
디지털컨버전스 기반 UI UX Front 전문 개발자 양성과정

Lecturer silenc3502 Sanghoon Lee (이상훈)
gcccompil3r@gmail.com

Student may_hz HyeonJeong Choi (최현정)
hyeonjeong9943@gmail.com

Dead-Lock이 발생하기 위해서는 아래의 **4가지 조건**이 반드시 만족되어야 한다.

1. **상호배제 (mutual exclusion)**

한번에 프로세스 하나만 해당 자원을 사용할 수 있다.

사용중인 자원을 다른 프로세스가 사용하려면 요청한 자원이 해제될 때까지 기다려야한다.

2. **점유와 대기 (hold-and-height)**

공유된 자원(shared data)를 가진 상태에서 또 다른 것을 요구할 때

내가 가진 자원이 있는데 다른 사람의 자원을 탐내는것.

3. **비선점 (no-preemption)**

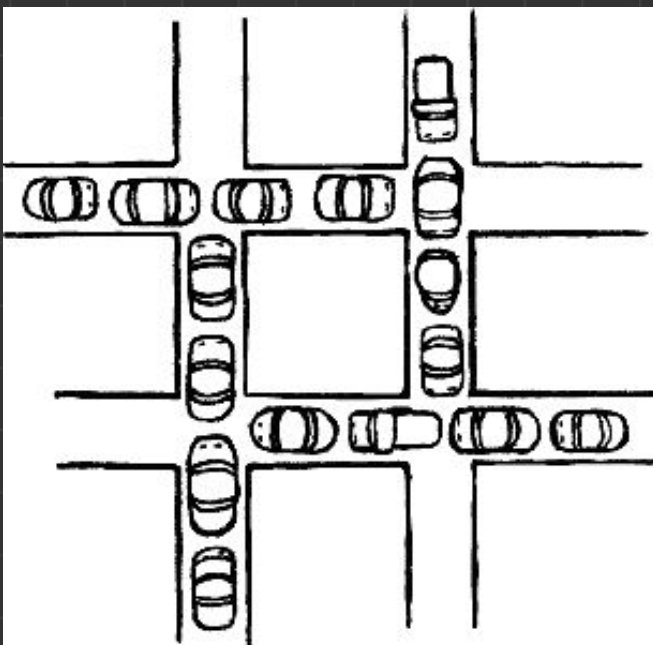
자원들이 점유하고 있는 프로세스로부터 도중에 해체 되지 않는다.

교통체증을 예로 들때 몇대의 차만 후진으로 양보해주면 데드락이 발생하지않는데, 양보하기 싫어서 오도 가도 못하는 상태

4. **순환대기 (circular-wait)**

프로세스와 자원들이 원형을 이루며, 각 프로세스는 자신에게 할당된 자원을 가지면서, 상대방 프로세스의 자원을 상호 요청하는 경우.

<< 그림으로 보는 Dead-Lock >>



- 하나의 차량은 그 순간에 하나의 구간만 차지할 수 있어요
- 각각의 차량들이 도로에서 하나의 구간을 차지하면서 나아가기를 기다리고만 있어요
- 하나의 구간이 양보할 수 없는 상황 (선점할 수 없는 상황), 차량을 뺄 수 없는 상황
- 서로 맞물리는 원형형태를 띄고 있어요

```
public SocketManager() {  
    scan = new Scanner(System.in);  
  
    in = new InputStream[ONE];  
    out = new OutputStream[ONE];  
}
```

<<Client>> 생성자

플레이어 서버에게만 입출력 해주기 때문에
ONE 이라는 상수를 사용하여 고정 !

