컴파일 시스템

각 c코드들은 다른 프로그램들에 의해 저급 기계어 명령어들로 번역되어야한다.
전처리단계-> 컴파일단계-> 어셈블리단계-> 링크단계 (전컴어링)

1) 전처리단계 : 주석 제거하고, #include를 만나면, 해당 소스 파일을 찾고
해당 내용을 복사한다. #define 지시문(매크로)의 내용들을 치환하고 적용한다.

2) 컴파일단계 : 컴파일은 인간이 이해할 수 있는 언어로 작성된 코드를 CPU가 이해할 수 있는 기계어로 번역하는 작업을 말함.
(언어의 문법 검사를 하고, static 영역에메모리할당)

3) 어셈블단계 : 어셈블리어는 사람이 이해할 수 있게 부호화한 것인데, 기계어와 1:1로 매칭. 어셈블단계는 어셈블리어를 오브젝트어로 바꾸는 작업을 한다.
*오브젝트 코드는 사람이 더 이상 알아볼 수 없는 기계어로 변환된 파일.

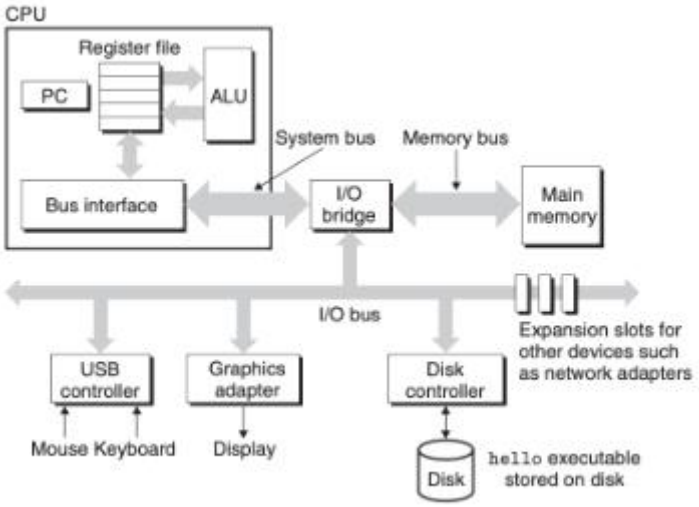
4) 링크 단계 : 링커를 통해 오브젝트파일을 묶어 실행파일로 만드는 단계이다.

시스템 하드웨어 조직

1) 버스: 시스템 내를 관통하는 전기적 배선-> 컴포넌트들 간에 바이트 정보들을 전송한다.
2) 입출력장치: 시스템과 외부세계와의 연결을 담당(키보드 마우스 모니터 기타...)
3) 메인메모리: 프로세서가 프로그램을 실행하는 동안 데이터와 프로그램을 모두 저장한다. (DRAM) -> 논리적으로메모리는연속적인바이트들의배열이다.
4) 프로세서 : CPU에 저장된 기계어들을 해독하는 엔진이다. 레지스터(저장장치)인 프로그램카운터(PC) 가있다.
-> 프로세서는 PC가 가리키는 곳의 명령어를 실행하고, PC값이 다음 명령어의 위치를 가리키도록 업데이트한다.
-> 프로세서는 PC가 가리키는 메모리로부터 명령어를 읽어오고, 명령어가 지정한 동작을 실행한 뒤, 다음 명령어로 업데이트한다.

Hello를 입력하면, 각각의 문자를 레지스터에 읽어 들인 후, 메모리에 저장한다.
레지스터는 cpu가 요청을 처리하는데 필요한 데이터를 일시적으로 저장하는 기억장치이다.

*레지스터 : CPU가 요청을 처리하는데 필요한 데이터를 일시적으로 저장하는 기억장치이다.
실제로 컴퓨터에서 데이터를 영구적으로 저장하기 위해서는 하드디스크를 이용.
임시적으로 저장하는 장소를 메모리(RAM)로 알고 있다.
그러나 메모리로 연산의 결과를 보내고 영구적으로 저장할 데이터를 하드에 저장하는 명령을 처리하기 위해선, 이들에 대한 주소와 명령의 종류를 저장할 수 있는 기억공간이 필요한데, 이것은 메모리보다 빨라야 한다. 이것이 레지스터다.



레지스터 공간은 작지만, cpu와 직접적으로 연결이 되어있어 연산속도가 메모리보다 실제 수십 배~수백 배 빠르다.

