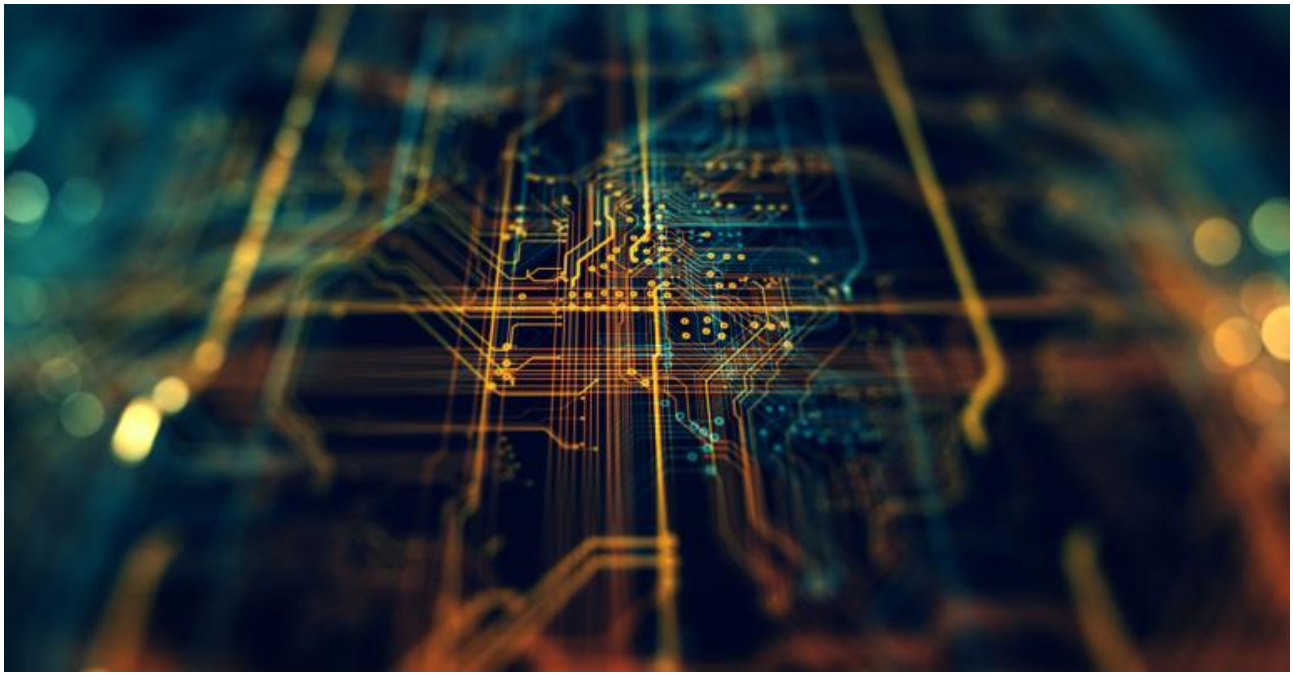
TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정



2018.04.05
31 일차
강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com

학생 – 신민철
akrn33@naver.com

비유 1:회사(일상생활)

얼마 전 회사를 차린 A 씨는 요즘 행복한 고민에 빠졌다. 처음 회사를 차릴 때 우려했던 바와는 달리 회사는 날로 번창하여 수입은 게속 늘어났지만, 이에 비례해서 일거리도 계속 늘어나면서 더 이상 모든 일을 혼자 처리하기엔 역부족이었기 때문이었다. 어쩔 수 없이 A 씨는 회사의 운영을 책임질 사장을 뽑기로 결정한다. 그래서 A 씨는 인력회사에 전화하여 컴퓨터 활용과 외국어 구사가 가능하며 성실한 사람을 보내달라고 요청하였다. 그런 뒤 A 씨는 새로 회사에 출근할 사람을 위해 책상과 의자, 그리고 사무를 보는데 필요한 기타 용품을 준비해 두었다. 다음날, A 씨의 회사로 출근한 B 씨는 A 씨로부터 어떻게 회사를 끌어가라는 지시를 받았다.

