

# Computer Network

## Term Project 1

- Client Server Program -

학	과	컴퓨터공학과
학	번	201211704
이	름	김기홍
제	출 일	2016.04.25



## <목 차>

1장 서론 .....	3
2장 배경 .....	3
3장 프로그램 설명 및 구현 .....	3
3.1 기능 소개 .....	3
3.1.1 모드 .....	3
3.1.2 다중스레드 .....	4
3.1.3 암호화 및 복호화 .....	4
3.1.4 사용자 등록 .....	4
3.1.5 탐색 성능 향상 .....	4
3.2 기능의 구현 .....	5
3.2.1 모드 .....	5
3.2.2 다중스레드 .....	6
3.2.3 암호화 및 복호화 .....	7
3.2.4 사용자 등록 .....	8
3.2.5 탐색 성능 향상 .....	10
4장 실행 예시 .....	10
5장 프로그램 분석 .....	13
6장 돌아보며 .....	16
7장 응용 프로그램 .....	17

## <그림 목차>

그림3.2.1 모드 입력 화면 .....	5
그림3.2.2 단일 스레드 환경과 다중 스레드 환경 비교 .....	6
그림3.2.3 _beginthread() .....	6
그림3.2.4 encrypt.dll 라이브러리 .....	7
그림3.2.5 암호화 및 복호화 다이어그램 .....	7
그림3.2.6 문자열의 암호화 및 복호화 .....	8
그림3.2.7 사용된 의사 난수 알고리즘 .....	8
그림3.2.8 같은 플레이어의 누계 승률 출력 .....	8
그림3.2.9 다른 플레이어의 누계 승률 출력 .....	9
그림3.2.10 이미 있는 아이디이거나 비밀번호가 틀렸을 경우 .....	9
그림3.2.11 테이블 탐색 다이어그램 .....	10
그림4.1.1 모드 선택 화면 .....	11
그림4.1.2 서버에 접속 후 입력 대기 상태 화면 .....	11
그림4.1.3 게임 결과 출력 화면 (CvP 모드) .....	11
그림4.1.4 동시에 두 개의 클라이언트가 플레이하는 화면 .....	11
그림4.2.1 먼저 접속한 플레이어가 다른 플레이어가 접속하길 기다리는 화면 .....	12
그림4.2.2 두 플레이어가 각자 입력을 대기하고 있는 화면 .....	12
그림4.2.3 게임 결과 출력 화면 (PvP 모드) .....	12
그림5.1 RawCap 프로그램에서 Loopback을 감시 중인 화면 .....	13
그림5.2 RawCap 프로그램에서 Loopback으로의 패킷 감지 화면 .....	14
그림5.3 Wireshark에서 dumpfile.pcap를 본 화면 .....	14
그림5.4 패킷의 정보를 보여주는 화면 .....	15
그림5.5 조금 더 자세한 패킷의 정보 (암호화 전) .....	15
그림5.6 조금 더 자세한 패킷의 정보 (암호화 후) .....	15
그림6.1 패킷 분석 실패 화면 .....	16
그림6.2 GSM over Ip가 사용하는 TCP 포트번호 .....	16
그림6.3 GSM over IP의 포트번호 수정 후 문제 해결된 화면 .....	17
그림7.1 보드게임 : 다빈치코드 .....	17

## 1장 서론

이번 Term Project의 주제는 컴퓨터 **네트워크를 이용한 클라이언트 서버 프로그램의 구현**이다. 이 프로그램은 컴퓨터로 가위 바위 보를 할 수 있는 기능을 가진다.

본 프로젝트는 **TCP와 소켓프로그래밍의 이해**, 더 나아가 컴퓨터 **네트워크의 흐름을 이해**하고자 진행되었으며 게임을 소재로 하여 다소 어려울 수 있는 내용에 흥미를 더한다.

해당 보고서는 **구현의 설명에 중점**을 두었으며 구현에 사용된 방법, 기술, 도구들을 간단하게 설명한 후 조금 더 자세하게 다루는 일련의 순서를 가진다.

후반부에는 본 프로젝트 내용의 번외로 구현한 다른 재미있는 게임을 소개한다.

## 2장 배경

본 프로젝트의 주제인 가위 바위 보를 할 수 있는 게임 프로그램(이하 가위 바위 보 프로그램)은 **Windows환경**에서 C언어로 작성되었으며 컴파일 및 디버깅 도구로는 Microsoft에서 제공하는 **Visual C++ 2008 Express**를 사용하였다. 그리고 loopback으로 전달되는 패킷을 감시하기 위한 프로그램으로는 **RawCap**과 **Wireshark**를 사용하였다.

## 3장 프로그램 설명 및 구현

프로그램의 각 기능에 대한 간단한 설명에 이어서 보다 자세한 설명과 구현 과정에 대해 살펴본다.

### 3.1 기능 소개

#### 3.1.1 모드

가위바위보 프로그램은 두 개의 서버 프로그램과 하나의 클라이언트 프로그램으로 구성된다. 서버 프로그램은 각각 **CvP(Computer vs Player) 모드**와 **PvP(Player vs Player) 모드**를 지원하며 클라이언트 프로그램은 실행 시 플레이어가 모드를 선택하여 접속할 서버를 선택할 수 있다.

### 3.1.2 다중스레드

각 서버는 클라이언트가 연결 요청을 하기까지 무한 대기하고 있으며 요청이 들어오고 정상적으로 연결이 이루어지면 **클라이언트를 위한 스레드**를 만들어 넘겨주고 자신은 다시 요청을 받기 위한 무한 대기상태로 돌아간다.

### 3.1.3 암호화 및 복호화

문자열의 각 문자에 대해 **중첩으로 rand()함수와 의사난수 알고리즘 그리고 xor 연산**을 사용하여 암호화 및 복호화를 하였다. 그리고 주목할 점은 암호화 및 복호화를 수행하는 부분을 **동적 라이브러리**로 만들어 여러 프로그램에서 효율적으로 사용할 수 있게 하였다.

### 3.1.4 사용자 등록

가위바위보 게임은 사용자 등록(=로그인) 기능이 있는데 이 기능은 클라이언트 프로그램에서 아이디와 비밀번호를 입력하면 서버에서는 그 정보를 개인정보 테이블에 저장하는 기능이다. 이 기능의 목적은 여러 **플레이어의 정보를 각각 저장**하기 위함이다. 이를테면 전체 게임 플레이 수, 누적승률 등의 정보 같은 것들이 있다.

### 3.1.5 탐색 성능 향상

사용자 등록 기능을 위해서는 플레이어들의 정보를 저장할 수 있는 테이블이 필요한데 이 프로그램에서는 연결 리스트를 사용하였다. 기본적인 연결 리스트는 탐색을 위해 순차 탐색을 필요로 하지만 이는 정보가 많아질수록 비효율적일 수 있다. 이를 조금 더 효율적으로 하기 위해 각 노드의 사용자 아이디를 키 값으로 하여 아이디의 첫 글자를 배열의 인덱스로 각각 사상시켜 인덱스마다 리스트를 가지는 구조를 사용하였다. 이는 **간단한 해시 테이블**로 이름 붙여질 수 있을 것이다.

## 3.2 기능의 구현

### 3.2.1 모드

클라이언트 프로그램은 실행 시 특정 모드를 선택할 수 있다. 모드 입력은 cvp, pvp, exit가 유효하며 나머지 문자열은 잘못된 입력으로 간주하고 다시 입력받는다.