그리고 cpu는 자체적으로 데이터를 저장할 방법이 없기 때문에 메모리로 직접 데이터를 전송할 수 없다.
때문에 연산을 위해서는 반드시 레지스터를 거쳐야하며, 이를 위해서 레지스터는 특정 주소를 가리키거나 값을 읽어올 수 있다.

1) hello를 입력하게 되면, 각각의 문자를 레지스터가 읽어 들이고 I/o bridge를 통해 메인 메모리에 저장한다.

2) 파일 내 코드와 데이터를 복사하는 명령어를 실행하여, hello를 디스크에서 메인 메모리로 옮긴다. 이때 디스크는 메인메모리로 직접 이동한다.

3) 데이터가 메모리에 적재된 후, 해당 프로그램은 main 루틴의 기계어가 명령을 실행한다.
이 명령어들은, 문자 hello를 메모리부터 레지스터 파일로 보내 복사하고
마지막으로 Display에 보여주면 된다.

* PCB 프로세스 컨텍스트 스위칭하고 그 전에 있던 작업 내용을 기억해야하는데,
프로세스 단위로 정보를 저장해주는 BLOCK을 process control block이라고 한다.

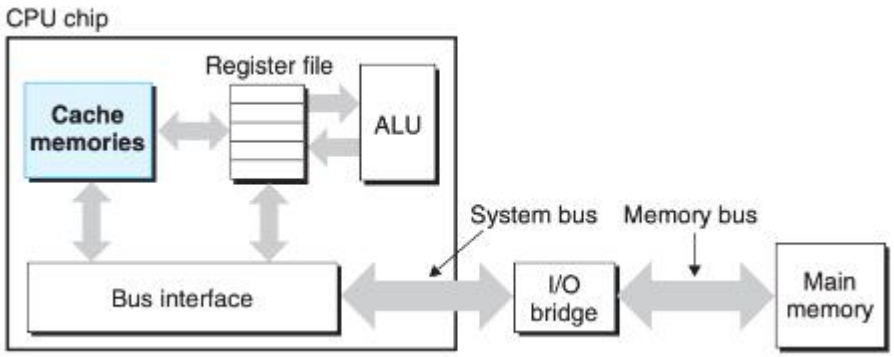
캐시

위 과정으로 시스템이 정보를 한 곳에서 다른 곳으로 이동시키는데 많은 시간을 보냈음을 알 수 있다.
본디 **데이터들은 하드디스크에 저장**되어 있는데,
프로그램이 로딩 될 때, 메인 메모리로 복사가 된다.

데이터 스트링 역시 디스크에 저장되어 있으나, 메인 메모리로 복사 되고, 디스플레이로 복사.

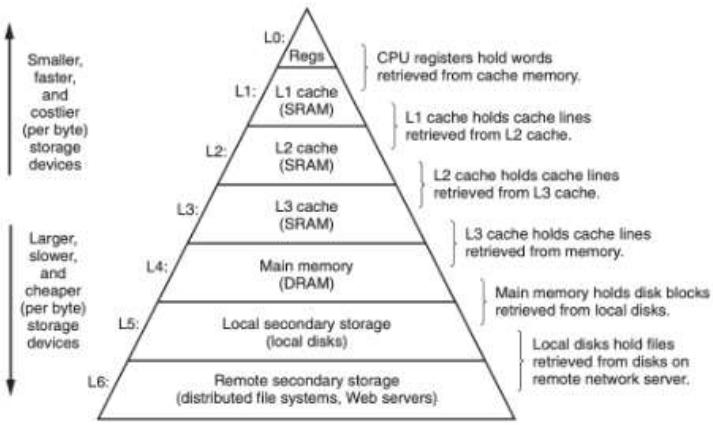
프로그래머의 관점에서 이러한 복사 과정들은 오버헤드이기에, 시스템 설계자들의 주요한 목적은 이러한 복사과정을 빠르게 하는 것.

캐시메모리 : 프로세서가 단기간에 필요할 가능성이 높은 정보를 임시로 저장하는 메모리



L1 캐시는 대략 수천 바이트의 데이터를 저장할 수 있고, 레지스터 파일만큼 빠르게 접근.
L2 캐시는 수백 킬로바이트~수메가 바이트 저장 가능. 전용 버스를 통해 연결한다.

이러한 캐시들은 SRAM이라는 하드웨어 기술을 이용해 구현한다.
* 프로그래머들은 저 캐시를 활용하여 자신의 프로그램 성능을 개선할 수 있다.



