오픈소스와 역사

HANYANG UNIVERSITY ERICA
Education Research Industry Cluster @ Ansan

목차

오픈소스

오픈소스란?

그림 1 open source

오픈소스 소프트웨어, OSS 라고도 한다. 소프트웨어의 설계도에 해당하는 소스코드를 인터넷 등을 통하여 무상으로 공개하여 누구나 그 소프트웨어를 개량하고, 이것을 재배포할 수 있도록 하는 것 또는 그런 소프트웨어를 말한다. 또한 소프트웨어 혹은 하드웨어의 제작자의 권리를 지키면서 원시 코드를 누구나 열람할 수 있도록 한 소프트웨어 혹은 오픈 소스 라이선스에 준하는 모든 통칭을 일컫는다.

소스 코드를 알면 그 소프트웨어와 비슷한 것을 만들거나 그 소프트웨어에서 이용하고 있는 기술을 간단히 적용할 수 있다. 이 때문에 기업 등에서는 자사에서 개발한 소프트웨어의 소스코드를 극비로 하고 있으며, 이를 다른 사람에게 제공할 때는 사용료(라이선스료)를 받는 경우가 많다.

이에 대해 오픈소스의 개념은, 소스코드를 공개하여 유용한 기술을 공유함으로써 전세계의 누구나가 자유롭게 소프트웨어의 개발, 개량에 참여할 수 있게 하는 것이 우수한 소프트웨어를 만드는 데 도움이 된다는 생각에 바탕을 두고 있다.

오픈소스 소프트웨어

그림 2 open source sofrware

사물인터넷이 화제가 되기 전의 ICT 전 분야에서 디바이스의 소프트웨어 플랫폼은 이미 전통적 OSS(Open Source Software) 패러다임이 강하게 자리 잡아 왔다. OSS 는 그 저작권자가 누구든지 어떤 목적으로든 학습하고 수정하고 배포할 수 있는 권한을 제공하는 라이선스로 이루어진 소스코드의 소프트웨어를 말한다. 가장 강한 대표적인 OSS 가운데 하나는 리눅스(Linux)이다. 현재에는 OSHW 와 OSS 를 구분하는 것은 의미가 적다.

사물인터넷 디바이스 소프트웨어 플랫폼의 대표적 효시라 할 수 있는 TinyOs 도 가장 잘 부합이 되는 레퍼런스 하드웨어들과 관련 설계 소스를 제시하였다. 그만큼 H/W 와 S/W 와의 최적화가 강조되어 왔기 때문이다. 사물인터넷 소프트웨어 플랫폼은 OS 뿐 아니라 OS 를 기반으로 상위에 존재하는 미들웨어 같은 공통기능까지 정의한 것이 존재한다.

오픈소스 장단점

오픈소스 장점

첫째로 이용에 비용이 들지 않거나 적다.

둘째, 일반적으로 Web 상에서 무료로 다운로드 및 소스 코드 수정/재배포가 가능하다.

셋째, 오픈소스의 개발과정을 볼 때, 전세계에 있는 수많은 우수한 개발자들이 직접 개발과 Debugging 과정에 참여하기 때문에 in-house 에서 폐쇄적으로 개발되는 독점 프로그램보다 비교적 안정적으로 동작하는 편이다.

넷째, 오픈소스가 확산된 가장 큰 이유는 다양하고 강력한 Networking 기능 지원이라 할 수 있다. Open Source Networking Solution 은 대부분 상용 프로그램과 호환이 되기 때문에 상품화에도 아주 잘 활용될 수 있을 것이다.

다섯째로, 보통 최신 기술 정보 및 문제점과 해결책을 공유하는 형태로 자유롭게 운영되기 때문에 독점 프로그램보다 기술 발전 속도가 빠르다.

마지막으로 주로 Open Formats 또는 Protocois 를 사용하기 때문에 서로 다른 소프트웨어간 상호 연동성이 보장되는 장점이 있다.

오픈소스 단점

먼저, 비숙련 사용자들은 사용이 어렵다.

둘째로, 상용 프로그램보다 체계적인 문서를 갖고 있지 못한다.

셋째, 오픈소스에 기업이 보유한 특허 및 소스코드를 포함할 경우 오픈소스 라이선스는 일반적으로 Royalty-free 를 요구하고 있다. 따라서 Royalty-free 를 원치 않을 때에는 해당 오픈소스를 사용할 수가 없으며, 또한 사용 후 Royalty 를 주장하게 되면 해당 오픈소스에 대한 사용 권한이 박탈되는 경우가 일반적이다. 따라서 오픈소스를 활용하여 특허를 구현하거나 기존 소스코드를 포함하고자 할 경우, 반드시 Royalty 에 대한 입장을 명확히 하여야 할 것이다.

넷째로, 오픈소스는 영리를 목적으로 하는 회사에서 개발되는 것이 아니라, 자발적 참여를 바탕으로 Web 상에서 자유롭게 개발되는 것이 특징이다. 그러므로 독점 프로그램에서 볼 수 있는 Roadmap 을 기대하기 힘든 면이 있다.

마지막으로 고객지원이 불리하다는 점이 있다.

오픈소스 소프트웨어 종류

역시 첫 번째로 리눅스(Linux)를 들 수 있다. 리눅스는 Unix 와 유사하고, OS 간의 호환성을 위한 표준인 POSIX 와 대부분 호환되는 OSS 움직임의 결과물의 효시라 할 수 있는 OS 이다. 리눅스는 짧지 않는 역사 속에서 임베디드 OS, 모바일 OS 를 거치면서 수많은 파생 OS 를 만들어 왔다.

하지만 사물인터넷 디바이스가 인터넷에 직접 연결되고 외부에서의 최소한의 관리가 필요한 디바이스에서는 몇 가지 태스크는 발생하며 이를 위해서는 최소한 스케쥴링 기능을 기본으로 하는 OS 는 요구되고 있으며 실제로 OSHW 플랫폼에서도 대부분 OS 로 리눅스와 리눅스의 변종(Yocto linux 등)이 적용되고 있다. 또한 리눅스 파운데이션은 디바이스 S/W 뿐만 아닌 IoT 표준화 컨소시엄인 AllSeen 얼라이언스나 OIC 에도 관여하고 있다.

TinyOS

센서네트워크를 위해 태어난 TinyOS 는 US 버클리대학교에서 BSD 라이센스로 개발한 OS 이다. 센서와 네트워크 기능을 동시에 갖춘 마이크로컨트롤러 기반의 단일 보드 기기에 유용하다.

몇 KB 의 RAM 과 몇십 KB 의 코드 공간을 갖춘 마이크로컨트롤러와 같은 극히 자원이 제한된 기기에 적합하게 설계되었다. 아두이노 이전인 1999 년부터 시작되어 2012 년까지 발전하였으므로 기술적 완성도는 성숙하였다.