그러자 B 씨는 원래 A 씨가 했었던 수많은 귀찮은 일들을 모두 처리해 주었다. A 씨는 너무도 편했고, 이에 따라 자신 본연의 임무에 충실할 수 있었다.

B 씨는 A 씨가 시킨 일들을 모두 완수하기 위해 부하직원을 채용했다. 24 시간 근무하며 회사의 전화를 받는 C 씨와, D 씨. C 씨와 D 씨는 평소 할 일이 없을 땐 그저 사무실에서 쉬고 있었다. 그러나 C 씨는 누가 일을 시키지 않더라도 회사로 전화가 걸려오면 언제든 그 전화를 받아 응대하는 일을 수행하였고, D 씨는 A 씨가 시키는 일이 있을 때 마다 그 일을 수행하였다.

A 씨의 회사가 날로 번창하자 B 씨는 더 많은 부하직원을 채용하기로 결심하였다. 우선 C 씨의 업무가 너무 과중한 것 같아 C 씨와 동일한 일을 수행하는 C2 씨와, 제품을 조립하는 E 씨, 조립된 제품을 포장하는 F 씨, 포장된 제품을 발송하는 G 씨를 추가로 채용하였다.

C, C2, D, E, F, 그리고 G 씨는 한정된 사무실 공간에 있었지만 E 씨가 제품을 조립해야 할 때에는 남들보다 더 많은 공간을 쓸 수 있도록 배려하였다. 또한 제품을 포장 해야 할 때에는 F 씨가, 제품을 발송할 때는 G 씨가 공간을 더 쓸 수 있도록 B 씨가 항상 공간 배정을 해 주었기에 모두들 사무실이 좁다는 생각을 하지 않고 자신의 맡은 바 임무를 완수할 수 있었다.

비유 1:회사(운영체제 동작)
Firmware Level 의 프로그래머 A 씨는 여러 가지 기능을 하는 프로그램을 만들고 있다. 프로그램의 흐름을 계획하며 모든 것을 Firmware Level 에서 코딩하다 보니 이젠 더 이상 모든 것을 다 직접 코딩하기엔 역부족임을 느끼게 되었다.

어쩔 수 없이 프로그래머 A 씨는 귀찮은 이런 저런 일들을 대신 해주는 운영체제 (Operating System)리눅스를 사용하기로 결정한다. 그래서 A 씨는 Configuration 하여 스케줄링과 파일 시스템, 메모리 관리 기능을 가지고 있는 zImage 를 생성하였다. 그런 뒤 A 씨는 zImage 를 동작시키기 위해 Root Filesystem 을 만들었다. 그리고, 리눅스를 위한 Bootloader 도 작성하였다.

그러자 리눅스는 원래 Firmware Level 의 프로그래머 A 씨가 일일이 코딩해야 했던 수많은 귀찮은 일들을 모두 대신 처리해 주었다. 프로그래머 A 씨는 너무도 편했고, 이에 따라 시스템 디자인에만 충실할 수 있었다.

리눅스는 Firmware Level 의 프로그래머 A 씨가 해야 했던 모든 이들을 완수하기 위해 태스크를 생성했다. 시스템이 정지되기 전까지 게속 동작하는 백그라운드 데몬(Daemon)과, 쉘(Shell)태스크, 태스크는 할 일이 없을 땐 게속 idle 상태를 유지하며 쉬게 된다. 그러나 데몬은 누가 일을 시키지 않더라도 해야 할 일이 생기면 있을 때마다 Running 상태가 되어 시킨 일을 수행하였다.

데몬, 데몬 2, E, F, G 그리고 쉘 태스크는 한정된 메모리상에서 돌고 있지만, B 는 각각의 태스크가 필요로 할 때마다 메모리를 잘 할당/해제해 주었기 때문에 모든 태스크가 성공적으로 자신의 작업을 완료할 수 있었다.

