과제 1: CSAPP 1장 요약 및 자신의 강의 목표 기술

시스템 프로그래밍(SW) 7분반
소프트웨어학과 32170578 김산

1. CSAPP 1장 1.1절 ~ 1.4절 주요내용 정리

1.1 정보는 비트와 문맥으로 구성되어있다.

우리가 C언어를 처음 배우게 될 때 작성하는 “hello world”를 출력하는 프로그램을 예시로 한다. 프로그래머가 에디터를 통해 ‘hello.c’라는 프로그램을 작성했을 때 해당 프로그램의 각 문자는 해당 문자를 대표하는 ASCII 코드를 통해 바이트 단위 정수들로 저장된다. 예를 들어 #include의 문자 ‘#’은 35의 정수 바이트로 저장된다. 이처럼 디스크 파일, 메모리의 데이터 그리고 네트워크를 통해 전달되는 데이터까지 시스템의 모든 데이터는 일련의 비트들로 표현할 수 있다.

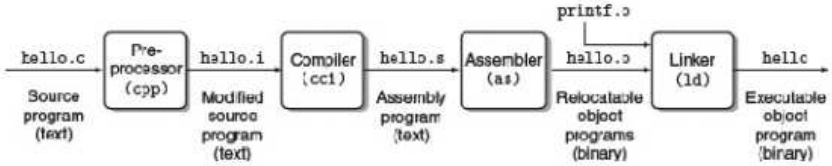
>C 프로그래밍 언어의 기원
C언어는 1969년부터 1973년까지 벨 연구소의 Dennis Ritchie에 의해 개발되었다. C언어는 1989년 미국의 연구소 ANSI가 C언어의 표준을 비준하였다. 이 표준은 C언어에 대한 정의와 C 표준 라이브러리에 관한 내용을 담고 있다.

>C언어의 성공의 원인
1. C언어의 Unix 계열의 운영체제와 관련성
- C언어는 Unix 시스템의 시스템 언어를 목적으로 설계되었다. Unix 시스템의 대부분의 커널과 툴, 라이브러리는 C언어로 작성되었다. Unix 시스템이 대학에서 인기를 얻기 시작하면서 많은 사람이 C언어에 익숙해졌고 C언어의 높은 이식성으로 많은 C언어 사용자가 생겨났다.
2. C언어의 단순함
- C언어는 여러 사람에 의해 개발되기보다 Dennis Ritchie라는 한 사람에 의해 개발되었기에 깨끗하고 일관되게 설계되었다. 이러한 단순함은 C언어를 상대적으로 배우기 쉽고 이식성이 높은 언어로 만들었다.
3. C언어의 실용성
- C언어는 Unix시스템 언어를 목적으로 설계되었기에 아무런 제약없이 원하는 프로그램을 개발할수 있었다.

C언어는 시스템 수준의 프로그래밍뿐만 아니라 응용프로그램 수준의 프로그램도 개발할 수 있지만, C++, Java와 같이 객체나 예외의 활용이 힘들다.

1.2 프로그램은 시스템 프로그램에 의해 다른 형태로 변환된다

'hello.c' 프로그램은 고수준의 언어로 작성된 프로그램으로 사람만이 읽을 수 있고 이해할 수 있다. 하지만 이 프로그램을 시스템에서 실행하기 위해서는 반드시 저수준의 기계어인 명령어로 변환되어야 한다. 유닉스 시스템에서는 다음과 같은 변환 과정을 거친다.



1. 전처리 : C 프로그램은 '#' 문자로 시작하는 '#include<stdio.h>'와 같은 명령어를 처리한다. 해당 명령은 'stdio.h'라는 헤더 파일을 포함시켜 '.i' 확장자를 가진 다른 프로그램으로 변환시킨다.
2. 컴파일 : 컴파일러(ccl)는 'hello.i' 파일을 어셈블리 언어로 변환하여 'hello.s'파일로 만든다.
3. 어셈블러 : 어셈블러(as)는 'hello.s' 파일을 기계어로 변환시켜 확장자 '.o'의 재배치 가능한 객체 프로그램으로 만든다.
4. 링커 : 링커(ld)는 'printf'함수와 같이 미리 컴파일된 프로그램을 현재 프로그램과 결합하고 최종적으로 실행 가능한 프로그램(binary)을 만든다.