그림 3 TinyOS 2.0 하드웨어 추상화 아키텍쳐

하드웨어의 확장성을 고려하지 않았으며 저전력 무선 통신 기능에만 중점을 두었고 전용 개발언어(NetC)를 적용하였으므로 사물인터넷 디바이스로의 생태계 적용 발전은 더딘 상황이다.

콘티키(Contiki)

콘티키는 TinyOS 와 같이 센서 네트워크 목적을 가진 가벼운 사물인터넷 디바이스를 위한 OS 이다. Contiki OS 는 TCP/IP 를 임베디드 환경에 적용하기 위해 가볍게 만든 uIP 를 중심으로 개발이 시작된 인터넷 지향의 태생 배경을 가지고 있다. 2002 년에 Contiki 의 코드명으로 이어져 Atmel, Cisco 등의 추가 참여를 통해 개발이 이어졌다. TinyOS 는 Non-IP 프로토콜에서 시작하여 IP 프로토콜을 수용하는 방향으로 발전하였지만 Contiki 는 원래부터 IP 통신을 지향했으므로 IPv4 와 IPv6 를 다 지원하며 초소형 구현체로는 세계 최초로 IPv6 Ready 인증을 받았다.

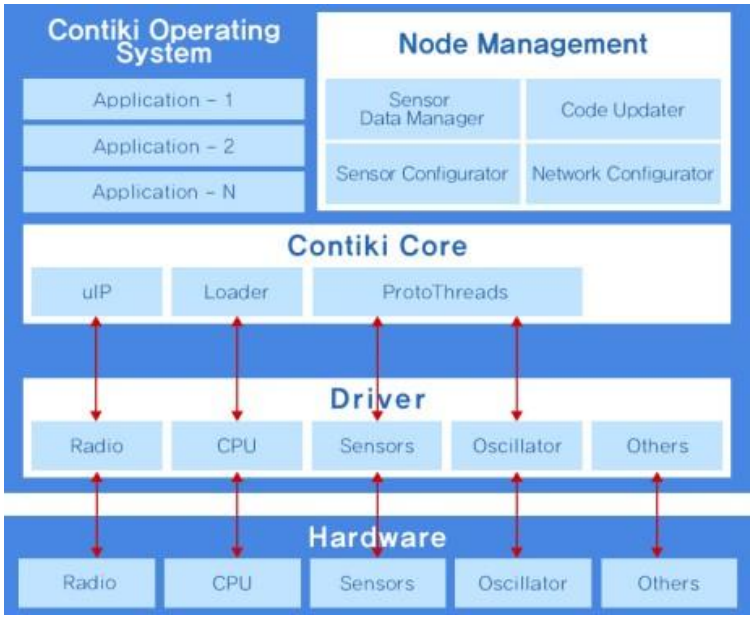

그림 4 콘티키(Contiki) 아키텍쳐

Mbed OS

mbed OS 는 사물인터넷 이전의 임베디드 기기 환경에서 MCU 의 주도권을 가져온 ARM 은 사물인터넷 디바이스에서도 그 주도권을 유지하려는 전략으로 mbed OS 를 개발하여 2014 년 10 월에 무료로 공개했다. mbed OS 는 Cortex-M 시리즈 위에서만 동작한다. 생태계를 고려하여 파트너들까지 라인업하여 사물인터넷 디바이스에 필요한 핵심 내용을 대부분 포함하는 아키텍처를 가진다. Cortex-M

시리즈의 시장 지배력을 계승하는 mbed OS 는 현재 가장 사물인터넷 디바이스 시장에서 영향력 있는 OS 의 하나라고 할 수 있다.

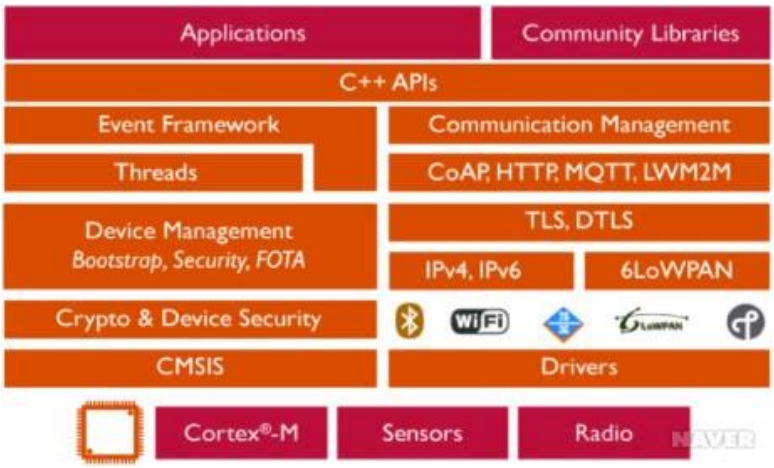

그림 5 Mbed OS 아키텍쳐

nanoQplus

nanoQplus 는 우리나라 ETRI 에서 2007 년에 최초로 공개 버전을 배포한 뒤로 C 언어로 개발되어 온 소형 OS 이다. TinyOS, 콘티키와는 달리 우선순위 기반 선점형 스케줄러를 갖는 멀티스레드 기반으로 리눅스 프로그래밍 방식을 사용하여 쉽게 접근할 수 있다. 다른 사물인터넷 디바이스 OS 와 마찬가지로 IPv6 통신을 지원하므로 CoAP, RPL, 6LoWPAN 이 포함되어 있으며 콘티키와 마찬가지로 2014 년 3 월에 IPv6 Ready 인증을 받았다. Atmega 128L 의 8bit MCU, MSP 430 등의 16bit MCU 그리고 ARM7 의 32bit MCU 도지원한다.

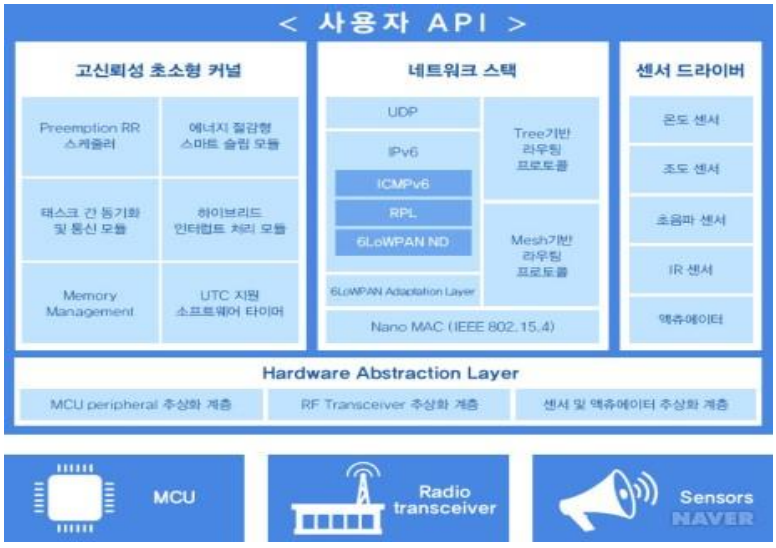

그림 6 nanoQplus 의 아키텍쳐

오픈소스 하드웨어

그림 7 open source hardware

OSHW 는 하드웨어의 설계 결과물(회로도, 자재명세서(BOM, Bill of Meterials), PCB 도면 등)뿐 아니라 그것을 목적에 맞게 구동하는 소프트웨어(Firmware, OS, 응용프로그램 등)의 소스 결과물까지도 무료로 공개함으로써 다수에 의해 공유되고 논의되고 발전/확대되면서 나름대로 생태계를 만들어 나가는 오픈 소스 문화와 그 결과물을 얘기한다.