비유 1 은 운영체제의 탄생이유와 운영체제가 하는 역할과 필요성을 제시 해준다. 사람이 덧셈,뺄셈,곱셈,나눗셈연산을 하는 것보다 컴퓨터가 연산을 하는것이 더 빠르기 때문에 컴퓨터를 이용하는데, 사람이 프로세스들을 전부관리하는 것 보다는 하드웨어의 자원을 효율적으로 관리할 수 있는 운영체제가 필요하다. 이 운영체제가 일상생활에서 비유한 사장이고, 데몬과 쉘 태스크, 태스크가 사장의 부하직원인데, 데몬은 중요한 프로세스를 쉽게 종료되지 않게 하기 위해 사용되고, 태스크들은 가상메모리를 관리 하게된다.

이렇게 된다면 일일이 손을 대지 않고 다른일을 더 해낼 수있다.

비유 2:카사노바 박씨 이야기(일상생활)
카사노바 박씨에겐 24 명의 여자 친구가 있다. 모두 내일 꼭 만나달라고 아우성이었다. 대체 내일 누굴 만날 것인가?

친구들에게 물어보았다. 하지만 친구들마다 다른 충고를 해주었다. 친구 이씨는 가장 예쁜 여자를 만나겠다고 하였고, 또 다른 친구 김씨는 가장 착한 여자를 만나겠다고 하였다.

하지만 진정한 카사노바 박씨는 어떻게 하면 더 많은 여자 친구를 만날 수 있을까 고민하던 중, 24 명의 여자 친구를 가장 예쁜 순서대로 줄을 서게 하고 순서대로 정해진 시간 만큼씩만 만나기로 하였다. 따라서 단 한명의 여자 친구만 카사노바 박씨를 만나는 상태이고, 다른 모든 여자 친구는 박씨를 기다리는 상태이다.

그런데 문제가 생겼다. 얼마나 오랫동안 만날 것인가? 1 시간으로 고정하면 어떨까? 할 얘기가 많은 여자 친구와도 1 시간, 할 얘기가 별로 없는 여자 친구와도 1 시간 만난다는 것은 비효율 적이었다. 그래서 카사노바 박씨는 할 얘기가 많은 여자 친구와는 보다 더 오랜 시간 만나기로 결정하였다.

또 다른 문제가 생겼다. 오늘 새로 알게 된 여자 친구가 급히 만나달라고 조르기 시작했다. 카사노바 박씨는 고민에 빠졌다. 여자 친구를 만나던 중 다른 여자 친구가 급히 만나자고 하면 응할 것인가 말 것인가. 카사노바 박씨는 급한 일이 있는 경우에만 다른 여자 친구를 만나기로 결정하였다.

그렇게 여러 여자친구를 만나고 있던 카사노바 박씨는 여러 여자친구를 만나고 있는 사실을 들키지 않게 치밀한 준비를 하였다. 지난번에 만났을 때 어디까지 얘기 했는지 수첩에 꼼꼼히 기록해 두었던 것이다. 그래서 박씨는 여러 여자친구를 만나더라도 했던 얘기를 또 하는 등의 실수를 범하지 않을 수 있었다.

비유 2:카사노바 박씨 이야기(운영체제 동작)
Linux 에겐 24 개의 태스크가 있다. 모두 CPU 스케줄링을 요청하고 대기 중이었다. Linux 는 대체 어떤 태스크를 스케줄링 해줄 것인가?

다른 OS 는 어떻게 하고 있는지 살펴보았다. 어떤 운영체제는 FIFO(First-in-First-Out)정책을 사용하고 있었고, 어떤 운영체제는 RM(Rate Monotonic)정책을 사용하고 있었다.

하지만 효율성이 높은 Linux 는 어떻게 하면 가장 많은 태스크를 수행시켜 줄 수 있을까 고민하던 중, 24 개의 태스크를 우선순위 순서대로 정렬한 뒤, 정해진 time slice 만큼씩만 CPU 자원을 할당하기로 하였다. 따라서 단 하나의 태스크만 실제 Running 상태고, 나머지 태스크는 모두 대기 중인 상태이다.

