TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 : Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 : 황수정
sue100012@naver.com
32 일차 (2018. 04. 06)

목차

1. Chapter 0 : 운영체제 이야기 - 동작비유

비유 1 : 회사 <<운영체제 동작>>

Firmware Level 의 프로그래머 A 씨는 여러 가지기능을 하는 프로그램을 만들고 있다. 프로그램 의흐름을 계획하여 모든 것을 Firmware Level 에서 코딩하다 보니 이젠 더 이상 모든 것을 다 직접 코딩하기엔 역부족임을 느끼게 되었다.

→ 스케줄링을 할 수가 없기 때문이다. Firmware 는 운영체제(OS)가 없는 상태에서 돌아가는 소프트웨어이다. OS 가 서포트 해주는 기능 중에 중요한 것은 스케줄링이다. 스케줄링은 각 프로세스마다 얼마나 시간을 할당해줄 것인지를 결정한다. 스케줄링은 타이밍 값을 다 계산해준다. 즉, 이 기능이 없다는 것은 모든 동작(CPU 가 명령을 내리거나 스택에 올라가거나 메모리에 접근하는 것 등)을 사람이 계산해야 한다는 것이다. 이는 레지스터 정보 저장하는 것과 CPU 클럭 타이밍을 맞추어서 계산해야 한다는 것인데, 이는 매우 어렵다.

예를 들어 '캐쉬(cache)'를 보자. 캐쉬가 적중하면 메모리에 접근하지 않는다. 반면 적중하지 않으면 메모리에 접근하게 되는데 이 때, 걸리는 시간은 다르다. 사람은 계산으로 이 시간을 구할 수 없다. 이는 CPU 칩만 알 수 있다. 이것은 사람이 CPU 칩을 설계해도 마찬가지이다. 언제 접근할지 알 수 없기 때문이다. 만약 이 타이밍 값이 꼬여서 연산에 사용되고 있는 레지스터 하나를 버리게 된다면, 값이 다 꼬이는 것이다.

운영체제 없이 여러 작업(태스크, 프로세스, 쓰레드 등)을 제어하려면 기본적으로 스케줄링이 필요한데 어떤 타이밍에 제어해줘야 할지 모르기 때문에 값이 다 꼬이게 되어버린다. context switching 과 multi-tasking 은 스케줄링을 기반으로 일어난다. 시간이 만료되면 context switching 을 하고 제어권을 넘겨준다. 따라서 스케줄링이 안되면 이 두 가지가 되지 않는다.

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

→ zImage 는 리눅스 커널의 압축된 이미지다. Bootloader 는 하드웨어를 초기화해준다. 기본적으로 모든 하드웨어들은 전원이 들어오면 모든 장치들을 처음에 초기화를 해주어야 한다. 초기화시키지 않으면 제대로 동작하지 않는다. 안 그러면 0 도 1 도 아닌 애매한 상태가 된다. 이 때, 값이 들어가면 정확한 값이 산출되지 않기 때문이다. 리눅스의 경우, 가상메모리 레이아웃도 잡아줘야 한다. Root Filesystem 은 '/'으로 나타나는데, vim 등으로 추상화시킨 디스크를 관리한다.

→ 역부족을 느꼈다는 부분은 스케줄링 부분이다. 그래서 운영체제를 사용한 것이다.

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.

→ 시스템 디자인은 하드웨어를 디자인한다는 것이다(input 이 들어가면 output 이 어떻게 나올게 할 것인가). 회로 설계하고 PCB 로 장치를 만드는 것에 집중했다는 뜻이다. 시스템 디자인이란 어떤 A 라는 신호가 들어왔을 때, 이 장치가 어떤 식으로 구동하게 할 것인지 설계하는 것을 뜻