아두이노(Arduino)

아두이노(Arduino)가 OSHW 의 철학을 처음으로 담았다. 이탈리아 작은 도시의 예술과 IT 의 융합을 가르치던 대학원에서 공학도가 아닌 예술학도도 쉽게 접근할 수 있는 저렴한 전자교육용으로 탄생한 보드 제품으로 나오자마자 꾸준히 전 세계적인 인기를 끌고 있다.

그림 8 Arduino

임베디드 시스템 개발 경험이 전혀 없는 사람들도 쉽게 접근하도록 펌웨어를 쉽게 만들어 탑재하도록 지원하는 통합개발환경(IDE), 다양한 코드소스, 회로도 등을 무상으로 제공하는 점, 가격이 30 달러 정도로 저렴한 점, 각종 센서/엑츄에이터 및 통신모듈 등을 탑재한 다양한 호환보드('쉴드'라 한다)들에 의해 쉽게 확장할 수 있는 점 등이 아두이노 열풍을 일게 만든 이유라 볼 수 있다.

Atmel 사의 AVR 이라는 마이크로 컨트롤러를 적용한 첫 제품 이후로 100 만 단위의 누적 판매대수를 오래전에 초과하며 '쉴드'라는 확장 보드, 다양한 변종 제품이 풍부해 뚜렷한 생태계를 형성하고 있다.

라즈베리 파이 (Raspberry Pi)

라스베리 파이(Raspberry Pi)는 영국의 라즈베리파이 재단이 학교에서의 기초 컴퓨터 과학 교육용 프로젝트의 목적으로 개발된 초소형/초저가 PC 이다. 2006 년에 개념이 형성되고 재단이 만들어져 2012 년 처음 제품이 나온 이후 2013 년 1 월 초에 백만 대가 판매되었다.

그림 9 Raspberry Pi

아두이노와 달리 키보드, 마우스, 모니터만 연결하면 PC 가 될 수 있다. 즉 일반 데스크톱과 유사하다는 것이 강조되는 제품이다. 리눅스 OS 를 기반으로 하고 세부적인 설정을 제공함으로써 초보 프로그래머에 맞춤형 환경을 제공한다.

갈릴레오(Galileo)

갈릴레오(Galileo)는 모바일과 임베디드 분야에서 암(ARM)쪽에 주도권을 빼앗긴 인텔이 사물인터넷 분야에서 주도권을 재탈환하기 위해 OSHW 의 트렌드에 편승하는 첫 제품이다. 이를 시작으로 '에디슨', '큐리' 등 다양한 시도를 인텔은 계속한다.

그림 10 Galileo

갈릴레오 보드는 인텔의 Quark SoC x1000(32bit 펜티엄 클래스 SoC) 프로세스를 기반으로 하였다. 아두이노 우노 R3 용 쉴드와 호환 가능한 H/W 와 S/W 아키텍처 기반으로 인텔이 출시한 보드이다. 즉 아두이노의 생태계를 수용한 것이다.

비글본 블랙(BeagleBone Black)

비글본 블랙(BeagleBone Black)은 라즈베리파이와 비슷한 배경에서 생겨난 오픈소스 하드웨어 플랫폼이다. 개발능력에 상관없이 쉽게 접근할 수 있는 환경을 제공하며 값싼 작은 컴퓨터에 원하는 주변기기를 붙여 초보 개발자를 포함한 누구나가 자신이 원하는 임베디드 시스템을 구성할 수 있도록 설계되어 교육용으로도 적합하다.

그림 11 BeagleBone Black

또 다른 메이저 칩 벤더인 TI (Texas Instrument)사가 사물인터넷 디바이스 시장을 겨냥한 OSHW 전략의 결과물이라고 할 수 있다.

오픈소스의 역사

자유 소프트웨어 운동의 태동

그림 12 Richard Matthew Stallman

자유 소프트웨어에 대한 리처드 스톨먼의 철학은 1971 년 MIT 대학의 인공지능 연구소에서 소프트웨어를 공유하는 공동체의 일원이 되면서였다. 이 공동체는 수년 전부터 존재했고 MIT 에만 있는 특별한 존재도 아니었다. 인공지능 연구소엔 ITS 라 불리는 운영체제를 사용했었는데, ITS 는 당시에 사용되던 DEC 사의 PDP-10 기종에 탑재할 목적으로 인공지능 연구소의 해커 연구원들에 의해 만들어진 운영체제이다.

당시엔 아직 자유 소프트웨어란 정의가 없었다. 하지만 연구원들이 만들었던 소프트웨어의 모습은 지금의 '자유 소프트웨어'가 의미하는 그것이었다. 다른 대학이나 기업에서 연구원들이 만든 프로그램을 요청하거나 다른 시스템에 이식하기 위해서 도움을 요청할 때면 언제든지 프로그램을 빌려주었다.

이 방식이 일반적인 것이기에 흥미로운 프로그램 사용하는 사람이 있으면 얼마든지 소스 코드를 보여 달라고 요청할 수 있었다. 공동체 시절은 소스 코드를

자유롭게 공유했기에 프로그램을 수정하거나 해당 프로그램을 기반으로 발전시킬 가능성이 풍부했던 때였다.

하지만 1980 년대 초,
DEC 사가 P-10 시리즈의 생산을 중단하며 공동체에 위기가 찾아온다.
DEC 사가 PDP-10 시리즈의 생산을 중지하면서 ITS 환경에서 만들어진 모든 프로그램들이 호환성의 문제로 더이상 사용될 수 없게 되었다. 인공지능 연구소를 주축으로 모인 공동체는 붕괴하였고, 해커들은 뿔뿔이 흩어졌다. 당시에 사용되었던 현대적 모델의 컴퓨터는 전용 운영체제를 탑재했는데, 이들은 자유 소프트웨어가 아니었다. 바이너리 형태의 프로그램조차 관련 자료를 유출하지 않겠다는 비공개 계약 조건에 동의해야만 했다. 이는 상호협력적인 공동체의 형성이 불가능해진 것을 의미했고 독점 소프트웨어 제도는 다른 사람과의 공유나 교환을 금지하였다.