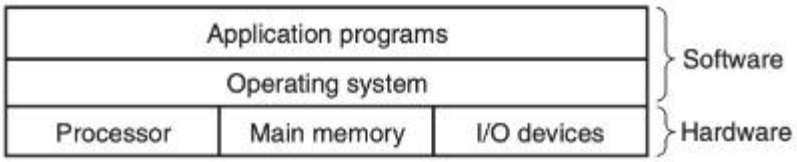
위 그림은 저장 장치 계층 구조이며, 아래로 갈수록 느리고 싸진다.

L1은 L2의 // L2는 L3의 // L3는 메인메모리의 // 메인메모리는 디스크의 캐시이다.

운영체제

시스템 하드웨어를 관리할 뿐 아니라 응용 소프트웨어를 실행하기 위하여,
하드웨어 추상화 플랫폼과 공통 시스템 서비스를 제공하는 시스템 소프트웨어

hello 프로그램을 로드하고 실행했을 때 / 메시지를 출력할 때 프로그램은
키보드, 디스플레이, 디스크, 메인 메모리에 엑세스하지 않고
운영체제가 제공하는 서비스를 활용한다.



운영체제는 HW, SW 사이에 위치한 소프트웨어 계층으로 생각할 수 있다.
고로, 응용 프로그램이 하드웨어를 제어하려면, OS가 필요하다.

OS는
1. 제멋대로 동작하는 응용 프로그램들이 HW를 잘못 사용하는 것을 막는다.
2. 응용 프로그램들이 단순하고 균일한 메커니즘을 사용하여 복잡하고 매우 다른 저수준
HW 장치들을 조작할 수 있도록한다.

프로세스

1. 프로세스와 프로그램의 차이
   프로그램 자체는 생명이 없다. 프로그램은 보조기억 장치(HDD, SSD)에 존재하며,
   실행되기를 기다리는 명령어와 정적인 데이터의 묶음이다.

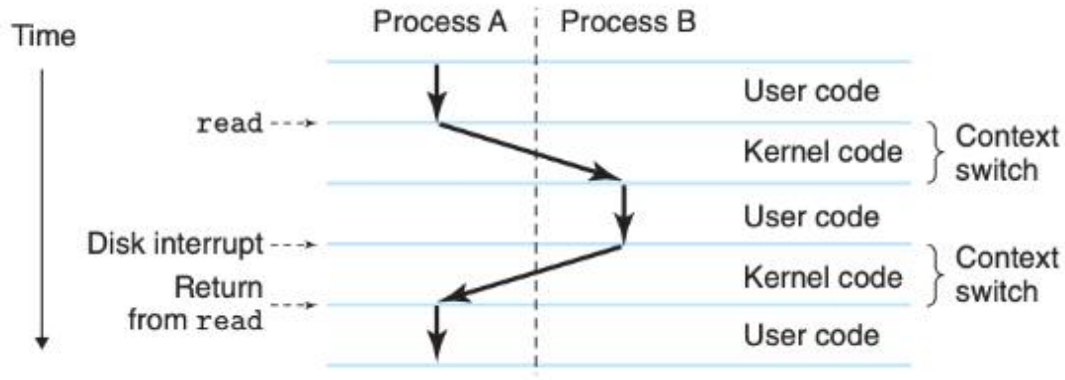
   이 프로그램의 명령어와 정적 데이터가 메모리에 적재가 되면, 생명이 있는 프로세스가 됨

즉 프로세스란, 실행중인 프로그램이다.

컴퓨터에서는 여러 개의 프로세스들이 동시에 실행된다. (카톡을 하면서 웹 서핑을 하거나..)
하나의 cpu는 한순간에 하나의 프로세스만 실행할 수 있는데...!
이는, 운영체제가 엄청나게 빠른 속도로 CPU가 실행할 프로세스를 교체하고 있기 때문이다.