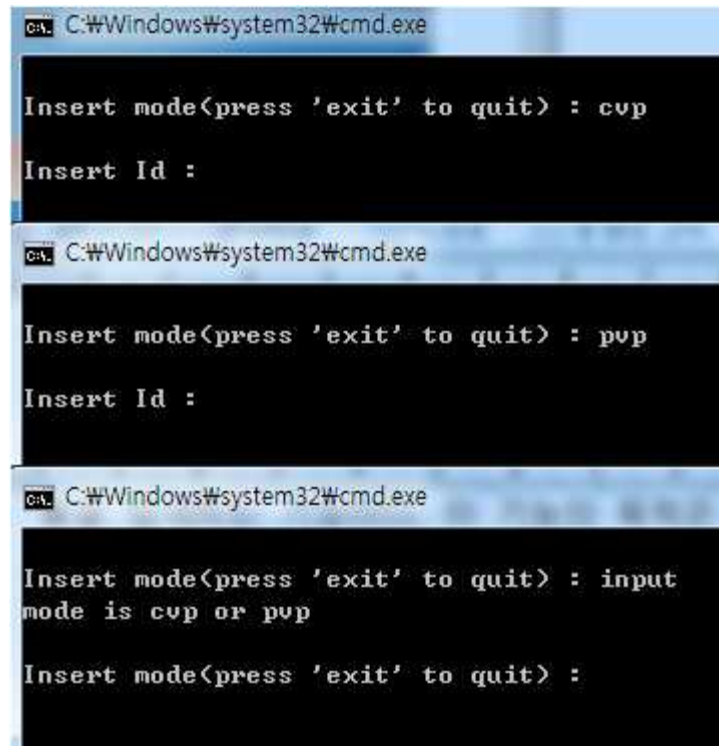
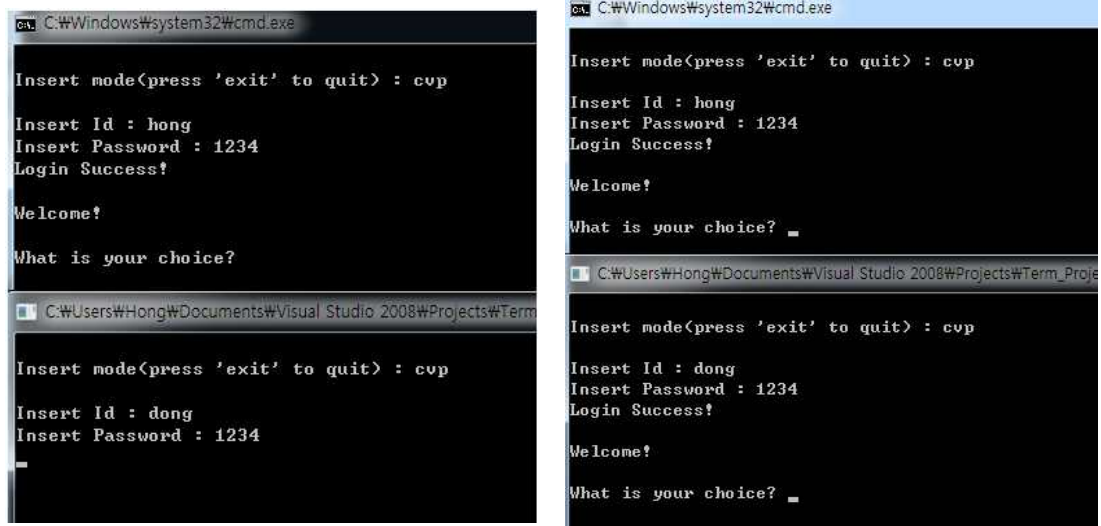


그림3.2.1 모드 입력 화면

모드의 구현은 서로 다른 포트 번호를 가진 각각의 서버로 접속을 요청하는 방법으로 구현하였다. 그림3.2.1에서 보이는 것과 같이 입력을 받아 문자열을 검사하고 해당하는 모드의 기능을 하는 **서버의 포트번호를 변수에 저장함**으로써 모드를 선택할 수 있다.

### 3.2.2 다중스레드

다중스레드 기능을 넣음으로써 한 플레이어가 특정 서버에 접속해 있을 때 다른 플레이어도 같은 서버에 접속할 수 있다. 즉, **여러 플레이어가 동시에 서버와 통신**을 할 수 있다. 구현은 `_beginthread()` 함수로 아주 쉽게 할 수 있다.



단일 스레드 환경

다중 스레드 환경

그림 3.2.2 단일 스레드 환경과 다중 스레드 환경 비교

```
_beginthread(Thread_processing, 0, (void*)C_sock);  
uintptr_t _beginthread(void (*)(void *)) _StartAddress, unsigned int _StackSize, void * _ArgList)
```

그림 3.2.3 `_beginthread()`

`_beginthread()` 함수의 매개인자로 생성한 스레드가 처리할 **함수의 포인터**, 새로운 스레드의 **스택 사이즈**, 해당 스레드로 **전달할 인자**, 총 세 가지가 있다. 인자 값이 그렇게 어렵지는 않은 것을 볼 수 있다. 다만 전달할 값이 하나가 아니라면 구조체의 주소를 넘겨주면 될 것이다.

### 3.2.3 암호화 및 복호화

암호화-및 복호화 함수는 서버 프로그램과 클라이언트 프로그램 모두에서 쓰이기 때문에 **라이브러리**로 만들어서 효율적으로 사용하였다.



그림3.2.4  
encrypt.dll  
라이브러리

암호화 방식은 우선 rand()함수의 시드 값을 문자열의 길이로 설정한 다음 문자열의 각각의 문자에 대해 **rand()함수**를 호출하여 그 반환 값으로 **xor 연산**을 한다. 이 과정을 **의사난수 알고리즘**을 이용한 **난수에 의해 반복**한다. 의사난수의 코드 역시 문자열의 길이로 설정한다. rand()함수와 의사난수 알고리즘을 함께 병행하는 이유는 같은 시드 값이라도 생성해내는 난수가 다르기 때문이다. rand()의 시드를 바꿔서 반복 횟수를 정할 수도 있지만 **시드는 결국 문자열의 길이와 관련성이 불가피하기 때문에**(복호화 시 암호화 할 때 사용했던 난수의 수열이 그대로 필요하기 때문에 시드는 공통점이 있는 문자열의 길이이기 때문이다.) 노출될 가능성이 있어 좋지 않은 방법이다. 이 일련의 과정이 그림 3.2.5에 잘 표현되어 있다.

