TI DSP, MCU, Xilinx Zynq FPGA Based Programming Expert Program

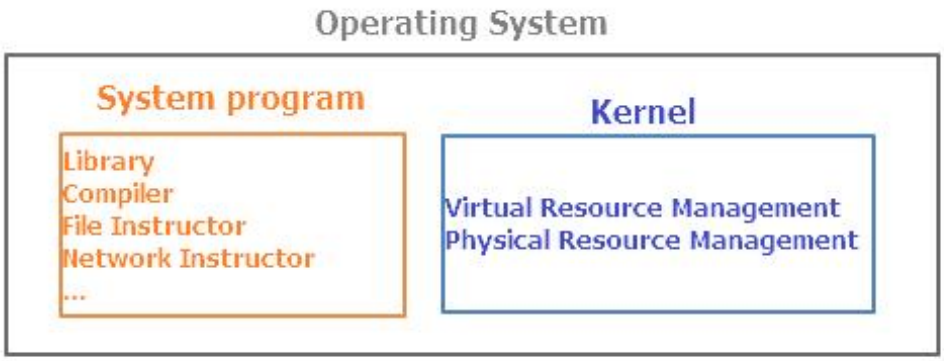
Instructor – Innova Lee (Sanghoon Lee)
gcccompil3r@gmail.com
Student – Howard Kim (Hyungju Kim)
mihaelkel@naver.com

배울 내용이 너무 많고 개념적인 부분이라 “리눅스 커널 내부 구조“ 파트만은 한글로 작성합니다.
제대로된 학습이 먼저!! 이후 포트폴리오 제작시 오개념 잡아가며 영작 예정

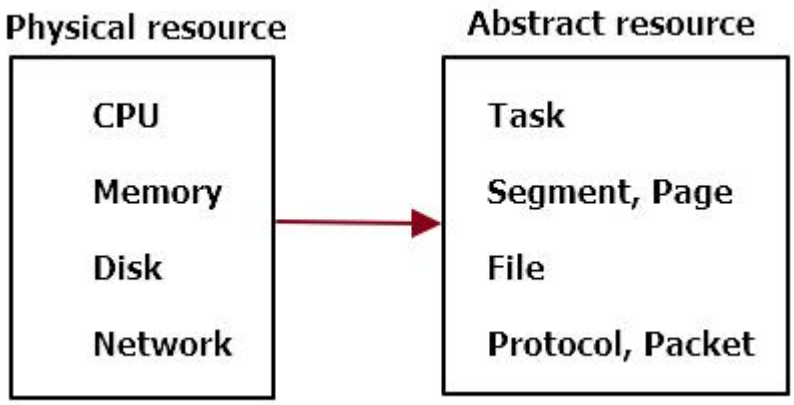
리눅스 커널 내부 구조

1.OS 란?

OS란 크게 시스템 프로그램과 커널로 이루어진 Operating System을 말한다.
시스템 프로그램에는 H/W control Library, Compiler, File instructor 등이 있다.
커널이란, 자원 관리자(Resource manager)로서 운영체제의 핵심 기능을 담당한다.



자원(resource)란 무엇일까?
자원(resource)에는 물리적 자원(physical resource)과 추상적 자원(virtual resource)가 있다.
CPU, Memory, Disk, Network 등이 물리적 자원이다. 추상적 자원은, 이 물리적 자원들을 추상화(abstraction) 한 것을 의미한다.

