1.1 Information Is Bits + Context

우리가 만든 hello 프로그램은 '소스 프로그램' 혹은 '소스 파일'로 시작되고 hello.c라고 불리는 텍스트 파일형식으로 저장된다. 소스 프로그램은 각각 0 또는 1의 값을 갖는 비트의 순서로, 바이트라고 하는 8비트 덩어리로 구성된다. 각 바이트는 프로그램에서 일부 텍스트 문자를 나타낸다. 시스템은 한 바이트 크기의 정수 값으로 각 문자를 나타내는 ASCII 표준을 사용하여 텍스트 문자를 나타낸다.
** 아스키 코드 - 한 문자당 한 가지의 정수로 대응시키는 정수를 나타낸 코드
hello.c와 같이 아스키 코드로 이루어진 파일을 'text file'이라 하고 그 외에 모든 파일들은 'binary file'이라 한다.

<C 프로그래밍 언어의 기원>
C는 Dennis Ritchie에 의해 1969 ~ 1973년에 개발되었다.
- C는 유닉스 운영체제와 밀접한 관련이 있고, 처음부터 유닉스의 시스템 프로그램이 언어로 개발되었다. 대부분의 우ㅠ닉스 커널과 모든 지원 도구 및 라이브러리는 C로 작성되었다.
- C의 단순성으로 인해 다른 컴퓨터를 배우고 이식하기가 비교적 쉬워졌다.
- C는 실용적인 목적으로 설계되었고, 유닉스 운영체제를 구현하도록 설계되었다.

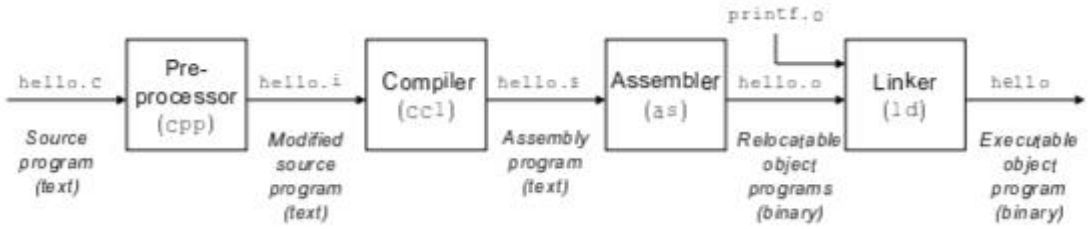
C는 모든 프로그래머와 모든 상황에 완벽하지 않다. C 포인터는 혼란과 프로그래밍 오류의 일반적인 원인이다. C에는 클래스, 객체 및 예외와 같은 유용한 추상화에 대한 명시적인 지원도 없다. 그래서 C++과 Java와 같은 최신 언어는 이러한 문제를 해결한다.

1.2 Programs Are Translated by Other Programs into Different Forms

hello.c 시스템을 실행하기 위해선 각각의 C문이 다른 프로그램에 의해 낮은 수준의 machine-language로 변환되어야 한다. 이러한 명령어들은 실행 가능한 오브젝트 프로그램이라는 형식으로 패키지되어 2진 디스크 파일로 저장된다.
유닉스 시스템에서 소스 파일에서 객체 파일로의 변환은 컴파일러 드라이버에 의해 수행된다.

unix > gcc -o hello hello.c

여기서 GCC 컴파일러 드라이버는 소스 파일 hello.c를 읽고 실행 가능한 오브젝트 파일 hello로 변환한다. 전처리기 / 컴파일러 / 어셈블러 / 링커를 수행하는 프로그램을 통칭하여 컴파일 시스템이라고 한다.



- 전처리 단계 : 전처리기(Pre-processor(cpp))는 C 프로그램에서 #문자로 시작하는 디렉티브에 따라 수정한다. 그 결과 hello.i라는 새로운 C 프로그램이 생성된다.
- 컴파일 단계 : 컴파일러(Compiler(ccl))은 hello.i를 hello.s로 번역하고 이 파일에 어셈블리어를 저장한다. 어셈블리 언어는 다른 고급 언어에 대해 다른 컴파일러에 공통 출력 언어를 제공하기 때문에 유용하다.
- 어셈블리 단계 : 어셈블러(Assember(as))는 hello.s를 기계 언어 명령어로 변환하고 재배치 가능한 객체 프로그램으로 알려진 형태로 패키지화 한 다음 결과를 개체 파일 hello.o에 저장한다.
- 링크 단계 : 링커(Linker(ld))는 print.o와 hello.o 파일을 결합하여 printf 함수를 호출 할 수 있도록 한다.