한다. 이걸 가능하게 해주는 기능이, 시스템 설계에만 집중할 수 있게 해주는 것이 디바이스 드라이버 덕분이다. 디바이스 드라이버는 어떤 장치가 들어오면 그 것이 뭔지 판별하고 그 장치가 구동시키기 위한 소프트웨어를 돌아간다(이 작업은 OS 전문가가 한다).
리눅스는 Firmware Level 의 프로그래머 A 씨가 해야 했던 모든 일들을 완수하기 위해 태스크를 생성했다. 시스템이 정지되기 전까지 계속 동작하는 백그라운드 데몬과 쉘 테스크. 태스크는 할일이 없을 땐 계속 idle 상태를 유지하며 쉬게 된다. 그러나 데몬은 누가 일을 시키지 않더라도 해야 할 일이 생기면 Running 상태가 되어 자신에게 주어진 일을 수행하였고, 쉘은 A 씨가 시키는 일이 있을 때마다 Running 상태가 되어 시킨 일을 수행하였다.
→ 데몬은 터미널을 꺼도 동작하기 때문이다. 섹션 부분이 '?'로 데몬을 확인 할 수 있다. 어디에도 속하지 않았기 때문에 컴퓨터를 끄기 전까지 안 꺼진다.

해야 할 일들이 점점 다양해지자 리눅스는 더 많은 태스크를 생성하였다. 우선기존의 데몬을 fork()하여 데몬 2 를 만들었고, fork()후 exec()하여 태스크 E, F, G 를 생성하였다.
데몬, 데몬 2, E, F, G 그리고 쉘 태스크는 한정된 메모리 상에서 동고 있지만, B 는 각각의 태스크가 필요로 할 때마다 메모리를 잘 할당/해제해 주었기 때문에 모든 태스크가 성공적으로 자신의 작업을 완료할 수 있었다.
→ 메모리 관리

비유 2 : 카사노바 박씨 이야기 <<운영체제 동작>>

Linux 에겐 24 개의 태스크 가 있다.
→ (중요) 어떻게 꼬이지 않게 하는가? 스케줄링이 답이다.

어떤 운영체제는 FIFO(First-In First-Out) 정책을 사용하고 있었고 어떤 운영체제는 RM(Rate Monotonic) 정책을 사용하고 있었다.
→ RM 은 비율로 우선순위 높은 프로세스 할당량을 많이 주는 것으로 실시간으로 적용해야 하는 계열에서 효과적이다. FIFO 는 실시간 적용이 중요한 곳에서는 사용할 수 없다.

정해진 time slice 만큼 CPU 자원을 할당하기로 하였다.
→범용적인 사항을 말하는 것으로 리눅스는 하이브리드 방식을 채택한다. 리눅스에도 우선순위가 변하는 것이 있고 안 변하는 것이 있다. 리눅스는 정적으로 고정된 우선 순위와 동적으로 변하는 우선순위가 있다.

Linux 는 선점 기능을 지원한다.
→ 싱글 코어 CPU 에서 말하는 것이다. CPU 는 오로지 한 순간에 하나의 작업만 할 수 있기 때문이다. 중요한 작업이 있으면 time slice 가 만료되지 않아도 wait queue 로 빠지고 우선순위 높은 것이 실행된다. 시스템 콜이 우선순위가 유저 영역보다 높아서 사용자 명령보다 먼저 수행된다.
'리눅스는 태스크 속성을 고려해서 time slice 를 결정한다.' 는 것은 우선순위를 보겠다는 뜻이다.

즉, 예제 2 는 Multi-Tasking 을 구현했다는 뜻이다.

2. Chapter 1 : 리눅스 소개

1) 리눅스의 탄생

리눅스는 리누스 토발즈가 386 보호모드(인텔 계열 머신에만 있다.)에서 작업 전환기를 만들고 거기에 페이징, 타이머 인터럽트 핸들러 등을 만들어 내놓게 된 운영체제이다. 거기에 자유 소프트웨어 재단(Free Software Foundation)이라는 단체의 GNU(Gnu is Non Unix)의 프로젝트가 합세하면서 탄생된 것이다.

