이기종 네트워크 환경에서 다중 응급 원격진료 모니터링 시스템 설계 및 평가

연세대학교 대학원

의 과 학 과

김 도 윤

이기종 네트워크 환경에서 다중 응급 원격진료 모니터링 시스템 설계 및 평가

지도교수 유 선 국

이 논문을 석사 학위논문으로 제출함

2008년 12월

연세대학교 대학원

의 과 학 과

김 도 윤

김도윤의 석사 학위논문을 인준함

심사위원 _____ 인

심사위원 _____ 인

심사위원 _____ 인

연세대학교 대학원

2008년 12월

감사의 글

석사과정을 마치고 논문의 마지막 마무리를 글로 남기려 하니 옛일이 스쳐 지나가며 베풀지는 못하고 받기만 한 삶을 반성하게 됩니다. 저의 석사 논문이 완성될 수 있도록 아낌없는 격려와 도움을 주셨던 많은 분들이 계셨기에 오늘과 같은 결과물이 있을 수 있었습니다.

참으로 부족한 저에게 인생의 올바른 길과 학문의 양심 그리고 진정한 학자로서 나아가야 할 길을 지도해 주신 유선국 지도 교수님께 진심으로 감사에 뜻을 전합니다. 또한 바쁘신 중에도 논문이 잘 완성될 수 있도록 조언과 격려를 해주신 신촌 세브란스병원 심장내과 하종원 교수님과 영동 세브란스병원 응급의학과 정현수 교수님께도 진심으로 감사드립니다.

연구실 인터과정과 석사과정을 지내면서 너무나도 좋고 훌륭한 선배님, 동기, 후배들 그리고 저의 연구와 인연을 맺으면서 지냈던 분들이 계셨기 때문에 저의 발전이 있었던 것 같습니다. 저와 함께 서로 격려하고 기쁨을 함께 나누었던 연구원들께 진심으로 감사드립니다. 우리 연구실 살림을 도맡아 관리하신 김동근 박사님, 학생들의 올바른 학문의 길로 갈 수 있도록 안내해준 정석명 박사과정, 인생에 아낌없는 조언과 항상 따뜻한 가르침을 주신 박순만 박사과정, 감성이라는 새로운 분야에 과감하게 자신에 인생을 바친 이충기 박사과정, 밤 세워 연구에 고민하고 인생의 고뇌 그리고 연구실 생활에 모든 것을 같이하며 저의 부족한 부분을 채워주기 위해 아낌없는 후원과 지원을 해준 김정채 박사과정, 항상 저를 보면 진진하게 연구에 방향과 문제점을 지적해주시는 원윤재 박사과정, 자신의 목표를 이루기 위해 뒤 늦께 연구실 생활에 합류한 이미희 박사과정, 항상 윗사람을 존경할 줄 알며 의리를 지킬 줄 아는 우영재 석사, 지금 다른 곳에서 자신의 꿈을 펼치며 더 큰 목표를 위해 노력하는 이동헌, 저와 생활을 같이 하면서 먼

저 졸업한 강호현, 강기원, 김수정, 이민규, 장봉문, 서국진, 서덕찬, 박정진, 한동훈, 임승경, 박윤정 선배님들께서 후배들을 이끌어 주셨기 때문에 오늘날 같이 현존하는 의학공학의 최고의 연구실이 될 수 있었다고 생각됩니다. 또한 저의 대학원 동기이며 후배들 카사 이용귀, 모성애 김도성, 크리스찬 김상용, 단가 이인호, 뇌파 조한규, 뺑기 임동규, 돌아이 김광수, 낙마오현택, 순뎅 이수호, 줌마 홍성혜, 주정 조은정, 울보 김주현, 막내 최우진, 송강일 에게도 진심으로 고맙다는 말을 전하고 싶습니다. 그리고 영원히 변치 안는 우정 유병철, 대학교 99학번 동기들 모두에게도 고맙다는 말을 전하고 싶습니다.

사랑하는 누나, 여동생, 매제, 연웅 그리고 주영이네 아버지, 어머님, 형님, 새언니, 종의, 정의 모두에게 감사하며 고마움을 전합니다. 또한 사랑하는 여자 친구 주영이가 있어서 외롭지 않고 올바른 길과 인생의 역경을 이겨 낼 수 있었던 것 같습니다. 앞으로 평생을 같이하며 지금 보다도 더 큰 목표를 위해 함께 노력하고 자 합니다.

오늘날 저의 논문이 나오기까지 항상 희생하시고 고생하신 아버지, 어머님께 진심으로 감사드리며 저의 지금까지 쌓아온 모든 공과 본 석사논문을 부모님께 받칩니다.

2008년 12월
김도윤 올림

차 례

그림 차례

표 차례

국문요약

이기종 네트워크 환경에서 다중 응급 원격진료 모니터링 시스템 설계 및 평가

최근 IT(Information Technology) 및 통신 기술의 발달로 의료 시스템 분야에 적용하여 응급 원격진료라는 서비스를 제공하게 되었다. 응급 원격진료 서비스는 인력 및 장비가 부족한 보건소, 학교, 도서지역 등에서 응급환자가 발생했을 때, 2, 3차 의료기관의 전문의와 연결하여 원격 자문과 원격 진단을 할 수 있다. 또한 이송중인 응급환자의 상태와 관련된 정보를 실시간으로 응급의 측에 전송하여 실시간으로 환자의 상태를 파악하고 환자가 응급실에 도착하기 전에 치료 준비를 할 수 있다.

현재의 응급 원격진료 서비스는 환자와 전문의가 1:1 통신 방식으로 환자의 연결 순서에 따라서 순차적으로 서비스가 수행되고 있다. 이러한 환경은 응급환자가 동시에 많이 발생 했을 때 원활한 서비스를 제공 받지 못하며, 협력진료가 요구되는 응급환자의 경우 여러 전문의와 협력진료를 할 수 없다.

원격진료 시스템을 이용하는 환자와 전문의, 응급의는 다양한 통신망을 사용한다. 즉, Wibro, HSDPA와 같은 무선통신망과 VDSL, ADSL과 같은 유선통신망을 사용하여 원격진료 서비스를 이용한다. 서로 다른 통신망은 대역폭의 차이와 공인IP, 사설IP의 특성으로 서로 통신하는데 많은 제약을 가지고 있다. 따라서 다양한 통신 환경에서도 원활한 원격진료 서비스를 이용 할 수 있도록 새로운 시스템 구성이 필요하다.

응급 원격진료 서비스를 다양한 네트워크 환경에서도 원활하게 서비스하고 환자와 의사의 연결 리스트를 효율적으로 제어하며 서비스를 할 수

있도록 다중접속 제어서버를 구현하였다. 본 논문에서 구현한 다중접속 제어서버는 응급 원격진료 서비스를 이용하고자 하는 구성원들의 통신망 상태와 특성을 분석하여 안정적으로 서비스 할 수 있도록 시스템을 설계하고 평가하는데 목적을 두었다.

핵심되는 말 : 응급 원격진료 시스템, 다자간 서비스, 이기종 네트워크, 다중 제어 서버

이기종 네트워크 환경에서 다중 응급 원격진료 모니터링 시스템 설계 및 평가

<지도교수 유선국>

연세대학교 대학원 의과학과

김 도 윤

Ⅰ. 서론

점진적인 통신망의 발달로 도서지역, 지방 중소병원, 보건소 및 교도소와 1, 2차 의료기관과 3차 의료기관간의 원격진료, 원격자문, 원격진단 서비스를 구축하여 원거리에 있는 환자들에게도 보다 높은 의료서비스를 제공하기 위하여 많은 노력을 기울이고 있다[1-2]. 응급 원격진료 시스템은 환자와 의사가 근거리 또는 원거리에서 통신망에 의존해서 환자의 상태를 보고 진단 및 처방을 할 수 있어야한다. 응급 원격진료 시스템을 이용하기 위해서는 환자의 의료데이터의 송수신이 안정적으로 서비스 가능해야하며, 데이터의 왜곡이나 끊김 불안정한 전송을 피하고 환자에게 최대한 빠르고 정확하게 응급조치를 취할 수 있도록 도와주는 역할을 해야 한다[3-5].

응급 원격진료 시스템을 운용하는데 있어서 가장 중요한 것은 통신망이다. 기존에는 정해진 장소에서 원격진료 서비스를 이용해야 하는 불편함이 있었지만, 최근에 Wibro(Wireless BroadBand Internet), HSDPA(High Speed Down Link Packet Access)와 같은 무선 통신망 신기술이 도입되어

1Mbps ~ 3Mbps에 달하는 높은 대역폭(Throughput)을 제공하는 기술을 응급 원격진료 시스템에 적용하여 무선 환경에서도 환자의 데이터를 안정적으로 전송 할 수 있어 응급 원격진료 서비스의 질을 높였다. 즉 응급 원격진료 서비스에 적용 할 수 있는 통신망이 다양해 졌으며, 시간적, 공간적, 이동적인 제약 없이 전문의는 원격으로 환자를 진료 및 진단을 할 수 있고 환자에게 필요한 조취를 취할 수 있게 되었다[6-8].

현재 상용화 되어있는 유무선 통신망은 실시간으로 멀티미디어 데이터를 전송 및 수신하는데 문제는 없지만, 지역적 네트워크 환경에 따라 음영지역과 각각의 통신망의 대역폭 차이로 원활한 통신을 하는데 어려움이 있다. 따라서 응급 원격진료 시스템을 이용 할 때 적합한 통신망을 선택하는 것은 중요한 문제이다. 기존의 원격진료 서비스는 유무선 네트워크 환경에서 1대1 통신 방식으로 제공되었다. 그러나 1대 1 통신 방식은 응급 원격진료 서비스를 받고자 하는 환자의 연결 순서에 따라 순차적으로 원격진료가 수행되는 방식이므로 응급환자 발생했을 때 서비스를 원활하게 제공하지 못하고, 협력 진료가 필요한 환자를 다수의 전문의가 환자의 상태를 동시에 모니터링 하지 못하는 단점을 가지고 있다. 또한 원격진료를 이용하는 구성원들은 Wibro, HSDPA의 무선 통신망과 VDSL, ADSL과 같은 유선통신 망을 이용하여 원격진료 서비스를 이용한다. 하지만 각각의 통신망이 가지고 있는 대역폭의 차이와 공인IP, 사설IP의 특성 때문에 원격 서비스의 제한을 받거나 서비스를 이용하는데 어려움이 있다[6-10].

응급 원격진료 시스템을 원활하게 서비스하고 제어하기 위해서 다중접속 제어서버를 구현하였다. 다중접속 제어서버는 다수의 환자와 전문의 접속 리스트를 관리하고 응급 원격진료 서비스를 이용하고자 하는 구성원들의 통신망의 특성에 맞게 서비스 정책을 결정한다. 이질적인 네트워크 환경에서도 다수의 환자와 전문의에게 고품질의 응급 원격진료 서비스를 제

공할 수 도록 설계하였다. 즉, 이기종 네트워크 환경 및 이기종 원격진료 장치에서도 실시간으로 환자의 정보를 다수의 전문의에게 원활하게 제공할 수 있도록 다중 제어서버 시스템을 설계하고 성능을 평가하고자 한다.

Ⅱ. 재료 및 방법

1. 응급 원격진료 시스템(High-quality Multimedia Real-time Emergency Telemedicine)

그림1. 응급 원격진료 시스템(환자측, 의사측) 어플리케이션

그림1는 환자와 전문의가 사용하는 응급 원격진료 시스템 어플리케이션이다. 응급 원격진료 시스템은 다음과 같은 기능을 제공한다.

① 환자의 생체신호 모니터

② 환자의 고화질 영상 모니터

③ 환자측과 전문측의 컨퍼런스 모니터

④ 채팅 모니터

⑤ 파일 전송 모니터

⑦ 메뉴 및 프로그램 설정 메뉴

2. 전체 시스템 구성도

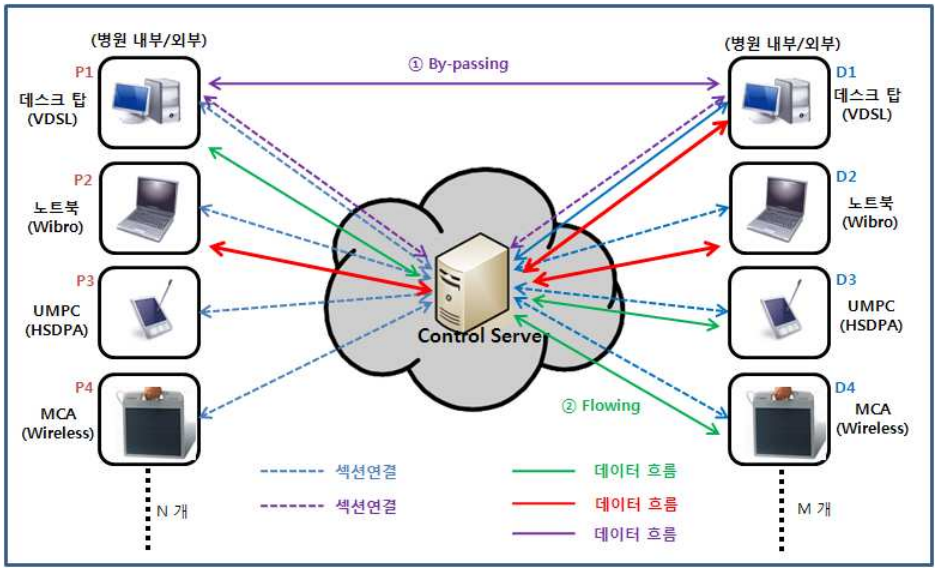

그림2. 다중접속 제어 서버를 이용한 응급 원격진료 서비스 시스템 구성도

다중 제어 서버(Multiple Control Server)를 이용한 응급 원격진료 서비스 시스템 전체 구조는 그림2와 같다. 응급 원격진료 시스템을 이용하는 전문의와 환자가 사용하는 통신망은 그림2에서 보는 봐와 같이 무선통신망 Wibro(Wireless Brodband Internet), HSDPA(High Speed Downlink Packet Access)와 유선통신망 VDSL(Very High-Data Rate Digital Subscriber Line), ADSL(Asymmetric Digital Subscriber Line) 그리고 병원 내부에서 사용하는 통신망을 이용하여 이기종 네트워크 환경에서 응급 원격진료 서비스를 가능하도록 구성되어 있다.

이기종 네트워크 환경에서는 각각의 통신망의 대역폭과 공인 IP, 사설 IP의 망의 특성이 서로 다르므로 응급 원격진료 서비스를 이용하고자 하는 환자와 전문의 측에서 원활한 서비스가 가능 하도록 디자인 되어야한다. 다중 접속제어 서버는 다수의 환자와 전문의가 연결을 하면 각각의 연결

리스트를 순차적으로 관리하고 통신망의 특성에 맞게 서비스 정책을 결정할 수 있도록 구현되어야 한다.