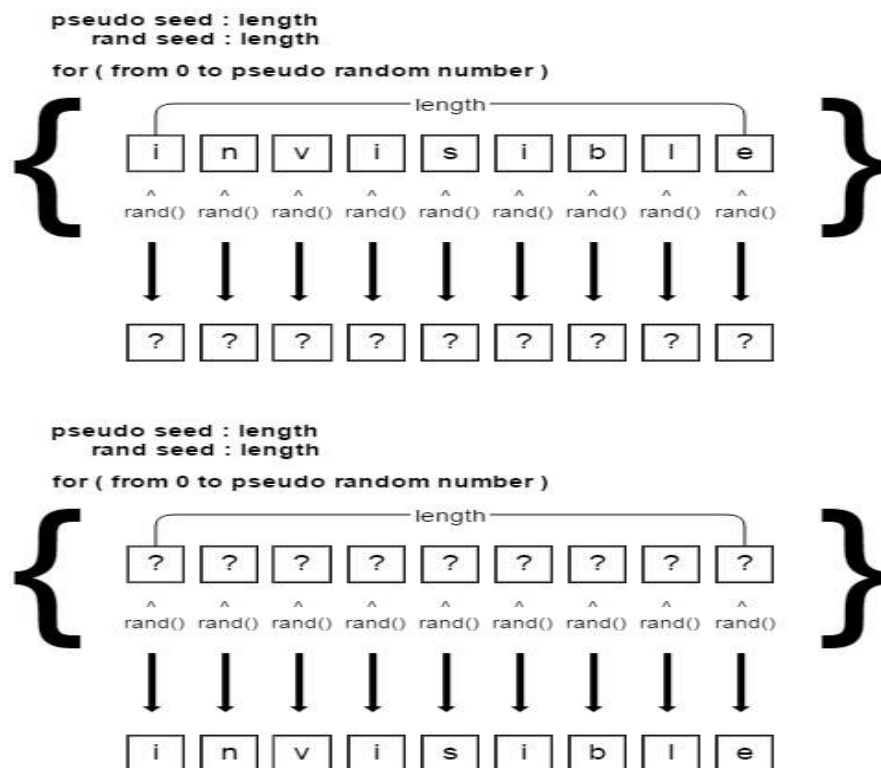


그림3.2.5 암호화 및 복호화 다이어그램



암호화 하기 전 문자 : invisible	암호화 하기 전 문자 : invisiblee
암호화 한 후 문자 : f'?'쑈      ↑,	암호화 한 후 문자 : ▲?'?q몇?
복호화 한 후 문자 : invisible	복호화 한 후 문자 : invisiblee

그림3.2.6 문자열의 암호화 및 복호화

xor연산만 한다면 비슷한 글자를 암호화 했을 때 비슷한 암호문이 생성되었을 것이다. 따라서 암호 방식이 쉽게 노출될 수 있지만 그림3.2.6을 보면 알 수 있듯이 여기서 제시한 암호화 방법은 사용한 **의사 난수 코드를 모른다면 쉽게 알아내지 못할 것이다.**

(ps. rand()함수 대신에 2개의 다른 의사 난수 코드를 사용하는 것도 좋은 방법일 것 같다.)

```
int seed= len;

seed ^= seed>>11;
seed ^= seed<<7 & 0x9D2C5680;
seed ^= seed<<15 & 0xEFC60000;
seed ^= seed>>18;
seed = (seed&0x7FFFFFFF)%10;
```

그림3.2.7 사용된 의사 난수 알고리즘

### 3.2.4 사용자 등록

사용자 등록 기능은 **플레이어 개개인을 식별할 수 있는 기능**을 제공한다. 플레이어들은 이 기능을 통해 자신의 **누적승률을 확인할 수 있다.**

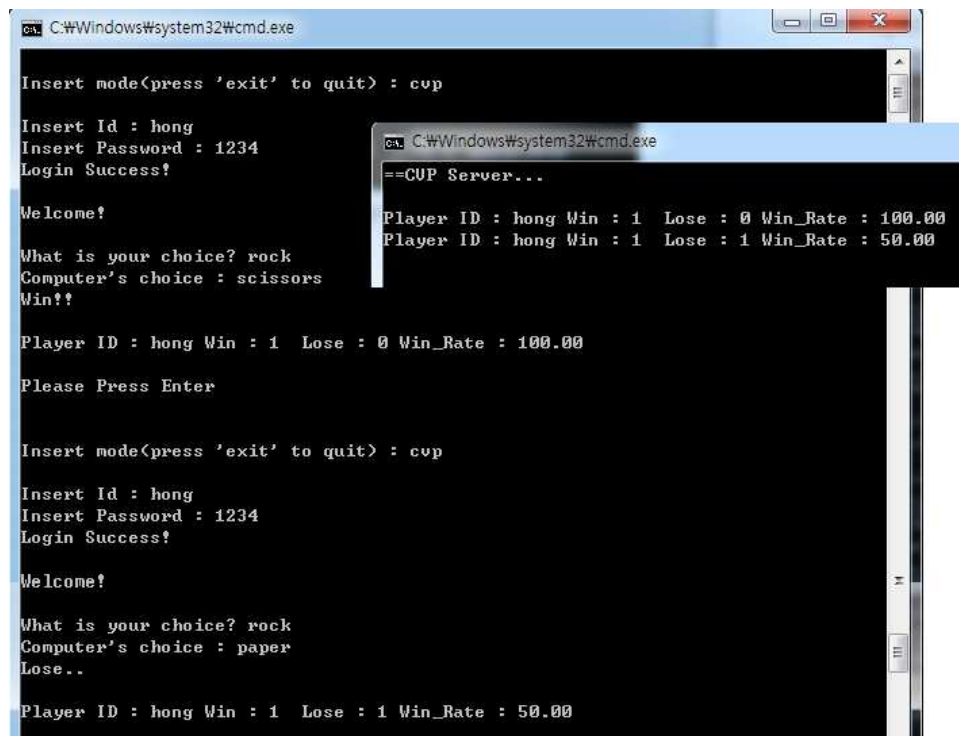
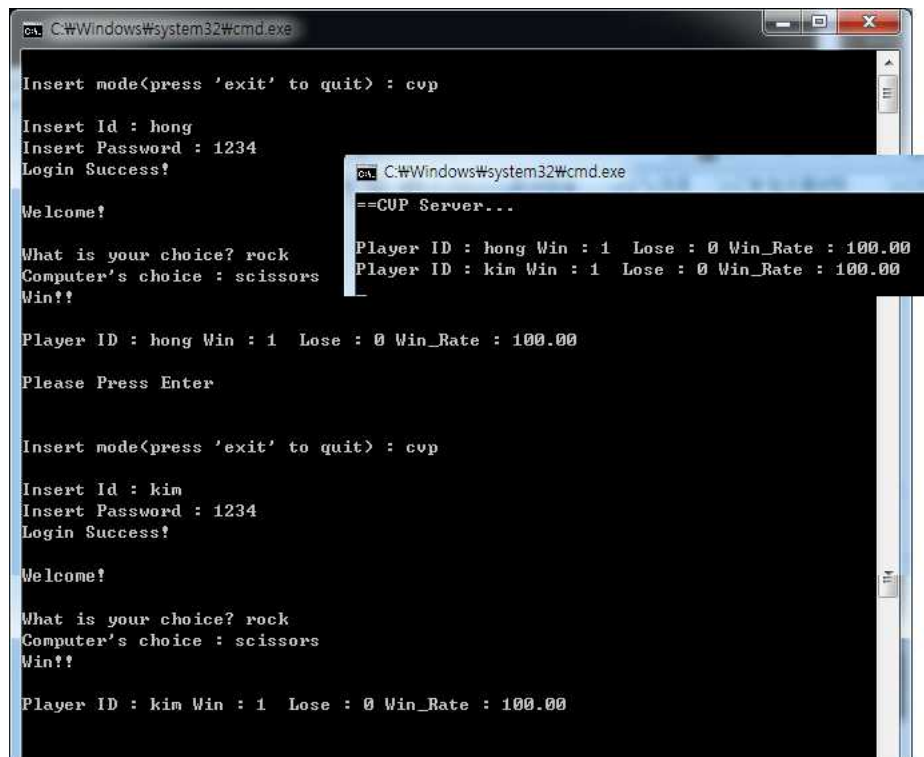