ONE = 1;

```
public SocketManager(int num) {  
  
    out = new OutputStream[num];  
    in = new InputStream[num];  
  
    arrRockScissorPaper =  
        new String[num];  
}
```

<<Server>> 생성자

플레이어 각각 입출력이 필요하기 때문에
배열을 사용하여 입출력배열을 생성

```
public void send(Socket sock)  
    throws IOException {  
    System.out.print("숫자를 입력하세요: ");  
    String str = scan.nextLine();  
  
    out[ZERO] = sock.getOutputStream();  
    out[ZERO].write(str.getBytes());  
}
```

<<Client>>

클라이언트가 사용가능 하게 만든
output 메소드이다.
위에서 배열의 크기를 ONE으로
지정하였기 때문에 클라이언트가
사용하는 IN / OUT의 배열은
[ZERO]번째 이다.

ZERO = 0;

```
public void send(Socket[] sock, int num)  
    throws IOException {  
    out[sendCount] =  
        sock[sendCount].getOutputStream();  
  
    writer = new PrintWriter  
        (out[sendCount++], true);  
  
    String str = convertNumber2RSP();  
  
    writer.println(str);  
}
```

<<Server>>

서버가 사용가능 하게 만든
output 메소드이다.

플레이어가 입력한 숫자를
문자로 바꾸는 메소드를 넣어
각 플레이어들이 어떠한 입력을
했는지 보내 줄 수 있게 하였다.

(1) = ' 가위 '
(2) = ' 바위 '
(3) = ' 보 '

```
public void recv(Socket[] sock, int num) throws IOException {  
  
    int tmp;  
  
    in[recvCount] = sock[recvCount].getInputStream();  
    // MainServer에서 while문을 사용하기 때문에 recvCount가 필요하다  
  
    reader = new BufferedReader(new InputStreamReader(in[recvCount]));  
    // 플레이어들의 입력을 받기위한 InputStreamReader  
  
    tmp = Integer.parseInt(reader.readLine()); // (3) -> (4) 변경할 수 있게 임시 저장소를  
선언  
  
    if(tmp == MAGICNUM) { // MAGICNUM = 3;  
        arrRockScissorPaper[recvCount] = Integer.toString(tmp + ONE);  
        // 플레이어의 입력값이 3이 나올경우 + 1  
    } else {  
        arrRockScissorPaper[recvCount] = Integer.toString(tmp);  
        // 아닐경우 그대로 저장  
    }  
    System.out.println("msg: " + arrRockScissorPaper[recvCount++]);  
}  
  
public boolean canWeGetWinner(int num) {  
  
    int bitORofAllInputString = ZERO;  
  
    for(int i = ZERO; i < num; i++) {  
        bitORofAllInputString |=  
            Integer.parseInt(  
                arrRockScissorPaper[i]);  
    }  
    if(bitORofAllInputString == 7) {  
        return false;  
    } else if(bitORofAllInputString == 1) {  
        return false;  
    } else if(bitORofAllInputString == 2) {  
        return false;  
    } else if(bitORofAllInputString == 4) {  
        return false;  
    }  
    return true;  
}
```

무승부 판정을 위해 비트연산자
OR를 사용하기로 하였다
플레이어들이 1, 2, 3 을 각각 입력 했을
경우, 무승부를 판단해야 하는데 3개의
숫자들을 OR 하면
결과가 3이 나오게 된다.
결과값이 3일 경우 이게 무승부인지,
아니면 '보' 를 말하는 것 인지
알 수가 없기 때문에
(3) -> (4) 임시 변경한다.
변경한 숫자를 OR연산할 경우 결과값이
7이 나오기 때문에 무승부 판단이
가능해 진다.