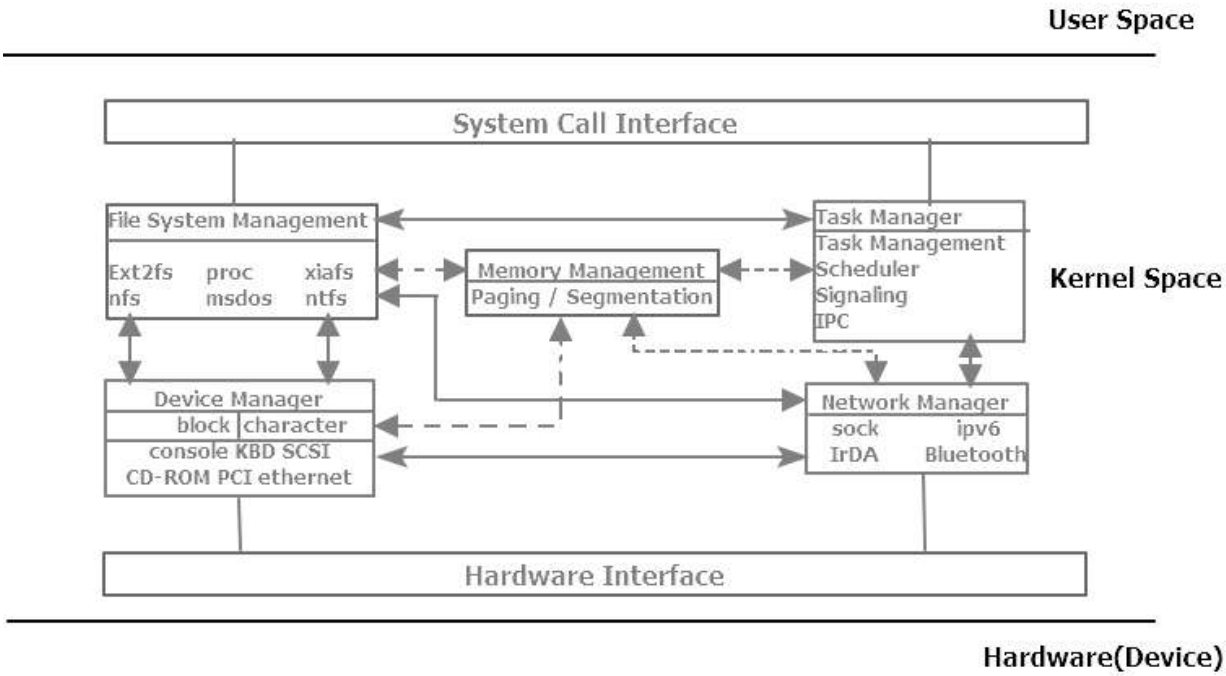


다만, 모든 추상적 자원(virtual resource)가 물리적 자원(physical resource)에 대응하는 것은 아니다.
보안(security)나 사용자 ID에 따른 접근 제어(access control) 등이 이러한 자원의 예이다.

리눅스 커널 내부 구조

3.간략하게 보는 커널의 내부 구조

커널의 내부에는 리소스의 종류에 따라 5가지로 구분된 자원 관리자들이 있다.



1)태스크 관리자 : 태스크의 생성(fork, pthread_create), 실행, 상태 전이, 스케쥴링, 시그널 처리, IPC 등의 서비스를 제공한다.

2)메모리 관리자 : 물리 메모리 관리, 가상 메모리 관리, segmentation, paging, Segmentation fault 처리, page fault 처리 등의 서비스를 제공한다.

3)파일 시스템 : 파일의 생성, 접근 제어(stat struct), inode 관리, 디렉토리 관리, 수퍼 블록 관리 등의 서비스를 제공한다.

4)네트워크 관리자 : 소켓 인터페이스, 통신 프로토콜 등의 서비스를 제공한다.

5)디바이스 드라이버 : 디스크나 터미널, CD, LAN카드 등 장치 구동에 필요한 firmware로 구성되어 있다.

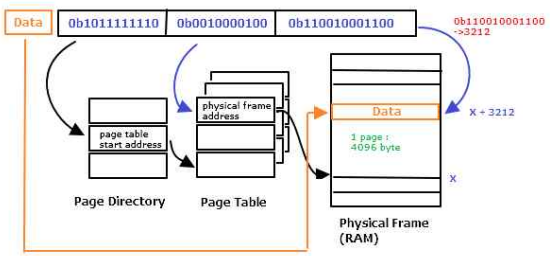
왜 위의 자원(resource)들을 추상화(abstraction) 할까?

user mode에서 접근이 가능할 때, 여러 가지 문제들을 초래할 수 있기 때문에.

즉, 운영체제라 하는 것은, user space의 task가 system call을 사용하여 여러 가지 자원(resource)들을 문제 없이 사용할 수 있도록 인터페이스를 제공해주는 자원 관리자(resource mansger)이다.