그런데 문제가 생겼다. Time slice 를 어떻게 결정할 것인가? 100Ms 정도로 고정하면 어떨까? 수행할 작업이 많은 태스크에게도 100ms, 수행할 작업이 적은 태스크에게도 100ms 씩 CPU 자원을 할당하는 것은 비효율적이었다. 그래서 Linux 는 태스크의 속성을 고려하여 time slice 를 결정한다.

또 다른 문제가 생겼다. 지금 새로 생성된 우선순위가 높은 태스크가 선점(Preemtion)을 요청하였다. Linux 는 고민에 빠졌다. 우선순위가 높은 태스크의 우선순귀가 낮은 태스크선점을 허용할 것인가 말 것인가? Linux 는 선점 기능을 지원한다.

동시에 여러개의 태스크를 하나의 CPU 에서 수행 시키고 있는 Linux 는 시스템의 원활한 동작을 위해 치밀한 준비를 하였다. 문맥 전환(Context Switch)시 현재 태스크가 어디까지 수행되었는지를 꼼꼼히 기록해 두었던 것이다. 그래서 Linux 는 여러개의 태스크를 동시에 수행시키더라도 문제없이 시스템을 운영할 수 있었다.

비유 2 는 CPU 가 프로세스를 동작시킬 때 어떠한 방식으로 동작되는가를 설명하였다. CPU 는 한방에 여러 연산을 처리하지 못하기 때문에, 한개 씩 우선순위가 높은 순서대로 처리한다. 우선순위를 두고 처리하지 않는다면 중요하지 않은 것에 시간을 빼앗기기 때문에 우선순위를 두고 처리한다
우선순위 1,2 가 있는데 1 이 하는일이 없어 2 가 CPU 연산을 점유하고 있

었는데 갑자기 우선순위 1 이 실행되어야 하는 상황이 오면 운영체제 안의 태스크에 우선순위 2 가 하던 작업을 저장해놓고 곧바로 우선순위 1 을 실행시킨다. 이것이 Context Switching 이다. 그런데 우선순위를 두지 않고 실행시킨다면, 우선순위 1 의 실행이 꼭 필요한 상황에 실행하지 못하기 때문에 효율성이 떨어진다.

비유 3:철수와 영희 이야기(일상생활)

철수는 오늘도 영희가 만나고 싶었다. 그래서 철수는 영희네 집 앞으로 찾아가 만나달라고 영희 방 창문에 돌을 던졌다.

돌을 던지면 창문은? 기본적으로는 깨진다. 창문이 깨지는 것이 싫다면, 영희는 누군가 돌 던질 때를 대비해서 항상 받을 준비를 하고 있어야 한다. 그런데 영희는 갑자기 궁금해졌다. 철수가 왜 돌을 던졌을까? 보고 싶으니 나오라고? 채팅 하고 싶으니 메신져에 들어오라고? 아님 화가 나서 그냥? 그래서 철수와 영희는 왜 돌을 던졌는지를 나타내는 번호를 돌에 쓰기로 결정하였다.

한편, 철수가 던진 돌을 받은 영희는 만나자는 번호가 써 있음을 알게 되었다. 그래서 세수도 하고 옷도 갈아입고, 만날 준비를 하고 있었다. 이 때 철수가 만나달라고 또 돌을 던지면 어떨까? 준비하고 있는데 또 돌을 던지니 매우 짜증이 날 것이다. 그래서 영희는 “나갈 준비하고 있으니 또 돌 던지지 마시오” 라고 창문에 써서 붙여 놓기로 하였다.

돌을 통해 간단한 번호라는 정보를 주고받던 철수와 영희는 문득 돌을 던지는 것이 빠르고 간단하긴 하지만 자세한 정보를 주고 받을 수 없다는 생각을 하게 되었다. 편지를 쓰면 어떨까? 전화를 하는 건 어떨까?

비유 3:철수와 영희 이야기(운영체제 동작)

A 태스크는 B 태스크에게 전달해 줄 사건이 발생했다. 그래서 A 태스크는 B 태스크에게 시그널(Signal)을 보냈다. 시그널을 받은 태스크는? 기본적으로 종료(Terminating)된다. 태스크가 종료되는게 싫다면 시그널 핸들러를 등록시켜 두어야 한다. 그런데 시그널을 받은 태스크는 왜 시그널을 보냈는지 궁금할 수 있다. 종료시키려고? IPC(Inter Process

Communication)을 위해서? 그래서 태스크 간 시그널을 보낼 때는 시그널의 이유별로 번호를 정한 뒤 그 번호를 보내주기로 하였다.