3. 네트워크 환경

응급 원격진료 시스템을 다중 제어서버를 통하여 서비스 할 수 있도록 구현한 통신망 환경은 Wibro, HSDPA, VDSL, ADSL, 병원망을 사용하였다.

가. Wibro

Wibro는 삼성전자와 한국전자통신연구원이 개발한 무선 광대역 인터넷 기술이다. 처음엔 고속 데이터 통신 기술을 가리키는 용어로 창안된 것이지만, 통신업체에서 기술명을 서비스명으로 이용하면서 기술 이름보다 서비스 이름으로 더 잘 알려져 있다. 현재 국내 사업자로 KT와 SK텔레콤이 선정되어, 2006년 6월 30일부터 국내에 서울과 경기도의 일부지역에서 세계 최초로 상용화 서비스가 시작되었다. 2007년 10월 18일 세계최초로 개발한 Wibro로 기술이 국제전기통신연합 전파총회에서 'IMT-2000'으로 통칭되는 3세대 이동통신(3G)의 6번째 국제표준으로 채택됐다.

표 1 와이브로와 하향고속패킷접속의 비교

구분	와이브로	하향고속패킷접속
서비스 개요	무선 인터넷 접속 서비스	이동통신 서비스의 데이터 다운로드 서비스
기반	인터넷 네트워크	이동통신 네트워크
다운로드 속도	3Mbps	2Mbps
서비스 사업자	KT, SKT	SKT, KTF

(1) 통신 네트워크 구성

(가) RAS

Wibro 기지국으로 무선접속이 가능하도록 하며, 무선자원의 효율적 관리, 제어 및 디지털 신호처리 기능을 한다. 이동성(핸드오프)을 지원하며, 하향링크 멀티캐스트를 하며, QoS제공과 인터페이스 기능, RF 송수신과 동기화가 진행된다.

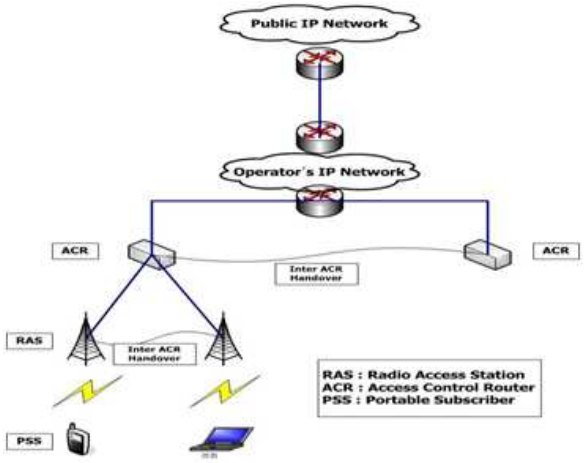

그림3. Wibro 통신 네트워크 구성

(나) ACR

ACR시스템이란 무선 액세스 스테이션과 연동을 통해 40G Cell Switch를 하여 다수의 DSP(Data Processing Shelf)를 하나로 결합하는 역할을 한다. 또한 이동 단말의 이동성 관리와 통계 정보의 생성 및 통보, QoS제고, 인증과 보안 및 무선자원 관리 및 제어를 한다.

나. HSDPA

3세대 비동기식 이동통신기술 표준화 기구인 3세대파트너십프로젝트

(3GPP:3rd Generation Partnership Project)가 2002년 3월 발표한 릴리스 5의 핵심기술인 고속데이터 패킷접속은 W-CDMA표준에서 기반의 데이터 서비스를 가리킨다. 이 기술을 사용하면 W-CDMA보다 5배 이상 빠른 속도로 통신할 수 있다. 다운링크 속도는 최대 14.4Mbps이다. 기지국에 대한 별도의 투자 없이 W-CDMA 시스템을 개량하는 방식으로 서비스를 제공할 수 있다는 것도 장점이다.

(1) 통신 네트워크 구성

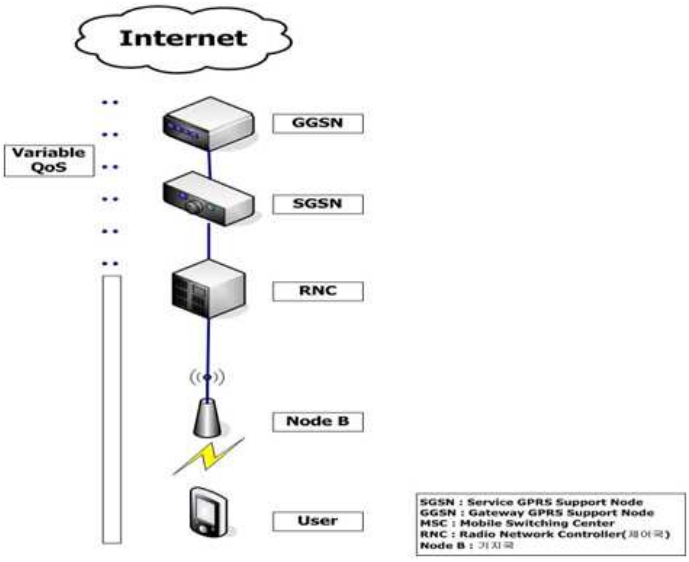

그림4. HSDPA 통신 네트워크 구성

(2) 통신원리

HSDPA은 사용자가 Signal-to-Noise Ratio(SNR) 값을 기반으로 다운링크 채널 상태가 가장 좋은 기지국을 선택하여 기지국으로부터 수신한 패킷이 제대로 전송되었는지에 대한 ACK 또는 NACK 정보(1bit)와 다운링

크 채널에 대한 채널 상황을 나타내는 Channel Quality에 대한 CQI 값(5bit)을 다운링크 전송을 위한 signaling 정보로 전송하는 업링크 채널이다. 기지국과 사용자는 하나의 HS-DPCCH를 통해 기지국으로부터 사용자가 패킷을 전송 받기 위해 필요한 제어 정보를 송신하기 위한 다운링크 채널이다.

다. ADSL

ADSL은 전화국과 각 가정이 직접 1:1로 연결되며 전화국에서 사용자까지 데이터가 내려가는 하향의 경우에는 일반적으로 최저 1.5Mb 이상의 고속 데이터 통신이 가능하고, 반대로 사용자로부터 전화국까지 상향 신호는 상당히 느리다. 따라서 상하향이 같은 대칭형 서비스가 아닌 비대칭형 서비스라고 한다. 장점은 전화선이나 전화기를 그대로 사용하면서도 고속 데이터 통신이 가능할 뿐만 아니라 한 전화선으로 일반 전화통신과 데이터 통신을 모두 처리할 수 있다는 것이다.

기존 모뎀은 전화 데이터를 통신을 동시에 사용할 수 없다. ISDN은 동시 사용이 가능하지만 데이터 통신 속도가 절반으로 떨어진다. 하지만 ADSL은 음성통신은 낮은 주파수 대역을 이용하고 데이터 통신은 높은 주파수 대역을 이용하기 때문에 혼선이 일어나지 않고 통신속도도 떨어지지 않는다. 그러나 쌍방향 서비스로 이루어지는 원격진료나 원격교육 같은 서비스에서는 효율이 떨어진다는 단점도 있다.

라. VDSL

일반 가정에서 기존의 전화선을 이용해 양방향으로 빠른 속도로 전송이 가능하고, 많은 양의 데이터를 초고속으로 전송할 수 있어 광섬유의 가정화를 위한 최종 단계로 평가되는 기술이다. 국낸에서는 2000년 1월 처음으

로 개발되어 2002년부터 상용화되었다. ADSL과는 달리 가입자에게 필요한 데이터만을 전송하고, 기존의 전화선을 그대로 이용하기 때문에 공급가격이 저렴할 뿐 아니라, 설치공간도 덜 차지한다는 장점이 있다. 이 기술은 실시간 비디오와 고화질의 영상회의를 위하여 제공되는 고속 유선통신 규격으로 2.74km 미만의 전화선 서비스 지역에 적합하다. 그리고 공공 네트워크나 기업용 데이터 서비스 같은 저렴하면서도 대용량 데이터 처리가 필요한 곳에 유용하다.

4. 다중접속 제어 서버를 이용한 응급 원격진료 서비스 통신 정책(Policy)

응급 원격진료 시스템을 사용하는 구성원들(환자, 전문의)의 통신망의 환경은 표2와 같다. 다중 제어서버를 통해 응급 원격진료 서비스를 이용하는 구성원들에게 안정적으로 서비스하기 위해서 각각의 통신만의 대역폭 적합한 정책을 결정하여 서비스해야 한다. 본 논문에서는 2가지 정책을 설계하였다.

표 2. 응급 원격진료 서비스에 사용한 통신망 환경

	네트워크 망	특 성
①	내부 유선망	병원 내부에서 사용하는 네트워크 망, 내부 사설망(Private Network)
②	내무 무선망	
③	외부 유선망	VDSL, ADSL 같은 네트워크 망, 공중망(Public Network)
④	외부 무선망	
⑤	Wibro 망	광대역 무선 망(일부 수도권 지역 서비스), 공중망(Public Network)
⑥	HSDPA SKT망	무선 망(휴대폰이 사용가능한 지역에서 서비스 가능) 무선 사설망(Wireless Personal Area Network)
⑦	HSDPA KTF망	

가. By-passing 정책

By-passing 정책은 응급 원격진료 서비스를 제공 할 때 전문의 측 시스템과 환자 측 시스템이 다중접속 제어서버를 거치지 않고, 응급 원격진료 서비스를 이용하는 구성원들 상호간에 연결을 맺어 환자 데이터를 송수신 하는 정책이다. By-passing 정책은 전문의 측 통신망이 공인된 통신망과 대역폭이 환자 측 통신망 보다 대역폭이 크고 1(환자):1(전문의) 응급 원격진료 서비스를 이용 할 때 적합한 정책 이다. 또한 By-passing 정책은

다중제어 서버에 부담을 줄이고 시스템의 효율성을 높여 서비스의 질을 높이기 위해 수립된 정책 이다. 이때 다중접속 제어서버는 환자와 전문의 서비스 연결 리스트를 관리한다.

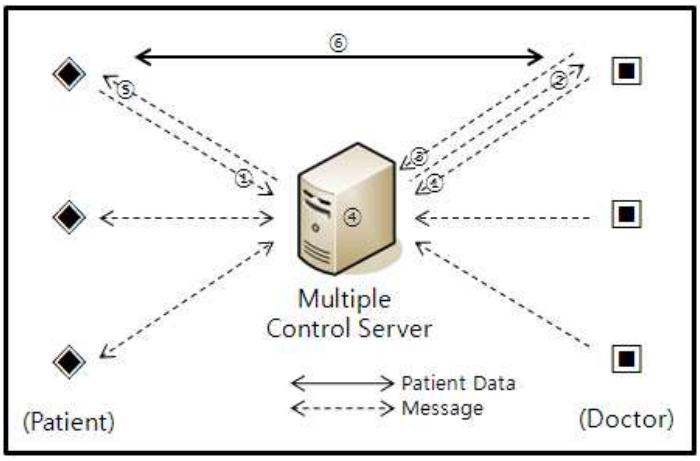

그림5. By-passing 흐름도

By-passing 정책 동작 순서는 다중접속 제어서버에 ①과 같이 환자와 전문의 측이 다중접속 제어서버에 접속을 하면 환자와 전문의 연결 리스트를 관리하고 환자와 전문의 측의 통신망과 장치타입을 저장한다. ②전문의는 다중접속 서버에 접속되어 있는 환자의 리스트를 제어 서버로부터 요청을 하면 다중접속 제어서버는 전문의가 요청한 환자의 이름과 IP정보를 전송한다. ③전문의는 원격진료 및 원격자문을 하고자 하는 환자를 선택고, 다중접속 제어서버로 선택된 환자의 정보가 전송된다. ④다중접속 제어서버는 전문의 측과 선택한 환자의 통신망을 비교하여 정책을 결정하고 결정된 정책이 By-passing이면 ⑤다중접속 제어서버는 환자 측으로 전문의 측의 공인 IP를 전송한다. 환자 측에서는 다중접속 제어서버로부터 넘겨받은 전문의 측의 공인 IP를 가지고 ⑥직접 전문의 측의 원격진료 시스템으로

연결을 하고 원격진료 및 자문을 한다. 이때 다중접속 제어서버는 환자와 전문의 간에 원격 진료 서비스가 수행되는 것을 다중접속 제어서버는 모니터링을 통하여 환자와 전문의 간에 접속 리스트를 관리한다.

나. Flowing 정책

Flowing 정책은 다중접속 제어서버를 통해서 응급원격 진료 서비스를 제공 할 때 환자와 전문의 연결을 맺어 환자의 데이터를 다중접속 제어서버를 통해 서비스하는 정책이다. 전문의 측 통신망이 사설망인 경우와 환자 측 통신망 보다 저 대역폭을 가지고 있고 1(환자):M(다수의 전문의) 진료서비스를 이용할 때 사용되는 정책이다.

Flowing 정책을 이용하여 다중접속 제어서버를 이용하여 응급원격 진료서비스를 수행할 때 동작 순서는 ①과 같이 다중접속 제어서버에 환자와 전문의가 접속을 하면 다중접속 제어서버는 환자와 전문의 리스트를 관리하고 통신망 정보와 장치타입을 저장한다. ②전문의는 다중접속 제어서버로 접속되어 있는 환자의 리스트를 요청하면 다중접속 제어서버는 전문의 측으로 환자의 리스트 정보를 전송한다. ③전문의 측은 응급원격 진료 및 진단을 할 환자를 선택하면 다중접속 제어서버로 전송이 되고 ④다중접속 제어 서버에서는 전문의 측과 환자 측의 통신망을 비교하여 정책을 결정하고 결정된 정책이 Flowing이면 ⑤다중접속 제어서버는 환자 측으로 부터 환자 정보 데이터를 받아 ⑥응급원격 진료서비스를 이용하는 전문의 측으로 환자의 데이터 서비스 한다.

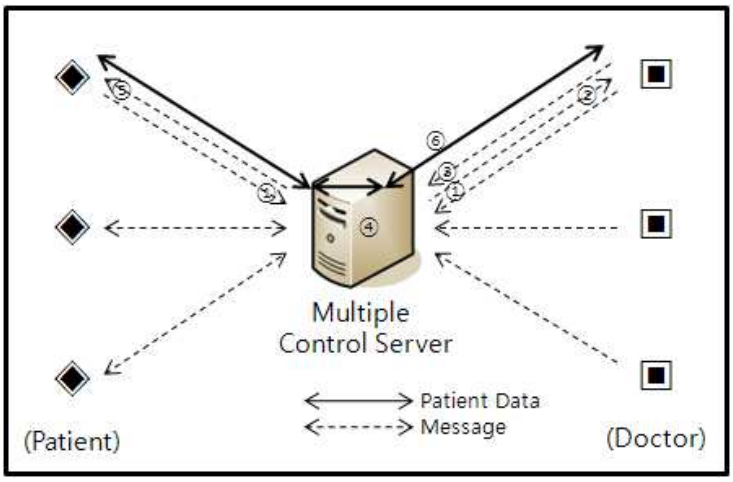

그림6. Flowing 흐름도

5. 다중접속 제어서버 프로그램 개발 환경

가. 개발언어

다중접속 제어서버는 윈도우즈 XP기반에 Microsoft Visual C++ 6.0을 이용하여 구현하였다.