1.3 컴파일 시스템의 작동방식을 이해하는 것은 많은 도움이 된다

1. 프로그램의 성능을 최적화할수 있다.
- 프로그래머로서 효율적인 코드작성을 위해 컴파일러의 내부의 작동방식을 알 필요는 없지만, C 프로그램 작성을 위한 좋은 코드를 선택하기 위해서는 기계어에 대한 기본적인 이해와 어떻게 컴파일러가 C언어를 기계어로 번역하는지 알 필요가 있다.

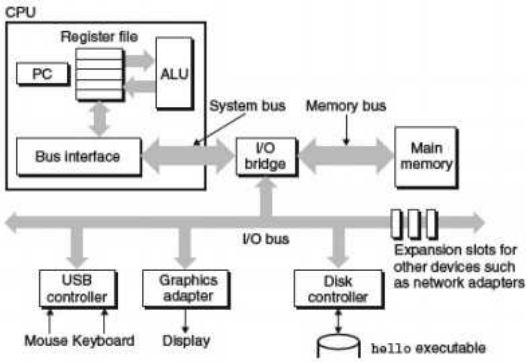
2. link-time 에러를 이해할 수 있다.
- 우리가 큰 소프트웨어 시스템을 구축할 때 발생하는 오류중 몇몇은 링크의 작동과 관련이 있다. 이러한 오류들을 링크의 작동방식을 이해하여 해결할 수 있다.

3. 보안 취약점을 피할 수 있다.
- 버퍼 오버플로우 취약점은 네트워크나 인터넷 서버 보안 취약점에서 많은 부분을 차지한다. 이러한 문제는 프로그래머가 신뢰할 수 없는 출처에서 받아들인 데이터의 양과 형식을 신중하게 제한하지 못하여 발생한다. 따라서 우리는 이러한 데이터와 제어 정보가 프로그램의 스택에 저장되는 방식을 이해해야 한다.

1.4 프로세서는 메모리에 저장된 명령을 읽고 해석한다.

우리가 쉘에 ‘./hello’라는 명령어를 입력했을 때 쉘은 이것이 쉘에 내장된 명령어가 아니라 실행 가능한 파일의 이름이라고 추정한다. 이 경우 쉘은 hello 프로그램을 로드하고 실행시킨 후 프로그램이 종료될 때까지 기다린다. 프로그램이 종료되면 다음 명령어가 입력될 때까지 기다린다.

1.4.1 시스템의 하드웨어 구성



1. Buses
- 버스는 시스템 전체에 걸쳐 바이트 정보를 앞뒤로 전달하는 전기 도관의 모음이다. 버스는 일반적으로 단어라 불리는 4byte 또는 8byte의 크기의 바이트 덩어리를 전송하도록 설계된다.

2. I/O Devices
- 키보드, 마우스, 모니터와 같은 입출력 장치는 시스템을 외부세계와 연결한다. 각장치는 컨트롤러 또는 어댑터에 의해 I/O 버스에 연결된다. 여기서 컨트롤러는 장치나 시스템 메인보드의 칩셋을 말하고 어댑터는 메인보드 슬롯에 연결되는 카드를 말한다. 둘 다 입출력 장치와 버스 사이에 정보를 주고받는 역할을 한다.

3. Main Memory(메인 메모리)
- 메인 메모리는 프로세서가 프로그램을 실행하는 동안 프로그램과 프로그램의 데이터를 저장하는 임시 저장장치이다. 물리적으로는 DRAM 칩들로, 논리적으로 선형 배열로 구성되며 각각은 0으로 시작하는 고유한 주소(Array index)를 가진다.
- Linux x86-64시스템에서 C 프로그램의 변수의 크기는 short는 2byte, int와 float은 4byte, long과 double은 8byte로 유형에 따라 다르다.