한편, A 태스크가 보낸 시그널을 받은 B 태스크는 해당 시그널 번호에 걸맞는 핸들러 함수를 호출하여 적당한 작업을 수행 중이었다. 그런데 이 때 같은 시그널이 또 배달 되지 않도록 해당 시그널을 블록(Block) 시켜놓기로 결정하였다.

시그널을 통한 통신 기법은 간단하고 빠르지만 자세한 혹은 대량의 정보를 주고받을 수 없다는 단점을 가진다. 메시지 큐(Message Queue)는 어떨까? 소켓(Socket)은 어떨까?

비유 3 은 시그널의 필요성과 처리 과정을 설명하고 있다.
태스크 종료를 창문이 깨진다에 비유했고, 시그널요청을 받아 핸들러를 동작시키는것을 영희의 행동에 비유했다. 시그널핸들러 동작시 블록 시켜놓는 것을 창문에 글을 써 놓은것으로 비유를 했다.

비유 4: 통신병 김군(일상생활)
오늘은 새벽 6 시부터 훈련이 시작되었다. 다른 소대/중대/대대와 연락을 주고 받을 필요가 없는 특수한 훈련이 아닌 이상 통신병 김군은 훈련의 시작부터 끝까지 반드시 지휘관 옆에 상주해야 한다.

언제 다른 부대에서 연락이 올지 몰랐기 때문에 통신병 김군은 항상 무전기를 지니고 다녔다. 그러다 연락이 오면 그 내용을 지휘관에게 보고하는 것이 바로 통신병의 임무였기 때문에..

그런데 갑자기 김군의 상관이 지시했다. “야 옆에 중대랑 연락좀 해봐라.” 그래서 김군은 무전기를 들고 연락을 시도했다. “여기는 1 중대, 2 중대 나오라 오버” 그러자 바로 응답이 왔다. “여기는 2 중대, 1 중대 말하라 오버.” 그러자 김군은 말했다. “잠시만 기다리시지 말입니다 오버” 그리곤 상관에게 무전기를 넘겼다.

한참 뒤 하고 싶은 대화를 끝낸 김군의 상관은 무전기를 김군에게 돌려주었다. 김군은 “이상 무전 끝” 이라고 말했고, 상대방은 “무전끝, 확

인” 이라고 대답했ㄷ. 그래서 김군은 “그럼 수고하시지 말입니다~”라고 익살스럽게 무전을 보냈다.

비유 4:통신병 김군(운영체제 동작)
컴퓨터에 전원이 켜졌다. 다른 컴퓨터와 통신할 필요가 없는 특수한 시스템이 아닌 이상 네트워크를 담당하고 있는 TCP/IP 스택은 컴퓨터가 종료될 때까지 활성화 되어 있어야 한다.

언제 다른 컴퓨터에서 패킷이 전송되어 올지 몰랐기 때문에 TCP/IP 스택은 항상 랜카드(Lan Card)를 살펴보고 있었다. 그러다 패킷이 전송되어 오면 그 내용을 커널에게 보고하는 것이 바로 TCP/IP 의 임무중 하나였기 때문에..

그런데 갑자기 응용프로그램에 의해 요청을 받은 커널이 지시했다. “다른 컴퓨터와 통신하게 연결 좀 설정해봐라~” 그래서 TCP/IP 스택은 다른 컴퓨터와 통신 연결을 시도했다. ‘SYN(Synchronization)’을 보냈고, 이에 대한 응답으로 ‘ACK(Acknowledge-ment)+SYN’을 받았으며, 최종적으로 다시 ‘ACK’를 전송하였다. 그리곤 통신이 가능한 소켓을 해당 응용프로그램에게 넘겼다.