이러한 사람을 고립시키고 서로를 돕는 것을 금지하는 제도에 대해 스톨만은 부정적이었다. 반면 소프트웨어 판매자들은 소프트웨어의 유통 형태에 대해 기존의 방식이 유일한 방법이라는 생각을 사람들에게 주입하기 위해 오랫동안 노력해왔다. 구체적인 예를 살펴보면 일반 사용자가 소프트웨어 회사의 권리를 '침해'한다든가 또는 소프트웨어 회사들이 그들의 정당한 '권리를 행사'한다는 표현 안에는 마치 소프트웨어 제조 회사가 소프트웨어를 유형, 무형으로 완전히 소유할 권리를 갖고 있어서 소프트웨어에 대한 사용자의 권리마저 통제해 버릴 수 있다는 전제가 포함되어있다.

사용자를 최우선으로 고려한 상식과 윤리를 통해서 이 문제를 판단한 스톨만은 사람들 간의 도움이 사회를 지탱하는 근간이라 판단했고, 프로그램을 자유롭게 수정하거나 공유해야한다는 결론에 도달했다. 공동체가 사라짐에 따라 예전처럼 지탱할 수 없게 되었고, 스톨만은 선택의 기로에 서게 되었다. 첫째는 비공개 협약에 서명하고 동료를 돕지 않을 것을 약속함으로써 독점 소프트웨어의 세계에 합류하는 것이었다. 만약 그랬다면 본인이 만든 소프트웨어 또한 비공개 협약에 의해 배포되었을 것이므로 다른 사람이 동료를 돕지 못하도록 하는 또 하나의 요인이 되었을 것이다. 그는 MIT 인공지능 연구소와 그에게 제어 프로그램의 소스 코드를 줄 것을 거부했던 사람으로 인하여 프린터의 사용에 곤란을 겪은 경험이 있었기에 이러한 비공개 협약이 가져올 결과를 알고 있었기 때문에 소스 코드를 공유해야 겠다는 생각을 하였다.

이러한 이유로 인해 스톨만은 공동체를 다시 부활시키기 위해 그가 직접 만들 프로그램이 없을까라는 생각을 하기에 이르렀다. 제일 먼저 필요한 프로그램은 바로 운영체제였다. 운영체제는 컴퓨터를 사용하기 위해 가장 핵심적인 소프트웨어이다. 따라서 자유롭게 사용가능한 운영체제가 있다면 상호 협력적인 해커들의 공동체를 다시 재건할 수 있을 것이며, 누구에게나 공동체에 합류하기를 권유할 수 있었다. 또한 운영체제의 구입에 수반되는 '재배포를 금지한다'는 등의 계약 조건에 구애받을 필요가 없어지므로 누구나 자유롭게 컴퓨터를 사용할 수 있게 될 것이었다.
운영체제의 개발자로서 그는 이러한 작업에 필요한 적합한 기술을 갖고 있었다. 그는 새롭게 개발할 운영체제를 유닉스와 호환되도록 만들려했다. 그렇게 해 기존의 유닉스 사용자들이 손쉽게 적응하고 GNU 로 유입될 수 있으리라 생각했기 때문이다. GNU 라는 이름은 GNU is Not Unix, 즉 "GNU 는 유닉스가 아니다."라는 뜻이 되도록 의도적으로 조합해서 만든 것이다. 따라서 GNU 는 유닉스와 호환되도록 만든 운영체제이지만 유닉스와는 다른 체제임을 알리기 위해 만든 이름이다.

그림 13 GNU 로고

완전한 운영체제를 만든다는 것은 매우 커다란 일에 해당하기 때문에 스톨만은 기존에 존재하던 자유 소프트웨어를 개작하거나 변형해서 시스템을 완성시켜 나갔다. 이러한 방식으로 인해 GNU 시스템은 단순히 GNU 소프트웨어를 모아 놓은 것 이상이 되었다. GNU 시스템은 GNU 프로젝트를 통해 구현하려는 유닉스와 호환되는 소프트웨어 시스템 전체를 의미하고, GNU 프로젝트는 이러한 시스템을 구현하기 위해 진행되는 프로젝트 자체를 의미했다.

1984 년 1 월, MIT 연구원직을 사직하고 스톨만은 GNU 소프트웨어를 만들기 시작했다. 그가 MIT 를 떠난 이유는 연구원이 개발한 소프트웨어에 대한 저작권을

소속 회사나 학교로 귀속시킬 수 있는 법률에 따라서 MIT 당국이 그의 소프트웨어를 자유 소프트웨어로 만들지 못하게 할 가능성이 있기 때문이다.

스톨만은 1984 년 9 월부터 GNU Emacs 를 만들기 시작해 1985 년 초에는 제법 완성된 상태가 되었다. 사람들의 GNU Emacs 에 대한 수요가 생길 때 즈음에서 그는 이 에디터를 어떠한 방식으로 배포할 것인가에 의문이 생겼다. 그 당시에는 Emacs 에 관심을 가진 사람 중에서 인터넷을 통해 구할 수 있는 사람이 거의 없었기에 어떠한 방식으로 Emacs 를 구할지 알려주는 것도 중요한 문제였다. 그는 반송용 우표가 붙어있는 봉투를 테이프와 함께 보내면 PDP-10 용 Emacs 를 복사해 주겠다고 했다. 즉, Emacs 를 배포함에 있어서 금전적인 비용이 청구되지 않는 방법을 말이다. 그는 150 달러의 비용을 지불하면 누구나 Emacs 가 든 테이프를 우성해주는 방식을 생각해내어, 이러한 판매 방식을 통해 자유 소프트웨어를 이용한 사업을 시작하고, 리눅스를 기반으로한 GNU 시스템 전체를 담은 배포판을 판매하는 현재의 리눅스 배포판 업체들의 시초가 되었다. GNU 의 목적은 GNU 를 널리 알리는 것이 아닌, 사용자에게 자유를 주는 것이기에 배포 기준에는 GNU 가 독점 소프트웨어로 변질되는 것을 막을 조항이 필요했고, 이를 위해 카피레프트라는 방식을 사용했다. 카피레프트의 핵심은 프로그램을 실행하고 복제할 권리와 함께 개작된 프로그램에 대한 배포상의 별도의 조건을 설정하지 않는다면, 개작과 배포에 대한 권리 또한 모든 사람에게 허용하는 것이다. 즉, 프로그램의 실행, 복제, 개작, 배포의 자유를 허용함으로써 자유 소프트웨어의 핵심인 자유를 보장하고 프로그램을 입수한 사람은 권리를 갖게 되는 것이다.

Emacs 의 사용에 대한 관심의 증가에 따라 사람들이 GNU 프로젝트에 참여하기 시작했고, 기금을 모을 적절한 시기가 되었다. 그래서 1985 년, 자유 소프트웨어의 개발을 위해 자유 소프트웨어 재단을 설립하게 된다. 자유 소프트웨어 재단, FSF 는 Emacs 의 테이프 배포 사업을 맡고, 점진적으로 다른 소프트웨어도 테이프에 담아 판매했다. FSF 는 기부를 받지만 대부분의 운영 자금이 자유 소프트웨어 판매와 관련된 부수적 서비스로 충당된다.