** GNU 프로젝트.
GNU 프로젝트는 소스 코드가 수정되거나 배포되는 방법에 대한 제한으로 인해 소스 코드가 제한되지 않는 완전한 유닉스 계열 시스템을 개발하려는 야심 찬 목표를 가지고 있습니다. GNU 프로젝트는 Linux 프로젝트에 의해 별도로 개발 된 커널을 제외하고 유닉스 운영 체제의 모든 주요 구성 요소가 있는 환경을 개발했습니다. GNU 환경에는 EMACS 편집기, GCC 컴파일러, GDB 디버거, 어셈블러, 링커, 이진 조작 유틸리티 및 기타 구성 요소가 포함됩니다. GCC 컴파일러는 다양한 기계에 대한 코드를 생성 할 수있는 능력으로 다양한 언어를 지원하도록 성장했습니다. 지원되는 언어에는 C, C ++, Fortran, Java, Pascal, Objective-C 및 Ada가 있습니다.

1.3 It Pays to Understand How Compilation Systems Work

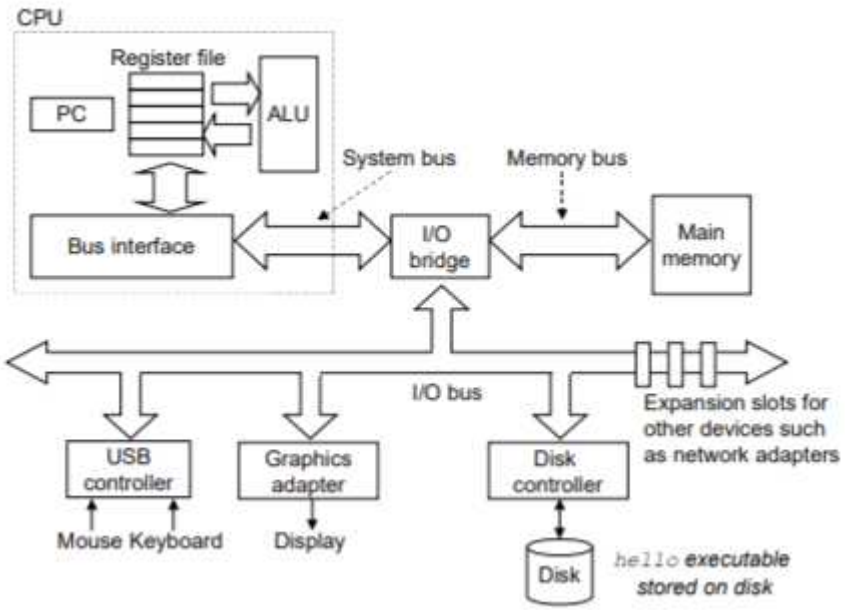
- 프로그램 성능 최적화 : 프로그래머는 효율적인 코드를 작성하기 위해 컴파일러의 내부 작동을 알 필요는 없지만 올바른 C 코딩을 위해선 컴파일러가 다른 C 명령문을 기계어로 변환하는 방법에 대한 기본 지식은 필요하다.
  Microprocesser와 컴파일러가 C 구문을 다른 언어로 변환하는 방법, 컴파일러가 C 코드를 간단하게 변환하여 프로그램의 성능을 향상시키는 방법 그리고 메모리 시스템의 계층적 특성을 배울 것이다.
- 링크 타임 오류 이해 : 복잡한 프로그래밍 오류 중 일부는 링커 작동과 관련이 있다.
  1) 링커에서 참조를 해결할 수 없다고 보고
  2) 정적 변수와 전역 변수의 차이점
  3) 이름이 같은 다른 C 파일에 두 개의 전역 변수 정의
  4) 정적 라이브러리와 동적 라이브러리의 차이점
  5) 라이브러리를 나열하는 순서가 중요한 이유
  6) 링커 관련 오류가 런타임까지 나타나지 않은 이유
- 보안 허점의 회피 : 오래동안 버퍼 오버플로우 취약점은 네트워크 및 인터넷 서버의 보안 허점을 설명해왔다. 보안 프로그래밍 학습의 첫 번째 단계는 데이터 및 제어 정보가 프로그램 스택에 저장되는 방식의 결과를 이해하는 것이다. 앞으로 스택 규율 및 버퍼 오버플로우 취약점과 컴파일러 및 운영체제에서 공격 위협을 줄이기 위해 사용할 수 있는 방법을 배운다.

1.4 Processors Read and Interpret Instructions Stored in Memory

hello.c 소스 프로그램은 hello라는 실행 가능한 오브젝트 파일로 번역되어 디스크에 저장된다. 유닉스 시스템에서 이 파일을 실행하기 위해선 쉘이라고 알려진 응용 프로그램에 입력해야한다. 쉘은 명령어를 입력 받아 그 명령어를 실행해주는데 만일 입력된 명령어가 내장 쉘 명령어가 아니면 쉘은 이를 실행파일의 이름으로 판단하고 그 파일을 로딩 해준다. 이 경우 쉘은 hello 프로그램을 로딩하고 실행 한 후 종료를 기다린다. hello 프로그램은 메시지를 출력하고 종료한다. 쉘은 다음 명령어를 입력하기를 기다린다.