한참 뒤 통신을 끝낸 응용 프로그램은 연결을 종료하려 하였다. 그래서 TCP/IP 스택은 ‘FIN(Finalizing)’을 전송하였고, 이에 대한 응답으로 ‘FIN+ACK’을 받았으며, 최종적으로 다시 ‘ACK’를 수신하였다.

비유 4 는 TCP/IP 소켓 통신을 비유했는데 통신병 김군을 TCP/IP 스택으로 비유하였다. TCP/IP 의 서버는 socket 생성과 주소셋팅및 초기화 이후 패킷이 전송될 것을 기다린다. 그러다가 요청패킷이 들어오면 응답을 해주고 요청해온 클라이언트와 통신을 시작한다.

이와 반대로 UDP 는 비연결지향형이어서 연결요청을 하지 않고 그대로 전송받는다. 연결 요청 과정이 없어서 UDP 가 TCP/IP 보다 빠르다.

비유 5: 크게 성공할 미래의 주방장 이군(일상생활)
식사 시간이 되어 주방장은 주방 보조 이군을 불렀다. “야 어서 창고에 가서 재료 가져와” 라고 이군에게 지시했다.

이군은 급히 창고로 이동하여 식량을 찾기 시작했다. 사실 이군은 전혀 당황하지 않았다. 각종 물건들을 창고 안에 여기 저기 대충 쌓아 놓으면 나중에 찾기가 불편할 것 이라고 판단한 이군은 평소 물건을 창고에 넣어둘 때 기가 막히게 정리를 해두었던 것이다.

우선 창고의 문 앞에는 창고 안에 어떤 물건들이 들어있고, 어떤 방식으로 정리되어 있으며, 또한 창고 안 어디에 빈 공간이 있는지를 나타내는 장부를 붙여 두었고, 창고 내부는 저장한 물건 종류별로 정리가 되어 있었으며, 각 종류별로 어떤 물건이 얼마나 있는지를 나타내는 정리표를 완벽하게 만들어서 붙여 놓았다.

비유 5: 크게 성공할 미래의 주방장 이군(운영체제 동작)

커널은 파일시스템(File System)에게 A.txt 라는 파일을 읽어오라고 명령을 내렸다.

파일시스템은 급히 하드디스크에서 A.txt 라는 파일을 찾기 시작했다. 사실 파일시스템은 전혀 당황하지 않았다. 각종 파일들을 하드디스크 안에 여기 저기 대충 저장해 놓으면 나중에 찾기 불편할 것이라고 판단한 파일시스템은 평소 파일을 디스크에 저장 할 때 Ext2(혹은 FAT, 혹은 LFS, 혹은 어떤 파일시스템)형식에 맞게 저장해 두었던 것이다.

우선 하드디스크의 제일 첫 부분에는 수퍼블록(Super Block)을 통해 해당 하드디스크를 위해 구축된 파일시스템의 전역적인 메타데이터를 담았고, 블록 할당/해제 정보를 담아 놓는 영역도 기록해 두었으며, 나머지 하드디스크 공간은 디렉토리 개념을 통해 정리가 되어 있었고, 파일 별로 메타데이터를 기록해 두었다.

비유 5 는 주방장 이군을 파일시스템에 비유했는데, 파일시스템이 파일을 저장할 때 어떠한 방식으로 저장하는 지 설명한다. 파일시스템은 블록으로 관