그림3.2.8 같은 플레이어의 누계 승률 출력

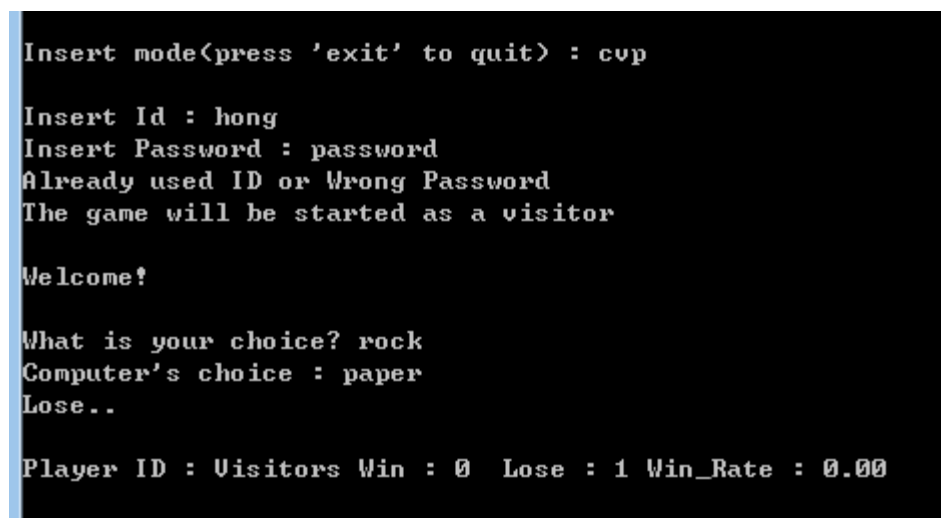


```
C:\Windows\system32\cmd.exe
Insert mode<press 'exit' to quit> : cvp
Insert Id : hong
Insert Password : 1234
Login Success!
Welcome!
What is your choice? rock
Computer's choice : scissors
Win!!
Player ID : hong Win : 1 Lose : 0 Win_Rate : 100.00
Please Press Enter
Insert mode<press 'exit' to quit> : cvp
Insert Id : kim
Insert Password : 1234
Login Success!
Welcome!
What is your choice? rock
Computer's choice : scissors
Win!!
Player ID : kim Win : 1 Lose : 0 Win_Rate : 100.00

C:\Windows\system32\cmd.exe
==CUP Server...
Player ID : hong Win : 1 Lose : 0 Win_Rate : 100.00
Player ID : kim Win : 1 Lose : 0 Win_Rate : 100.00
```

그림3.2.9 다른 플레이어의 누계 승률 출력

새로운 아이디를 입력하면 서버는 항상 로그인 성공 메시지를 클라이언트에게 전송한다. 하지만 서버의 플레이어들의 정보를 담는 테이블에 이미 존재하는 아이디를 입력하고 테이블에 저장된 비밀번호와 다른 입력을 하게 되면 서버는 **방문자 아이디로 로그인**을 시킨다. 방문자 아이디는 방문자로 접속한 모두의 누계승률을 저장한다.



```
Insert mode<press 'exit' to quit> : cvp
Insert Id : hong
Insert Password : password
Already used ID or Wrong Password
The game will be started as a visitor
Welcome!
What is your choice? rock
Computer's choice : paper
Lose..
Player ID : Visitors Win : 0 Lose : 1 Win_Rate : 0.00
```

그림3.2.10 이미 있는 아이디이거나 비밀번호가 틀렸을 경우

### 3.2.5 탐색 성능 향상

플레이어들의 정보를 저장하고 있는 리스트를 모두 순차적으로 탐색하는 데에는 프로그램의 **규모에 따라 다르겠지만 아주 비효율적**일 수 있다. 눈에 띄는 성능 변화는 없지만 탐색은 그림3.2.11과 같이 변형되어 이루어진다. (소문자, 대문자는 같은 인덱스를 사용하며, 알파벳으로 시작하지 않는 모든 아이디는 하나의 인덱스에 사상시킨다.)

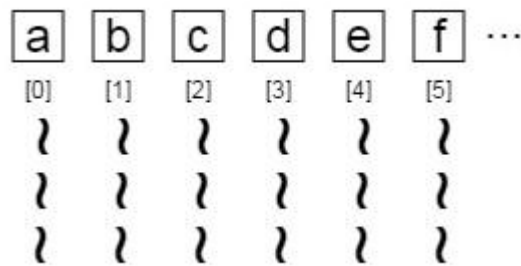


그림3.2.11 테이블 탐색 다이어그램

알파벳마다 미리 선언되어진 배열의 인덱스 값을 사상하고 해당 배열부터 연결리스트의 순차탐색이 이루어진다. 간단하게 생각해서 일반 순차탐색 알고리즘보다 **대략 27배**(알파벳 26개의 인덱스와 알파벳이 아닌 문자의 인덱스) **효율적**이라고 할 수 있다.

(아이디마다 첫 글자의 빈도가 다르기 때문에 정확한 계산이라고는 할 수 없다.)

## 4장 실행 예시

이번 장에서는 각 모드별 **전체적인 실행의 흐름**을 살펴본다.