1.4.1 Hardware Organization of a System

hello 프로그램이 어떻게 실행되는지 이해하려면 하드웨어 구성을 이해해야 한다.



- Buses :
시스템 전체에서 Bus로 구성 요소 사이에 바이트 정보를 주고 받는다. Bus는 고정된 크기의 바이트 청크를 전송하도록 설계되었다.

- I/O Devices :
입출력 장치(I/O)에는 사용자 입력을 위한 키보드 및 마우스, 출력을 위한 디스플레이, 데이터 및 프로그램의 장기 저장을 위한 디스크 등이 있다. 각 I/O장치는 컨트롤러 또는 어댑터를 통해 I/O 버스에 연결된다. 컨트롤러는 장치 자체 또는 시스템의 기본 인쇄 회로 기판에 있는 칩셋이다. 어댑터는 마더 보드의 슬롯에 꽂는 카드이다. 각각의 목적은 I/O 버스와 I/O 장치 간에 정보를 주고 받는 것이다.

- Main Memory :
메인메모리는 프로세서가 프로그램을 실행하는 동안 조작하는 데이터와 프로그램을 모두 저장하는 임시 저장 장치입니다. 물리적으로 메인메모리는 DRAM(Dynamic Random Access Memory) 칩 모음으로 구성된다. 메모리는 바이트들의 배열이고 각 배열은 각자의 주소를 가지고 있다.

- Processor :
중앙 처리 장치 / 프로세서(CPU)는 메인 메모리에 저장되어 있는 기계어들을 실행하는 역할을 한다. CPU 중심에는 레지스터(storage device)인 PC(program counter)가 있다. 시스템에 전원이 공급되는 순간부터 꺼질 때까지 CPU는 PC가 가리키는 명령어를 수행한다. CPU는 기계어를 수행하고 PC는 다음 수행할 기계어가 있는 메모리를 가리키고 프로세서는 이 명령어를 읽고 해석하고 수행한다. 레지스터 파일은 고유 이름을 갖는 워드 크기의 레지스터의 집합이고, ALU는 새 데이터와 주소 값을 계산한다.

- Load : 메인 메모리에서 레지스터에 워드 또는 한 바이트 단위로 이전 값에 덮어쓰는 방식으로 복사하는 것.
- Store : 레지스터에서 메인 메모리로 한 바이트 또는 워드 단위로 이전 값에 덮어쓰는 방식으로 복사하는 행위.
- Operate : 두 레지스터 값을 ALU로 복사한 뒤, 수식연산을 수행하고, 결과를 레지스터에 덮어쓰는 방식으로 저장하는 것.
- Jump : 인스트럭션 자신으로부터 한 개의 워드를 추출하고, 이것을 PC에 덮어쓰는 방식으로 복사.

현대 프로세서는 복잡한 메커니즘을 사용하여 프로그램 실행 속도를 높인다.

1.4.2 Running the hello Program

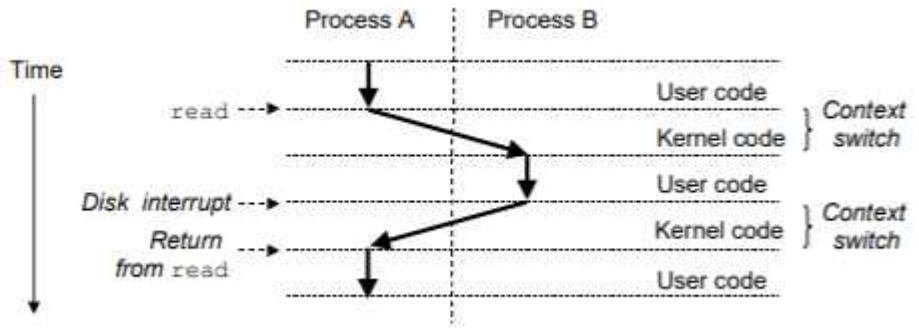
시스템의 하드웨어 구성 및 운영을 이해하면 프로그램을 실행할 때 어떤 일들이 발생하는지 이해할 수 있다.

1.7 The Operating System Manages the Hardware

쉘이 hello 프로그램을 로드하고 실행할 때, hello 프로그램이 메시지를 인쇄할 때 프로그램이 키보드, 디스플레이, 디스크 또는 메인메모리에 직접 접근하지 않고 운영체제에 의존한다. 그림 1.10과 같이 운영체제를 응용프로그램과 하드웨어 사이에 있는 소프트웨어 계층으로 생각할 수 있다. 응용프로그램이 하드웨어를 조작하려는 모든 시도는 운영체제를 거쳐야한다.운영체제는 두 가지의 주요 목적이 있다. (1) 런어웨이 응용프로그램에 의한 하드웨어 오용 방지, (2) 단순하고 균일한 메커니즘을 응용프로그램에 제공.