<<Client>>

```
public class ClientManager extends SocketManager {
    private Socket clntSock;

    public ClientManager(String hostIp, int portNum) throws IOException {
        super();
        clntSock = new Socket(hostIp, portNum);
    }
```

<<SocketManager>> 생성자

```
scan = new Scanner(System.in);
in = new InputStream[ONE];
out = new OutputStream[ONE];
```

<<Server>>

```
public class ServerManager extends SocketManager {
    public ServerManager(int portNum, int max) throws IOException {
        super(max);

        System.out.printf("%d 명이 접속해야 게임을 시작할 수 있습니다. \n", max);

        servSock = new ServerSocket(portNum);

        clntCnt = 0;
        maxClnt = max;
        clntSockArr = new Socket[max];
    }

    public void waitForClientRequest() throws IOException {

        System.out.println("사용자 접속을 대기합니다. ");

        clntSockArr[clntCnt] = servSock.accept();
        // 접속 대기 상태 접속이 되면 아래에 있는 메시지를 출력한다.
        System.out.println(
            "[" + clntSockArr[clntCnt++].getInetAddress() +
            "] client connected");
    } // [ 소켓[clntCnt번째]의 ip ] 연결되었음
}
```

<<SocketManager>> 생성자

```
out = new OutputStream[num];
in = new InputStream[num];
arrRockScissorPaper = new String[num];
```

```
public static void main(String[] args)
    throws IOException, InterruptedException {

    ServerManager ssm = new ServerManager(9303, MAX);
    int cnt = 0;

    while (cnt != 3) { // 플레이어가 3명이 될때까지 while문 실행
        ssm.waitForClientRequest();
        ssm.recv(ssm.getClntSockArr(), ssm.getMaxClnt());
        cnt++; // 플레이어들의 입력(숫자)를 받는다. cnt는 플레이어의 수 while을 한번 실행하면 증가한다.
    }
    System.out.println("다음을 진행합니다.");

    if (ssm.canWeGetWinner(ssm.getMaxClnt())) {
        System.out.println("승패가 결정되었습니다.");
    } else {
        System.out.println("무승부: 게임을 다시 시작합니다.");
    }

    cnt = 0;
    ssm.setClntCnt(0);

    while (cnt != 3) {
        System.out.println(
            "사용자들에게 결과를 전달합니다.");
        ssm.waitForClientRequest();
        ssm.send(
            ssm.getClntSockArr(), ssm.getMaxClnt());
        cnt++;
    }
    System.out.println(
        "모든 사용자에게 입력 결과 전달 완료!");
}
```

실행결과

<<Server>>

3 명이 접속해야 게임을 시작할 수 있습니다.

사용자 접속을 대기합니다.

[/127.0.0.1] client connected

recvCnt = 0

msg: 1

사용자 접속을 대기합니다.

[/127.0.0.1] client connected

recvCnt = 1

msg: 1

사용자 접속을 대기합니다.

[/127.0.0.1] client connected

recvCnt = 2

msg: 1

다음을 진행합니다.

무승부: 게임을 다시 시작합니다.

사용자들에게 결과를 전달합니다.


```
public static void main(String[] args) throws IOException,
                                   InterruptedException {
```

```
    String ip = "127.0.0.1";
    int port = 9303;
```

```
    Scanner scan = new Scanner(System.in);
```

```
    ClientManager csm = new ClientManager(ip, port);
```

```
    System.out.println("접속 요청 완료!");
```

```
    System.out.println("가위는 1, 바위는 2, 보는 3");
```

```
    csm.send(csm.getClntSock());
```

```
    System.out.println("전송 완료!");
```

```
    csm.close(csm.getClntSock());
```

```
    // 소켓 연결을 종료한다.
```

```
    scan.nextLine();
```

```
    csm = new ClientManager(ip, port);
```

```
    // 결과를 받기 위해 다시 연결한다.
```

```
    csm.recv(csm.getClntSock());
```

```
    // 게임 결과를 받는다.
```

```
public void send(Socket sock)
    throws IOException {
    System.out.print("숫자를 입력하세요: ");
    String str = scan.nextLine();

    out[ZERO] = sock.getOutputStream();
    out[ZERO].write(str.getBytes());
}
```

실행결과

<<Client>>

21_01_21_Seventeenth.MainClient

접속 요청 완료!

가위는 1, 바위는 2, 보는 3

숫자를 입력하세요: 1

전송 완료!

1번 사용자: 가위, 2번 사용자: 가위, 3번 사용자: 가위