4. Processor(프로세서)
- 프로세서는 메인 메모리에 저장된 명령을 실행하는 엔진으로 그 코어에는 프로그램 카운터라 불리는 단어 크기의 저장장치가 있다. 프로세서는 시스템의 전원이 차단될 때까지 프로그램 카운터가 가리키는 메모리의 명령을 읽고, 수행한 다음 지속해서 프로그램 카운터를 업데이트하여 다음 명령을 실행한다. 이러한 연산은 메인 메모리, 레지스터 파일, ALU를

중심으로 반복된다. 레지스터 파일은 단어 크기의 레지스터들로 구성되며 각각 고유한 이름을 가진 저장장치이다. ALU는 새로운 데이터와 주소값을 계산하는 산술/논리 유닛이다.

- 프로세서는 다음과 같은 명령을 수행할 수 있다.
>Load : 메인 메모리에서 바이트나 단어를 레지스터로 복사하여 덮어씀
>Store : 레지스터의 바이트 또는 단어를 주메모리 위치로 복사하여 덮어씀
>Operate : 두 레지스터의 내용을 ALU에 복사, 산술연산을 수행 후 결과를 레지스터에 저장
>Jump: 명령 자체에서 단어를 추출하여 단어를 프로그램 카운터에 복사하여 덮어씀

1.4.2 hello 프로그램의 실행

hello 프로그램의 실행과정은 크게 다음과 같은 순서로 실행된다.

1. 쉘에 ‘./hello’를 키보드를 통해 입력했을 때 쉘 프로그램은 각 문자를 레지스터로 읽은 다음 메인 메모리에 저장한다.
2. 엔터키를 입력하면 쉘은 명령의 입력이 종료됨을 알고 ‘hello’ 파일의 코드와 데이터를 복사하는 명령을 수행하여 실행 가능한 ‘hello’ 파일을 load 한다.
3. 데이터는 디스크에서 메모리로 DMA 기술을 통해 곧바로 이동한다. hello 파일의 코드와 데이터가 메모리에 load되면 프로세서는 기계어 명령을 hello 프로그램의 main routine 내에서 수행한다. 이러한 명령은 “hello word\n” 문자열을 메모리에서 레지스터로 복사하고 디스플레이 장치를 통해 화면에 표시한다.

2. CSAPP 1장 1.7절 주요내용 정리

1.7 The Operating System Manages the Hardware(운영체제의 하드웨어 관리)
운영체제는 추상화를 통해 하드웨어를 응용프로그램의 리소스 낭비로부터 보호하고 low-level 하드웨어 장치를 조작하기 위한 간단하고 통일된 메커니즘을 응용프로그램에 제공한다.

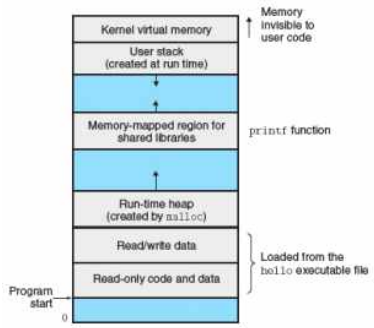
1.7.1 Processes(프로세스)
현대 시스템에서 프로세스는 실행 중인 프로그램에 대한 추상화다. 여러 프로세스가 같은 시스템에서 동시에 실행될 수 있지만, 각 프로세스는 하드웨어를 독점적으로 사용하는 것처럼 작동한다. 대부분 시스템에서 CPU의 숫자보다 프로세스의 숫자가 더 많다.

운영체제는 context switching이라는 메커니즘으로 하나의 CPU가 여러 프로세스를 동시에 실행하는 것처럼 보이게 할 수 있다.

운영체제는 프로세스가 실행되는 데 필요한 모든 상태정보를 추적한다. 이러한 상태정보를 context는 프로그램 카운터의 현재값, 레지스터 파일, 메인 메모리의 내용 등의 정보를 포함한다. 운영체제는 현재 프로세스에서 새로운 프로세스를 실행하기 위해 현재 프로세스의 context를 저장하고 새로운 프로세스의 context를 복원한 후 새로운 프로세스에 대한 제어를 시작한다. 새로운 프로세스는 그 프로세스가 중단되었던 부분에서 시작된다.