자유 소프트웨어가 가진 철학은 만연한 특정한 형태의 상업적 관행을 반대하는 것이지, 상업 행위 자체를 중단시키려는 것은 아니다. GNU 의 주된 목적은 자유 소프트웨어가 되는 것이다. GNU 가 유닉스에 비해 기술적 장점을 갖지 못하더라도, 사용자의 협력을 통해 사회적 이점을 가질 것이고, 사용자의 자유를 존중한다는 윤리적인 이점도 있다. 하지만 우수한 기술의 표준을 GNU 에 수용하는 것은 당연한 일이었다. GNU 프로젝트의 명성이 높아짐에 따라 유닉스가 탑재된 시스템을

기증하는 사람이 늘어났다. GNU 를 구성하는 프로그램 개발의 가장 쉬운 방법은 유닉스 환경에서 먼저 개발한 뒤 이를 GNU 에 맞게 수정하는 것이기 때문에 이러한 시스템은 매우 유용했다. 하지만 기본적인 GNU 의 철학은 독점 소프트웨어를 사용하지 않는 것이라 독점 소프트웨어인 유닉스를 통해 자유 소프트웨어를 만든다는 것은 모순이라는 주장이 있었다.

따라서 어느 시점부터 독점 소프트웨어를 사용하지 않기로 결정하고 자유 운영체제를 찾기 시작했다. 오늘날에는 유닉스를 대체할 수 있는 자유 운영체제를 갖고 있기 때문에 유닉스를 시스템에 전혀 사용하지 않고 있다. GNU 프로젝트가 진행됨에 따라 발견하거나 개발한 자유 운영체제의 구송 요소들이 많아짐에 따라 남은 부분에 대한 목록을 만드는 것이 필요한 시점에 도달했다. 이 목록을 이용해 남아 있는 부분을 만드는 데 필요한 개발자를 모집했는데, 이 목록은 GNU "테스크 리스트"라는 이름으로 불리었다. 여기엔 아직은 개발되지 않은 유닉스 시스템의 구성요소만이 아니라 하나의 완벽한 운영체제가 되기 위해 필요한 다양한 소프트웨어와 문서 프로젝트를 이 목록에 추가시켰다.

에릭 레이먼드는 소프트웨어에 있어서 "모든 성과는 개발자 자신의 가려운 곳을 긁는 것으로부터 시작된다."라고 주장했다. 이는 타당한 말이지만 GNU 소프트웨어의 핵심 부분은 거의 완전한 자유 운영체제를 만들기 위한 목적을 개발된 것이기 때문에 가려운 곳을 긁기 위해 그때그때 만든 것이 아니라 비전과 계획을 확고하게 가지고 만들어진 것이다. 예를 들어 GNU C 라이브러리를 개발한 이유는 유닉스 형태의 운영 환경에서 C 라이브러리가 필요해서고, BASH 를 개발한 이유는 유닉스 환경이 셸을 요구해서이다. 어떤 프로그램은 자유에 대한 위협에 대처하기 위해 만들어졌다.

GNU 프로젝트를 시작할 당시엔 GNU 시스템이 모두 완성된 뒤, 전부를 공개하려 했지만 아래와 같은 이유로 인해 그렇게 되지 않았다. GNU 는 알다시피 유닉스에서 요소를 구현한 뒤 이를 GNU 로 옮겼기 때문에 GNU 시스템이 완성되기 전부터 유닉스에서 사용되었다. 이들 중 일부는 유명해져 사용자가 프로그램을 확장하거나 다양한 종류의 유닉스로 이식시켰다. 또한 유닉스 이외에 운영체제로 이식이 되기도 했다. 이러한 과정을 통해 프로그램은 좀 더 강력한 기능을 갖게 되어 기부와 공헌자를 GNU 프로젝트로 끌어들였다. 하지만 이는 GNU 개발자로 하여금 GNU 시스템을 완성시키는 것보다 오히려 기존의 구성 요소에 새로운 기능을 덧붙이거나 다른 시스템으로 이식하는 데 시간을 투자하게 만들어버렸다. 결국 작동 가능한 최소한의 시스템을 완성하는 데 몇 년이 지연되는 결과를 가져왔다.

1990 년 무렵 GNU 시스템이 거의 완성되었지만, 운영체제를 구성하는 핵심 부분인 커널이 누락되었었다. 따라서 프로젝트의 일원들은 MACH 를 기반으로 하는 한 서버 프로세스를 연결해 커널을 구현하기로 결정했다. 커널은 운영체제를 구성하는 핵심 요소를 의미하는 말이다.
MACH 는 자율적 프로세스의 집합이며, 유닉스 커널이 가진 다양한 기능을 그대로 수행할 수 있는 것이었다.

Linus Torvalds & Linux

Linus Torvalds 는 누구인가?

그림 14 Linus Torvalds

리누스 베네딕트 토르발스(Linus Benedict Torvalds , 1969 년 12 월 28 일 ~)는 핀란드 헬싱키에서 태어난 스웨덴계 핀란드인으로서 소프트웨어 개발자이자 리눅스 커널과 깃을 최초로 개발한 사람으로 잘 알려져 있다. 후에 그는 리눅스 커널 개발 최고 설계자가 되었고, 현재 프로젝트 코디네이터로 활동하고 있다. 그는 커널의 플랫폼 독립적인 부분과 인텔 IA-32 아키텍처로 구체화되는 핵심 커널의

컴포넌트들을 관리한다. 저명한 오픈소스 소프트웨어 개발리더들에게 부여되는 명예 타이틀직인 BDFL(Benevolent Dictator for Life) 중의 한 사람이기도 하다.

Linus Torvalds 가 만든 Linux

리눅스는 1989 년 핀란드 헬싱키대학에 재학중이던 리누스 토르발스(Linus Torvalds)가 유닉스를 기반으로 개발한 공개용 오퍼레이팅시스템(OS) (오픈소스 운영체제) 이다.

리누스 토발즈가 당초 공개한 리눅스 커널(kernel, 핵심 구성 요소)의 첫 번째 버전(0.01)은 약 1 만행 정도의 소스 코드로 구성되었다. 하지만 이를 보고 흥미를 느낀 개발자들이 하나 둘이 프로젝트에 참가해 힘을 보태기 시작했고 리눅스는 성능과 기능이 급격히 향상되기 시작했다. 불과 1 년 후에 출시된 버전 0.96 의 소스 코드는 4 만행 정도로 덩치가 커졌다 (참고로 2012 년을 즈음해 리눅스 커널의 소스 코드는 1,500 만 행을 돌파했다).

리눅스는 특히 소프트웨어의 소스 코드를 무료로 공개하는 오픈 소스(open source) 확산 운동을 주도하던 리처드 스톨먼(Richard Matthew Stallman)이 설립한 자유소프트웨어재단(FSF, Free Software Foundation)의 주목을 받았다. 그리고 리눅스는 그들이 추진하는 GNU(GNU's Not Unix) 프로젝트의 핵심 중 하나로 부상하게 된다. 이로 인해 전세계 개발자들이 자유롭게 리눅스의 개발에 참여할 수 있는 토대가 마련되었고, 1994 년 3 월, 마침내 첫 번째 완성 버전인 리눅스 커널 1.0.0 이 공개되었다. 이후에도 리눅스 커널은 여전히 무료 공개 원칙을 유지하고 있으며, 이를 변형시키고 재배포 하는 것도 자유롭다는 점 역시 변함이 없다.

