**오픈소스 SW기초 중간 보고서**

**9관조**

2016006526 조은한

<보고서 목차>

1. 자료 취합 진행사항
2. 웹페이지 진행사항
3. 소책자 디자인 진행사항
4. 자료 취합 진행사항

목차

1. ‘오픈소스’란?
2. 오픈소스의 정의
3. 오픈소스의 조건
4. 오픈소스의 특징
5. 오픈소스의 장단점
6. 오픈소스의 역사
7. 상용 소프트웨어의 등장 & unix 유료화
8. 자유 소프트웨어 연합(FSF)의 등장과 GNU
9. 리누스토발즈의 리눅스
10. 현재의 오픈소스
11. 오픈소스의 전망
12. 최근 오픈소스 시장의 추세
13. 오픈소스의 활용분야
14. ‘**오픈소스’란?**

* 오픈소스(Open Source), 또는 공개 소프트웨어는 저작권자가 소스코드를 공개하여 누구나 제작자의 권리를 지키면 특별한 제한 없이 소프트웨어 혹은 하드웨어를 자유롭게 사용, 복제, 배포, 수정할 수 있는 소프트웨어이다. 어원에 대해서는 History of the OSI 자료에 따르면 1998년 2월 3일에 넷스케이프 브라우저의 원시코드에 대해 어떠한 형태로 공개할까 하는 전략회의에서 붙여진 새로운 용어라고 설명되어 있다.
* OSI(Open Source Initiative)에서 말하는 오픈소스 조건

1. 자유로운 재배포

* 소프트웨어의 판매나 기부에 대해서 그 어떤 누구도 배제되어서는 안된다. 로열티 또는 판매 비용이 없다.

1. 소스코드

* 프로그램에는 소스 코드가 포함되어 있어야 하며 소스 코드 형태와 컴파일이 완료된 형태로 배포될 수 있어야 한다. 특정 형태의 제품이 소스 코드로 배포되지 않으면 소스코드를 구매해야 하며 인터넷을 통해 무료로 다운로드 해야 한다. 또한 알아보기 힘든 소스 코드는 사용될 수 없다.

1. 파생작업(Derived work)

* 변경과 파생작업이 가능해야 하며 원래의 소프트 라이선스와 같은 조건 하에 배포되어야 한다.

1. 소스 코드의 무결성

* 구현 시 프로그램을 변경할 목적으로 소스 코드와 함께 “패치 파일”의 배포를 허용하는 경우에만, 라이선스는 소스 코드가 변경된 형태로 배포하는 것을 제한한다.

1. 개인 또는 그룹의 평등

* 라이선스는 어떤 개인이나 그룹에 차별을 두어서는 안된다.

1. 분야에 대한 평등

* 특정 분야에서 프로그램을 사용하는 것에 대해 제한을 두어서는 안된다.

1. 라이선스 배포

* 프로그램에 대한 권한은 프로그램이 재배포 된 모든 곳에 적용되어야 한다. 추가 라이선스를 발행할 필요가 없다.

1. 제품 스펙에 따른 라이선스

* 프로그램에 대한 권한은 특정 소프트웨어 배포판의 프로그램의 일부에 해당하지 않는다. 프로그램이 그 배포판에서 추출되었고, 그 프로그램의 라이선스 조건 하에 사용 및 배포된다면 재배포된 프로그램을 사용하는 모든 당사자들은 원래의 소프트웨어 배포판에서 허용된 것과 같은 권한을 갖게 된다.

1. 다른 소프트웨어를 제한하지 않는 라이선스

* 라이선스는 라이선스를 받은 소프트웨어와 함께 배포된 다른 소프트웨어에 제약 사항을 두어서는 안된다.

1. 중립적인 기술의 라이선스

* 라이선스는 기술이나 인터페이스 스타일은 한정해서는 안된다.
* 오픈소스의 특징

1. 특정 라이선스에 따라 소프트웨어의 소스 코드가 공개되어 있다.
2. 일반적으로 오픈소스 소프트웨어에 대한 코드를 자유롭게 열람하거나 사용, 복사, 수정, 개작 및 재배포의 권한을 부여 받았다.
3. 누구나 무료로 이용할 수 있으며 공개된 코드를 기반으로 프로그램을 마음대로 변형 할 수 있다.
4. 하드웨어에도 적용된다.
5. 사용 시, 원제작자의 권리를 지키는 것도 중요하다.
6. 라이선스는 배포되어 각각의 제품 스펙에 따른 라이선스가 존재한다.
7. 라이선스는 다른 소프트웨어를 제한하지 않으며 기술 중립적 이어야 한다.