여기서 자유 소프트웨어 재단은 컴퓨터 전반의 모든 분야에 공개(free) 소프트웨어를 개발하고 사용하게 함으로써 상용 소프트웨어 프로그램에 대한 복사, 배포, 이해, 수정에 대한 모든 제한을 없애기 위해 노력하는 단체이다. 이는 코드 개발 등으로 돈을 벌 수 없다는 뜻이다. 따라서 라이선스를 등록할 때, 프리로는 돈을 벌 수 없으며 오픈는 코드는 공개하지만, 돈을 벌 수 있다. 따라서, 여러 라이센스들이 있고 잘 선택해야 한다. 리눅스는 '원하는 사람은 누구나 그 소스를 수정하여 성능을 향상시킬 수 있으며 내용도 공개해야 한다.'라는 정신이 있 dj 막강한 성능과 안정성을 가지게 되었다.

2) 유닉스와 리눅스

리눅스는 유닉스 계열의 운영체제이다.

생명을 제공하는 태스크와 장소를 제공하는 파일이라는 두 가지 객체로 모든 것을 지원한다 → '모든 것은 파일이다.' 라는 리눅스의 핵심 철학을 뜻한다. 파일은 디스크에 있으며 프로세스가 실행되지 않은 상태이다.

리눅스는 모널리식 구조를 기반으로 만들어져 있다. 모놀리식 커널은 탈부착이 불가능하다. 이에 리눅스는 디바이스 드라이버만큼은 마이크로 커널로 만들었다. 이 구조는 탈부착이 가능하다. 그래서 USB 를 꽂았다 뺐을 때 컴퓨터가 재부팅이 되지 않는 이유이다. 만약 이 구조가 아니라면 컴퓨터가 나가게 된다.

3) 리눅스의 배포판

4) **리눅스의 장점**

- **사용자 임의대로 재구성이 가능**하다.
  → 커스텀 마이징이 가능하다. 언어, 어셈블리어를 공개했기 때문이다.
- 열악한 환경에서도 HW 자원을 적절히 활용하여 동작한다.
  → 이는 성능이 좋지 않은 컴퓨터에서도 리눅스를 깔 수 있게 해준다. 모놀리식 방식에 필요한 건 마이크로 커널로 하이브리드이기 때문이다. **최적화가 매우 잘 되어있다**는 뜻
- 커널의 크기가 작다.
  → 윈도우보단 리눅스가 작다.
- 완벽한 멀티유저, 멀티태스킹 시스템
- **뛰어난 안정성**
  → 네트워크 장비나 기상 기지국에서 사용될 정도로 안정성을 검증 받았다.
  네크워크 고성능 장비에 들어간다는 것 자체가 안정성이 증명된 것이다.
- 빠른 업그레이드

- 강력한 네트워크 지원→ TCP/IP
- 풍부한 소프트웨어 → 스톨만의 GNU 덕분이다.
- 사용자를 위한 여러 가지 공개 문서들

C 언어 프로그래머들이 가져야 할 올바른 자세 : 운영체제의 소스 자체가 오픈 되어 있으니 안 된다면 내가 되게 하면 된다는 마인드! 물론 실력이 뒷받침 되어야 하므로 공부 또한 열심히 하자!

5) 쉘(Shell) 사용해보기

Chapter 2 : 리눅스 커널 구조

커널이란 운영체제의 핵심을 이루는 부분이다. CPU(태스크)나 메모리(페이지) 그리고 기타 디바이스(드라이버) 등의 시스템 리소스를 관리하고, 사용자 프로그램이 이를 상용할(System call) 수 있도록 한다.

1) 리눅스 커널 구조

운영체제는 자원관리자이다.

운영체제는 자원공간은 task_struct 이다. int main() 이건 유저영역이다. 단, 만든 프로그램이 구동되면 자원이다. 프로그램 만든 파일 또한 자원이다. 커널이 관리하면 자원이란 것이다. 로직자체는 사용자 관리라 자원이 아니다.