* 문맥전환 (context switch)
운영체제는 컨텍스트 스위칭이라는 방법을 통해 교차 수행을 실시한다.
컨텍스트라 부르는 상태 정보는 PC, 레지스터 파일, 메인 메모리의 현재 값을 포함하고 있다.
운영체제는 현재 프로세스에서 다른 프로세스로 제어를 옮기려고 할 때, 현재 프로세스의
컨택스트를 저장하고 새 프로세스의 컨텍스트를 복원시키는 문맥 전환을 실행하여

제어권을 새프로세스로 넘겨주고, 새 프로세서는 이전에 중단했던 그 위치부터 다시 실행

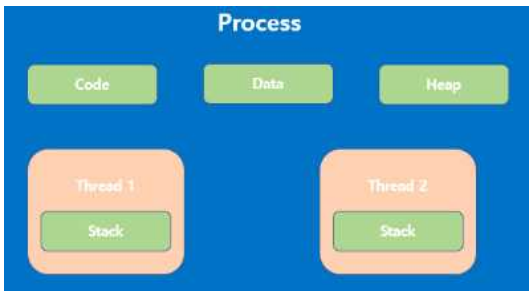


위 그림처럼 두 개의 동시성 프로세스가 존재한다.
처음에 A프로세스가 동작하다가 명령줄에서 입력을 기다림.
그러다가 B프로세스를 실행하라는 명령을 받으면, A는 **시스템 콜**이라는 함수를 호출하여 운영체제로 제어권을 넘겨준다.
OS는 A의 컨텍스트를 저장하고 새로운 B 프로세스와 컨텍스트를 제어권을 B프로세스로 넘김
B의 프로세스가 종료되면 다시 컨텍스트를 복구 시키고 제어권을 넘겨주면서 다음 명령줄 입력을 기다린다.

하나의 프로세스에서 다른 프로세스로의 전환은 **OS 커널**에 의해 관리된다.
커널은 OS 코드의 일부분으로 메모리에 상주한다.
프로그램 -> OS 로 작업 요청 시, **파일 읽기, 쓰기**와 같은 특정 시스템 콜을 실행

쓰레드

프로세스는 실제로 쓰레드라고 하는 다수의 실행 유닛으로 구성되어 있다.
각각의 쓰레드는 해당 프로세스의 컨텍스트에서 실행되며, 동일한 코드와 전역 데이터를 공유한다.



즉 프로세스는 프로세스 내에서 실행되는 흐름의 단위 혹은 CPU의 스케쥴링 단위이다.
1. 쓰레드는 STACK 영역을 보유한다. (최소한 자신의 레지스터 상태를 보유한다)
2. 쓰레드는 프로세스 내에서 code, data, heap 영역을 공유한다.
3. 쓰레드를 생성하고 switching 하는 것은 그리 비용이 들지 않다.

각 프로세스는 별도의 주소 공간에서 실행되며, 한 프로세스는 다른 프로세스의 변수나 자료구조에 접근 불가.

쓰레드와 프로세스 차이

1. 프로세스는 각자 프로세스 간 통신에 IPC가 필요하다.
   그러나 쓰레드는 쓰레드간 통신에 IPC가 필요하지 않다.

2. 각 프로세스는 code, data, heap, stack 영역을 각자 보유한다.
   그러나 쓰레드는 code, data, Heap영역은 공유하고 Stack 영역만 각자 보유한다.

3. 프로세스는 생성과 컨텍스트 스위칭에 비용이 많이 들어간다
   그러나 쓰레드는 생성과 컨텍스트 스위칭에 비용이 적게 들어간다

* IPC는 프로세스간 통신 (한 프로세스가 다른 프로세스의 자원에 접근하려는 것)

<멀티 프로세스와 멀티 쓰레드>

멀티 프로세싱은 여러 개의 프로세스가 각자 하나의 작업을 맡아 처리하는 것을 뜻함.
멀티 쓰레딩은 여러 개의 쓰레드가 각자 하나의 작업을 맡아 처리하는 것을 뜻함

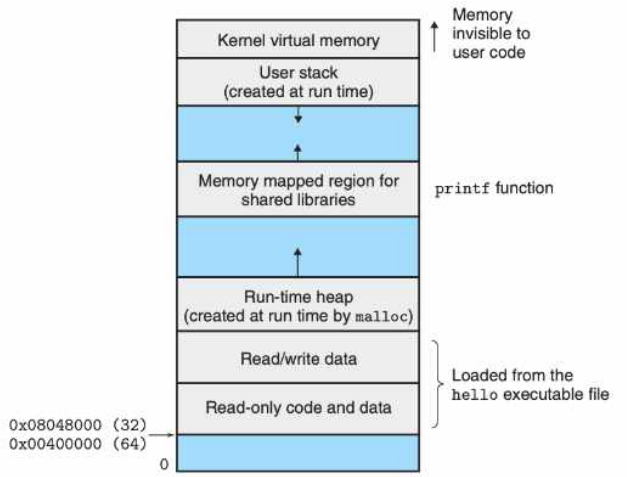
<멀티 프로세스>
1) context switching 발생시 캐시에 존재하는 모든 데이터를 리셋하고 다시 캐시 정보를 불러와야한다. -> 오버헤드 비용이 크다.
2) 프로세스간의 통신에 복잡한 IPC를 사용해야한다.
3) 여러개의 자식 프로세스 중 하나에 문제가 발생하면 그 자식 프로세스에만 이상이 생기고 다른 프로세스에 영향을 주지 않는 장점도 존재.