* 오픈소스의 장점

1. 낮은 진입비용 : 오픈소스는 무료로 다운로드 및 소스코드의 수정/재배포가 가능하므로 일반적으로 초기 개발비용이 새로 개발하는 것에 비해 1/2 정도인 것으로 알려져 있다.
2. 빠르고 유연한 개발 : 오픈소스 커뮤니티는 보통 최신 기술 정보 및 문제점과 해결책을 공유하는 형태로 자유롭게 운영되기 때문에 독점 프로그램에 비해 기술 발전 속도가 빠르다.
3. 다양성 : 특정 SW 및 HW의 종속적인 부분에서 탈피되기 때문에 다양한 소스코드가 존재한다.
4. 호환성 : 오픈소스는 주로 오픈 포맷 또는 표준화 프로토콜을 사용하기 때문에 서로 다른 소프트웨어간의 상호연동성이 보장된다. 모든 기기들이 서로 다른 네트워크를 통해 하나로 연결되는 유비쿼터스 시대에 필수적인 요소이다.
5. 신뢰성과 안정성 : 오픈소스의 개발 과정을 보면 전 세계에 있는 수많은 우수한 개발자들이 직접 개발과 디버깅 과정에 참여하기 때문에 In-house에서 폐쇄적으로 개발되는 독점 프로그램에 비해 비교적 안정적으로 동작한다. 하지만 신뢰성과 안정성은 많은 개발자들이 적극적으로 참여할 때에만 가능하므로, 사용하고자 하는 오픈소스의 개발과정, 평판 등을 주의 깊게 살펴보아야 한다. 실제로 잡다한 수많은 오픈소스들이 있기 때문에 쓸 만한 오픈소스를 가려내는 것이 중요하다.

* 오픈소스의 단점

1. 어플리케이션의 부족: 대부분의 이용자들이 MS 윈도우즈 기반의 GUI에 익숙한 반면, 오픈소스는 GUI가 일반적이지 않는다. 또 오픈소스는 리눅스 기반으로 개발된 애플리케이션이 많기 때문에 윈도우즈 기반 애플리케이션과 호환되지 않는 문제점도 가지고 있다.
2. 빈약한 문서: 오픈소스를 수정하여 원하는 애플리케이션을 제작하고자 할 경우 문서화가 중요한데, 상용 프로그램에 비해 오픈소스는 체계적인 문서를 갖지 않은 경우가 많다. 이에 따라서 개발과정을 지체 시키는 일이 생기게 됩니다.
3. 유지보수의 어려움: 오픈소스는 영리를 목적으로 하는 회사에서 개발되는 것이 아니라 개인의 자발적인 참여를 통해 개발되는 경우가 많기 때문에 독점 프로그램에서 볼 수 있는 로드맵을 기대하기 어렵다. 어느 날 갑자기 단종 되고, 업그레이드가 중단되는 경우도 있다.
4. 이에 따라 관리, 모니터링, 제어 등 유지보수 및 업데이트 및 업그레이드 비용, 기술지원비용이 생겨나게 된다.
5. 지적재산권: 일반적으로 오픈소스를 수정한 프로그램은 사용료 없이 배포할 것을 요구하고 있어 기업이 보유한 특허를 소스코드에 포함시켜 재배포하려는 경우 반드시 명확한 입장을 밝히고 오픈소스 저작권자의 정책을 고려해야 한다. 하지만 라이센스 미준수에 따른 저작권법 위반 및 특허소송의 위험성이 존재 할 수 있다.
6. 이전 및 통합: 기업이 보유한 소프트웨어를 오픈소스로 전환할 경우 Migration과 Integration 비용이 발생한다. 특히 OS와 같은 인프라성 오픈소스가 변경된 경우 해당 오픈소스와 관련된 H/W, S/W, Network를 고려해야 한다.
7. 오픈소스의 역사

* 상용 소프트웨어의 등장 & unix 유료화

[](https://github.com/sue991/SOS/blob/master/assets/Unix.png)

