시스템 프로그래밍 system programming 요약보고서

학번: 201520908
이름: 유성민
학과: 소프트웨어

시스템 프로그래밍은 컴퓨터 시스템 소프트웨어를 프로그래밍하는 활동이다. 응용 프로그래밍과 다른 점은 다른 소프트웨어에 서비스를 제공하는 소프트웨어 및 소프트웨어 플랫폼을 생산하는 것이라고 볼 수 있다. 수업 시간에 소개된 대표적인 시스템 프로그램으로는 운영체제, 어셈블러, 링커, 로더, 마이크로프로세서, 컴파일러, I/O Device Driver가 있다. 이 프로그램들은 응용적으로 데이터를 가공해 주는 프로그램인지 살펴보면 아니다. 운영체제가 사용자가 원하는 아웃풋을 내기 위한 프로그램인가? 아니다. 운영체제는 하드웨어와 응용프로그램 간의 소통을 매개하기 위한 프로그램이다. 컴파일러도 마찬가지로 응용프로그램을 위한 high-level의 코드를 어셈블리어로 번역하는 역할을 하고, 어셈블러는 어셈블리어로 된 코드를 기계어(이진 스트림)으로 번역하는 역할을 한다. 이 과정에서 링커가 라이브러리와 합쳐주는 역할을 하고, 로더는 메모리에 프로그램을 적재하는 역할을 한다. 이렇듯 시스템 프로그램들은 사용자가 원하는 아웃풋을 내는 프로그램이 아닌 그런 프로그램을 만들기 위해 만들어진 소프트웨어라고 할 수 있겠다. 그렇다면 이 목적을 달성하기 위해 시스템 프로그램들은 무엇이 중요할까? 상당한 수준의 하드웨어 인식이 필요하고, 하드웨어와 직접적으로 맞닿는 프로그램이므로 하드웨어를 효율적으로 사용하는 것 또한 빠른 컴퓨팅을 위해 성능 또한 중요할 것이다. 역사를 살펴보면 본래 시스템 프로그래머들은 항상 어셈블리 언어로 프로그래밍을 하였다. 1960년대 후반에 들어 high-level 언어로 하드웨어를 제어하는 실험을 해서 pl/s, bliss, bcpl, burroughs 대형 시스템을 확장 algol과 같은 언어가 사용되었고, 1970년대에 들어 요즘에도 흔히 사용하는 c / c++ 언어의 하위집합이 일부 사용되기 시작했다. 본인이 배운 바로는 요즘 시스템 프로그램들은 high-level 언어와 assembly 언어 script 언어로 이루어진다. 다만 객체지향적 언어(java나 python같은) 성능 상의 문제로 사용이 많이 제한된다. 이제 특징을 보면 첫 번째로는 프로그래머는 프로그램이 실행되는 시스템의 하드웨어 및 기타 속성에 대해 가정 할 수 있으며, 예를 들어 특정 하드웨어와 함께 사용할 때 효율적인 것으로 알려진 알고리즘을 사용하여 이러한 속성을 악용하는 경우가 많다. 두 번째로는 일반적으로 low-level 언어가 다음과 같이 사용된다. 프로그램은 자원이 제한된 환경에서 작동 할 수있다. 런타임 라이브러리가 있거나 전혀 없음 프로그램은 메모리 엑세스 및 제어 흐름을 직접적으로 제어할 수있음 프로그래머는 프로그램의 일부를 어셈블리 언어로 직접 작성할 수 있다. 세 번째 특징으로는 종종 시스템 프로그램을 디버거에서 실행할 수 없는데, 이는 lock과 많이 관련되어 있다. 소프트웨어 공학에서 lock을 구현 할 때 디버깅이 영향을 미치는 경우가 많기 때문이다. 이를 테면, semapore lock을 구현하고 이를 테스

트하기 위해 printf라는 함수를 사용했다고 하자. 운영체제의 스케쥴링에 따라 이게 될 수도 있고 안 될 수도있고 뒤죽박죽이 되어버린다. 이런 특징들이 존재한다. 지금까지 정의 특징 역사에 대해 알아보았는데, 응용프로그램 프로그래밍과는 확연한 차이가 난다. 디버깅이라든지 목적이라든지 시스템 프로그래밍을 하기 위해서는 하드웨어에 관한 배경지식도 필요한 것 같다. 응용프로그램에서는 생각하지 않고 프로그래밍 해도 되는 문제 이를 테면 메모리 부족 여부, 주소 공간 등등 이 시스템 프로그램 프로그래밍에서는 필요하게 되고, 디버깅이 안되니 모니터링이나 로깅 같은 대체 방법으로 프로그램을 검사하는 것이 필요하다. 또한 프로그래밍 하기위해 어셈블리어를 공부해야 하고, 어느 부분은 이 언어로 구현하는 것이 최적인가? 또 저 부분은 이 언어로 구현하는 것이 최적인가? 이런 프로그래밍적 특징도 잘 숙지해놔야 좋은 시스템 프로그램을 작성할 수 있을 것 같다.