6. 서버모델

다중 접속 서버란 여러 클라이언트들이 동시에 접속할 수 있는 서버를 의미한다. 조금 더 구체적으로 말하면, 단순히 동시에 접속을 하는 것이 아니라 서비스를 동시에 받을 수 있는 서버를 의미한다. 다중 접속 서버 구현 방법으로는 일반적으로 세 가지를 언급하게 된다.

① 프로세스 생성을 통한 멀티태스킹(Multitasking) 서버의 구현

② select 함수에 의한 멀티플렉싱(Multiplexing) 서버의 구현

③ 쓰레드 기반으로 하는 멀티쓰레딩(Multithreading) 서버의 구현

가. 멀티태스킹(Multitasking) 서버 모델

(1) 멀티 프로세스

멀티 프로세스 모델은 다중 접속 처리를 위한 소켓 프로그래밍에서 가장 쉽게 그리고 가장 간단하게 다중 접속 처리를 구현할 수 있는 방법이다. 소켓 프로그래밍의 다중 접속 처리 모델의 기초는 멀티 프로세스 모델이다. 멀티프로세스 모델은 클라이언트의 통신 요청이 들어오면 새로운 프로세스를 생성한다. 이처럼 새롭게 생성된 프로세스가 클라이언트와의 통신을 담당하기 때문에 많은 클라이언트가 통신을 요청하더라도 모두 처리해 줄 수 있는 것이다. 멀티 프로세스 모델의 최대 장점은 구현하기가 간단하다는 것이다. 따라서 동시 접속 수가 적은 서버를 빠른 시간에 개발하고자 할 때 손쉽게 선택할 수 있는 방법이다. 하지만 이런 멀티 프로세스 모델을 사용할 때는 다음과 같이 몇 가지 주의해야 할 점이 있다.

① 클라이언트의 접속이 많아질 경우 시스템 성능이 저하 될 수 있다.

모든 다중 접속 처리 서버의 공통된 단점이기는 하지만 멀티 프로세스 모델의 경우, 클라이언트의 다중 접속 처리를 위해서 프로세스를 사용하기 때문에 클라이언트의 접속이 많아지면 너무 많은 프로세스가 생경서 전체적인 시스템의 성능을 저하시킬 수 있다.

② 프로세스간의 데이터 공유가 불편하다.

프로세스는 서로 메모리가 분리되어 있기 때문에 프로세스 사이에 데이터를 공유해야 할 필요가 있는 경우에는 프로세스 사이에 데이터를 공유할 수 있게 해주는 IPC(Inter-Process Communication)를 사용해야 하는 번거로움이 생긴다.

③ 데이터 공유를 위해서 IPC를 사용하면 프로그램의 복잡성이 증가 할 수 있다.

IPC에는 시그널, 파이프, 메시지 큐, 공유 메모리 등이 있다. 이런 IPC를 프로그램에 추가하면 프로그램이 복잡해져서 가독성이 떨어지는 원인이 된다.

(2) 멀티 프로세스 모델 흐름

① 서버에서 listen함수를 사용해서 접속 대기 상태로 클라이언트의 접속을 기다린다.

② 클라이언트가 접속되면 접속을 허가한 후 새로운 프로세스를 만들어서 해당 클라이언트와 통신한다.

③ 접속에 성공한 클라이언트 프로그램은 서버로 문자열 데이터를 전송한다.

④ 문자열 데이터를 받은 서버는 해당 문자를 그대로 클라이언트 프로그램에서 전송한다.

⑤ 문자열을 돌려받은 클라이언트 프로그램은 그 문자를 출력한다.

⑥ 문자열을 전송을 주고받는 작업을 반복해서 진행한다.

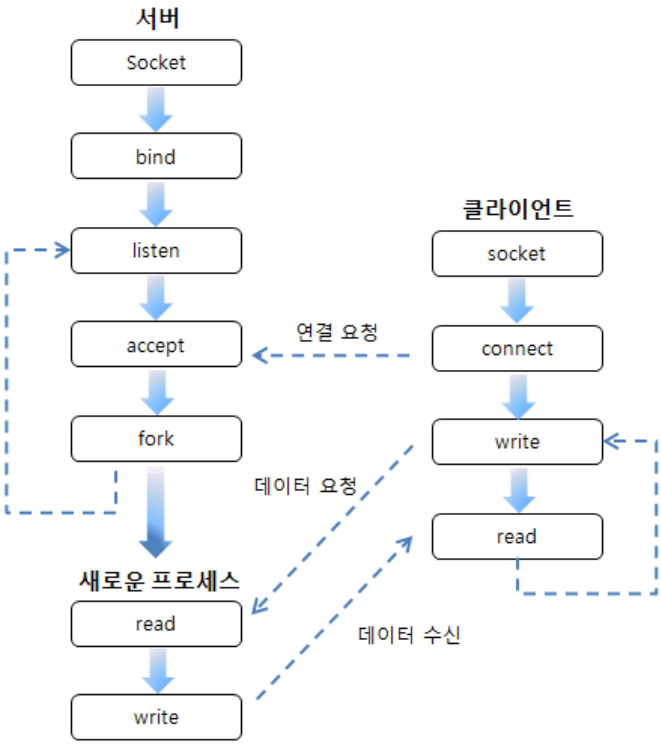

그림7. 멀티 프로세스 모델의 함수 사용 흐름

(3) 프로세스 ID

모든 프로세스는 운영체제로부터 할당된 프로세스 ID를 가지게 된다. 프로세스 ID란 프로세스마다 할당되는 유일한 숫자이면 2에서부터 32768을 범위로 가지게 된다. 숫자 1은 운영체제가 시작되자마자 실행되는, init 프로세스에게 할당되기 때문에 우리가 만들어 내는 프로세스는 1이라는 값을 가질 수 없다.

(4) Fork 함수 호출을 통한 프로세스 생성

fork 함수는 호출한 프로세스의 복사본 프로세스를 생성해 낸다. 만약에 함수 호출이 실패 한다면 -1을 리턴 한다. 그러나 성공하는 경우에는 프로

세스가 둘로 나뉘어서 원본 프로세스(복사의 대상이 되는 프로세스)와 복사본 프로세스(복사되어 새롭게 생성된 프로세스)에게 전달되는 리턴 값이 달라진다. 따라서 우리는 리턴 값을 이용해서 프로그램의 흐름을 나눌 수 있다.

```
#include <sys/types.h>
#include <unistd.h>

pid_t fork (void);
```

성공 시 프로세스 ID, 실패 시 -1을 리턴

그림8. fork 함수

또한 함수 호출이 성공하면, 복사본에 해당하는 프로세스는 원본 프로세스의 모든 메모리 공간(데이터 영역, 힙(heep) 및 스택(stack)을 그대로 복사하게 된다. 따라서 똑같은 프로그램을 완전히 독립된 프로세스가 실행하는 모습을 갖추게 된다.

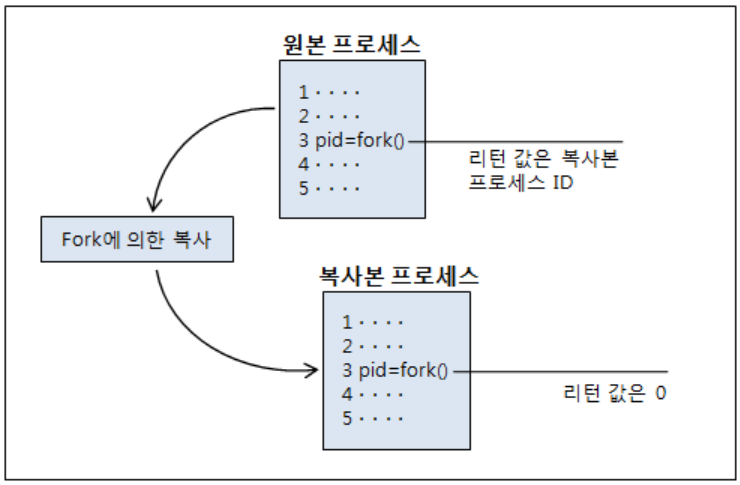

그림9. fork 함수에 의한 프로세스 복사

여기서 fork 함수를 호출하는 원본 프로세스를 '부모 프로세스(Parent Process)'라 하고, fork 함수 호출에 의해 복사된 프로세스를 '자식 프로세스(Child Process)'라 한다.

fork함수 호출 성공 시, 부모 프로세스는 자식 프로세스의 ID를 리턴받고 fork 함수 호출 이후를 실행하게 되고, 자식 프로세스 역시 0을 리턴받고 fork 함수 호출 이후를 실행하게 된다. 그러나 부모 프로세스와 자식 프로세스는 같은 프로그램 코드를 서로 다른 메모리 공간에서 실행하기 때문에 데이터의 공유는 일어나지 않는다. 말 그대로 완전히 독립된 두 개의 프로그램이 실행된다고 생각하면 된다.

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    pid_t pid;
    int data=10;
          ①
    pid=fork();
    if(pid==-1)
        printf("fork 실패, 프로세스 id : %d ₩n", pid);
    printf("fork 성공, 프로세스 id : %d₩n", pid);

  ②if(pid==0)           /* 자식 프로세스라면 */
        data+=10;
    else                /* 부모 프로세스라면 */
        data-=10;

    printf("data : %d ₩n", data);
    return 0;
}
```

그림10. 프로세스 생성 예제

- ①에서 fork 함수 호출을 통해서 자식 프로세서를 생성하고 있다. 따라서 ①을 분기점으로 부모 프로세서의 경우 자식 프로세스 ID를 리턴받고

이후를 실행하게 되고, 자식 프로세스인 경우 0을 리턴받고 이후를 실행하게 된다. 즉 실행의 흐름이 두 갈래로 나뉘게 되는 것이다.

- ②을 보면 자식 프로세서의 경우 data변수에 10을 더하고, 부모 프로세스의 경우 10을 빼고 있다. 그러나 서로 독립된 메모리 구조를 지니고 있으므로 서로의 메모리 영역에 아무런 영향도 미치지 않는다. 즉 독립된 data 변수를 유지하고 있다.

(5) fork 함수 호출에 의한 파일 디스크립터 복사

그림11은 fork 함수 호출 이후의 상황을 보여 주고 있다. fork 함수 호출 이후에 serv_sock과 clnt_sock가 복사되었음을 볼 수 있다(clnt_sock은 클라이언트와 연결되어 있는 소켓의 파일 디스크립터를 의미한다).

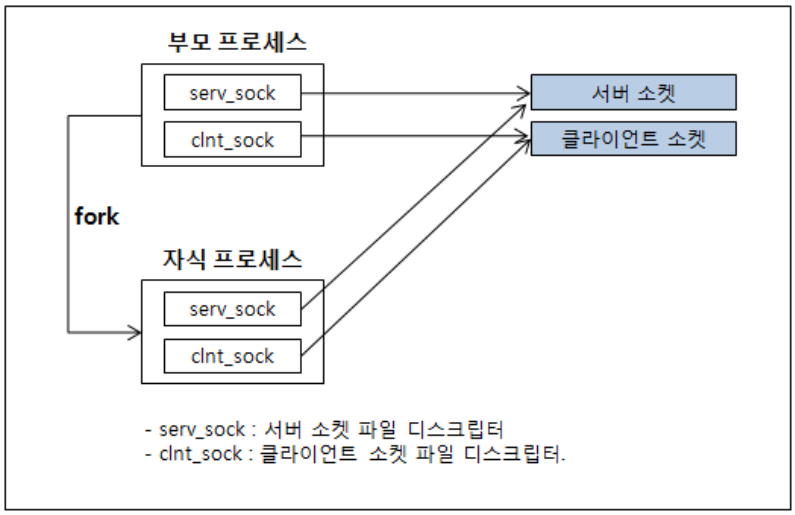

그림11. fork 함수 호출 이후의 파일 디스크립터 상황1

그러나 부모 프로세스에서는 clnt_sock이 필요 없고, 서버의 자식 프로세스에서는 serv_sock이 필요 없으므로 바로 종료해 준다. 그림12는 이러

한 상황을 설명한다.

참고로, 그림11과 같이 파일 디스크립터가 복사되는 경우 하나의 소켓에 두 개의 파일 디스크립터가 존재하게 된다. 이런 경우 하나의 파일 디스크립터가 종료되어도 소켓은 종료되지 않는다. 반드시 두 개의 파일 디스크립터가 모두 종료되어야 소켓이 종료된다.

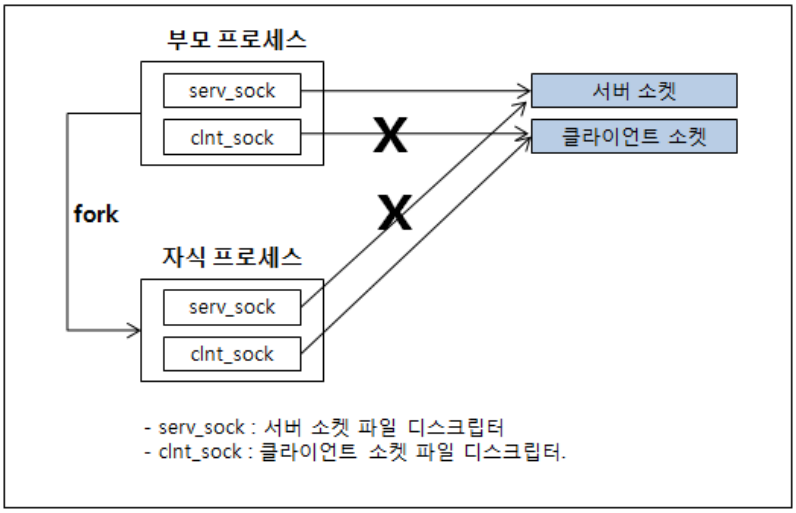

그림12. fork 함수 호출 이후의 파일 디스크립터 상황2

따라서 fork 함수 호출 이후에 그림11과 같은 모습으로 프로그램을 유지한다면, 나중에 자식 프로세스가 종료되면서 클라이언트 연결 소켓을 종료하려 해도 종료되지 않고 계속 남아있게 된다. 왜냐하면 복사된 파일 디스크립터 하나가 아직 남아 있기 때문이다. 따라서 fork 함수 호출 이후에 반드시 파일 디스크립터를 종료해서, 그림12와 같은 모습이 되도록 서버를 구현해야 한다.