1. Unix의 등장

1960년대 중반 당시 AT&T 와 General Electric 사, Honeywell 사, MIT 공과대학에서 MULTICS 라는 프로젝트를 미 국방성(ARPA)으로부터 자금 지원을 받아서 수행하게 된다. 멀틱스 프로젝트의 목적은 끊이지 않고 돌아갈 수 있는 다중 사용자, 다중 프로세서, 다중 프로세스 기반의 시스템을 만드는 것이었다. 물론 이 프로젝트에는 군사적인 목적도 있어서 보안과 관련해서도 다중 레벨 보안의 개념도 지원할 계획이었다. 그 당시로써 이상적인 시스템으로 계획되었고, 디자인되었던 시스템이었다. 1969년 켄 톰슨은 당시 AT&T사의 연구원으로 멀틱스 프로젝트에 참가하고 있었는데, 그는 사무실에 있는 컴퓨터에 멀틱스와 비슷한 개념을 가진 운영체재를 한번 구현해 보게 되었다. 그러다 그는 여기에 재미를 느끼고 함꼐 멀틱스 프로젝트에서 일했던 데니스 리치와 피터 뉴만과 이 프로젝트를 같이 하게 되었고, 이 프로젝트의 이름을 '유닉스(Unix)'라고 지었다.

1. 퍼듀 대학의 톰슨은 당시 대부분의 시스템처럼 버클리에 설치된 시스템에서는 관여하고 있지 않았다. 그렇지만 몇 가지 이상한 시스템 충돌이 일어나면서 그의 전문 지식이 필요하게 되었으며, 이 때문에 연구에 참여하게 되었다. 버클리에는 자동 응답 기능이 없었으므로 톰슨은 기계실에 있는 스탠디포드에게 전화를 하여 전화를 모뎀에 삽입하도록 하였다. 대부분의 충돌은 문서에 있는 것과는 달리 디스크 컨트롤러가 중첩된 탐색을 신뢰성 있게 수행하지 못했기 때문에 일어났다. 톰슨의 원력디버깅은 버클리와 벨 연구소 사이에 이루어진 최초의 협동 작업이었다. 버클리와 작업을 공유하려 했던 벨 연구소 연구원들의 자발성은 버클리에서 해당 소프트웨어를 사용할 수 있도록 신속하게 개선하는 데 도움이 되었다. 기존의 멀틱스 프로젝트에서는 너무 많은 것을 구현하려고 했지만 유닉스 시스템에서는 단지 프로그램을 실행시키는 것에만 중점을 두었다. 이런저런 기능을 전부 제외하고 굉장히 단순하고 간단하게 만든 것이다.
2. 1970년대, AT&T 연구원들은 유닉스 시스템을 점점 보안하고 발전시키면서 하나의 기능만 하는 자그마하고 단순한 프로그램들을 함께 사용함으로써 엄청난 일을 수행할 수 있게 만드는 유닉스 시스템의 철학을 세우게 되었다. 그들은 이 시스템을 잘 활용하기 위해 작은 프로그램들의 사용을 극대화 했고 또 프로그램의 소스 코드도 수정해 가며 계속 유닉스를 발전시키게 된다.
3. 그러다 1973년 데니스 리치가 C 프로그래밍 언어를 사용해 대부분의 유닉스 시스템의 코드를 다시 쓰게 된다. C 언어는 다른 기계로 포티잉 매우 쉽기 때문에, 유닉스 시스템이 C로 다시 쓰여지면서 산불처럼 크게 퍼지기 시작했다. 이로 인해 유닉스 시스템은 빠르고 크게 사람들 사이에서 확산되었고 어느새 널리 대중적으로 사용되는 운영체제가 되었다. 이에 따라 자연스레 대규모 기업들에서도 유닉스를 주목하게 되었다. 실제로, 1977년경에 인터넷의 500개 가량의 사이트에서 유닉스를 운용하고 있다는 통계가 있으며, 그 중 125개는 대학에서 운용되었다. 하지만 그 당시 유닉스가 운용되는 대부분의 기계들은 실험실 등에 위치해 있었는데, 당시의 네트워크 환경도 그랬지만 대부분 격리된 장소에서 네트워크로의 접근 등이 허용되지 않는 환경이었다. 따라서 보안과 같은 요소가 거의 필요하지 않았다. 단지 몇명의 허가된 사용자들만이 시스템에 접근할 수 있었다. 그러면서 유닉스의 사용자들은 계속적으로 툴과 유틸리티들을 개발하고, 보다 나은 성능을 가지도록 향상시키고 있었다.
4. 1970년대 말, 버클리대학은 유닉스 소스코드가 담긴 테이프를 구입하였다. 그런데 빌 조이와 척 핼리라는 학생이 그 소스 코드를 조금씩 수정하였고, 1978년 자기들이 수정한 코드로 만든 유닉스와 유틸리티들을 포함하는 배포판을 만들어 팔았다. 이것이 유닉스 운영체제 중 하나인 BSD(Berkeley Software Distribution)버전의 유닉스 시스템의 최초 버전이다. BSD 시스템이 기존의 AT&T의 유닉스와 다른 점은 동시에 실행되고 있는 여러 개의 프로그램 간에 작업 전환이 가능하다는 것이다. 또한 네트워킹 소프트웨어 개발이 가능해졌다. 이 네트워킹 프로그램들로 해 유닉스를 사용하여 근거리 통신망을 구축하는 것이 매우 수월해졌고, 네크워크 자체가 컴퓨터가 될 만큼 컴퓨터에 빠질 수 없는 존재가 되었다.
5. **1984년, AT&T가 해체되고 유닉스가 최초로 상업 제품이 되었다. 썬과 다른 회사가 만들어낸 워크스테이션급 기계들은 해커들에게 새로운 세계 를 열어 주었다. 해당 기계들은 고성능 그래픽이 가능하고 네트워크를 통해 공유 데이터를 전송할 수 있도록 만들어졌다. 1980년대에 해커 문화는 이러한 기능에 서 가장 많은 것을 얻어낼 수 있는 소프트웨어와 도구를 만드는 도전에 심취해 있었다. 버클리 유닉스는 ARPAnet 프로토콜의 내장 지원을 개발하였으며, 이는 네트워킹 문제에 대한 해결책을 제공하고 이후 인터넷의 성장을 뒷받침하였다.**
6. 1990년대가 열리자 80년대의 워크스테이션 기술은 새롭고, 저가 개인용 컴퓨터에 의해 분명히 위협받는 것처럼 보이기 시작하였다. 처음으로 해커들은 각자 10년 전 미니 컴퓨터의 성능과 저장 능력에 맞먹는 개인용 컴퓨터를 가질 수 있었다. 바로 전체 개발환경을 지원하고 인터넷과 통신할 수 있는 유닉스 엔진이다.
7. 오늘날의 유닉스 시스템은 위에서 언급한 매우 많은 표준들과 방식들에 기반해서 발전해왔고 또한 제안된 표준들에 의해 비교적 비슷하고 공통적인 인터페이스를 가지게 되었으며, 여러가지 분야에서 기본 플랫폼으로서의 역할을 충실히 수행하고 있다. 후엔 유닉스를 기반으로 리눅스를 만들고 발전시킬 수 있었기에 유닉스는 소프트웨어의 발전에 기반이 된 매우 중요한 역할을 맡고 있다