### (1) CvP 모드

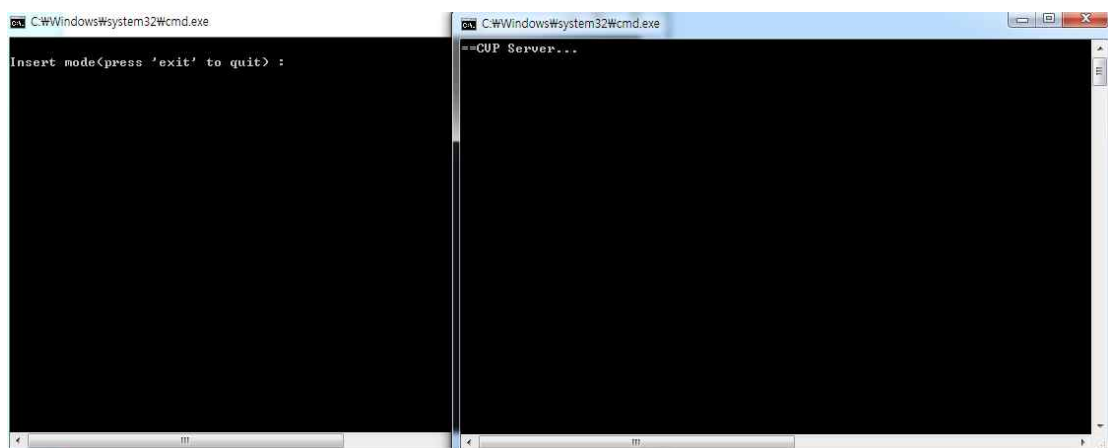


그림4.1.1 모드 선택 화면

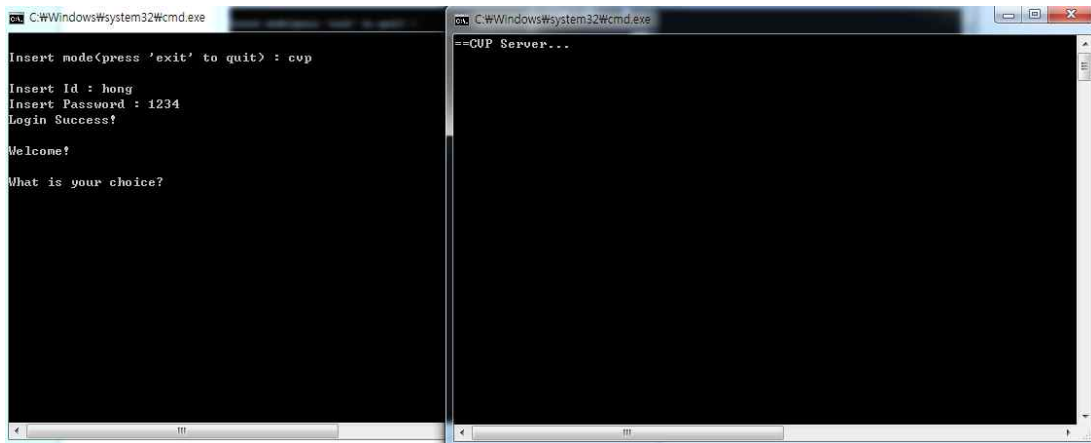


그림4.1.2 서버에 접속 후 입력 대기 상태 화면

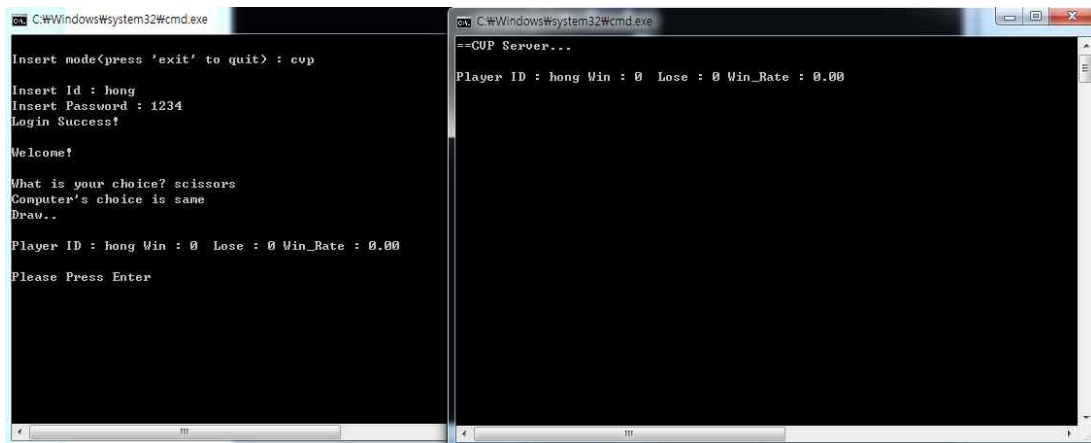


그림4.1.3 게임 결과 출력 화면 (CvP 모드)