```
while(1)
{

        if((pid=fork())==-1)// fork 실패시
        {
                close(clnt_sock);
                continue;
        }

        else if(pid>0)      // 부모 프로세스의 경우
        {
                puts("연결 생성");
                close(clnt_sock);
                continue;
        }

        else                // 자식 프로세스의 경우
        {
                close(serv_sock);

                /* 자식 프로세스의 처리 영역 : 데이터 수신 및 전송*/
                while((str_len=read(clnt_sock, message, BUFSIZE)) !=0)
                {
                        write(clnt_sock, message, str_len);
                        write(1, message, str_len);
                }

                puts("연결 종료");
                close(clnt_sock);
                exit(0);
        }
}
```

그림13. fork 함수 호출 이후의 파일 디스크립터 상황2 코딩

(6) 프로세스 기반의 다중 접속 서버의 구현 모델

1:1 서버의 경우 한번에 하나의 클라이언트에 대한 연결 요청만 수락했었다. 즉 동시에 둘 이상의 클라이언트에게 서비스를 해 주지는 못했다. 그러나 프로세스 기반의 다중 접속 서버의 모델을 이용하여 다중 접속 서버를 구현 할 수 있다.

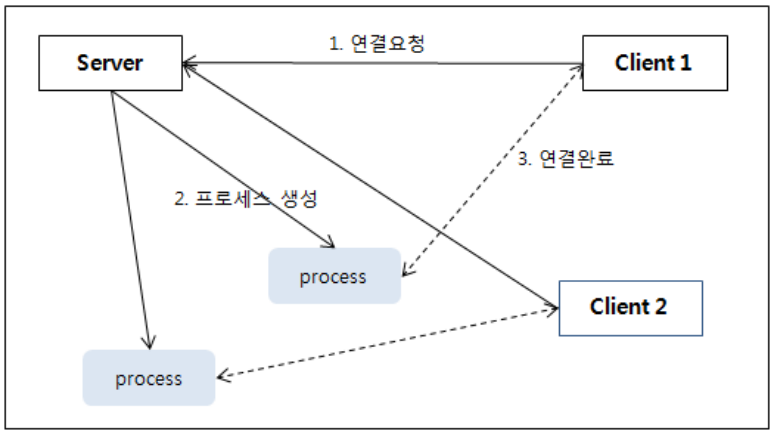

그림14. 다중 접속 서버 모델

나. 멀티스레드 서버 모델

(1) 멀티스레드

멀티 스레드 모델은 다중 접속 처리를 위해서 스레드를 생성해서 처리한다. 멀티 스레드 모델이 멀티 프로세스 모델에 비해서 자원을 적게 쓰고 클라이언트의 요구를 처리하는 이유는 스레드를 사용하기 때문이다. 스레드는 스스로 프로그램이 되어서 실행할 수는 없지만 프로그램 안에서만 동시에 실행시킬 수 있는 작업 단위이다. PC에서 여러 가지 프로그램을 동시에 실행하기 위해서는 프로세스를 사용하지만 프로그램 안에서 여러 가지 작업을 동시에 하고 싶으면 스레드를 이용해서 각각의 작업을 실행한다.

스레드는 프로세스와 비교해서 공통점과 차이점을 모두 가지고 있다. 스레드와 프로세스의 공통점은 자신의 스택을 가지고 주어진 코드를 실행한다는 것이다. 스레드도 프로세스와 마찬가지로 자신만의 스택을 가지고 자신의 코드를 실행한다는 것이다. 스레드도 프로세스와 마찬가지로 자신만의 스택을 가지고 자신의 코드를 실행한다. 그러나 스레드와 프로세스의

중요한 차이점은 프로세스는 새로운 프로세스가 생성되면 메모리 역시 다른 영역에 생성해서 사용하지만 스레드는 메모리를 같이 공유해서 사용한다는 점이다.

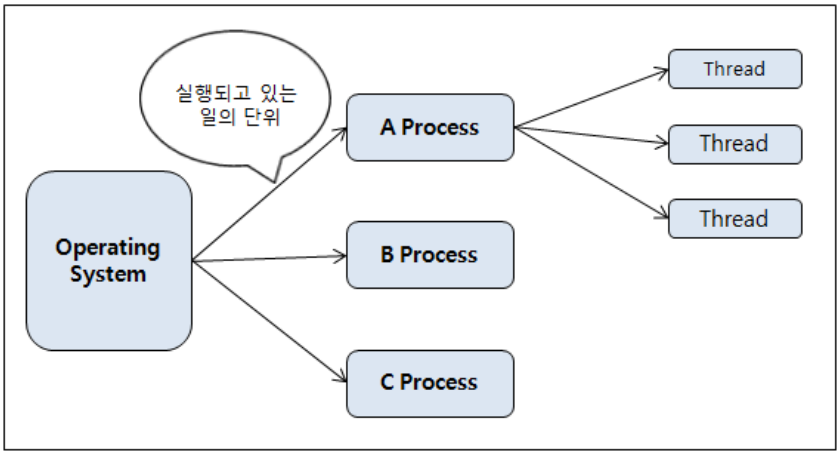

그림15. 프로세스와 스레드

스레드를 사용하면 여러 가지 작업을 동시에 할 수 있는 장점이 있다. 채팅을 하면서도 파일을 전송 하면서 계속적으로 채팅을 수행 할 수 있다.

시스템에서는 시스템 자원을 적게 사용하는 것을 가리켜 '경량 프로세스'라고 명명하는 경우도 있다. 이렇게 스레드가 가벼운 이유는 메모리에서 찾아 볼 수 있다. 스레드는 메모리를 서로 공유해서 사용하기 때문에 새로운 스레드를 생성할 때 메모리 할당에 대한 로드가 적어 생성 시간이 프로세스에 비해서 짧다. 그리고 메모리를 서로 공유하기 때문에 프로세스처럼 스레드 사이의 통신에 IPC같은 복잡한 루틴을 사용하지 않고 간단하게 스레드 사이의 통신을 구현할 수 있다.

또 하나의 장점을 들자면 스레드의 '컨텍스트 스위치'가 프로세스의 컨텍스트 스위치보다 빠르다는 것이다. 스레드의 컨텍스트 스위치에 소모되

는 시간이 프로세스의 컨텍스트 스위치에 소모되는 시간보다 적게 들기 때문에 결과적으로 스레드의 컨텍스트 스위칭이 프로세스의 컨텍스트 스위칭에 비해서 빠르게 동작한다. 이것은 동시에 여러 클라이언트의 요구를 처리할 때 얼마나 많은 클라이언트의 요구를 짧은 시간 안에 처리할 수 있는지에 대해서 영향을 미친다. 물론, 컨텍스트 스위치 시간이 빠른 스레드가 프로세스에 비해서 더 나은 성능을 보이게 된다. 스레드는 속도가 빠르고 가볍다는 장점이 있지만 다음과 같은 몇 가지 단점도 있다.

① 한 스레드가 메모리를 잘못 사용하면 모든 스레드에 영향을 준다.
스레드는 메모리를 공유해서 사용하기 때문에 하나의 스레드에서 메모리를 잘못 사용하면 모든 스레드에 잘못된 영향을 줄 수 있는 문제점이 있다. 하지만 프로세스는 각자 다른 메모리 영역을 사용하기 때문에 하나의 프로세스가 잘못되더라도 해당 프로세스만 잘못된 동작을 한다.
② 하나의 머신에서만 사용할 수 있다.
스레드의 또 다른 단점으로 프로세스는 다른 머신에서도 프로세스를 동작시킬 수 있는 것에 비해서 스레드는 해당 프로그램 안에서만 실행 할 수 있기 때문에 단지 하나의 머신에서만 사용할 수 있다는 것이다.

이러한 단점에도 불구하고 가볍고 처리속도가 빠르다는 장점으로 인해 멀티 스레드 모델을 자주 사용한다. 따라서 이러한 장점과 단점을 잘 고려해서 자신의 프로그램이 멀티 스레드 모델에 적합한지 판단하는 것은 프로그래머의 몫이라고 할 수 있다.

멀티 스레드 모델은 전체적인 동작 구조가 멀티 프로세스 모델과 비슷하다. 차이점은 클라이언트의 접속 요청이 들어오면 멀티 프로세스 모델에서는 프로세스 하나를 새로 생성해서 접속한 클라이언트와의 통신을 처리

하게 하지만 멀티 스레드 모델에서는 스레드 하나를 새로 생성해서 클라이언트의 요구를 처리한다는 것이다. 멀티 스레드 모델의 서버 프로그램에서 클라이언트의 접속 요청을 처리하는 과정을 살펴보면 다음과 같다.

① 서버가 listen 함수를 호출한 상태로 대기한다.

② 클라이언트가 서버에 접속을 요청한다.

③ 서버에서 클라이언트와 통신할 스레드를 생성한다.

④ 스레드에서 클라이언트의 데이터를 수신한다.

⑤ 스레드에서 클라이언트에 데이터를 송신한다.

⑥ 읽을 데이터가 더 이상 없으면 스레드를 종료한다.

멀티 스레드 모델에서의 클라이언트 접속 요구 처리 방식이 멀티 프로세스 모델과 비슷하기 때문에 위 과정에서도 스레드를 생성한다는 것 외에는 멀티 프로세스 모델에서의 처리 방식과 유사한 과정을 보여주고 있다.

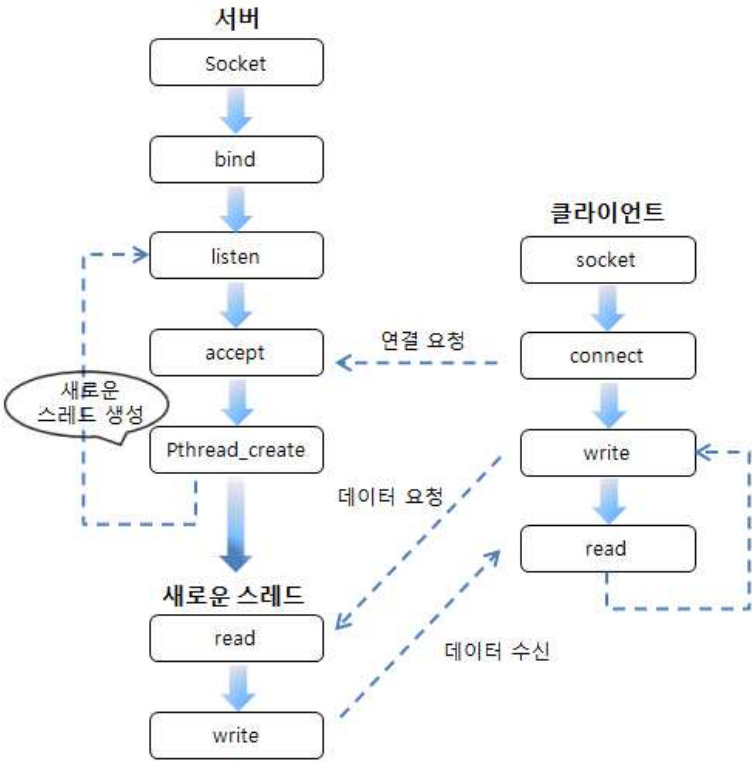

그림16. 멀티 스레드 모델

멀티 프로세스 모델과 비교해보면 멀티 프로세스 모델에서는 fork 함수를 호출해서 새로운 프로세스를 생성했지만 멀티 프로세스 모델에서는 pthread_create 함수를 호출해서 새로운 스레드를 생성하는 것을 볼 수 있다.

(2) 스레드 생성

```
#include <pthread.h>

int pthread_create(pthread_t * thread,
                   pthread_attr_t * attr,
                   void *(*start_routine)(void *),
                   void * arg
);
```

성공 시 0, 실패 시 이외의 값 리턴

그림17. 스레드 생성

• thread : 생성된 스레드의 ID를 저장할 변수의 포인터를 인자로 전달한다. 함수가 호출되고 나면 스레드가 생성되는데, 생성되는 모든 스레드는 프로세스처럼 ID를 할당받게 된다.

• attr : 생성하고자 하는 쓰레드의 특성(attribute)을 설정할 때 사용하는데, 일반적으로 NULL을 전달한다.

• start_routine : 자세히 보면 함수 포인터를 인자로 요구하고 있음을 알 수 있다. 리턴 타입이 void*이고 인자도 void*인 함수를 가리키는 포인터인데, 스레드가 생성되고 나서 실행해야 하는 함수 루틴을 설정해 주게 된다. 따라서 스레드가 생성되자마자 여기서 인자로 전달된 함수를 호출하게 될 것이다. 더불어 이 함수의 종료와 동시에 스레드도 소멸된다.

• arg : 스레드에 의해 호출되는 함수(start_routine 포인터가 가리키는 함수)에 전달하고자 하는 인자 값을 넘겨준다.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>

void *thread_function(void *arg);

int main(int argc, char **argv)
{
        int state;
        pthread_t t_id;
        void * t_return;

      ①state=pthread_create(&t_id, NULL, thread_function, NULL);
        if(state !=0)
        {
                puts("스레드 생성 오류");
                exit(1);
        }
        printf("생성된 스레드의 ID : %d \n", t_id);
        sleep(3);
        puts("main함수 종료");

        return 0;
}
  ②
void *thread_function(void *arg)
{
        int i;
        for(i=0; i<3; i++)
        {
                sleep(2);
                puts("스레드 실행 중");
        }
}
```

그림18. 스레드 생성 코딩

- ①에서 스레드를 생성하고 있다. 여기서 생성 된 스레드에 의해서 thread_function 함수가 호출 될 것이다.
- ②는 스레드가 호출할 함수 루틴이다. 2초를 주기로 “스레드 실행 중”이라는 메시지를 총 세 번 출력하게끔 되어 있다.

다. 멀티플렉싱 서버 모델

(1) I/O 멀티플렉싱 모델

I/O 멀티플레싱 모델(I/O Multiplexing Model)은 앞서 소개한 멀티 프로세스 모델과 멀티 스레드 모델 중에서 가장 효율이 좋은 방법이다. I/O 멀티플렉싱 모델이라는 이름에서 알 수 있듯이 I/O를 이용해서 소켓을 처리하기 때문에 기존의 프로세스나 스레드를 이용한 서버 소켓 프로그램에 비해서 좋은 효율을 낸다.

I/O는 Input/Output(입력/출력)을 의미하고 다른 말로는 입출력 장치라고도 한다. 우리가 자주 사용하는 키보드나 마우스 같은 장치를 I/O라고 하고 파일 시스템에서 파일을 읽고 쓰고 하는 것이나 소켓을 이용해서 데이터를 주고받는 것도 I/O라고 한다. 여기서는 소켓에서 파일을 주고받는 것에 대한 I/O를 의미한다.