* 자유 소프트웨어 연합(FSF)의 등장과 GNU

1. 자유 소프트웨어에 대한 리처드 스톨먼의 철학은 1971년 MIT 대학의 인공지능 연구소에서 소프트웨어를 공유하는 공동체의 일원이 되면서였다. 이 공동체는 수년 전부터 존재했고 MI에만 있는 특별한 존재도 아니었다. 인공지능 연구소엔 ITS라 불리는 운영체제를 사용했었는데, ITS는 당시에 사용되던 DEC사의 PDP-10 기종에 탑재할 목적으로 인공지능 연구소의 해커 연구원들에 의해 만들어진 운영체제이다.
2. 당시엔 아직 자유 소프트웨어란 정의는 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.

#### [https://github.com/sue991/SOS/raw/master/assets/%EC%98%88%EC%88%982.png](https://github.com/sue991/SOS/blob/master/assets/%EC%98%88%EC%88%982.png)<Richard Matthew Stallman>

1. 이 방식이 일반적인 것이기에 흥미로운 프로그램 사용하는 사람이 있으면 얼마든지 소스 코드를 보여 달라고 요청할 수 있었다. 공동체 시절은 소스 코드를 자유롭게 공유 했기에 프로그램을 수정하거나 해당 프로그램을 기반으로 발전시킬 가능성이 풍부했던 때이다. 하지만 1980년대 초, DEC사가 P-10 시리즈의 생산을 중단하며 공동체는 붕괴되었다. 1960년대를 차지했던 PDP-10 시리즈도 1980년대가 되며 시대의 발전 흐름에 따라 구조적 한계를 직면하고 DEC사는 PDP-10 시리즈의 생산을 중단했다. 이는 ITS 환경에서 만들어진 모든 프로그램이 호환성의 문제로 더 이상 사용될 수 없음을 의미했다.
2. 인공지능 연구소를 주축으로 모인 공동체도 PDP-10과 함께 붕괴해 1981년에는 대부분의 인원이 직장을 옮기게 되었다. 해커들의 이직에 따라 공동체를 유지할 인원이 없어지고, ITS가 아닌 자유 운영체제가 아닌 DEC의 운영 체제를 도입하는 데까지 이르렀다.
3. 당시에 사용되었던 현대적 모델의 컴퓨터는 전용 운영체제를 탑재했는데, 이들은 자유 소프트웨어가 아니었다. 바이너리 형태의 프로그램조차 관련 자료를 유출하지 않겠다는 비공개 계약 조건에 동의해야만 했다.
4. 이는 컴퓨터를 사용하는 처음 단계부터 단절적인, 주위 사람과의 협력을 하지 않겠다며 약속하는 것을 의미했다. 상호 협력적인 공동체의 형성이 불가능해진 것이다. 독점 소프트웨어의 소유자들은 소프트웨어를 공유하는 것을 저작권 침해라며 마치 해악한 것과 동일시했고, 프로그램에 대한 수정은 전적으로 그들을 통해야 했다. 독점 소프트웨어 제도는 소프트웨어를 다른 사람과의 공유나 교환을 금지하기 때문에 절대 발전적인 제도라 할 수 없다. 이는 분명히 비윤리적인 제도이지만 누군가에겐 이것이 충격적인 사실로 받아들여질 수 있다. 그러나 사람을 고립시키고 서로를 돕는 것을 금지하는 방법으로 무력하게 만드는 제도에 대해 스톨만은 부정적이었다. 반면 소프트웨어 판매자들은 소프트웨어의 유통 형태에 대해 기존의 방식이 유일한 방법이라는 생각을 사람들에게 주입하기 위해 오랫동안 노력해왔다. “저작권 침해 행위를 중단해라”, “우리의 권리를 행사한다.”는 등의 표현 뒤에 숨겨진 의도는 법률적으로 과격한 표현을 통해 대중들이 그들의 주장을 비판 없이 수용하게 만드는 데 있다. 또한 ‘권리의 침해’나 ‘권리의 주장’이라는 표현 안에는 소프트웨어를 창작하고 수정하는 모든 권한이 제조 회사에 귀속되어 있으므로 회사의 주장을 인정하고 수용하지 않으면 특정한 작업을 수행할 수 있는 프로그램이나 그 밖의 유용한 소프트웨어를 전혀 사용할 수 없게 되리라는 강압적인 잘못된 가정이 포함되어 있다. 이러한 가정은 꽤나 설득력 있게 들릴 수 있지만 자유 소프트웨어 운동이 구속과 통제가 없는 많은 양의 자유 소프트웨어를 만들며 그 오류를 불식할 수 있었다. 사용자를 최우선으로 고려한 상식과 윤리를 통해서 이 문제를 판단한 스톨만은 사람의 서로의 도움이 사회를 지탱하는 근간이라 판단했고, 프로그램을 자유롭게 수정하거나 공유해야 한다는 결론에 도달했다. 공동체가 사라짐에 따라 예전처럼 지탱할수 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. 따라서 스톨만은 사람들을 단절시키고, 세상을 발전시키지 못해 아니 퇴행 시킬 것이라고 판단했다. 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 공유하기를 거부했을 때 그는 결코 같은 일을 다른 사람이 겪게 하지 않겠다고 다짐한 것이다.
5. 그누(GNU) 프로젝트 시작

[](https://github.com/sue991/SOS/blob/master/assets/GNU.png)