이러한 프로세스에서 프로세스의 전환은 운영체제의 커널에 의해 관리된다. 커널은 항상 메모리에 상주하는 운영체제 코드의 일부로 응용프로그램이 파일을 읽거나 쓰는 것과 같은 운영체제에 의한 조치가 필요할 때, 특별한 시스템 호출을 수행한다. 커널은 프로세스를 관리하는 데 사용하는 코드 및 데이터 구조의 모음으로 프로세스가 아니다.

1.7.2 Threads(스레드)



프로세스는 스레드라 불리는 여러 실행 단위로 구성될 수 있다. 스레드는 프로세스의 context에서 실행되며 동일한 코드와 전역 데이터를 공유할 수 있다. 스레드는 네트워크 서버의 동시성 요구와 스레드 간의 데이터 공유가 프로세스 간 데이터 공유보다 쉬우므로 점점 더 프로그래밍 모델에서 중요해지고 있다. 다중프로세서를 이용할 수 있을 때 멀티스레딩은 프로그램을 더 빠르게 실행할 수 있는 방법중 하나다.

1.7.3 Virtual Memory(가상메모리)

가상메모리는 메인메모리에 대한 추상화다. 각 프로세스에는 가상 주소 공간이라고 알려진 메모리에 대한 뷰가 존재한다. 주소 공간의 최상위 영역은 프로세스에 공통적인 운영체제의 코드와 데이터를 위해 예약되며 주소 공간의 하단 영역은 사용자 프로세스에 의해 정의된 코드와 데이터를 보관한다.

- 프로그램 코드 및 데이터 : 코드는 모든 프로세스의 동일한 고정 주소부터 C 전역 변수에 해당하는 데이터의 위치까지다. 코드와 데이터 영역은 실행 가능한 개체 파일의 내용에서 직접 초기화 된다.
- Heap : 프로세스가 실행되면 크기가 고정된 코드나 데이터 영역과 달리 'malloc' 'free' 와 같은 표준 C라이브러리 함수에 의해 런타임에서 동적으로 확장되고 수축된다.
- Stack : 스택은 프로그램 실행 중에 함수의 호출과 반환에 의해 동적으로 확장되고 수축된다.
- Kernel virtual memory : 응용프로그램과 달리 주소 공간의 상단 영역은 커널에 예약되어 있다. 이 영역의 내용을 읽거나 쓰기 위해 커널 코드에 정의된 함수를 직접 호출하려면 커널을 호출해야 한다.

3. 이번 학기 시스템 프로그램을 수강하면서 배우고 싶은 목표

이번 학기 시스템 프로그래밍 과목을 수강하면서, 소프트웨어를 전공하고 프로그래밍 언어를 공부하면서 정작 그 기반이 되는 컴퓨터 시스템에 관한 공부는 소홀하지 않았나 생각이 들었습니다. 제가 최근 텐서플로우를 공부하면서 가장 많이 접하고 활용했던 것이 윈도우의 리눅스 하위시스템(WSL)이었습니다. WSL이 Hyper-V를 기반으로 만들어졌다는 이야기만 알고 있었지 Hyper-V가 어떤 것을 의미하는지는 몰랐습니다. 이번 주 시스템 프로그래밍 강의에서 Virtual machine은 'hypervisor'라는 논리적인 플랫폼 위에서 실행된다는 내용을 듣고 WSL의 Hyper-V를 떠올렸습니다. 찾아보니 VMware는 ESX/ESX i Server, 마이크로소프트는 Hyper-V, Oracle의 VM Server등 다양한 hypervisor 종류가 있었고 생각보다 시스템 프로그래밍이 가깝게 느껴졌습니다. 동시에 제가 활용하고 있는 리눅스 시스템에 대한 흥미가 생겼습니다. 저는 시스템 프로그래밍 과목이 int[i][j] 와 int[j][i]의 수행시간 차이 예제처럼 프로그래밍에서 활용할 수 있는 시스템에 대한 이해는 물론, 개발자로서 자주 활용하게 될 시스템과 플랫폼에 대한 이해력을 높이는 데 많은 도움을 줄 것 같습니다.

시스템 프로그램과목을 수강하면서 시스템이라는 기본에 충실한 개발자가 되는 것이 저의 목표입니다.