멀티플렉싱(Multipliexing)은 우리말로 ‘다중화’라는 의미이다. 다중화는 통신에서 하나의 회선을 분할해서 각기 다른 신호를 동시에 송수신 할 수 있게 한다.

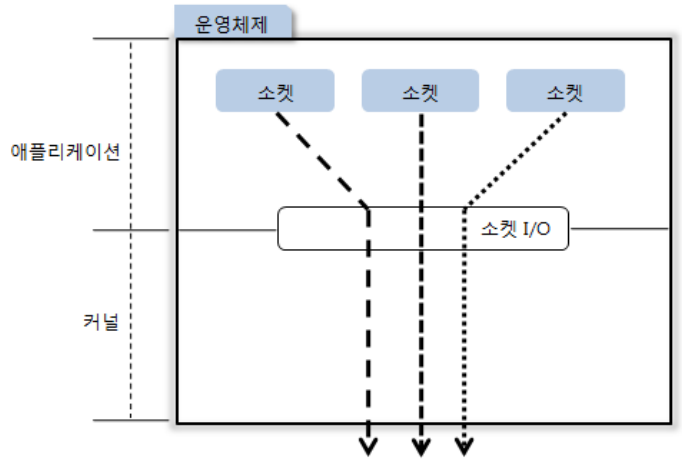

그림19. I/O 멀티플렉싱

위 그림과 같이 애플리케이션에서 생성된 소켓이 각자의 소켓 I/O를 이용해서 통신하는 것이 아니라 하나의 소켓 I/O를 통해서 통신하는 것을 I/O 멀티플렉싱 이라고 한다. 이런 I/O 멀티플레싱 방법을 이용해서 서버 소켓 프로그램에 걸리는 부하를 줄여준 것이 I/O 멀티플렉싱 모델이다.

I/O 멀티플렉싱 모델에서는 소켓의 I/O를 감시하고 있다가 소켓에 패킷이 들어오거나 패킷이 전송되는 I/O가 발생하면 즉시 감시하는 루틴에 I/O가 일어났다는 결과 값을 반환한다. I/O를 감시하는 루틴에서는 이 결과 값을 확인하고 해당 소켓의 동작에 맞는 함수를 호출하게 된다.

소켓의 I/O를 감시하기 위해서는 select 시스템을 콜을 사용한다. select 시스템 콜은 지정된 파일 디스크립터 테이블을 감시하고 있다가 I/O가 발생하면 결과 값을 반환한다. 파일 디스크립터 테이블은 1024개의 파일 디스크립터 배열로 되어 있다. 파일 디스크립터 테이블에 생성한 소켓을 지정하고 select 함수를 이용해서 감시하면 해당 소켓에 패킷이 도착했을 때 select 함수가 인식할 수 있게 된다.

하지만 select 함수는 파일 디스크립터 전체에서 I/O가 발생했다는 것을 아려줄 뿐 정확하게 몇 번째 파일 디스크립터에서 I/O가 발생했는지는 알려주지 않는다. 따라서 루프를 돌면서 파일 디스크립터 테이블의 몇 번째 파일 디스크립터에서 I/O가 발생했는지를 찾는 과정을 거친다.

여기서 현재 발생한 I/O의 종류를 구분해야 한다. 현재 발생한 I/O가 연결을 요청하는 것일 수도 있고 데이터가 수신되는 것일 수도 있기 때문이다. 발생한 I/O의 종류를 구분하기 위해서 하는 것이므로 accept 함수를 호출해서 새로운 연결을 받아들인다. 클라이언트 소켓인 경우에는 데이터가 수신되는 것이므로 데이터를 읽고 다시 클라이언트 프로그램에게 데이터를 전송하게 된다. 이 때 데이터가 더 이상 읽히지 않는다면 클라이언트가 연결을 종료한 것으로 간주하고 해당 클라이언트와 통신을 위해 사용하던 파일 디스크립터를 해제한다.

그림20. 파일 디스크립터

(2) select 함수의 기능과 호출 순서

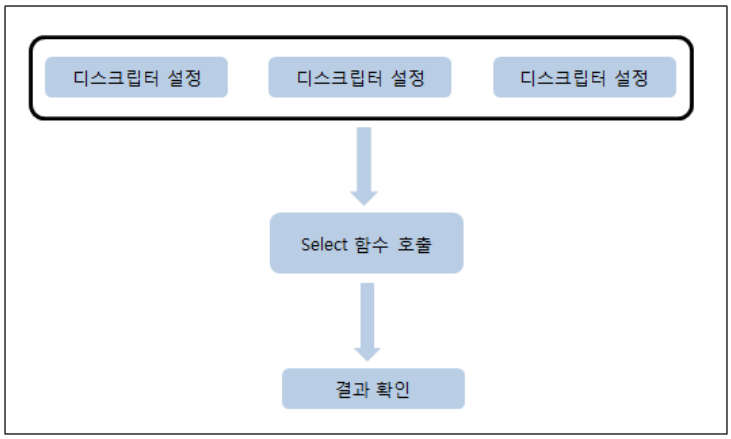

그림21. select함수 호출 과정

그림21은 select 함수를 호출해서 결과를 얻기까지의 과정을 간략하게 보여주고 있다. 그림에서 보면 select함수를 호출하기 전에 뭔가 준비를 해야 하고, 또 호출한 다음에도 결과를 확인 하는 과정이 따로 존재하고 있는 것을 볼 수 있다. 함수 호출 전에 디스크립터 설정, 검사범위 설정, 타임아웃 설정 과정을 거쳐야 함을 알 수 있다

(가) 디스크립터 설정

select 함수를 사용하게 되면, 여러 개의 파일 디스크립터를 동시에 관찰할 수 있다고 하였다. 파일 디스크립터를 관찰할 수 있다는 말은 결국 소켓을 관찰할 수 있다는 말과 같다. 일단 관찰하고자 하는 파일의 디스크립터들을 모아야 한다. 모을 때도 관찰되는 항목에 따라서 따로 모아야 한다.

• 수신할 데이터를 지니고 있는 소켓이 존재하는가?
(입력 버퍼에 데이터 존재)
• 데이터를 전송할 경우 블로킹되지 않는 소켓은 무엇인가?

(출력 버퍼에 충분한 여유 공간 존재)

• 예외 상황이 발생한 소켓이 있는가?

이렇게 관찰 항목이 셋이기 때문에 세 묶음으로 파일 디스크립터를 준비해야 한다. 파일 디스크립터를 세 묶음으로 모아놓기 위해서 사용되는 것이 fd_set 데이터 타입의 자료형이다. fd_set은 0과 1을 나타내는 비트들의 배열이라고 생각하면 된다.

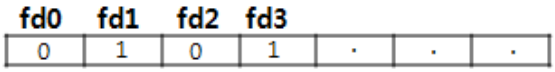

그림22. fd_set 자료형

그림22에서 보듯이 0과 1을 표현할 수 있는 비트 단위 데이터 배열이다. 가장 왼쪽 비트는 파일 디스크립터 0을 나타낸다. 1로 설정된 비트가 관찰 대상이 되는 파일 디스크립터를 의미한다. 그렇다면 파일 디스크립터 1과 3을 관찰 대상으로 하고 있다.

(나) 검사 범위 설정

select함수는 여러 파일 디스크립터를 관찰하고, 그 결과를 전달해 준다. 그렇다면 여러 개의 디스크립터를 확인해야 하는데 이왕이면 확인해야 하는 범위를 설정해 주면, 보다 효율적으로 수행 할 수 있을 것이다.

(다) 타임아웃(timeout) 설정

타임아웃을 설정한다는 것은, 블록킹 상태를 빠져 나가기 위한 시간 설정을 의미한다. select 함수는 호출했을 때 관찰 대상들에게서 변화(관찰 항목에 부합되는 소켓의 변화를 의미한다)가 발생해야 리턴하며, 그렇지

않으면 변화가 발생될 때까지 무한정 블록킹 상태에 있게 된다. 그러나 타임아웃을 설정해 놓으면 관찰 대상들에게서 변화가 없더라도 설정 시간이 지나면 무조건 리턴한다. 따라서 무한 대기 상태에 빠지는 것을 피할 수 있다.

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int select(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
           struct timeval *timeout)
```

성공 시 0 이상, 오류 발생 시 -1 리턴

그림23. select 함수

• n: 검사 대상이 되는 파일 디스크립트 수

• readfds : “입력 스트림에 변화가 발생했는지” 확인하고자 하는 소켓들의 정보를 전달한다. 여기서 입력 스트림에 변화가 발생했다는 것은 수신할 데이터가 있다는 뜻이다.

• writefds : “데이터 전송 시, 블로킹되지 않고 바로 전송이 가능한지” 확인하고자 하는 소켓들의 정보를 전달한다.

• exceptfds : “예외가 발생했는지” 확인하고자 하는 소켓들의 정보를 전달한다.

• timeout : 함수 호출 후, 무한 대기 상태에 빠지지 않도록 타임-아웃(time-out)을 설정하기 위한 인자를 전달한다.

• 리턴 값 : -1이 리턴되는 경우, 오류 발생을 의미한다. 또한 0이 리턴된 경우에는 타임아웃에 의해 리턴되었음을 의미한다. 마지막으로 리턴된 값이 0보다 큰 경우는 변경된 파일 디스크립터의 수를 의미한다.

표 3. fd_set 자료형 데이터 조작 함수

함수 선언	기능
FD_ZERO(fd_set *fdset);	fdset 포인터가 가리키는 변수들의 모든 비트들을 0으로 초기화한다.
FD_SET(int fd, fd_set *fdset);	fdset 포인터가 가리키는 변수에 fd로 전달되는 파일 디스크립터 정보를 설정한다.
FD_CLR(int fd, fd_set *fdset);	fdset 포인터가 가리키는 변수에서 fd로 전달되는 파일 디스크립터 정보를 삭제한다.
FD_ISSET(int fd, fd_set *fdset);	fdset 포인터가 가리키는 변수 fd로 전달되는 파일 디스크립터 정보를 지니고 있는지 확인한다.

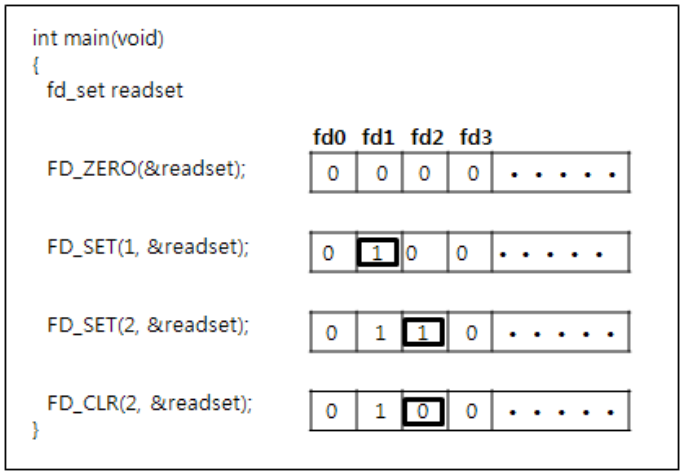

그림24. fd_set 데이터 조작 함수 사용

readset이라는 이름으로 fd_set의 변수를 선언하고, FD_ZERO 함수를 통해서 모든 비트를 0으로 초기화하였다. 그 다음에 FD_SET 함수 호출을 통하여, 파일 디스크립터 1과 2를 나타내는 위치의 비트를 1로 설정하였다.

마지막에 FD_CLR 함수를 사용해서 디스크립터 2를 나타내는 위치의 비트를 0으로 설정하였다.

select 함수 선언을 보면 readfds, writefds 그리고 exceptfds라는 이름으로, 총 세 개의 fd_set변수의 포인터를 요구하고 있다. select 함수는 총 3가지 항목에 대해 관찰한다.

• 파일 디스크립터를 통해 수신할 데이터가 존재 하는가?
- 수신할 데이터가 존재한다는 것은 비교적 쉽게 이해 할 수 있다. 일단 입력 버퍼로 데이터가 수신되어서 읽어 들일 데이터가 존재하는 경우라고 생각할 수 있다. 여기서 주의할 것이 한자기 있다. 소켓을 통해서 들어오는 클라이언트의 연결 요청 또한 수신할 데이터가 존재하는 경우에 해당한다. 연결 요청도 일종의 데이터 전송에 해당되고, 이를 수신하는 것도 데이터를 수신하는 것이기 때문이다.

• 파일 디스크립터를 통해서 데이터를 전송할 경우 블로킹되지는 않는가?
- write 함수와 같은 출력 함수가 블로킹되는 경우는, 출력 버퍼에 아직 전송하지 못한 데이터가 많이 남아 있어서, 데이터 전송을 바로 할 수 없는 경우에 발생한다. 따라서 출력 버퍼가 여유 있는 경우에는 블로킹되지 않고, 데이터를 전송할 수 있는 경우가 된다.

• 파일 디스크립터가 가리키는 소켓에서 예외가 발생하였는가?
- TCP 기반의 Out-of-band data가 수신되었을 경우를 의미하는 것이다. 여기서 “수신할 데이터가 존재하는가”이므로 readfds를 제외한 나머지 인자에 대해서는 NULL 포인터를 전달할 것이다.

다음 타임아웃을 설정하기 위한 timeval구조체를 보여주고 있다.