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 구애 받을 필요가 없어지므로 누구나 자유롭게 컴퓨터를 사용할 수 있게 될 것이었다. 운영체제의 개발자로써 그는 이러한 작업에 필요한 적합한 기술을 갖고 있었다. 그는 새롭게 개발할 운영체제를 유닉스와 호환되도록 만들려 했다. 그렇게 해 기존의 유닉스 사용자들이 손쉽게 적응하고 GNU로 유입될 수 있으리라 생각했기 때문이다. GNU라는 이름은 GNU’s Not Unix, 즉 “GNU는 유닉스가 아니다.”라는 뜻이 되도록 의도적으로 조합해서 만든 것이다. 따라서 GNU는 유닉스와 호환되도록 만든 운영체제이지만 유닉스와는 다른 체제임을 알리기 위해 만든 이름이다. 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. GNU시스템은 GNU 프로젝트를 통해 구현하려는 유닉스와 호환되는 소프트웨어 시스템 전체를 의미하고, GNU 프로젝트는 이러한 시스템을 구현하기 위해 진행되는 프로젝트 자체를 의미했다. 1984년 1월, MIT 연구원직을 사직하고 스톨만은 GNU 소프트웨어를 만들기 시작했다. 그가 MIT를 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 1984년 9월부터 GNU Emacs를 만들기 시작해 1985년 초에는 제법 완성된 상태가 되었다. 사람들의 GNU Emacs에 대한 수요가 생길 때 즈음에서 그는 이 에디터를 어떠한 방식으로 배포할 것인가에 의문이 생겼다. 그 당시에는 Emacs에 관심을 가진 사람 중에서 인터넷을 통해 구할 수 있는 사람이 거의 없었기에 어떠한 방식으로 Emacs를 구할지 알려주는 것도 중요한 문제였다. 그는 반송용 우표가 붙어있는 봉투를 테이프와 함께 보내면 PDP-10용 Emacs를 복사해 주겠다고 했다. 즉, Emacs를 배포함에 있어서 금전적인 비용이 청구되지 않는 방법을 말이다. 그는 150달러의 비용을 지불하면 누구나 Emacs가 든 테이프를 우성해주는 방식을 생각해내어, 이러한 판매 방식을 통해 자유 소프트웨어를 이용한 사업을 시작하고, 리눅스를 기반으로 한 GNU 시스템 전체를 담은 배포판을 판매하는 현재의 리눅스 배포판 업체들의 시초가 되었다. GNU의 목적은 GNU를 널리 알리는 것이 아닌, 사용자에게 자유를 주는 것이기에 배포 기준에는 GNU가 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.