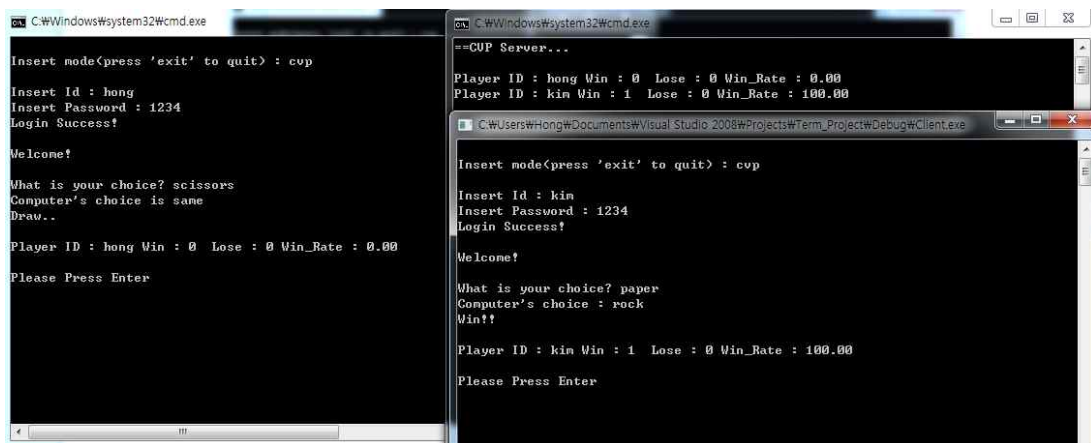


그림4.1.4 동시에 두 개의 클라이언트가 플레이하는 화면

## (2) PvP 모드

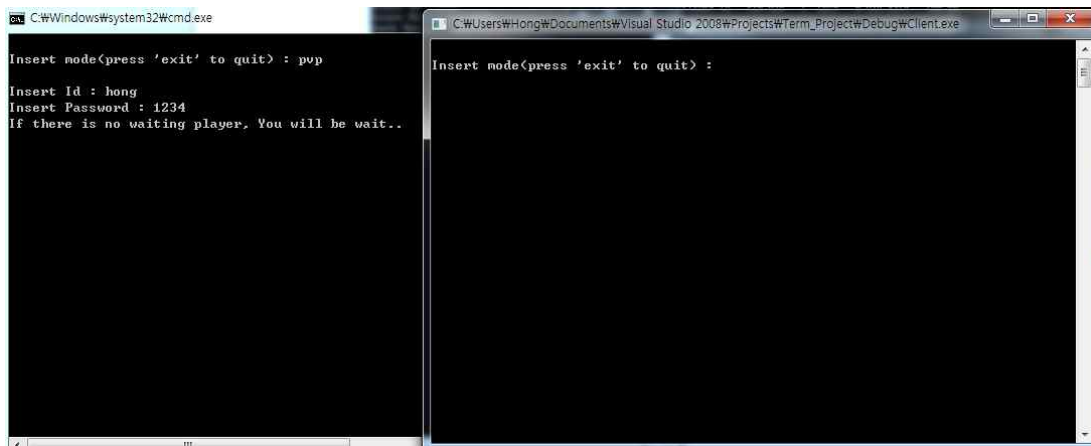


그림4.2.1 먼저 접속한 플레이어가 다른 플레이어가 접속하길 기다리는 화면

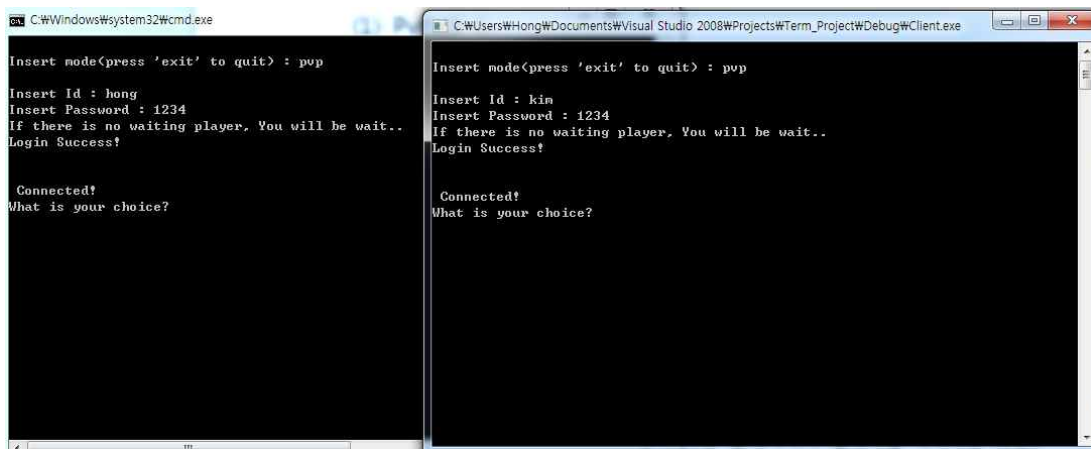


그림4.2.2 두 플레이어가 각자 입력을 대기하고 있는 화면

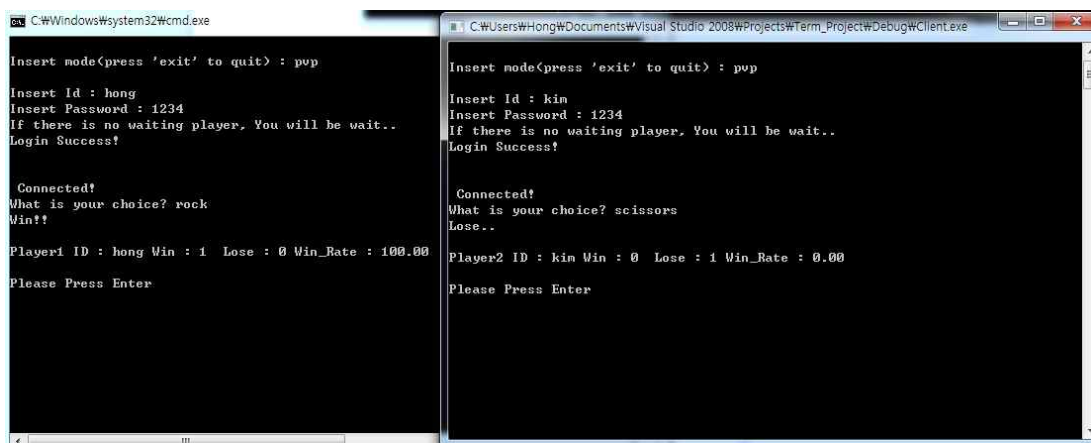


그림4.2.3 게임 결과 출력 화면 (PvP 모드)

## 5장 프로그램 분석

이번 장에서는 서버 프로그램과 클라이언트 프로그램 사이에 어떻게 연결이 이루어지고 패킷이 어떻게 전달되어지는지 **패킷 감시 툴을 이용해** 살펴본다.

먼저 설명되어져야 할 부분이 있다. 본 프로그램의 통신은 루프백으로 이루어지는데 **Windows 환경에서 Wireshark**를 통해 루프백으로 전송되는 패킷을 감지할 수 없다고 하기에 검색 결과 패킷 감지는 **RawCap** 프로그램으로 하고 생성된 .pcap 파일을 **Wireshark**로 분석하기로 한다.