```
struct timeval
{
        long    tv_sec;         /* seconds */
        long    tv_usec;        /* microseconds */
}
```

그림25. 타임아웃 구조체

예를 들어서 tv_sec가 3이고 tv_usec가 500000이면 타입아웃은 3.5초로 설정된다. 이렇게 설정한 timeval 구조체 변수의 포인터를 select 함수의 마지막 인자(timeout 인자)로 넘겨주게 되면, 파일 디스크립터에 아무런 변화가 없더라도 3.5초가 지나면 무조건 리턴하게 된다. 만약에 타임 아웃을 설정해 주지 않을 경우, NULL 포인터를 인자로 전달하면 된다.

(마) 파일 디스크립터 범위 설정하기

select 함수는 여러 파일 디스크립터를 검사하고, 그 결과를 전달해 준다. 그렇다면 select 함수는 여러 개의 파일 디스크립터를 확인해야 하는데, 이왕이면 확인해야 하는 파일 디스크립터의 범위를 제한해 주면, 보다 효율적으로 수행 할 수 있을 것이다.

그래서 select 함수의 첫 번째 인자로 검사해야 하는 총 디스크립터의 개수를 넘겨주게 된다. 그러나 일반적으로 디스크립터는 생성될 때마다 1씩 증가하기 때문에 가장 큰 파일 디스크립터 값에 1을 더해서 인자로 전달하면 된다.

1을 더하는 이유는 디스크립터 값이 0부터 시작하기 때문이다. 따라서 인자로 n이라는 값을 넘겨 주게 되며, select함수는 검사하게 되는 파일 디스크립터의 범위를 0부터 n-1로 설정된다. 따라서 반드시 1을 더해줘야 한다.

(바) select 함수 호출 이후 결과 확인

select 함수가 리턴되고 나서 무엇보다도 중요한 것은 결과를 얻는 것이다. 일단 함수 호출이 정상적으로 리턴했다는 것은 파일 디스크립터에 변화가 있었거나, 아니면 타임아웃이 발생했거나 둘중에 하나이다.

일단 리턴 값이 -1인 경우는 오류발생을 의미한다. 또한 0이 리턴 된 경우에는 타임아웃에 의해 리턴되었음을 의미한다. 즉 0이 리턴된 경우 파일 디스크립터에 아무런 변화도 발생하지 않았다는 의미가 된다.

그러나 리턴된 값이 0보다 큰 경우에는, 변화가 발생한 파일 디스크립터의 수를 의미하게 된다. 예를 들어서 수신할 데이터가 존재하는 파일 디스크립터가 두 개 발생했다면, 2가 리턴될 것이다.

select 함수 호출 전에 fd_set변수를 생성하고, 파일 디스크립터 0과 3을 나타내는 위치를 1로 설정하였다. 그리고 select 함수의 두 번째 인자로 이 변수의 포인터를 넘겼다.(세 번째 네 번째 인자는 NULL 포인터를 전달 했다고 가정). 파일 디스크립터 0이나 3에서 수신할 데이터가 존재하는 경우에 select 함수는 리턴을 할 것이다.

select 함수 호출 후에 fd_set 변수를 살펴보면, 파일 디스크립터 3의 위치는 여전히 1로 설정되어 있거, 파일 디스크립터 1의 위치는 0으로 변경된 것을 알 수 있다. 이것이 의미하는 바는 파일 디스크립터 3으로부터 수신할 데이터가 존재한다는 의미이다. 즉 select 함수를 리턴하게 한 장본인이라는 뜻이다. 즉, select 함수 호출이 끝나고 나서도, fd_set 변수에서 1로 설정되어 남아 있는 파일 디스크립터가 변화를 일으킨 파일 디스크립터가 된다.

(사) select 함수 호출 코딩

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/time.h>

#define BUFSIZE 30

int main(int argc, char **argv)
{
        fd_set reads, temps;
        int result;

        char message[BUFSIZE];
        int str_len;
        struct timeval timeout;

     ① FE_ZERO(&reads);
        FD_SET(0,&reads);       /* standard input 설정 */

        while(1)
        {
             ② temps=reads;
                timeout.tv_sec=5;
                timeout.tv_usec=0;

             ③ result = select(1, &temps, 0, 0, &timeout);
                if(result==-1)
                {
                        puts("selcet 오류 발생");
                        exit(1);
                }
                else if(result==0)
                {
                        puts("시간이 초과되었습니다. : select");
                }
             ④ else
                {
                        if(FD_ISSET(0,&temps))
                        {
                                str_len=read(0,message,BUFSIZE);
                                message[str_len]=0;
                                fputs(message, stdout);
                        }
                }
        }

}
```

그림26. select함수 사용

- ①에서 fd_set구조체 변수를 초기화하고, 파일 디스크립터 0을 나타내는 위치를 1로 설정해 주고 있다. 즉 표준 입력에 변화가 있는지 관심을 두고 보겠다는 뜻이다.
- ② 이미 준비해둔 fd_set 변수를 임시 변수에 복사해 두고 있다. 여기에

는 그럴만한 이유가 있다. select 함수 호출이 끝나면 변화가 생긴 파일 디스크립터 위치를 제외한 나머지 위치의 비트들은 0으로 초기화된다. 따라서 원본 변수를 직접 select 함수의 인자로 전달해 버리면 또 다시 변수를 설정하는 것을 가지고 select 함수를 호출한다.

- ③에서 select 함수를 호출하고 있다. 만약에 콘솔로부터 입력된 데이터가 있다면 0보다 큰 수가 리턴 될 것이며, 입력이 없어서 타임아웃이 발생하는 경우에는 0이 리턴 될 것이다.

- ④는 select 함수가 0보다 큰 값을 리턴했을 경우의 실행 영역에 해당된다. 변화를 보인 파일 디스크립터가 표준 입력인지 확인해 보고, 맞으면 데이터를 읽어서 콘솔로 데이터를 출력해 주고 있다.

Ⅲ. 결과

1. 프로토콜 구현

다중접속 제어서버는 접속되어 있는 환자와 전문의 리스트를 관리하고 이질적인 통신망 환경에서 응급원격 진료 서비스를 이용하고자 하는 구성원간의 망의 특성에 맞게 안정된 서비스를 제공하기 위해서 By-passing과 Flowing 정책을 적용하였다. 전문의와 환자 측에서 다중접속 서버에 접속될 때 헤더에 통신망의 정보를 포함하기위해 프로토콜을 디자인하여 각각의 통신망의 정보와 장치 타입의 정보를 저장한다.

가. 네트워크 타입 정보

표 4. 네트워크 타입 프로토콜

Classification	Type	Value	Description
NETWORK_TYPE	NET_INWIRED	0x11	내부 유선망
	NET_INWIRELESS	0x12	내부 무선망
	NET_HSDPASKT	0x14	HSDPA SKT망
	NET_HSDPAKTF	0x15	HSDPA KTF망
	…	…	…
	NET_WIRED	0x21	외부 유선망
	NET_WIRELESS	0x22	외부 무선망
	NET_WIBRO	0x23	Wibro 망

네트워크 통신망의 정보를 다중제어 서버는 7가지 타입으로 정리하여 저장하였다. 크게 사설IP망과, 공인IP망으로 분류하여 Value 값을 Byte 값

으로 받아 전문의 측과 환자 측의 통신망을 구별 할 수 있다. 통신망의 네트워크 타입은 표4와 같다.

```
CString CControlServerDlg::Call_Network_Type_Interpret(BYTE Network)
{

    CString strTemp;
    switch(Network)
    {
    case NET_INWIRED:
        strTemp="병원내부 유선망";
            break;
    case NET_INWIRELESS:
        strTemp="병원내부 무선망";
        break;
    case NET_WIRED:
        strTemp="외부 유선망";
        break;
    case NET_WIRELESS:
        strTemp="외부 무선망";
        break;
    case NET_WIBRO:
        strTemp="와이브로 망";
        break;
    case NET_HSDPASKT:
        strTemp="HSDPA SKT 망";
        break;
    case NET_HSDPAKTF:
        strTemp="HSDPA KTF 망";
        break;
    }
    return strTemp;
}
```

그림27. Network Type Coding

나. 연결 정보

표 5. 연결 정보 프로토콜

Classification	Type	Value	Description
CONNECT_TYPE	CONN_PATIENT	0x03	환자측 접속
	CONN_MONITOR	0x04	모니터 접속

다중접속 제어 서버로 환자와 전문의가 연결을 하면 다중제어 서버는 환자인지 전문의인지 구별 할 수가 없다. 따라서 환자와 전문의 측을 구별할 수 있도록 헤더에 Client 구별 정보를 포함하여 다중접속 제어 서버 측

으로 전송하게 된다.

다. 장치 타입 정보

표 6. 장치타입 프로토콜

Classification	Type	Value	Description
COMPUTER_TYPE	COM_DESKTOP	0x31	데스크 탑 컴퓨터
	COM_NOTBOOK	0x32	노트북 컴퓨터
	COM_UMPC	0x33	UMPC 컴퓨터
	COM_MCA	0x34	MCA 컴퓨터
	COM_STCART	0x35	Style Cart 컴퓨터
	…	…	…

```
CString CControlServerDlg::Call_Computer_Type_Interpret(BYTE Computer)
{
    CString strTemp;
    switch(Computer)
    {
    case COM_DESKTOP:
        strTemp="데스크탑";
        break;
    case COM_NOTEBOOK:
        strTemp="노트북";
        break;
    case COM_UMPC:
        strTemp="UMPC";
        break;
    case COM_MCA:
        strTemp="MCA";
        break;
    case COM_STCART:
        strTemp="스타일 카트";
        break;
    case COM_PDA:
        strTemp="PDA";
        break;
    }
    return strTemp;
}
```

그림28. Computer Type Coding

2. 정책 결정

네트워크 통신망의 타입(표.3)을 이용하여 다중접속 제어 서버는 정책을 결정한다. 환자와 전문의 측이 같은 망에 포함되어 있고 1(환자):1(전문의) 서비스 할 때 By-passing 정책을 결정한다. 그 외 모든 조건은 Flowing 정책을 결정한다.

```
//정책 결정
BYTE CControlServerDlg::Decide_Network_Policy(BYTE _PatientNetworkValue, BYTE _MonitorNetworkValue)
{
    if(_PatientNetworkValue > 0x16 && _MonitorNetworkValue > 0x16)
    {
        return BYPASSING;
    }
    else if((_PatientNetworkValue > 0x13 && _PatientNetworkValue < 0x16)
          &&(_MonitorNetworkValue > 0x13 && _MonitorNetworkValue < 0x16))
    {
        return BYPASSING;
    }
    else if(_PatientNetworkValue < 0x13 && _MonitorNetworkValue < 0x13)
    {
        return BYPASSING;
    }
    else
    {
        return FLOWING;
    }

    return FLOWING;
}
```

그림29. 정책 결정 Coding

가. By-passing 서비스

환자와 전문의 측이 다중접속 제어서버로 접속을 하면 다중접속 제어서버는 환자 측과 전문의 측의 대역폭과 통신망의 종류를 저장하게 된다. 전문의 측에서 환자를 선택하면 선택된 환자와 정책을 비교하고 1:1 서비스를 원하거나 같은 통신망에 존재 하면 By-passing 정책을 적용하여 서비스를 결정한다. 다중접속 제어서버는 전문의 측의 IP을 환자 측으로 전달하여 직접 전문의 측의 원격 진료 시스템과 접속하여 서비스 할 수 있도록 설계 하였다. 이때 다중접속 제어서버는 환자와 전문의 간에 원격 진료 서

비스가 원활히 진행 되고 있는지를 체크하여 환자와 전문의에 리스트를 관리한다. By-passing 정책의 연결 및 데이터 흐름도는 그림 30과 같다.

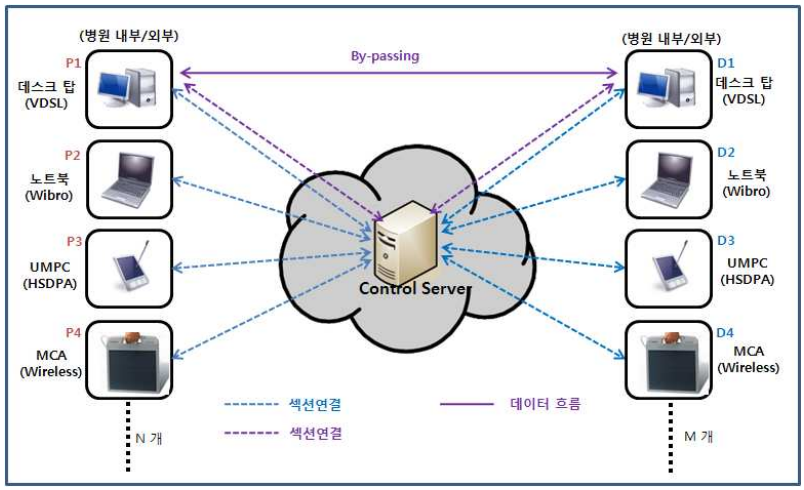

그림30. By-passing 서비스

```
else // 5. 바이패싱 일 경우
{
    // 5-1. 전송 버퍼 정보 수정
    PClient_Information pCliInfo = NULL;
    CopyMemory(CmdDataBlock, &ClientHeader, sizeof(Client_Header));
    CopyMemory(&CmdDataBlock[sizeof(Client_Header)], &ExReqHeader, sizeof(ExReq_Header));
    // 5-3. 환자의 상태 호출
    pCliInfo = Call_Client_Information(1, _ExSelIP->str_Selected_IP);
    if(pCliInfo)
    {
        // 5-2. 의사에게 메세지 전달
        Send_CtrlMsg_To_MClient(_ExSelIP->str_Selected_IP, CmdDataBlock, nLenDataBlock);
        // 5-4. 환자에게 메세지 전달
        Send_CtrlMsg_To_PClient(_SenderIP, CmdDataBlock, nLenDataBlock);
        // 5-5. 의사와 환자의 정보를 불러와 수행 결과를 업데이트 한다.
        pCliInfo = Call_Client_Information(0, _SenderIP);
        pCliInfo = Call_Client_Information(1, _ExSelIP->str_Selected_IP);

        //화면 업데이트
        Bypassing_List_Update(_ExSelIP->str_Selected_IP, _SenderIP);
    }
}
```

그림31. 환자 측으로 전문의 측의 IP주소 전송

나. Flowing 서비스

Flowing 정책은 환자와 전문의의 통신망이 이질적인 통신망에서 대역폭과 1:M(Multi) 서비스를 할 때 다중 접속 서버가 중간에서 환자의 데이터를 중재하여 서비스 하도록 설계한 정책이다. Flowing 정책의 데이터 흐름은 그림 32와 같다.

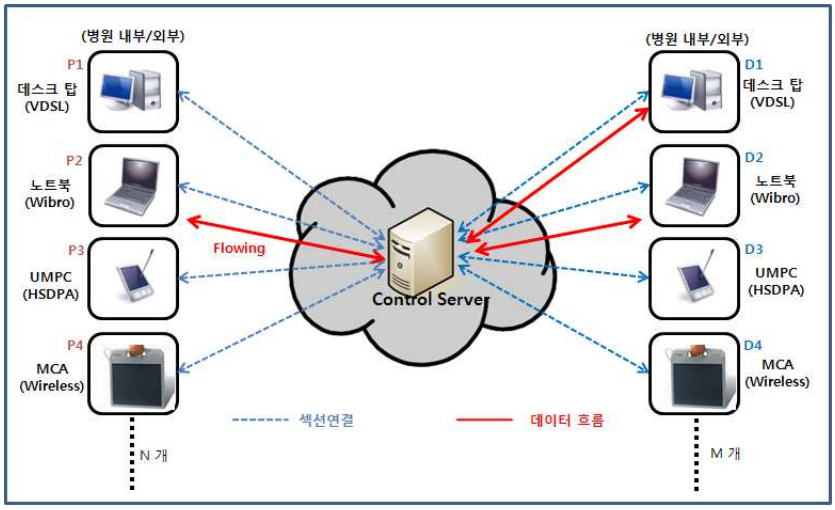

그림32. Flowing 서비스

환자측의 통신망의 대역폭이 전문의측의 통신망의 대역 폭보다 클 경우와 여러 전문의에게 협력 진료를 할 때 적용한다. 환자와 전문의가 다중 접속 제어서버로 접속을 하면 다중접속 제어서버는 환자와 전문의 측의 통신망과 대역폭을 저장한다. 전문의 측에서 환자를 선택하면 다중 접속 서버에 정책에 따라서 서비스를 결정한다. Flowing 정책으로 서비스가 결정되면 환자 측의 시스템은 다중접속 제어서버로 환자의 데이터를 보내고 다중 접속 서버는 데이터를 중개하여 전문의 측의 시스템으로 환자의 데이터를 전송 한다. 이때 모든 메시지 및 데이터는 다중 접속 서버를 통해서 제

어 및 제공이 된다.