파일구성이나 시스템기능의 일부는 유닉스를 기반으로 하면서, 핵심 커널 부분은 유닉스와 다르게 작성되어 있다. 인터넷 프로토콜인 TCP/IP 를 강력하게 지원하는 등 네트워킹에 특히 강점을 지니고 있으며, 유닉스와 거의 유사한 환경을 제공하면서 무료라는 장점 때문에 프로그램 개발자 및 학교 등을 중심으로 급속히 사용이 확대되고 있다. 리눅스는 각종 주변기기에 따라 혹은 사용하는 시스템의 특성에 맞게 소스를 변경할 수 있으므로 다양한 변종이 출현하고 있다.

그림 15 턱스(Tux)라는 이름의 펭귄은 1996 년 래리 유윙이 창조한 리눅스의 마스코트이다.

Linus Torvalds 의 행보

리눅스는 IT 세계 전반에 강력한 힘을 발휘하게 되었지만, 정작 리눅스의 아버지라고 할 수 있는 리누스 토발즈가 이를 통해 막대한 경제적 이득을 거두지는 않았다. 리눅스가 유명세를 얻게 된 이후에도 리누스 토발즈는 이를 상업적 목적으로 이용하지 않았기 때문이다. 1996 년에 대학을 졸업한 그는 이후 미국의 반도체 회사인 트랜스메타(Transmeta)에 취업, 1997 년부터 2003 년까지 개발자로 일하며 생계를 유지했다.

미국에서 Linux 라는 상표권을 취득해 보유하고 있는 것도 리누스 토발즈 자신이지만, 정작 그 자신은 이를 상업적으로 이용한 적이 없다. 단지 다른 사람이 리눅스의 상표권을 멋대로 취득해 부당한 이득을 얻는 것을 막기 위한 조치라고 밝힌 바 있다.

이후 그는 리눅스 재단의 산하에서 리눅스의 개발 및 표준 규정을 주도하는 단체인 OSDL(Open Source Development Labs, 오픈 소스 개발 연구소)로 자리를 옮겼다. 그리고 특이하게도 여전히 그는 경영자나 고문이 아닌 소프트웨어 개발자(Software engineer)로 분류된다.

리눅스 이후의 오픈소스

Qt

그림 16 Qt 로고

1996 년과 1998 년 사이에는 이른바 Qt 라고 불리는 또 다른 GUI 툴킷 라이브러리가 등장했다.

Qt 는 데스크톱 환경의 자유 소프트웨어인 KDE 에 사용되는 핵심 요소다. 그러나 자유 운영체제인 GNU/Linux 시스템에서는 Qt 의 라이선스와 관련해서 KDE 를 사용할 수 없었다. GNU/Linux 를 판매하는 일부 상용 배포판 업체에서는 KDE 를 함께 패키징해서 활용성을 중대시킨 시스템을 만들기도 했지만 결국 이것은 자유를 감소시키는 것이었다. KDE 그룹은 좀 더 많은 프로그래머가 Qt 를 사용하도록 적극 권장했고, 새롭게 리눅스를 사용하게 된 수많은 사람은 이러한 문제를 미처 생각하지도 못한채 KDE 를 사용하게 되었다. 상황이 무척 심각해진 것이다. 자유 소프트웨어 공동체는 GNOME 과 Harmony 라는 두가지 방법으로 이 문제에 대응했다.

GNU 는 먼저 GNOME 이라고 불리는 데스크톱 환경의 개발을 시작했다. 이 프로젝트는 1997 년 부터 시작되었고 레드햇 소프트웨어가 이를 후원했다. GNOME 은 데스크톱 환경과 유사한 기능을 제공하지만 오직 자유 소프트웨어만으로

만들어진다. 또한 C++ 이외에 다양한 종류의 언어를 지원함으로써 기술적인 장점도 함께 제공한다. 그러나 GNOME 의 주된 목적은 자유라고 할 수 있다. GNOME 환경은 자유 소프트웨어가 아닌 어떠한 프로그램의 기능도 자유 소프트웨어 안에서 충족시키기 위한 것이다. 또 하나의 대안인 Harmony 는 KDE 소프트웨어들을 Qt 없이도 사용할 수 있게 하기 위해서 개발된 라이브러리다.

1998 년 11 월에 Qt 개발자들은 Qt 를 자유 소프트웨어로 변경할 것이라고 발표했다. 그러나 이는 아직까지 확실한 것이 아니며, 부분적으로 Qt 가 자유 소프트웨어가 아니었을 때 가졌던 문제에 단호하게 대처함으로써 나타난 결과라고 할 수 있다.

자유 소프트웨어

자유 소프트웨어로 변경하기로 한 Qt 개발자들이 당면한 가장 큰 위협은 바로 소프트웨어에 대한 특허에서 발생했다. 알고리즘과 새로운 기술에 부여되는 특허는 20 년까지 보장되므로 이러한 영역은 소프트웨어가 아직 접근할 수 없는 것이다. LZW(Lempel-Ziv-Welch, 비손실 데이터 압축 알고리즘, 빠른 이식을 위해 고안되었지만 데이터의 제한된 분석만 수행하기 때문에 그리 최적으로 동작하지는 않는다.) 압축 알고리즘에 대한 기술 특허가 1983 년에 신청되었기 때문에 여전히 정상적인 GIF 압축을 생성할 수 있는 자유 소프트웨어를 공개하지 못하고 있다. 특허권에 대처할 수 있는 몇 방법은 특정한 특허가 특허로 인정될 수 없다는 반대 증거를 찾거나 특허가 취득된 기술을 대체할 수 있는 방법을 찾는 것이다. 이러한 방법은 때로 실패할 때가 있다. 자유 소프트웨어가 가진 자유의 가치를 중요하게 생각한다면, 언제라도 자유 소프트웨어를 버리는 일은 없을 것이다. 하지만 자유소프트웨어가 가진 기술적 우위를 기대하고 이를 이용하려 했던 사람은 특허에 의해 기술적 장점이 사라진 후, 자유소프트웨어가 실패했다고 단원할 것이다. 따라서 자유 소프트웨어가 가진 안정성과 성능에 대해서 말하는 것이 사람들에겐 유용한 동안에는 자유와 자유를 지켜나가기 위한 원칙도 함께 이야기해야 한다.