1. Emacs의 사용에 대한 관심의 증가에 따라 사람들이 GNU 프로젝트에 참여하기 시작했고, 기금을 모을 적절한 시기가 되었다. 그래서 1985년, 자유 소프트웨어의 개발을 위해 자유 소프트웨어 재단을 설립하게 된다. 자유 소프트웨어 재단, FSF는 Emacs의 테이프 배포 사업을 맡고, 점진적으로 다른 소프트웨어도 테이프에 담아 판매했다. FSF는 기부를 받지만 대부분의 운영 자금이 자유 소프트웨어 판매와 관련된 부수적 서비스로 충당된다.
2. 자유 소프트웨어가 가진 철학은 만연한 특정한 형태의 상업적 관행을 반대하는 것이지, 상업 행위 자체를 중단시키려는 것은 아니다. GNU의 주된 목적은 자유 소프트웨어가 되는 것이다. GNU가 유닉스에 비해 기술적 장점을 갖지 못하더라도, 사용자의 협력을 통해 사회적 이점을 가질 것이고, 사용자의 자유를 존중한다는 윤리적인 이점도 있다. 하지만 우수한 기술의 표준을 GNU에 수용하는 것은 당연한 일이었다. GNU 프로젝트의 명성이 높아짐에 따라 유닉스가 탑재된 시스템을 기증하는 사람이 늘어났다. GNU를 구성하는 프로그램 개발의 가장 쉬운 방법은 유닉스 환경에서 먼저 개발한 뒤 이를 GNU에 맞게 수정하는 것이기 때문에 이러한 시스템은 매우 유용했다. 하지만 기본적인 GNU의 철학은 독점 소프트웨어를 사용하지 않는 것이라 독점 소프트웨어인 유닉스를 통해 자유 소프트웨어를 만든다는 것은 모순이라는 주장이 있었다.
3. 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.
4. 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 C라이브러리가 필요해서고, BASH를 개발한 이유는 유닉스 환경이 셸을 요구해서이다. 어떤 프로그램은 자유에 대한 위협에 대처하기 위해 만들어졌다.
5. GNU 프로젝트를 시작할 당시엔 GNU 시스템이 모두 완성된 뒤, 전부를 공개하려 했지만 아래와 같은 이유로 인해 그렇게 되지 않았다. GNU는 알다시피 유닉스에서 요소를 구현한 뒤 이를 GNU로 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.
6. 1990년 무렵 GNU 시스템이 거의 완성되었지만, 운영체제를 구성하는 핵심 부분인 커널이 누락되었었다. 따라서 프로젝트의 일원들은 MACH를 기반으로 하는 한 서버 프로세스를 연결해 커널을 구현하기로 결정했다. 커널은 운영체제를 구성하는 핵심 요소를 의미하는 말이다.
7. MACH는 자율적 프로세스의 집합이며, 유닉스 커널이 가진 다양한 기능을 그대로 수행할 수 있는 것이었다.
8. 현재까지도 GNU HURD(GNU 커널의 명칭)은 하나의 제품으로 사용될 수 있는 상태가 아니다. 하지만 다행스럽게도 1991년, 리누스 토르발스가 유닉스 커널과 호환가능하도록 만든 리눅스라는 이름의 커널이 그것이었다. 1년뒤인 1992년, GNU 시스템과 리눅스를 결합함시킴으로써 하나의 완전한 자유 운영체제를 만들 수 있었다. 따라서 현재 사용되는 GNU 시스템은 리눅스 덕분에 가능했던 것이었다. 이 시스템을 GNU/Linux라고 부른다.
9. 자유 운영체제에서 사용가능한 자유 소프트웨어가 아닌 라이브러리는 자유 소프트웨어 개발자에게 함정과 같다. 이러한 라이브러리가 가진 매혹적인 기능을 외면하는 것은 개발자에게 있어서 어렵다. 그러나 유혹에 넘어가는것은 스스로 자신의 무덤을 파는 것과 같다. 왜냐하면 이러한 라이브러리를 이용해서 만들어진 프로그램은 자유 운영체제에 포함될 수 없기 때문이다.