운영체제가 관리해야 할 자원은 물리적인 자원과 추상적인 자원으로 구분된다.

물리적인 자원 : CPU, 메모리, 디스크, 터미널, 네트워크 등 시스템을 구성하고 있는 요소들과 주변 장치 등

추상적인 자원 : CPU 를 추상화 시킨 태스크(task), 메모리를 추상화 시킨 세그먼트와 페이지, 디스크를 추상화 시킨 파일, 네트워크를 추상화 시킨 통신 프로토콜, 패킷 등 물리적인 자원에 대응되지 않으면서 추상적인 객체만으로 존재하는 보안과 접근제어(소프트웨어 자원)도 있다.

태스크 관리자는 태스크의 생성(fork(), pthread_create), 실행(exec), 상태전이(wait queue, run queue), 스케줄링, 시그널 처리, 프로세스 간 통신(IPC) 등의 서비스 제공한다.

메모리 관리자는 물리메모리 관리(DRAM 관리), 가상메모리 관리(stack, data, heap, text), 세그멘테이션, 페이징, 페이지 부재결함(segmentation fault) 처리 등의 서비스 제공한다.

파일 시스템은 파일의 생성(open-O_CREAT), 접근제어, inode 관리, 디렉터리 관리, 수퍼 블록 관리 등의 서비스 제공한다.

네트워크 관리자는 소켓 인터페이스, TCP/IP 같은 통신 프로토콜 서비스 제공한다.

디바이스 드라이버는 디스크나 터미널, CD, 네트워크 카드 등 주변장치를 구동하는 드라이버들로 구성된다.

그럼 운영체제는 무엇을 위하여 이렇게 자원을 관리하는 것일까? **사용자에게 서비스를 제공하기 위함**이다. 서비스는 시스템 호출을 의미한다.
< 리눅스 커널의 소스트리 구조>

kernel 디렉토리
태스크의 생성과 소멸, 프로그램의 실행, 스케줄링, 시그널 처리 등의 기능이 이 디렉토리에 구현되어 있다.

arch 디렉토리
리눅스 커널 기능 중 하드웨어 종속적인 부분들이 구현된 디렉터리이다. architecture 의 줄임말인 arch 이다. 이 디렉터리는 CPU 의 타입에 따라 하위 디렉터리로 다시 구분된다.

fs 디렉토리
리눅스에서 지원하는 다양한 파일시스템과 open(), read(), write()등의 시스템 호출이 구현된 디렉토리이다. 가상파일시스템도 이 디렉토리에 존재한다.

mm 디렉토리
메모리 관리자가 구현된 디렉토리이다. 물리메모리관리, 가상 메모리 관리, 태스크마다 할당되는 메모리 객체 관리 등의 기능이 구현되어 있다.

driver 디렉토리
리눅스에서 지원하는 디바이스 드라이버가 구현된 디렉터리이다. 디바이스 드라이버란 디스크, 터미널, 네트워크 카드 등 주변장치를 추상화 시키고 관리하는 커널 구성요소이다. 디바이스 드라이버를 3 가지로 구분하는데 파일시스템을 통해 접근되는 블록 디바이스 드라이버, 사용자 수준 응용 프로그램이 장치파일을 통해 직접 접근하는 문자 디바이스 드라이버, TCP/IP 를 통해 접근되는 네트워크 디바이스 드라이버이다.

net 디렉토리
리눅스가 지원하는 통신 프로토콜이 구현된 디렉터리이다. 통신 프로토콜의 추상화 계층이며 사용자 인터페이스를 제공하는 소켓 역시 이 디렉터리에 구현되어 있다.

ipc 디렉토리
리눅스 커널이 지원하는 프로세스간 통신 기능이 구현된 디렉토리이다.