리하여 파일을 저장해 두었다가 필요할 때 마다 메모리에 바로 찾기 쉽도록 도와준다.

```
//c 언어로 구현하는 웹서버.
//web_serv.c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#include<arpa/inet.h>
#include<sys/socket.h>
#include<pthread.h>

#define BUF_SIZE 1024
#define SMALL_BUF   100

typedef struct sockaddr_in     si;
typedef struct sockaddr *      sp;
void error_handling(char* msg){
        fputs(msg, stderr);
        fputc('\n',stderr);
        exit(1);
}

void send_error(FILE* fp)
{
        char* protocol = "HTTP/1.0 400 Bad Request\r\n";
        char* server = "Server:Linux Web Server\r\n";
        char* cnt_len = "Content-length:2048\r\n";
        char* cnt_type = "Content-type:text/html\r\n\r\n";
        char*                                   content                                 =
"<html><head><title>Network</title></head>"
```

```
                "<body><font size=+5><br> 오류 발생! 요청 파일명
및 방식 확인!"
                "</font></body></html>";
        fputs(protocol, fp);
        fputs(server, fp);
        fputs(cnt_len, fp);
        fputs(cnt_type, fp);
        fflush(fp);
}

char* content_type(char* file)
{
        char extension[SMALL_BUF];
        char file_name[SMALL_BUF];
        strcpy(file_name, file);
        strtok(file_name, ".");
        strcpy(extension, strtok(NULL, "."));

        if(!strcmp(extension, "html")|| !strcmp(extension, "htm"))
                return "text/html";
        else
                return "text/plain";
}
void send_data(FILE* fp, char* ct, char* file_name)
{
        char* protocol = "HTTP/1.0 200 OK\r\n";
        char* server = "Server:Linux Web Server\r\n";
        char* cnt_len = "Content-length:2048\r\n";
        char* cnt_type[SMALL_BUF];
        char* buf[BUF_SIZE];
        FILE* send_file;

        sprintf(cnt_type, "Content-type:%d\r\n\r\n", ct);
        send_file = fopen(file_name, "r");
```

```
	if(send_file == NULL)
	{
		send_error(fp);
		return;
	}

	fputs(protocol, fp);
	fputs(server, fp);
	fputs(cnt_len,fp);
	fputs(cnt_type, fp);

	while(fgets(buf, BUF_SIZE, send_file) != NULL)
	{
		fputs(buf, fp);
		fflush(fp);
	}

	fflush(fp);
	fclose(fp);
}

void* request_handler(void* arg)
{
	int clnt_sock = *((int*)arg);
	char req_line[SMALL_BUF];
	FILE* clnt_read;
	FILE* clnt_write;

	char method[10];
	char ct[15];
	char file_name[30];

	clnt_read = fdopen(clnt_sock, "r");
	clnt_write = fdopen(dup(clnt_sock), "w");
```

```
	fgets(req_line, SMALL_BUF, clnt_read);

	if(strstr(req_line, "HTTP/") == NULL)
	{
		send_error(clnt_write);
		fclose(clnt_read);
		fclose(clnt_write);
		return;
	}
	strcpy(method, strtok(req_line, " /"));
	strcpy(file_name, strtok(NULL, " /"));
	strcpy(ct, content_type(file_name));

	if(strcmp(method, "GET") != 0)
	{
		send_error(clnt_write);
		fclose(clnt_read);
		fclose(clnt_write);
		return;
	}

	fclose(clnt_read);
	send_data(clnt_write, ct, file_name);
}

int main(int argc, char** argv)
{
	int serv_sock, clnt_sock;
	si serv_addr, clnt_addr;
	int clnt_addr_size;
	char buf[BUF_SIZE];
	pthread_t t_id;

	if(argc != 2)
	{
```

```
		printf("Use: %s <port>\n",argv[0]);
		exit(1);
	}
	serv_sock = socket(PF_INET, SOCK_STREAM, 0);
	memset(&serv_addr, 0, sizeof(serv_addr));
	serv_addr.sin_family = AF_INET;
	serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
	serv_addr.sin_port = htons(atoi(argv[1]));

	if(bind(serv_sock, (sp)&serv_addr, sizeof(serv_addr)) == -1)
		error_handling("bind() error");
	if(listen(serv_sock, 20) == -1)
		error_handling("listen() error");

	for(;;)
	{
		clnt_addr_size = sizeof(clnt_addr);
		clnt_sock = accept(serv_sock, (sp)&clnt_addr,
&clnt_addr_size);
		printf("Connection Request: %s %d\n",
inet_ntoa(clnt_addr.sin_addr), ntohs(clnt_addr.sin_port));
		pthread_create(&t_id, NULL, request_handler,
&clnt_sock);
		pthread_detach(t_id);
	}

	close(serv_sock);

	return 0;
}
```