10. 이러한 경우의 첫 번째 사례는 1980년대에 사용되던 모티프 툴킷에서 찾아볼 수 있다. 그 당시에는 아직 자유 운영체제가 존재하지 않았지만 모티프가 갖고 있던 문제가 어떤 결과를 초래하리라는 것은 이미 자명한 일이었다. GNU 프로젝트는 두 가지 방법으로 여기에 대응했다. 첫째는 자유 소프트웨어 프로젝트가 모티프뿐만 아니라 자유 소프트웨어에 해당하는 X 툴킷 위젯도 지원하도록 권고한 것이고, 또 하나는 모티프를 대체할 수 있는 라이브러리의 개발을 추진한 것이다. 이 작업은 몇 년이 소요되었는데, 열성적인 프로그래머들 덕분에 1997년에 이르러 모티프를 사용하는 대부분의 프로그램을 지원할 수 있는 레스티프 라이브러리가 완성되었다.
11. 1996년과 1998년 사이에는 이른바 Qt라고 불리는 또 다른 GUI 툴킷 라이브러리가 등장했다. Qt는 데스크톱 환경의 자유 소프트웨어인 KDE에 사용되는 핵심 요소다. 그러나 자유 운영체제인 GNU/Linux 시스템에서는 Qt의 라이선스와 관련해서 KDE를 사용할 수 없었다. GNU/Linux를 판매하는 일부 상용 배포판 업체에 서는 KDE를 함께 패키징해서 활용성을 증대시킨 시스템을 만들기도 했지만 결국 이것은 자유를 감소시키는 것이었다. KDE 그룹은 좀 더 많은 프로그래머가 Qt를 사용하도록 적극 권장했고, 새롭게 리눅스를 사용하게 된 수많은 사람은 이러한 문제를 미처 생각하지도 못한채 KDE를 사용하게 되었다. 상황이 무척 심각해진 것 이다. 자유 소프트웨어 공동체는 GNOME과 Harmony라는 두 가지 방법으로 이 문제에 대응했다.
12. GNU는 먼저 GNOME이라고 불리는 데스크톱 환경의 개발을 시작했다. 이 프로젝트는 1997년부터 시작되었고 레드햇 소프트웨어가 이를 후원했다. GNOME은 데스크톱 환경과 유사한 기능을 제공하지만 오직 자유 소프트웨어만 으로 만들어진다. 또한 C++ 이외에 다양한 종류의 언어를 지원함으로써 기술적인 장점도 함께 제공한다. 그러나 GNOME의 주된 목적은 자유라고 할 수 있다. GNOME 환경은 자유 소프트웨어가 아닌 어떠한 프로그램의 기능도 자유 소프트웨어 안에서 충족시키기 위한 것이다. 또 하나의 대안인 Harmony는 KDE 소프트웨어들을 Qt 없이도 사용할 수 있게 하기 위해서 개발된 라이브러리다.
13. 1998년 11월에 Qt 개발자들은 Qt를 자유 소프트웨어로 변경할 것이라고 발표했다. 그러나 이는 아직까지 확실한 것이 아니며, 부분적으로 Qt가 자유 소프트웨어가 아니었을 때 가졌던 문제에 단호하게 대처함으로써 나타난 결과라고 할 수 있다.
14. 이들이 당면한 가장 큰 위협은 바로 소프트웨어에 대한 특허에서 발생했다. 알고리즘과 새로운 기술에 부여되는 특허는 20년까지 보장되므로 이러한 영역은 소프트웨어가 아직 접근할 수 없는 것이다. LZW 압축 알고리즘에 대한 기술 특허가 1983년에 신청되었기 때문에 여전히 정상적인 GIF압축을 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, 자유 소프트웨어가 실패했다고 단원할 것이다 따라서 자유 소프트웨어가 가진 안정성과 성능에 대해서 말하는 것이 사람들에겐 유용한 동안에는 자유와 자유를 지켜나가기 위한 원칙도 함께 이야기해야 한다.
15. 리누스토발즈의 리눅스