init 디렉토리
커널 초기화 부분, 커널의 메인 시작함수가 구현된 디렉토리이다. 이 함수가 커널 전역적인 초기화를 수행한다.

include 디렉토리

리눅스 커널이 사용하는 헤더 파일들이 구현된 디렉터리이다. 헤더 파일 중에서 하드웨어 독립적인 부분은 include/linux 하위 디렉토리에 정의되어 있고 하드웨어 종속적인 부분은 include/asm-$(ARCH)디렉토리에 정의되어 있다.

others 디렉토리
리눅스 커널의 주요 기능이 구현된 디렉토리 외에도 리눅스 커널 및 명령어들이 자세한 문서 파일들이 존재하는 Documentation 디렉토리, 커널 라이브 함수들이 구현된 lib 디렉토리, 커널 구성 및 컴파일 시 이용되는 스크립트들이 존재하는 scripts 디렉토리 등이 존재한다.

2) 리눅스 커널 컴파일

Chapter 3 : 태스크 관리

리눅스는 태스크를 통해 다양한 생명과 변화를 제공한다.
태스크는 프로세스와 쓰레드를 가지고 있다. 다양한 생명이란 (fork()와 thread 로) 프로그램이 구동되어 프로세스가 살아나서 제어권을 갖게 된다는 뜻이다. 변화는 exec() 등이 가능하기 때문이다.

1) 프로세스와 쓰레드 그리고 태스크

태스크란 무엇일까? 프로세스나 쓰레드와의 차이점은 무엇일까?
Vi –t task_struct 명령어로 task_struct 내부를 살펴보게 되면 pid 와 tgid 가 선언 되어있는 것을 볼 수 있다. pid 는 프로세스 아이디이고, tgid 는 쓰레드 그룹 아이디이다. 프로세스와 쓰레드의 차이점은 pid 와 tgid 가 같으면 프로세스이고 pid 와 tgid 가 다르면 쓰레드로 프로세스 구성원이다. 쓰레드 리더일 때 프로세스라고 볼 수 있다.

ELF – 리눅스의 실행파일 포맷
DWARF – 리눅스의 디버깅 파일 포맷

프로세스는 동작중인 프로그램이며 커널로부터 할당 받은 자신만의 자원을 가지고, CPU 가 기계어 명령들을 실행함에 따라 끊임없이 변화하는 동적인 존재이다. 즉, **실행파일은 메모리에 올라가야 프로세스**다. 이 때, 커널이 시스템에 존재하는 여러 개의 프로세스 중 CPU 라는 자원을 어느 프로세스에게 할당해 줄 것인가를 결정하는 작업을 스케줄링이라고 부른다.

2) 사용자 입장에서 프로세스 구조

사용자 프로세스가 수행되기 위해서는 여러 가지 자원들을 커널로부터 할당 받아야 한다. 프로세스의 구조를 살펴보면 유저 영역과 커널 영역으로 분리되어 있다. 공간 할당은 32bit, 64bit 에 따라 다르다. 사용자 프로그램 중에서 명령, 함수 등으로 구성되는 텍스트 영역은 프로세스 주소 공간 중 가장 하위공간을 차지한다. 텍스트 영역 다음에는 전역변수 등을 담아 놓는 데이터 영역

이 차지한다. 한편 프로세스는 수행 중에 malloc()이나 free() 등의 함수를 사용하여 동적으로 메모리 공간을 할당 받고 그 영역을 heap 이라 한다. 이 영역이 데이터 영역의 다음 부분을 차지한다. 함수의 지역변수 등을 담는 스택 영역은 사용자 공간과 커널 공간의 경계위치부터 아래방향으로 공간이 자라고 힙은 아래에서 위쪽으로 자란다.
프로세스는 크게 텍스트, 데이터, 스택, 힙이라는 세그먼트로 구분하고 vm_area_struct 가 관리한다. 이 때, 각 영역을 세그먼트 또는 가상 메모리 객체라고 부른다.