‘자유’ → ‘오픈 소스’

‘오픈소스’라는 용어가 처음 소프트웨어에 적용된 것은 1998 년 2 월 3 일 미국 캘리포니아 마운틴 뷰의 VA 리눅스 시스템즈 사무실에서 열린 전략회의 에서였다. (오픈소스라는 용어 자체는 나노기술에 중점을 둔 씽크탱크 조직인 포어인사이트 인스티튜드(Foresight Institute)의 사장 크리스틴 패터슨(Christine Patterson)이 만들었다.) 그 후로 새로운 사용자들에게 자유에 대해 이야기하는 것이 더욱 어려워졌다. 왜냐하면 공동체의 일부에서 ‘자유 소프트웨어’라는 말 대신 ‘오픈 소스 소프트웨어’라는 표현을 사용하기 시작했기 때문이다. 이 용어를 선호하는 사람들의 의도는 ‘자유’를 의미하는 영단어인 ‘Free’가 가진 무료와 자유라는 의미상의 혼란을 방지하기 위한 것으로 이는 올바른 변화라 할 수 있다. 그러나 또 다른 이들은 ‘자유’라는 단어가 함축한 자유 소프트웨어 운동과 GNU 프로젝트의 정신을 일부러 퇴색시키기 위해 ‘오픈소스’라는 표현을 사용하며 자유나 공동체보다는 이윤에 더 높은 가치를 부여하는 기업 경영인과 사용자에게 다가서기 위해서 이 말을 사용하기도 했다. 즉, ‘오픈 소스’라는 용어는 높은 품질과 강력한 성능을 가진 소프트웨어를 만들 수 있다는 가능성에 초점을 맞춘 것이기에 공동체 그리고 원칙과 같은 개념을 의도적으로 전달하지 않으려는 표현으로도 볼 수 있다.

OSI

그림 17 OSI 로고

1998 년 2 월 오픈소스 소프트웨어를 인증하는 단체인 OSI(Open Source Initiative, www.opensource.net)가 에릭 레이몬드(Eric Raymond)와 브루스 페렌스(Bruce Perns)등에 의해 결성되면서 오픈소스 소프트웨어 운동은 궤도에 오르게 된다.

이 단체는 넷스케이프 커뮤니케이션스 코퍼레이션의 대표제품인 넷스케이프 커뮤니케이터에 대한 소스코드를 공개한것에 대해 고무되어 설립되었다.

넷스케이프는 넷스케이프 내비게이터의 개방형 개발을 스스로 주도하지 않고 흘러가는 대로 내버려뒀다. 이것이 모질라 프로젝트로 이어졌다. 이념적인 접근보다는 실용적인 목적이 있다면 많은 기업이 자유 소프트웨어를 도입한다는 것이 분명해졌다.

참고로 OSI 로고는 콜린 비브록이 만들었는데, O 는 개방성을, 열쇠구멍은 소스코드의 잠금 해제를 암시한다.

1999 년 데비안 프리 소프트웨어 지침 (Debian Free Software Guideline)의 작성자인 브루스 페렌스는 이 지침을 소프트웨어 라이선스가 어떻게 오픈소스로 인식될 수 있는지에 대한 객관적인 정의로 개작했다. 바로 OSD(Open Source Definition)이다. 이 기준을 만족하는 만족해야만 오픈소스 소프트웨어로 인정받게 된다. OSD 로 10 가지 '라이선스 조건'이 있다.

1. 자유로운 재배포

오픈소스 라이선스(license)는 몇 개의 다른 출처로부터 모아진 프로그램들로 구성된 집합 저작물 형태의 배포판의 일부로 소프트웨어를 판매하거나 무상 배포하는 것을 제한해서는 안된다. 또한 그러한 판매에 대해 사용료나 그 밖의 다른 비용을 요구해서도 안된다.

사용 허가에 자유로운 재배포를 규정하도록 강제함으로써 우리는 단기간의 적은 판매 수익을 얻 기 위해 많은 장기적인 이익을 포기하는 유혹을 없앨 수 있다. 만약 이렇게 하지 않는다면, 협력 자들에게 많은 변심의 압력이 있을 것이다.

2. 소스 코드

오픈 소스 프로그램에는 소스 코드(source code)가 포함되어야 하며, 컴파일된 형태 뿐 아니라 소 스 코드의 배포도 허용되어야 한다. 만약 소스 코드가 함께 제공되지 않는 제품이 있다면 소스 코드를 복제하는데 필요한 합당한 비용만으로 소스 코드를 구할 수 있는 널리 알려진 방법이 제공되어야만 한다.

이러한 경우에 있어 가장 권장할 만한 방법은 별도의 비용없이 인터넷을 통해 소스 코드를 다운 받을 수 있도록 하는 것이다. 소스 코드는 프로그래머가 이를 개작하기에 용이한 형태여야 하며, 고의로 복잡하고 혼란스럽게 만들어진 형태와 선행 처리기나 번역기에 의해 생성된 중간 형태의 코드는 인정되지 않는다.

소스 코드를 불분명하지 않은 형태로 제공하도록 규정하는 이유은 프로그램을 발전시키기 위해서 소스 코드에 대한 개작이 선행되어야 하기 때문입니다. 우리의 목적은 발전을 용이하게 만들기 위한 것이므로 개작이 용이하게 이루어 질 수 있는 방법을 요구한다.

3. 파생 저작물

오픈 소스 라이센스에는 프로그램의 개작과 2 차적 프로그램의 창작이 허용되어야 하며, 이러한 파생 저작물들이 원프로그램에 적용된 것과 동일한 사용 허가의 규정에 따라 배포되는 것을 허용해야만 한다.

단순히 소스 코드를 열람할 수 있는 것만으로는 독립된 등위 검토(peer review)와 빠른 발전 경쟁 에서의 생존을 지원할 수 없습니다. 프로그램을 빠르게 발전시키기 위해서는 사람들에게 개작된 프로그램을 실험하고 재배포할 수 있도록 허용할 필요가 있다.

4. 저작자의 소스 코드 원형 유지

오픈 소스 라이센스는 바이너리를 생성할 시점에서 프로그램을 수정할 목적으로, 소스 코드를 수 반한 ``패치 파일''의 배포를 허용한 경우에 한해서 패치로 인해 변경된 소스 코드의 배포를 제한할 수 있다. 그러나 이 경우에도 변경된 소스 코드를 통해 만들어진 소프트웨어의 배포는 명시적 으로 허용해야만 한다. 오픈 소스 라이센스는 파생 저작물에 최초의 소프트웨어와 다른 판 번호 (version)와 이름이 사용되도록 규정할 수 있다.

소프트웨어에 많은 향상이 이루어지도록 장려하는 것은 좋은 일입니다. 그러나 사용자에게는 그 들이 사용하고 있는 소프트웨어를 누가 책임지고 있는 지를 알 권리가 있다. 또한 저작자와 관리 자에게도 반대 입장에서 사용자들이 그들에게 어떤 지원을 요구하고 있는 지를 알 권리와 그들의 명성을 보호할 권리가 있다.

따라서 오픈 소스 라이센스는 소스 코드가 쉽게 이용될 수 있도록 보증해야만 하지만 변형되지 않은 최초의 소스 코드가 패치 파일과 함께 배포되도록 규정할 수도 있다. 이러한 방법을 통해 `` 비공식'' 수정들을 이용할 수 있으면서 소스 코드의 원형이 쉽게 구별될 수 있다.