**RawCap** 프로그램을 실행시키고 Loopback 인터페이스를 감시하도록 시키고 CvP 모드의 가위바위보 게임을 한 번 실행해 보았다. 그림5.2에서 보이는 바와 같이 패킷 17개가 감지되었다.

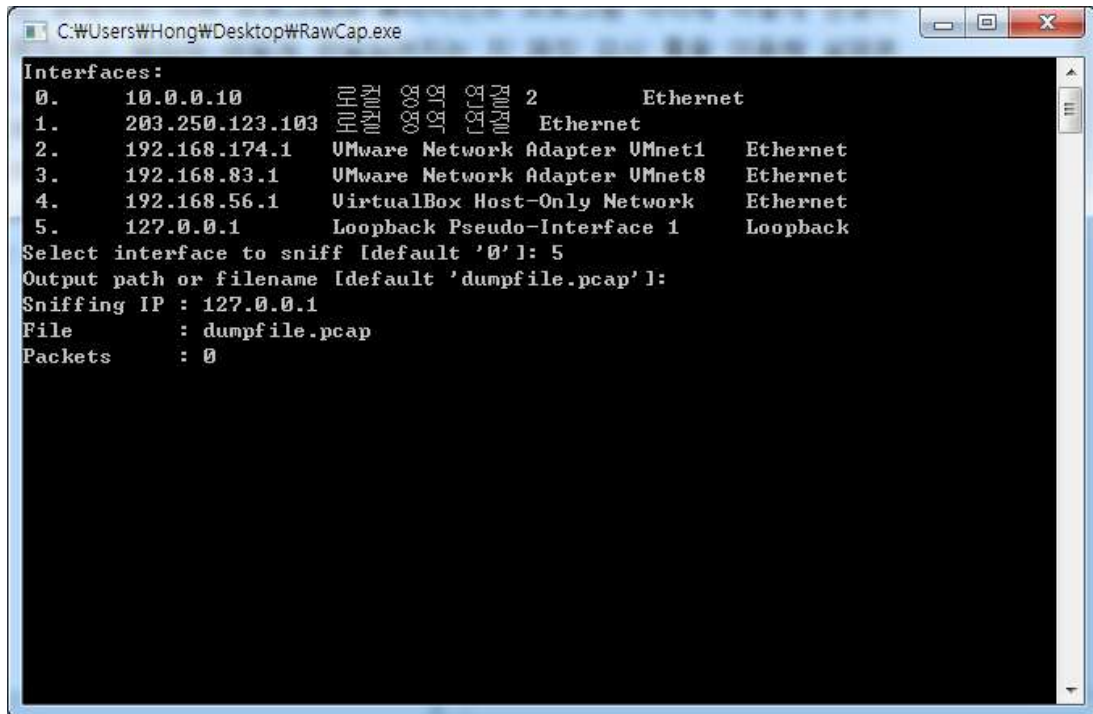


그림5.1 RawCap 프로그램에서 Loopback을 감시 중인 화면

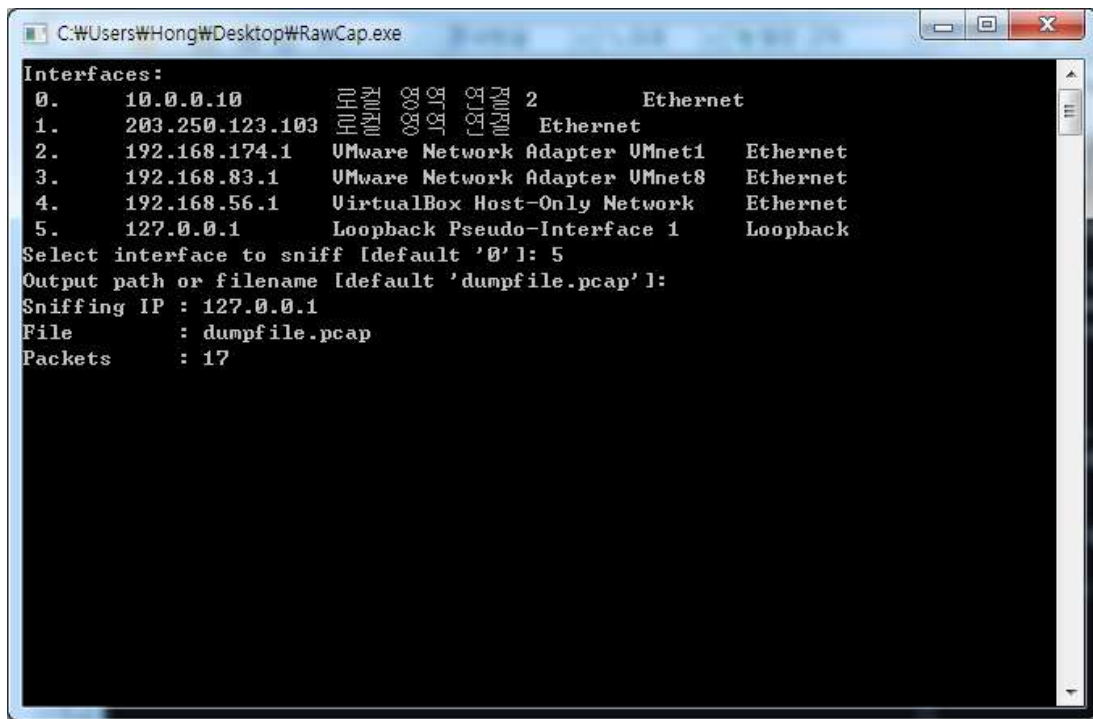


그림5.2 RawCap 프로그램에서 Loopback으로의 패킷 감지 화면

더 많은 패킷은 분석하는 데 필요가 없으므로 RawCap 프로그램은 Ctrl+C로 종료한다. 그러면 같은 디렉터리에 dumpfile.pcap 파일이 생성된다. 이 파일을 Wireshark에서 열어서 보면 그림5.3과 같다.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	52	59018 → 5000 [SYN] Seq=0 Win=8192 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	0.001000	127.0.0.1	127.0.0.1	TCP	52	5000 → 59018 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	0.001000	127.0.0.1	127.0.0.1	TCP	40	59018 → 5000 [ACK] Seq=1 Ack=1 Win=8192 Len=0
4	10.529603	127.0.0.1	127.0.0.1	TCP	49	59018 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=9
5	10.529603	127.0.0.1	127.0.0.1	TCP	40	5000 → 59018 [ACK] Seq=1 Ack=10 Win=7936 Len=0
6	10.529603	127.0.0.1	127.0.0.1	TCP	56	5000 → 59018 [PSH, ACK] Seq=1 Ack=10 Win=7936 Len=16
7	10.529603	127.0.0.1	127.0.0.1	TCP	40	59018 → 5000 [ACK] Seq=10 Ack=17 Win=7936 Len=0
8	10.529603	127.0.0.1	127.0.0.1	TCP	50	5000 → 59018 [PSH, ACK] Seq=17 Ack=10 Win=7936 Len=10
9	10.529603	127.0.0.1	127.0.0.1	TCP	40	59018 → 5000 [ACK] Seq=10 Ack=27 Win=7936 Len=0
10	16.841964	127.0.0.1	127.0.0.1	TCP	48	59018 → 5000 [PSH, ACK] Seq=10 Ack=27 Win=7936 Len=8
11	16.841964	127.0.0.1	127.0.0.1	TCP	40	5000 → 59018 [ACK] Seq=27 Ack=18 Win=7936 Len=0
12	16.841964	127.0.0.1	127.0.0.1	TCP	66	5000 → 59018 [PSH, ACK] Seq=27 Ack=18 Win=7936 Len=26
13	16.841964	127.0.0.1	127.0.0.1	TCP	40	59018 → 5000 [ACK] Seq=18 Ack=53 Win=7936 Len=0
14	16.841964	127.0.0.1	127.0.0.1	TCP	48	5000 → 59018 [PSH, ACK] Seq=53 Ack=18 Win=7936 Len=8
15	16.841964	127.0.0.1	127.0.0.1	TCP	40	59018 → 5000 [ACK] Seq=18 Ack=61 Win=7936 Len=0
16	16.841964	127.0.0.1	127.0.0.1	TCP	92	5000 → 59018 [PSH, ACK] Seq=61 Ack=18 Win=7936 Len=52
17	16.841964	127.0.0.1	127.0.0.1	TCP	40	59018 → 5000 [ACK] Seq=18 Ack=113 Win=7936 Len=0

그림5.3 Wireshark에서 dumpfile.pcap를 본 화면

조금 더 자세히 보자. 그림5.4를 보면 출발지 포트번호와 목적지 포트번호가 보이고 이어서 패킷의 플래그가 보인다. 그 다음으로 시퀀스 번호, ack 번호, 자신의 윈도우 크기, 데이터의 길이 순으로 보인다.



Info	
59018 → 5000	[SYN] Seq=0 Win=8192 Len=0 MSS=65495 WS=256 SACK_PERM=1
5000 → 59018	[SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=65495 WS=256 SACK_PERM=1
59018 → 5000	[ACK] Seq=1 Ack=1 Win=8192 Len=0
59018 → 5000	[PSH, ACK] Seq=1 Ack=1 Win=8192 Len=9
5000 → 59018	[ACK] Seq=1 Ack=10 Win=7936 Len=0
5000 → 59018	[PSH, ACK] Seq=1 Ack=10 Win=7936 Len=16
59018 → 5000	[ACK] Seq=10 Ack=17 Win=7936 Len=0
5000 → 59018	[PSH, ACK] Seq=17 Ack=10 Win=7936 Len=10
59018 → 5000	[ACK] Seq=10 Ack=27 Win=7936 Len=0
59018 → 5000	[PSH, ACK] Seq=10 Ack=27 Win=7936 Len=8
5000 → 59018	[ACK] Seq=27 Ack=18 Win=7936 Len=0
5000 → 59018	[PSH, ACK] Seq=27 Ack=18 Win=7936 Len=26
59018 → 5000	[ACK] Seq=18 Ack=53 Win=7936 Len=0
5000 → 59018	[PSH, ACK] Seq=53 Ack=18 Win=7936 Len=8
59018 → 5000	[ACK] Seq=18 Ack=61 Win=7936 Len=0
5000 → 59018	[PSH, ACK] Seq=61 Ack=18 Win=7936 Len=52
59018 → 5000	[ACK] Seq=18 Ack=113 Win=7936 Len=0

그림5.4 패킷의 정보를 보여주는 화면

더 자세한 정보를 보기 위해서 그림5.5를 살펴보면 프레임의 총 크기와 IP 헤더, TCP 헤더의 정보에 이어 데이터가 보인다. 데이터 부분에 hong 1234가 보이는데 이는 가위바위보 프로그램에서 입력한 아이디와 비밀번호이다.

▶ Frame 4: 49 bytes on wire (392 bits), 49 bytes captured (392 bits)	
Raw packet data	
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1	
▶ Transmission Control Protocol, Src Port: 59018 (59018), Dst Port: 5000 (5000), Seq: 1, Ack: 1, Len: 9	
▲ Data (9 bytes)	
Data: 686f6e672031323334	
[Length: 9]	
0000	45 00 00 31 10 f8 40 00 80 06 00 00 7f 00 00 01 E..1..@. ....
0010	7f 00 00 01 e6 8a 13 88 87 41 20 03 2f dc 2b f1 ..... .A ./..+
0020	50 18 00 20 57 41 00 00 68 6f 6e 67 20 31 32 33 P.. WA.. hong 123
0030	34 4

그림5.5 조금 더 자세한 패킷의 정보 (암호화 전)

이렇듯 암호화를 하지 않으면 패킷 감시자가 손쉽게 개인정보를 가로챌 수 있다는 것을 볼 수 있다.

이번에는 암호화를 한 다음 같은 부분을 살펴보자.