4.물리 메모리 접근법, Page기법과 page fault, segmentation fault

우리가 사용하는 주소는 가상 주소(virtual address)이다. 실제 메모리(DRAM 등)에 있는 주소로 접근하기 위해서, Paging 기법을 사용한다. Paging 기법이란 아래와 같이 가상 주소(virtual address)의 각 비트를 나누어, 물리적 주소(physical address)로 접근하는 방법을 말한다.



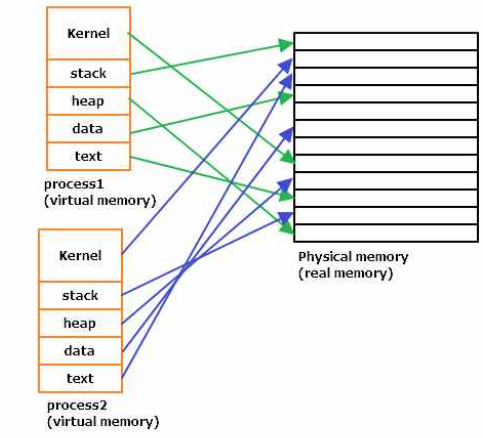
리눅스 커널 내부 구조

가상 메모리(virtual memory) 또는 segment를 물리적 주소로 mapping하는 것은 CPU 내부에 있는 H/W인 MMU(Memory Management Unit)이 해준다. MMU를 S/W로 구현한다면, MMU의 기능을 하는 S/W 자체도 segment가 존재하고 물리적 주소(physical address)로 mapping되어야 하기 때문에, 성능이 매우 떨어진다. address mapping operation은 상당한 overhead를 발생시키기 때문. 그렇기에 MMU가 없는 MCU에는 os를 올릴 수 없다.(RTOS는 가능)

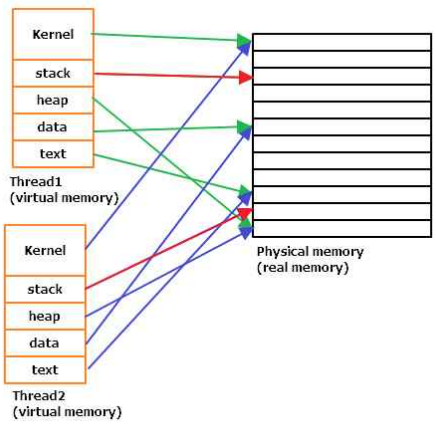
컴퓨터가 처음 부팅되고, process가 하나 실행됐을 때를 가정해보자.
그 전에 실행되었던 process가 없기 때문에, segment가 나타내는 가상 주소(virtual address)에 있는 page에는 물리 주소(physical address)가 mapping 되어 있지 않을 것이다. process가 호출하였지만, 해당 page에 해당하는 물리 주소(physical address)가 없는 상태가 "page fault"이다. page fault가 발생하면, 권한을 확인하여 user mode인지 kernel mode인지를 판별한다. 판별 후 kernel mode 일 시, MMU가 interrupt를 발생하여 page fault handler라는 S/W를 실행하고 새로운 물리 주소(physical address)를 해당 페이지에 할당하여 문제를 해결한다.
user mode로서 mapping되지 않은 page에 접근할 때에, 권한이 없기 때문에 "Segmentation fault"를 발생시키고 process를 종료한다.

4-1.COW(Copy On Write)
자식 프로세스를 생성할 때에는, fork()함수를 사용한다. fork()함수를 사용하면, 그 시점에서 process를 pause하고, 부모 프로세스의 segment를 그대로 복사하여 새로운 프로세스를 생성한다. 즉, fork()만 사용해서는 전혀 다른 프로그램을 만들 수 없었다.
그래서 생각해낸 기법이 fork() 후 exec()하는 방법이었다. exec()을 하면, segment에 있는 정보들을 모두 overwrite하여 전혀 다른 프로세스를 생성할 수 있다. 그러나, 이미 있는 segment를 지우고 overwrite하는 것 자체가 매우 비효율적인 방법이었다. 그래서 fork()할 때에, 한 번에 모든 segment를 복제하지 않고, 직접 메모리에 쓸 때마다(copy on write) 복제하게 되었다. 그래서 생긴 개념이 COW 이다.