5. 개인 및 단체에 대한 차별 금지

오픈 소스 라이센스는 특정 개인이나 단체를 차별해서는 안된다. 오픈 소스의 공정으로부터 최대의 이익을 끌어내기 위해 최대한 다양한 개인과 단체에게 오픈 소스에 기여할 수 있는 동등한 자 격이 부여되어야 한다. 따라서 우리는 어떠한 오프 소스 사용 허가도 특정인을 오픈 소스의 공정 으로부터 제외시키는 것을 금지한다.

미국을 포함한 몇몇 국가에서는 특정한 종류의 소프트웨어에 대한 수출이 금지되고 있다. OSD 를 준수하는 사용 허가는 피양도자에게 이러한 종류의 제한에 대해 경고하고 해당 법률을 준수해야 한다는 사실을 상기시킬 수 있다. 그러나 사용 허가 자체에 그러한 제한이 통합되어서는 안된다.

6. 사용 분야에 대한 차별 금지

오픈 소스 라이센스는 프로그램이 특정 분야에서 사용되는 것을 금지하는 제한을 설정해서는 안 된다. 예를 들면, 기업이나 유전학 연구에 프로그램을 사용할 수 없다는 등과 같은 제한을 설정 해서는 안된다.

이 조항의 주된 목적은 오픈 소스가 상업적으로 이용되지 못하게 방해하는 규정이 사용 허가에 포함되는 것을 금지하기 위한 것이다. 우리는 상업 이용자들도 오픈 소스 공동체에 동참하기를 원하며 이들이 공동체로부터 소외감을 느끼지 않기를 바랍니다.

7. 사용 허가의 배포

프로그램에 대한 권리는 배포에 따른 각 단계에서 배포자에 의한 별도의 사용 허가 없이도 프로 그램을 재배포받은 모든 사람에게 동일하게 인정어야만 한다. 이 조항은

비공개 계약을 요구하는 것과 같은 간접적인 수단을 통해 소프트웨어가 제한되는 것을 금지하기 위한 것이다.

8. 특정 제품에만 유효한 사용 허가의 금지

프로그램에 대한 권리는 프로그램이 특정한 소프트웨어 배포판의 일부가 될 때에 한해서만 유효 해서는 안된다. 만약 특정 배포판에 포함되어 있던 프로그램을 별도로 분리한 경우라 하더라도 프로그램에 적용된 사용 허가에 따라 프로그램이 사용되거나 배포된다면 프로그램을 재배포받은 모든 사람에게 최초의 소프트웨어 배포판을 통해 프로그램을 배포받은 사람과 동일한 권리가 보장되어야만 한다. 이 조항은 또다른 형태의 사용 허가상의 제한을 방지하기 위한 것이다.

9. 다른 소프트웨어를 제한하는 사용 허가의 금지

오픈 소스 라이센스는 오픈 소스 라이센스가 적용된 소프트웨어와 함께 배포되는 다른 소프트웨어에 대한 제한을 포함해서는 안된다. 예를 들면, 사용 허가 안에 동일한 매체를 통해 배포되는 다른 소프트웨어들이 모두 오픈 소스 소프트웨어여야 한다는 제한을 두어서는 안된다.

오픈 소스 소프트웨어의 배포자들은 그들의 소프트웨어에 대한 스스로의 선택 권리를 갖고 있다. 물론, GPL 은 이러한 규정을 충족시키고 있다. GPL 라이브러리와 결합되는 소프트웨어는 하나의 단일 저작물을 형성할 때에 한해서 GPL 이 계승되는 것이지 단순히 함께 배포된다는 것만으로 GPL 소프트웨어가 되어야 하는 것은 아니다.

10. 라이센스 기술 중립성

오픈 소스 사용 라이센스는 특정 소프트웨어의 기술이나 인터페이스에 의존해서는 안된다. 이는 특정 소프트웨어만을 위한 라이센스가 오픈 소스로 인정 받을 수 없음을 의미한다.

이어서, OSI 의 공동설립자인 브루스 페렌스(Bruce Perns)는 아래와 같이 설명했다.

"기존에 있던 자유 소프트웨어 개념을 기업에 홍보하고 또 라이선스를 인증하기 위해 오픈소스가 이 운동에 적합한 이름이다."

2000 년 이후의 오픈소스

2000 년대 중반, 오픈소스에 대한 인기가 높아지면서 라이선스의 무분별한 확산이 문제가 됐다. 2004~2006 년 기간에 OSI 는 이문제를 처리하기 위해 공공 의견 수렴 과정을 진행했다.

2004 년 리처드 스톨만이 '자바의 함정'이란 글을 통해 그 사악함을 지적했지만, 자바 플랫폼은 2006 년 오픈소스화 됐다.

2000 년대 말 너무나 많은 오픈소스 비영리 단체가 생겨나자 OSI 는 산하 단체 모델로 전환하고 개인 회원제도도 도입해 새로운 이니셔티브를 가능하게 했다.

새로운 경영진과 클라우드 솔루션 구축에서 오픈소스를 피할 수 없게 된 마이크로소프트는 오픈 소스에 대한 공개적인 적대감 표현을 중단했다. 물론 마이크로소프트는 임베디드 리눅스에 대한 특허 소송이 진행되고 있다.

계속해서, 많은 기업들이 오픈소스를 사용하고 있으며 상용 소프트웨어 벤더에서도 오픈소스를 가져다가 사용하기 시작했다. 특히, 웹 기반 서비스(예: SNS) 업체에서 오픈소스 활용이 폭발적으로 증가했다.

구글은 오픈소스 기술에 가장 적극적으로 관여하는 세계적인 기업 중 하나로, 오픈소스 개발자 플랫폼인 Git Hub 에는 900 명이 넘는 구글 소속 공여자와 1000 개가 넘는 구글 관련 저장소가 있다. 그 예로는 GWT(Google Web Toolkit) 등이 있다.

페이스북 또한 개발자 문화가 발달한 대표 기업으로 오픈소스 또한 장려되어 페이스북이 관리하고 있는 오픈소스 프로젝트 상당히 많다. 대표적인 예로는 React.JS 가 있다.

이 외에도 Twitter, Netflix, Naver, kakao, Samsung 등 오픈소스에 참여하는 기업들과 오픈소스 문화 기반의 회사들 및 스타트업들이 많이 생겨나고 있다.

오늘날 새로운 이니셔티브는 바로 오픈소스로 간다. 오픈소스는 20 년만에 위협에서 기회로, 암적인 존재에서 기념비적인 존재로, 파생기술에서 혁신으로 바뀌었다.

사이트 주소

https://bean3.github.io/open-source_project1/index6.html

Github 주소

https://github.com/bean3/open-source_project1

대표 이메일

khn9115@naver.com

대표 번호

+82 1037031114

공동 작업자

LEE HYE-JIN	2015061230
KANG HA-NEUL	2018044257
PARK SUNG-SOO	2018044584
BAE JIN-WOO	2018044684
YIM CHONG-HUON	2018045005