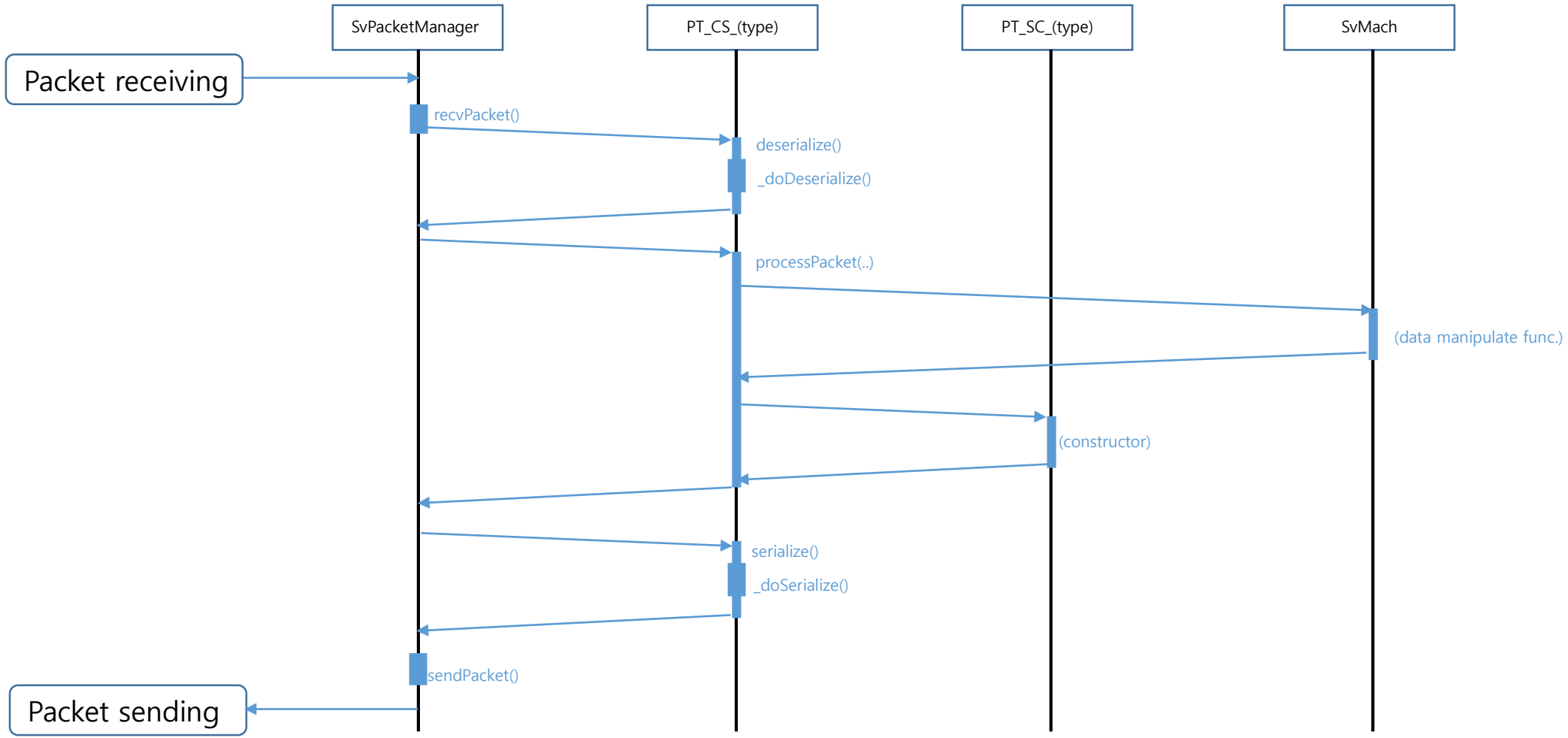
  - PK_CS_CHAT_CHAT: packet id(20), user key(13), room key(10), 입력 내용("HELLO!")

    header space(16)

Class Diagram : Packet system

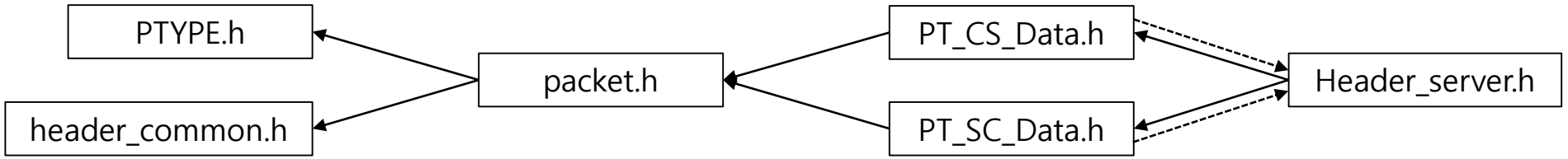


Sequence Diagram : Packet system



File structure : Packet system

- header_common.h : 서버와 클라이언트에 공통적으로 쓰이는 data class를 정의.
- Header_server.h : 서버에 쓰이는 data class를 정의
- PTYPE.h : packe에 따른 고유 번호를 정의.
- packet.h : 모든 packet의 base class 정의. Packet manage의 base class 정의.
- PT_CS_Data.h : 클라이언트에서 서버로 가는 packet 정의.
- PT_SC_Data.h : 서버에서 클라이언트로 가는 packet 정의.