<멀티 쓰레딩>
1) 시스템 자원 소모가 감소한다. -> 프로세스를 생성하는 시스템 콜이 줄어,
자원을 효율적으로 관리가 가능하다.
2) 시스템 처리량이 증가한다.(throughput). concurrenct와 parallelism을 얻을 수 있다.
3) 프로세스 내의 data, code, heap 영역을 공유해서 쓰레드간 통신이 원활하다.
4) 쓰레드간 자원을 공유하기 때문에 동기화 문제가 발생할 수 있다.
5) 디버깅이 까다롭다.
6) 하나의 쓰레드에서 문제가 생기면 전체 프로세스에 영향을 준다.

--> 프로세스는 어떻게 구성되어 있을까?
PID (process Identification) -> 운영체제가 각 프로세스를 식별하기 위해 부여된 번호
프로세스 상태 -> cpu가 빠르게 프로세스들을 교체하기 때문에 대기, 실행 중인 상태를 말함
프로그램 카운터 -> CPU가 다음 실행할 명령어를 가리키는 값. 다음 메모리 주소를 가리킴.
스케줄링 우선순위 -> 여러개의 프로세스가 실행되는 순서를 결정하는 것을 스케줄링이라 함
권한 -> 프로세스마다 어디까지 접근할 수 있는지에 대한 권한이 주어진다.
프로세스의 부모와 자식 프로세스 -> 최초 init을 제외하고 모든 프로세스는
부모 프로세스로부터 복제되어 생성된다.
프로세스의 데이터와 명령어가 있는 메모리 위치를 가리키는 포인터 ->
프로세스는 실행중인 프로그램이다. 그러므로 프로그램에 대한 정보를 갖고 있어야한다.
프로그램에 대한 정보는 프로세스가 메모리에 가지는 자신만의 주소공간에 저장됨.

<가상메모리>

가상 메모리는 각 프로세스들이 메인 메모리 전체를 독점적으로 사용하고 있는 것 같은
환상을 제공하는 추상화다. -> 각 프로세스는 아래 그림처럼 가상 주소 공간이라고 하는
균일한 메모리의 모습을 갖게 됨



주소공간의 최상위 영역은 모든 프로세스들이 공통으로 사용ㅇ하는 운영체제의 코드와
데이터를 위한 공간이다.

1) 프로그램 코드와 데이터 : 코드는 모든 프로세스들이 같은 고정 주소에서 시작.
   그 뒤, 전역변수에 대응되는 데이터 위치들이 따라온다. 코드와 데이터 영역은 실행 가능
   목적파일로부터 직접 초기화 됨

2) 힙(HEAP) : 코드&데이터 영역 다음으로 런타임 힙이 따라온다.
   크기가 고정되어 있는 코드, 데이터 영역과 달리, 힙은 프로세스가 실행되면서
   런타임에서 동적으로 크기가 줄었다 늘었다 한다.

3) 공유 라이브러리 : 주소 공간의 중간 부근에 C 표준 라이브러리나 수학 라이브러리와 같은 공유 라이브러리와 같은 공유 라이브러리의 코와 데이터를 저장하는 영역

4) 스택 : 사용 가상 메모리 공간의 맨위에 컴파일러가 함수 호출을 구현하기 위해 사용자 스택이 위치하고, 프로그램 실행시간 동안 늘었다가 줄었다 한다.

5) 커널 가상 메모리 : 주소 공간의 맨 윗부분은 커널을 위한 공간으로 할당됨.
응용프로그램은 접근 자체가 안되며, 호출도 금지되어있다. 사용 원할시, 커널 호출

우선 메모리는 프로그램에 필요한 코드나 데이터를 저장하는 장치
내부기억장치 (주기억 장치), 외부기억장치 (보조 기억장치) 로 나뉨
DRAM(RAM, DDR4)등의 메모리, CPU 안에 있는 레지스터와 캐시 등이 전자에 해당.
SSD, HDD 가 후자에 해당함.