5.process, thread and task??
프로세스와 쓰레드, 태스크의 개념을 알아보자.
메모리 계층 구조(Memory hierachy)를 보면, 모든 c언어 프로그램은 stack, heap, text, data의 Segment로 이루어져 있고, 각 Segment마다 역할이 있다. 실행되기 전까지는 file로 저장되어 disk 영역에 존재한다. 파일이 실행되고 나면, task_struct -> mm_struct -> vm_area_struct를 생성하여 process가 된다. process 내에 thread를 생성하는 함수가 있다면, stack을 제외한 나머지 영역을 공유하는 thread를 생성한다.



1)2개의 프로세스, segment가 서로 독립적



2)2개의 쓰레드, stack을 제외한 segment 공유

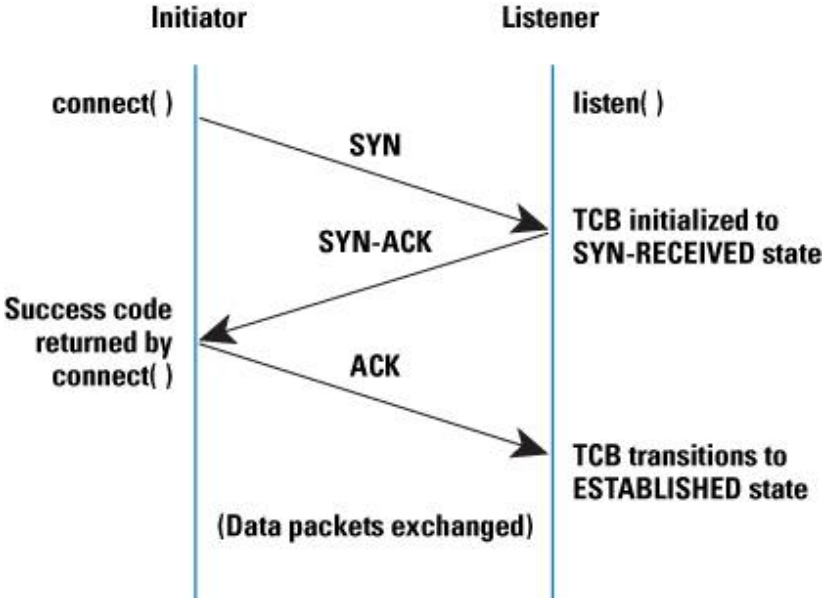
즉, process는 프로그램의 단위이고, thread는 동작의 단위이다. thread를 생성하지 않으면, process 내의 leader thread 자체가 process가 된다. 물론 detach함수를 통해 process로부터 독립할 수도 있다. getpid()를 통해 얻는 값은 정확히는 thread ID가 되고, gettgid()를 통해 얻는 값이 process ID가 된다. getpid() == gettgid()라면, process이고, 다르다면, gettgid()에 해당하는 process에 종속된 thread가 된다.

task란 process와 thread 모두를 말한다. 초기 컴퓨터에서는 multi-tasking이 지원이 되지 않았다. 즉, 한 번에 한 가지 프로그램(그 당시엔 job이라고 불렸음)을 수행하였고, 그것을 task로 정의하였다. 시간이 지남에 따라 process의 개념이 생겨나고, multi-tasking하는 데 있어 process의 비효율성을 보완하기 위해 thread가 생겨났다. task라는 개념은 굳이 지우지 않고 전해져 온 것으로, process와 thread 모두를 내포해도 문제가 없을 듯 하다.

리눅스 커널 내부 구조

6.TCP 통신을 위한 network 연결 방식 3-way handshaking

네트워크 연결을 위해, 3-way handshaking이라는 방식을 사용한다.



먼저, Client에서 Server로 SYN 신호를 보내 Client가 통신가능하다고 알린다.
Server에서 SYN 신호를 받으면, Client가 통신가능하다는 것을 확인했다는 신호로 ACK를 보낸다.
ACK와 함께, Server 자신도 통신가능하다고 알리기 위해 SYN 신호를 같이 송신한다.
SYN-ACK 신호를 수신한 Client는 Server가 통신 가능하다는 것을 확인했다는 신호로 ACK를 송신한다.
Server가 ACK를 수신하면, 통신망이 구축된다.
통신망 구축에 데이터 송/수신이 3번 일어나기 때문에, 3-way handshaking 이라고 한다.