다. Multi to Multi 서비스

N to M 서비스는 By-passing과 Flowing 서비스를 통합하여 그림33과 같이 환자와 전문의간의 독립적인 By-passing 연결과 환자와 전문의간의 Flowing 연결을 통하여 다중 연결이 혼합되어 서비스를 한다. 즉, 다수의 환자와 다수의 전문의가 다중 제어 서버를 통해서 다자간의 서비스를 제공한다.

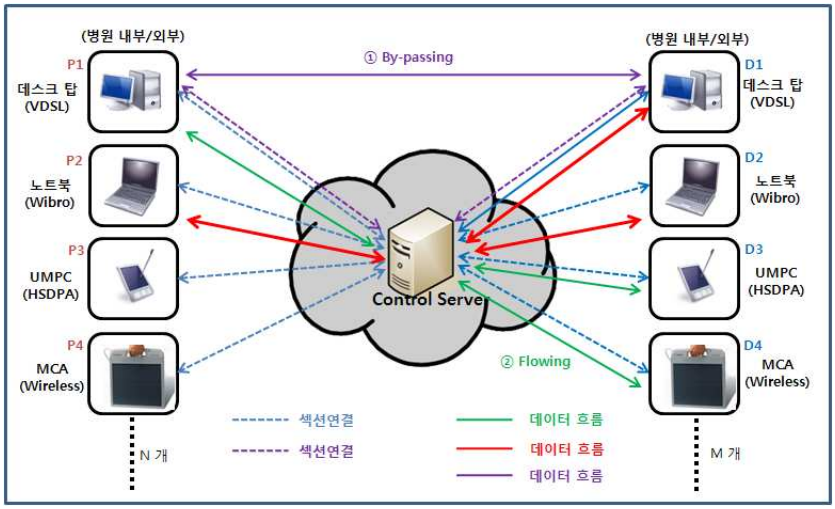

그림33. Multi to Multi 서비스

(1) 멀티 플렉싱 소켓 관리

다중접속 제어서버에 IP주소를 할당하고 서버를 실행하면 다중접속 제어 서버는 Server Listen Socket을 생성하고 환자와 전문의 연결을 대기하게 된다. Server Listen Socket은 연결 요청이 들어오는 환자와 전문의 순서에 따라 Client와 통신을 할 수 있는 Accept Socket을 생성하고 생성

된 디스크립터(핸들)를 배열에 순차적으로 저장한다. 저장된 환자와 전문의 구별을 위해서 원격진료 시스템은 Client(환자, 전문의) 헤더에 접속자 정보를 포함하여 다중접속 제어서버 측으로 전송한다. 다중접속 제어서버는 접속 할 때 헤더 정보를 저장하고 환자와 전문의를 구별하여 리스트로 분류하여 순차적으로 관리한다.

다중접속 제어서버는 Select함수를 이용하여 Client 소켓(Accept Socket)을 100ms 마다 모니터링을 한다. 즉 소켓의 변화가 있는지를 검사하여 변화가 있다면 소켓의 옵션에 해당하는 명령을 수행한다. Client 소켓으로부터 읽거나 쓸 수 있는 소켓이 있는지를 검사하는 것이다.

(2) 멀티 스레드 서비스

멀티 플렉싱으로 환자와 전문의에 접속 리스트가 관리가 되면 원격 진료를 원하는 환자와 전문의 정책을 결정하여 각각의 통신망의 특성에 맞게 서비스 할 수 있도록 다중접속 제어서버는 정책을 결정하여 서비스를 제공한다.

전문의 측에서 원격 진료를 하고자 하는 환자를 선택하면 다중 접속 서버는 환자의 통신망과 전문의 통신만을 비교하여 정책을 결정한다. 결정된 정책이 By-Passing 이면 다중접속 제어서버는 환자가 직접 전문의 측으로 접속할 수 있도록 전문의 통신망의 IP주소를 전달한다. 환자 측 원격진료 시스템은 전문의 측 IP주소를 가지고 접속을 요청하고 원격진료 서비스를 수행한다. 이때 다중접속 제어서버는 환자와 전문의 양자간 서비스가 원활하게 진행되는 것을 확인하기 위해 양자간의 접속을 모니터링한다. 서비스가 종료되면 제어 서버 측으로 연결 종료 메시지를 전송하여 환자와 전문의 측 서비스가 종료되었다는 것을 다중접속 제어서버로 알려준다.

다음으로는 결정된 정책이 Flowing 정책이면 제어 서버는

SessionManager에 의해서 스레드를 생성하고 관리한다. 스레드의 생성 순서는 원격진료를 받고자 하는 환자의 순서에 따라서 스레드가 생성되고 관리된다. 1번의 환자을 전문의 1번과 2번이 선택을 하게 되면 SessionManager에 의해서 스레드 1번이 생성되고, 스레드 1번에 의해서 전문의에 수에 따라 서비스 받도록 소켓을 생성한다. 즉, 2명의 전문의가 있으므로 소켓을 각각 7개씩 생성하여 서비스 한다. 만약 환자 번호가 2번이면 SessionManager에 의해서 번호가 2번인 스레드가 생성되고 원격 진료를 하고자 하는 의사 수에 따라서 소켓이 생성된다. 측 멀티 스레드 방식으로 서비스한다.

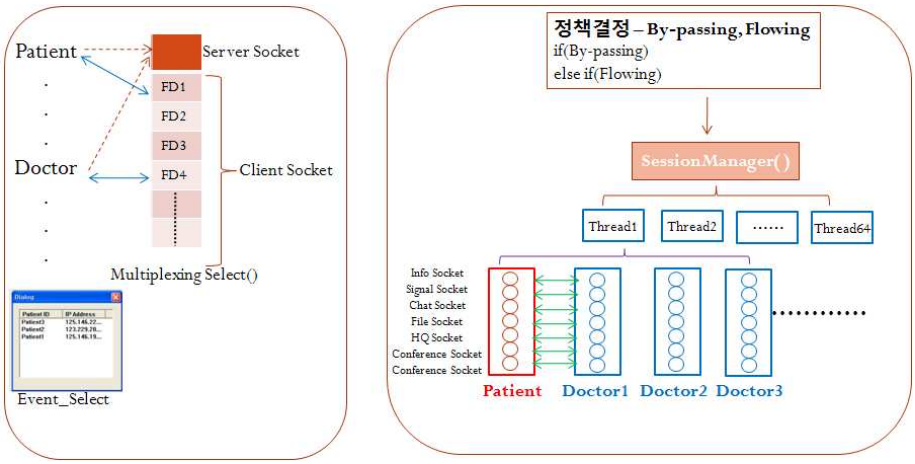

그림34. 섹션 매니저에 의한 멀티 스레드 서비스

3. 다중접속 제어 서버

다중접속 제어서버는 서버 실행과 종료하는 버튼, 접속된 환자와 전문의 리스 관리 화면 그리고 Flowing과 By-passing이 서비스 되는 것을 모니터링 할 수 있는 화면으로 구성되어 있다. 다중 제어 서버는 그림35와 같다.

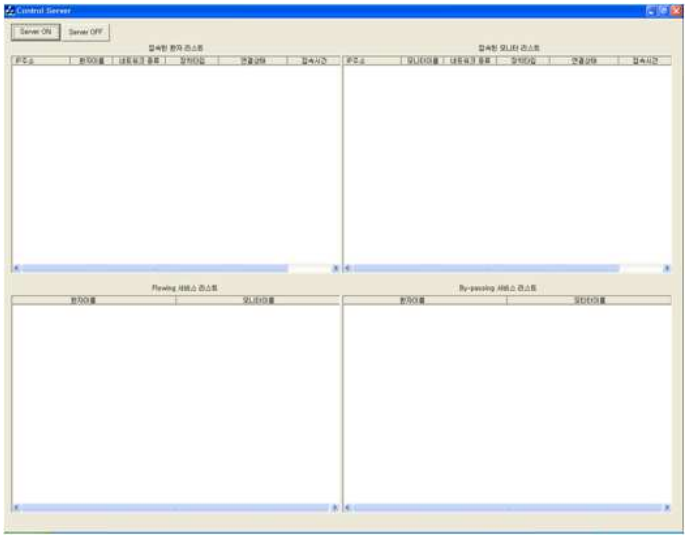

그림35. 다중접속 제어서버 실행 화면

먼저 환자와 전문의가 접속을 하면 접속되어 있는 리스트 화면에 관리되고 서비스를 중 혹은 종료를 하면 접속 리스트의 접속 상태를 나타 낼 수 있도록 디자인 하였다. 또한 Flowing과 By-passing 정책으로 서비스 중일 때는 Flowing List View와 By-passing List View에서 서비스 중인 환자와 전문의 이름을 모니터링 할 수 있다.

4. 다중접속 제어서버 실험 및 평가

가. 통신망 대역폭 측정

각각의 통신망이 가지고 있는 대역폭은 표7과 같다.

표 7. 각각의 통신망의 대역폭

VDSL(서버)	Down	40.4 Mbps
	Up	7.7 Mbps
Wibro	Down	353.87 kbps
	Up	992.13 kbps
HSDPA	Down	2.59 Mbps
	Up	51.15 kbps
병원 내부유선	Down	91.1 Mbps
	Up	79.7 Mbps
병원 내부무선	Down	2.88 Mbps
	Up	2.83 Mbps

나. 서버와 이기종 네트워크 대역폭 측정

(1) Iperf 측정

Iperf는 그림36과 같이 컴퓨터 A에서 컴퓨터 B까지 현재 사용 가능한 최대 속도를 구하려고 할 때 사용된다. 측정 원리는 서버와 클라이언트 각각의 양단에 Iperf를 실행하고 서버와 클라이언트로 설정을 한다. 더미 데이터를 만들어 서버 측으로 전송하여 대역폭을 측정한다.

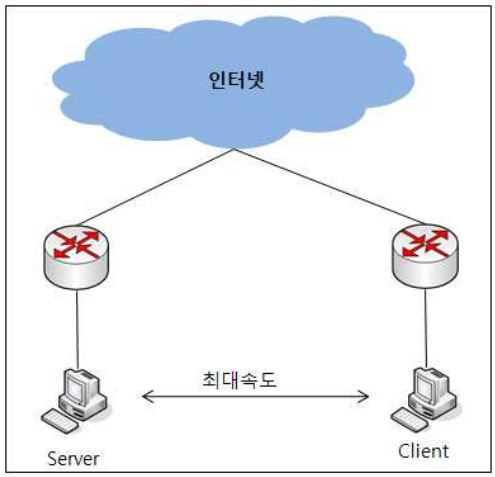

그림36. Iperf을 이용한 서버와 클라이언트 대역폭 측정

(2) TCP 측정

서버 측에 "iperf -s"를 실행하면 서버 측 설정이 완료되고 클라이언트로부터 접속 대기를 하고 있다. 클라이언트로부터 접속이 들어오면 그림37과 같이 서버와 클라이언트 간에 대역폭을 측정한다.

표 8. 서버와 클라이언트 간 대역폭(TCP)

VDSL (서버)	Wibro	769 kbps
	HSDPA	58.5 kbps
	병원 내부유선	7.7 Mbps
	병원 내부무선	1.78 Mbps

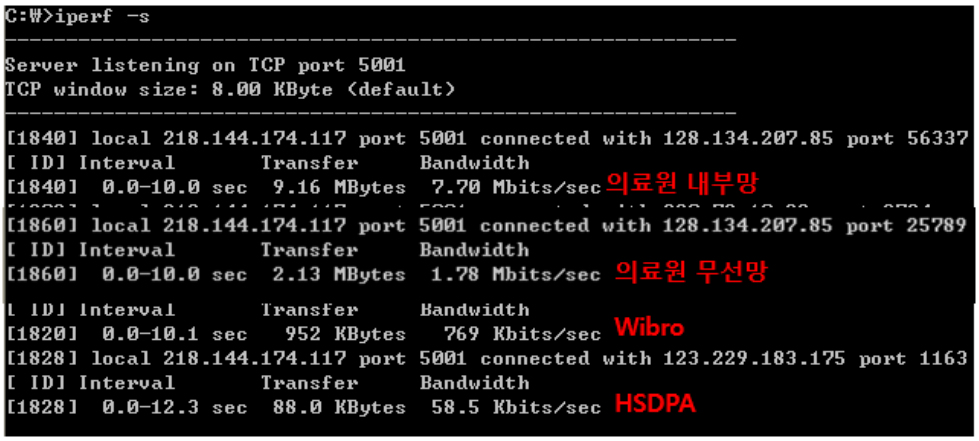

그림37. Iperf를 이용한 서버와 클라이언트 간 대역폭 측정(TCP)

(3) UDP 측정

표 9. 서버와 클라이언트 간 대역폭(UDP)

VDSL (서버)	Wibro	1.05 Mbps
	HSDPA	1.05 Mbps
	병원 내부유선	1.05 Mbps
	병원 내부무선	972 kbps

서버 측에서 “iperf -s -U″로 설정을 하고 클리언트 측에서는 ”iperf -c IP -U″로 설정하면 서버 측으로 접속하여 서버와 클라이언트 사이의 대역폭을 측정 할 수 있다. 측정 결과는 그림 38과 같다.