▶ Frame 4: 49 bytes on wire (392 bits), 49 bytes captured (392 bits)	
Raw packet data	
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1	
▶ Transmission Control Protocol, Src Port: 52179 (52179), Dst Port: 5000 (5000), Seq: 1, Ack: 1, Len: 9	
▶ Data (9 bytes)	
0000	45 00 00 31 3c bf 40 00 80 06 00 00 7f 00 00 01 E..1<.@. ....
0010	7f 00 00 01 cb d3 13 88 41 3a 8f f1 54 ef 6b 08 ..... A:..T.k.
0020	50 18 00 20 61 b8 00 00 67 61 e4 08 bd b7 59 47 P.. a... ga....YG
0030	7d }

그림5.6 조금 더 자세한 패킷의 정보 (암호화 후)



## 6장 돌아보며

우선 가위바위보 프로그램이라는 점에서 가위바위보의 결과가 잘 도출되는 것은 성공적이라고 할 수 있다. 또한 네트워크를 사용하여 패킷을 전달하는 것 또한 성공적이다. 다만 프로그램을 **구현하면서 아쉬웠던 점**은 분명 존재한다. 이번 장에서는 그러한 점들에 대해 기록한다.

우선 **프로그램의 전체적인 가독성 문제**이다. 게임은 단순한 가위바위보라 할지라도 사용자가 보기에 조금은 불편한 인터페이스가 아니었나 생각한다. 조금 더 게임답게 수정할 수도 있었겠지만 이 점은 본 프로젝트가 멋진 게임을 만드는데 비중을 두는 것이 아니기에 크게 고려할 사항은 아니다.

다음은 암호화 및 복호화 방법이다. 이전에 '이산수학' 과목을 수강하면서 **RSA 암호화**를 공부한 적이 있다. 소수와 mod 연산을 활용한 정말 좋은 방법이었다. 하지만 역시 알고리즘을 코드로 구현하는 것은 생각만큼 쉽지는 않았다. RSA 자체는 구현했지만 문자열을 하나의 정수로, 정수를 다시 본래의 문자열로 변환하는 데 있어서 어려움을 겪었다.

정수를 적절하게 잘라서 문자로 변환하는 어떤 프로토콜을 작성해야 구현할 수 있을 것 같다. lex 와 yacc를 사용했다면 왠지 쉽지 않았을까 생각해본다.

사용자 등록 기능에도 아쉬운 점은 존재한다. 지금의 로그인 기능은 서버가 종료되면 플레이어들 정보 테이블이 저장되지 않는다. 이를 보완하기 위해 파일에 저장하면 어떨까 생각해봤지만 앞서 설명한 다중 스레드 환경을 구현했기 때문에 **공유 메모리의 일관성 문제**에 봉착하여 실패했다. 동기화하는 방법을 다시 생각해봐야 할 것 같다.

마지막으로 Wireshark를 통한 초기 분석 시 IPA라는 프로토콜과 함께 패킷 정보는 **unknown로 표시되는 문제**가 있었다.

TCP	40 49811 → 5000 [ACK] Seq=1 Ack=1 Win=8192 Len=0
IPA	49 unknown 0x6b
TCP	40 5000 → 49811 [ACK] Seq=1 Ack=10 Win=7936 Len=0
IPA	56 unknown 0x67
TCP	40 49811 → 5000 [ACK] Seq=10 Ack=17 Win=7936 Len=0
IPA	50 unknown 0x6c

그림6.1 패킷 분석 실패 화면

Google에서 검색한 결과 IPA는 ip.access이고, 문제는 GSM(global system for mobile communication) over IP 프로토콜이 같은 5000번 포트를 사용하기 때문에 **포트가 중복**이 일어났기 때문이었다. 그림6.3과 같이 포트번호를 수정하면 원하는 결과를 얻을 수 있다.

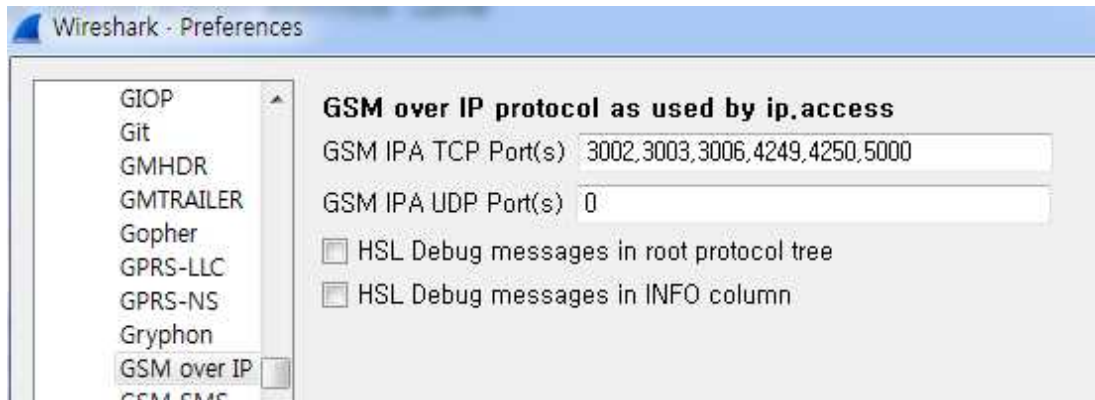


그림6.2 GSM over Ip가 사용하는 TCP 포트번호

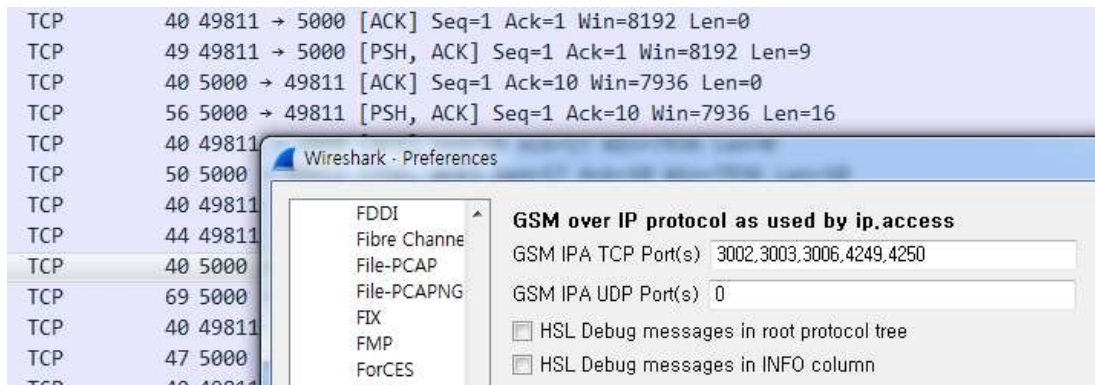


그림6.3 GSM over IP의 포트번호 수정 후 문제 해결된 화면

## 7장 응용 프로그램

이번 가위바위보 프로그램 중 특히 PvP 모드를 응용하여 **재미있는 보드게임을** 한번 만들어 보았다. 사실 자세한 TermProject 내용이 나오기 전에 게임이라는 주제가 나왔을 때 즐겨 했던 보드게임을 만들어 보았던 프로그램이다. 그냥 넘어가면 아쉽기에 번외로 짧게 소개한다.

본 프로그램의 이름은 다빈치 코드(보드게임 중 하나)이다. 개발 환경은 Linux Mint이고 언어는 역시 C언어를 사용하였다.



그림7.1 보드게임 : 다빈치코드

이하 응용 프로그램에 대한 내용은 본 TermProject와는 별개이므로 별첨 **파일로 첨부**한다.