1.7.1 Processes

프로세스는 실행중인 프로그램에 대한 운영체제의 추상화이다. 동일한 시스템에서 여러 프로세스를 동시에 실행할 수 있으며 각 프로세스는 하드웨어를 독점적으로 사용하는 것으로 보인다. 대부분의 시스템에는 실행할 CPU보다 많은 프로세스가 있다.
운영 체제는 프로세스를 실행하는 데 필요한 모든 상태 정보를 추적한다. 어느 시점에서나 단일 프로세서 시스템은 단일 프로세스에 대한 코드만 실행할 수 있다. 운영 체제가 현재 프로세스에서 일부 새로운 프로세스로 제어를 전송하기로 결정하면 현재 프로세스의 컨텍스트를 저장하고 새 프로세스의 컨텍스트를 복원 한 다음 새 프로세스로 제어를 전달하여 컨텍스트 전환을 수행한다. 그림 1.12는 예제 hello 시나리오의 기본 아이디어를 보여줍니다.



예제 시나리오에는 쉘 프로세스와 hello 프로세스의 두 가지 동시 프로세스가 있어서 처음에는 셸 프로세스가 단독으로 실행되고 명령 줄에서 입력을 기다리다가 hello 프로그램을 실행하도록 요청할 때, 쉘은 제어를 운영 체제에 전달하는 시스템 호출이라는 특수 기능을 호출하여 요청을 수행한다. 운영 체제는 셸 컨텍스트를 저장하고 새 hello 프로세스와 해당 컨텍스트를 만든 다음 새 hello 프로세스로 제어를 전달한다. hello가 종료 된 후 운영 체제는 쉘 프로세스의 컨텍스트를 복원하고 제어를 다시 전달하여 다음 명령 행 입력을 기다린다.

1.7.2 Threads

현대 시스템에서는 프로세스가 실제로 스레드라는 여러 실행 단위로 구성된다. 여러 프로세스간에보다 여러 스레드간에 데이터를 공유하기가 쉽고 스레드가 일반적으로 프로세스보다 효율적이다.

1.7.3 Virtual Memory

가상메모리는 각 프로세스에 메인메모리만 사용함을 보여주는 추상화이다. 각 프로세스는 가상 주소공간으로 알려진 동인한 메모리 뷰를 가지고 있다.

- Program code and data
코드는 모든 프로세스에 대해 동일한 고정 주소에서 시작하고 코드와 데이터 영역을 실행 가능한 객체 파일의 내용에서 직접 초기화된다.

- Heap
프로세스 실행이 시작되면 크기가 고정 된 코드 및 데이터 영역과 달리, malloc 및 free와 같은 C 표준 라이브러리 루틴에 대한 호출의 결과로 힙이 런타임에 동적으로 확장 및 축소된다.

- Shared libraries
주소 공간의 중앙 근처에는 C 표준 라이브러리 및 수학 라이브러리와 같은 공유 라이브러리에 대한 코드와 데이터가 들어있는 영역이 있다.

- Stack
사용자의 가상 주소 공간 맨 위에는 컴파일러가 함수 호출을 구현하는 데 사용하는 사용자 스택이 있다. 힙과 마찬가지로 사용자 스택은 프로그램 실행 중에 동적으로 확장 및 축소된다.

- Kernel virtual memory
커널은 항상 메모리에 상주하는 운영 체제의 일부입니다. 주소 공간의 최상위 영역은 커널 용으로 예약되어 있고 응용 프로그램은 이 영역의 내용을 읽거나 쓰거나 커널 코드에 정의 된 함수를 직접 호출 할 수 없다.

가상 메모리가 작동하려면 프로세서와 프로세서가 생성 한 모든 주소의 하드웨어 변환을 포함하여 하드웨어와 운영 체제 소프트웨어 간에 정교한 상호 작용이 필요하다. 기본 아이디어는 프로세스의 가상 메모리 내용을 디스크에 저장 한 다음 주 메모리를 디스크의 캐시로 사용하는 것이다.

1.7.4 Files

파일은 일련의 바이트이고 디스크, 키보드, 디스플레이 및 네트워크를 포함한 모든 I/O 장치는 파일로 모델링된다. 시스템의 모든 입출력은 Unix I/O를 사용하여 파일을 읽고 쓰는 방식으로 수행됩니다. 디스크 파일의 내용을 조작하는 응용 프로그램 프로그래머는 특정 디스크 기술을 인식하지 못하고 동일한 프로그램이 다른 디스크 기술을 사용하는 다른 시스템에서 실행된다.