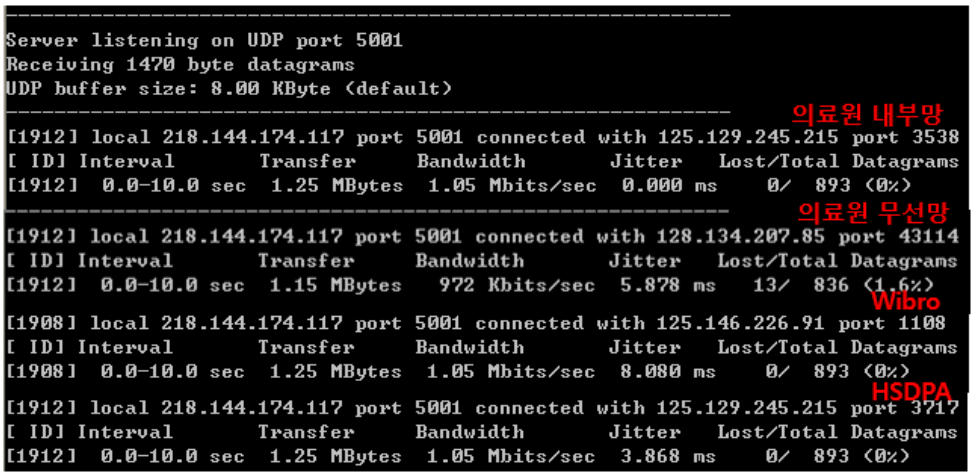

그림38. Iperf를 이용한 서버와 클라이언트 간 대역폭 측정(UDP)

다. 다중접속 제어서버 실험

다중접속 멀티서버를 디자인 후 테스트 하기위해서 일단 실험실에서 응급원격 진료 서비스에 사용할 장치 타입을 배치하여 네트워크 타입을 다양하게 사용하여 테스트를 실행 하였다. 테스트 환경은 그림 39와 같이 설치하고 실험을 하였다. 이론적으로 환자와 전문의측이 각각 128명씩 접속이 서비스 받을 수 있도록 설계하였다. 실제 실험환경에서는 환자(128명) : 전문의(128명)을 대상으로 실험을 할 수 없기 때문에 by-passing 1개 섹션과 Flowing 2개 섹션을 맺어 총 3개의 섹션을 연결하여 실험 하였다.

표 10. 다중접속 제어 서버 실험

서비스	구분	통신망
Flowing 서비스 (환자1 : 전문의3)	환자	Wibro
	전문의	ADSL, Wibro, HSDPA
Flowing 서비스 (환자1 : 전문의1)	환자	Wibro
	전문의	HSDPA
By-passing 서비스 (환자1 : 전문의1)	환자	HSDPA
	전문의	Wibro

그림 39. 다중접속 제어 서버 실험

라. 다자간 응급원격 진료 서비스

다자간 응급원격 진료 서비스 실험에서 다수의 환자와 다수의 전문의가 다양한 통신망과 위치가 서로 다른 지역에서 분산되어 다중접속 제어 서버에 접속하여 환자와 전문의 리스트를 관리하고 3개의 섹션을 맺어 서비스 실험을 하였다. 각각의 위치와 통신망은 표1과 같으며 그림40 은 테스 환

경을 나타 낸 것이다.

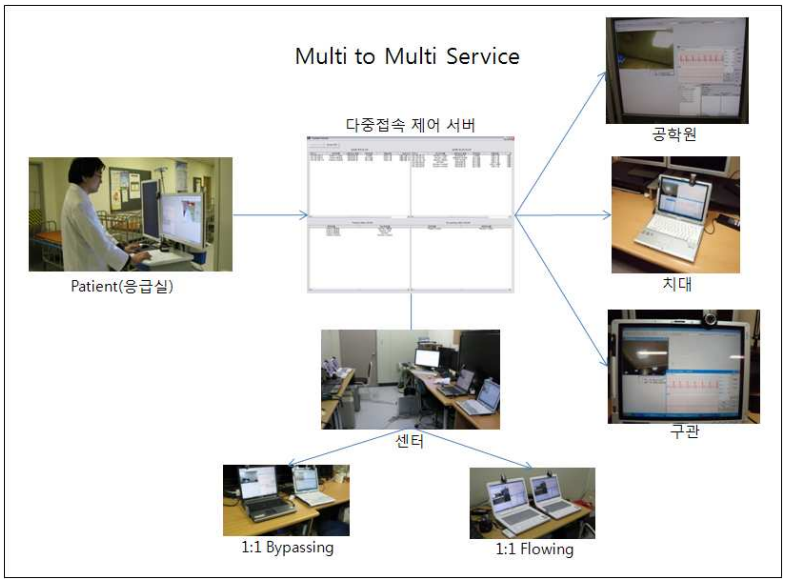

그림40. 다자간 응급원격 진료 서비스

표 11. 다자간 응급원격 진료 서비스

서비스	구분	위치	통신망
Flowing 서비스 (환자1 : 전문의3)	환자	응급실	HSDPA
	전문의	공학원,치대,구관	유선망, 병원망, 유선망
Flowing 서비스 (환자1 : 전문의1)	환자	실험실	Wibro
	전문의	실험실	Wibro
By-passing 서비스 (환자1 : 전문의1)	환자	실험실	HSDPA
	전문의	실험실	Wibro

다중접속 제어서버에서 Ethereal을 이용하여 서비스 중에 패킷을 측정하였다. 그림41은 환자의 영상을 100kbps로 전송하고 3명의 전문의 측에 서비

스 할 때 패킷의 변화량을 측정한 결과이다. 응급실에서 발생한 환자를 3명의 전문의가 협진을 할 수 있는 상황을 만들어 실험하였다. 본 실험은 다중접속 제어 서버를 통해서 환자와 전문의간의 세션을 맺어 충분한 서비스를 제공할 수 있는지에 대해 다중접속 제어 서버의 업 링크 대역폭의 변화를 측정 하였다. 그림41에서 보는 것처럼 UDP 최대 업 링크는 1.05Mbps이다. 환자 측에서 100kbps 환자 영상을 다중접속 서버로 보내고, 다중접속 서버는 3대의 전문의 측으로 환자의 영상을 전송한 결과이다. MPEG-4에서 영상 데이터를 압축해서 전송할 때 이전 프레임과 현재 프레임의 오차 정보를 계산하여 유사한 블록의 방향 벡터 정보만을 압축하는 방법을 사용한다. 따라서 영상 움직임이 거의 없으면 매우적은 압축 결과를 나타낼 수 있다. 그림 41에서 주기적으로 뾰족한 부분이 보이는 것은 전송해야 될 전체 영상중에서 변화된 영상만을 전송하다가 주기적으로 전체 영상을 보내기 때문에 피크점이 주기적으로 나타나게 되는 것이다.

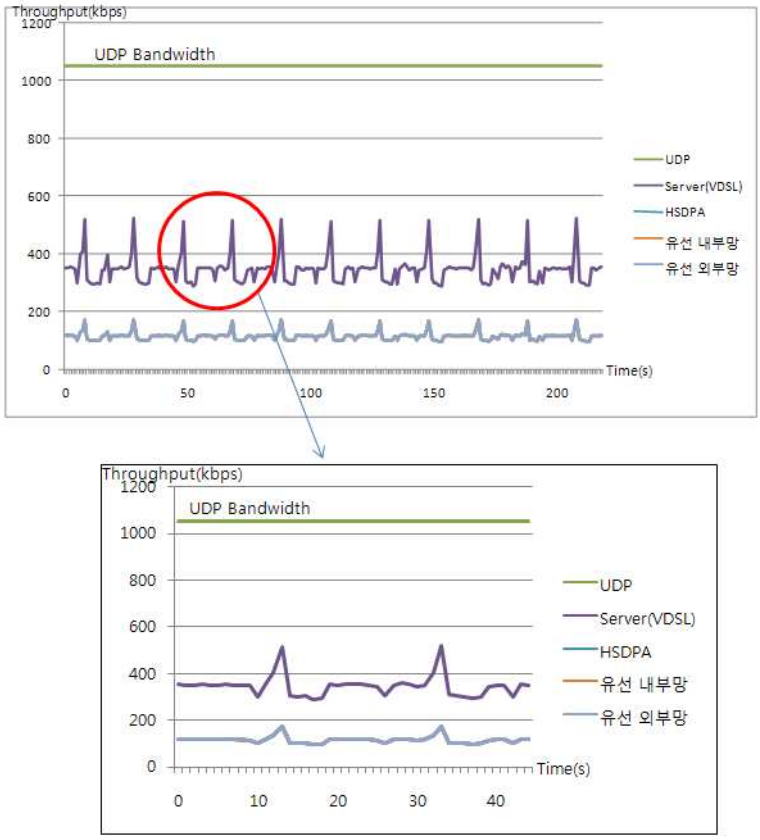

그림41. 섹션별 Throughput 측정 결과

Ⅳ. 결론 및 토의

응급원격 진료 시스템은 전문의가 환자를 원격으로 진단 및 자문을 할 수 있는 시스템이다. 응급원격 진료 시스템은 의료 사각지대인 도서지역, 지방 중소 병원, 가정, 군부대 지역에서 응급 환자가 발생 할 경우 응급 전문의로부터 신속한 조치를 받기위해 응급원격 진료 시스템이 필요하다. 환자와 다수의 전문의가 협력진료를 통해서 응급환자를 신속하게 처치하기 위해서는 다자간 멀티 서비스를 할 수 있는 시스템이 필요하다. 본 논문에서는 다중접속 제어서버를 디자인 하여 다수의 환자와 다수의 전문의가 각각의 연결 섹션을 맺어 독립적으로 서비스하고 협력진료가 필요한 응급환자에 대해서도 서비스가 이루어 질 수 있도록 설계하였다. 기존의 응급원격 진료 시스템은 환자의 연결 순서에 따라서 순차적으로 서비스가 되고, 다수의 전문의가 협력 진료를 할 수 없는 단점을 극복하였다.

응급원격 진료 서비스를 이용하는 환자와 전문의는 다양한 유무선 통신망을 이용하여 서비스를 이용한다. 통신망에 따라 사설IP, 공인 IP 그리고 대역폭의 차이로 인하여 1(환자) : 1(전문의) 통신 연결은 응급원격 진료 서비스를 이용하는데 많은 제약을 가지고 있었다. 이러한 문제점을 해결하기 위해 환자와 전문의가 사용하는 각각의 통신망 대역폭 상태에 따라 서비스 할 수 있도록 다중접속 제어서버를 디자인 하여 극복 하였다.

다중접속 제어서버는 환자와 전문의 간에 연결 리스틀 관리하고 원격진료를 사용하고 있는 환자와 전문의 통신망정보를 저장하여 안정적으로 서비스 할 수 있도록 디자인 하였다. 이러한 서비스를 효율적으로 하기위해서 By-passing정책과 Flowing 정책을 디자인하여 다중접속 제어서버에 큰 부하가 걸리지 않도록 디자인 하였다.

다중접속 제어 서버는 앞으로 다음과 같은 연구가 추가적으로 보완되어

야 할 것이다. 환자와 전문의는 이기종 통신망에서 응급원격 진료 서비스를 이용 할 때 시간, 장소에 따라 통신 대역폭이 변한다. 즉, 서비스 중 원격 자문 및 진단을 하는 전문의 측의 대역폭이 환자 측보다 급격히 낮아지면 연결이 끊어지거나 다운되는 현상이 발생한다. 이러한 환경 속에서도 안정적으로 서비스를 하기 위해 다중접속 제어서버는 네트워크의 통신 대역폭을 수시로 체크하고, 네트워크의 통신 대역폭에 적합하게 데이터를 가공하여 서비스 할 수 있도록 트랜스 코딩에 관한 연구가 필요하다.

참고문헌

1. Mehmet R. Yuce, Peng Choong NG, Chin K. Lee, Jamil Y. Khan, and Wentai Liu, A Wireless Medical Monitoring Over a Heterogeneous Sensor Network, Conference of the IEEE EMBS, Aug, 2007.

2. Jaemin Kang, Il Hyung Shin, Yoonseo Koo, Min Yang Jung, Gil Joon Suh, Hee Chan Kim. HSDPA(3.5G)-Based Ubiquitous Integrated Biotelemetry System for Emergency Care, Conference of the IEEE EMGS. Aug, 2007.

3. Ren-Guey Lee, Kuei-Chien Chen, Chun-Chieh Hsiao, Chwan-Lu Tseng, A Mobile Care System With Alert Mechanism. IEEE Transactions On Information Technology In Biomedicine, Vol. 11, No.5, Sep, 2007.

4. Kang, Ho Hun. Design of Multiple Emergency Telemedicine System in Wired and Wireless Integrated Network, Yonsei University. 2005. 12.

5. E.A. Mendoca, E. S. Chen, P. D. Stetson, L.K Mcknight, J. Lei, J. J. Cimino, Approach to mobile information and communication for healthcare, International Journal of Medical Informatics 2004;73:631-638.

6. Mehmet R. Yuce, Peng Choong Ng, Chin K. Lee, Jamil Y. Khan, Wentai Liu, A Wireless Medical Monitoring Over a Heterogeneous Sensor Network, Conference of the IEEE EMBS. Aug, 2007.

7. Kuk Jin. The Transmission Performance Analysis of Remote Healthcare Date over Secure Wireless Networks, Yonsei University. 2007. 7.

8. G.Demiris, S. Vijaykumar, A comparison of communication models of

traditional and video-mediated health delivery, International Journal of Medical Informatics 2005;74:851-856.

9. Y.C. Lu, Y. Xiao, A. Sears, J.A.Jacko. A review and a framework of handled computer adoption in healthcare, International of Medical Informaics 2005;74:409-422.

10. J.R. Barrett, S. M. Strayer, J. R. Schubart. Assessing medical residents usage and perceived needs for personal digital assistants, International Journal of Medical Informatics 2004;73:25-34

11. 윤성우, TCP/IP 소켓 프로그래밍, 프리렉, 2004, p. 227-504

12. 백창우, TCP/IP 소켓 프로그래밍, 한빛미디어, 2005, p. 237-698

13. 김성훈, 성공과 실패를 결정하는 1% 네트워크 원리, 성안당, 2005, p. 32-341

14. 진강훈, 시스코 네트워킹, 사이버, 2005, p. 16-281

Abstract

Design and Evaluation of Multiple Emergency Telemedicine Monitoring System over Heterogeneous Networks

Doyoon, Kim

Department of Medical Science
The Graduate School, Yonsei University

(Directed by Professor Sun K. Yoo)

We can adapt emergency telemedicine systems in advancement of information technology capabilities and increase of network bandwidth. The emergency telemedicine service can be applied to a public health center, a school, a prison and islands in lacks of medical equipments and medical staffs. Also we send the patient's vital sign and video data at the hospital when a emergency patient at the transport using ambulance. Accordingly the emergency telemedicine services which can be provided high quality medical services. However, currently telemedicine systems have offered one(Doctor) to one(Patient) telemedicine service. So,

I suggested multi to multi telemedicine service in heterogeneous network environments.

I designed the multiple control server system consisting 4 sub-function, patients and doctors list, network types, connection states and computer equipments. The emergency telemedicine link configuration was decided as 'Flowing', or 'By-passing' in accordance the network type and bandwidth of patient systems or doctor systems. The multiple control server system was performed the best communication configuration over heterogeneous networks. This system was achieved high quality emergency telemedicine services through dynamic wired and wireless networks at real-time.

This study represented a hybrid multimedia telemedicine system over heterogeneous networks in emergency cases. I expected that the designed system could provide not only the high quality services, tele-diagnosis and tele-consultation, but also the effective emergency telemedicine services to multi-patients in the heterogeneous network environments.

Key Words : Heterogeneous Networks, Emergency Telemedicine System, Hybrid Multimedia Telemedicine System, Multiple Control Server