Code Sample : Packet system 1 / 2

```
/*************************************************************************
 * Packet_Base class
 *************************************************************************/
class PacketManager_Base;
struct Packet_Base
{
public:
	static const size_t HEADER_SIZE = sizeof(size_t);
public:
	/* Member method */
	Packet_Base(PTYPE pType, const char * buf, size_t bufLen);
	virtual ~Packet_Base();

	static int ptoi(PTYPE);

	void serialize();		// Template method for serialize process
	void deserialize();		// Template method for serialize process
	virtual std::shared_ptr<Packet_Base> processPacket(MachObject& targetMObject) = 0;  //
Process received packet using strategy pattern
	void setProcessInfo(ProcInfo::ProcCode resCode, std::string&& msg = "");

	// Accessor
	size_t get_packetSize(); // The size of whole packet include header space.
	size_t get_bufSize();			// The size of buffer(serialized information).
	const char * get_bufAddr() const;
	const std::stringstream& get_buf() const;
	const ProcInfo& get_ProcInfo() const;

	Packet_Base& operator<<(const char * buf);
public:
	/* Member field */
	const PTYPE id;
	std::vector<SOCKET> sockList;
protected:
	/* Member method */
	Packet_Base(PTYPE);

	virtual void _doSerialProc() = 0;
	virtual void _doDeserialProc() = 0;
protected:
	/* Member field */
	std::stringstream _buf;
	ProcessInfo _ProcInfo;
private:
	/* Member method */
	Packet_Base();
	void _setHeaderSpace();  // Make (header) space for packet(_buf) size in _buf.
	void _writeHeader();	// At the end of serialize process, write _buf size on header
space calling at the end of serialization.
	size_t _bufSize();			// The size of serialized information excluding
header space
};
```

```
/*************************************************************************
 * PK_CS_LOGIN_REQUEST class
 *************************************************************************/
struct PK_CS_LOGIN_REQUEST : public Packet_Base
{
public:
	/* Member method */
	PK_CS_LOGIN_REQUEST();
	PK_CS_LOGIN_REQUEST(PTYPE ptype, const char* buf, size_t bufLen);

	virtual std::shared_ptr<Packet_Base> processPacket(MachObject& targetMObject);	//
Received packet process procedure using strategy pattern
public:
	/* Member field */
	std::string userId;
	std::string userPassword;
protected:
	/* Member method */
	virtual void _doSerialProc();
	virtual void _doDeserialProc();
protected:
	/* Member field */

};
```

Code Sample : Packet system 2 / 2

```
/*************************************************************************
 * PacketManager_Base class
*************************************************************************/
class PacketManager_Base
{
public:
        /* Member method */
        virtual ~PacketManager_Base();

        void setAgent(MachObject * pMObj);
        MachObject& getAgent();

        virtual void sendPacket(std::shared_ptr<Packet_Base>) = 0;                  // add
packet to outgoing queue.
        virtual std::shared_ptr<Packet_Base> recvPacket() = 0;      // get packet from incoming
queue.
public:
        /* Member field */
protected:
        /* Member method */
        PacketManager_Base();

protected:
        /* Member field */
        MachObject * _pMachObject;
};
```

```
/*************************************************************************
 * SvPacketManager class
        - Singleton pattern.
*************************************************************************/
class SvPacketManager : public PacketManager_Base
{
public:
        /* Member method */
        static SvPacketManager& Instance();

        void sendPacket(std::shared_ptr<Packet_Base> spPk);// transmit packet via network.
        std::shared_ptr<Packet_Base> recvPacket();// get packet from incoming packet queue.
public:
        /* Member field */
protected:
        /* Member method */
protected:
        /* Member field */
        static SvPacketManager _instance;
        std::queue<std::shared_ptr<Packet_Base>> _mapQueue;        // incoming packet queue

        friend DWORD WINAPI recvThreadMain(LPVOID);
        friend DWORD WINAPI packetProcessWorkerThreadMain(LPVOID);
};
```

블로그 & Github 주소

- 블로그 주소 : https://devwoodo.blogspot.kr
- Portfolio repos. 주소 : https://github.com/devwoodo/chatters