1991년 Linus Torvalds 는 ``리눅스" 라고 이름 지은 운영체제 커널을 개발하기 시작했다 [Torvalds 1999]. 이 커널은 자유로이 수정될 수 있고 매우 유용한 운영체제를 만들기 위해 FSF 산물과 다른 컴포넌트들 (특히 BSD 컴포넌트들의 일부와 MIT 의 X 윈도우 소프트웨어) 과 병합될 수 있었는데 이 책에서 커널 자체는 ``리눅스 커널"로 전체 결합은 ``리눅스"로 지칭할 것이다. 많은 사람들은 대신 ``GNU/Linux" 라는 용어를 사용함을 주목해라.

리눅스 공동체에서 여러 조직들은 사용할 수 있는 컴포넌트들을 다르게 조합하였는데 각 조합은 ``배포판" 으로 부르며 배포판들을 개발한 조직들은 ``배포업자"라고 부른다. 일반적으로 널리 알려진 배포판으로는 레드햇 (Red Hat), 맨드레이크 (Mandrake), 수제 (SuSE), 칼데라 (Caldera), 코렐 (Corel) 및 데비안 (Debian) 이 있다. 다양한 배포판들 간에는 차이점들이 있지만 모든 배포판들은 동일한 기반인 리눅스 커널 및 GNU glibc 라이브러리들에 기초하고 있다. 두가지 모두는 ``copyleft" 스타일의 라이센스로 다뤄지기 때문에 이러한 기반에 대한 변경은 일반적으로 모든 사람이 사용할 수 있어야 한다. 이는 BSD 와 AT&T 에서 파생된 유닉스 시스템 간에는 존재하지 않는 리눅스만의 기반에서 리눅스 배포판들을 통합시키는 힘이 원천이다. 이 책은 특정 리눅스 배포판에 고유한 것은 아니며 리눅스를 논의할 때 이 책은 본질적으로 모든 현재 주요 리눅스 배포판들에 대해 유효한 가정인 리눅스 커널 버전 2.2 이상과 C 라이브러리 glibc 2.1 이상을 가정한다.

* 현재의 오픈소스
* 거의 모든 분야에서 활용되고 있다.

18
오픈소스의 확대 – 이제는 거의 모든 분야
운영체제 웹서버/미들웨어
데이터베이스 기타
 

1. 전통적인 공개SW 동향
   * (초기 공개SW) 공개SW 생태계가 구축되기 시작한 초기에는 인터넷 서비스를 제공하기 위해 리눅스 계열 운영체제, MySQL 같은 데이터베이스, Apache HTTP 서버 같은 공개SW들을 활용하여 낮은 비용으로 인터넷 서버를 구축하는 용도로 주로 활용되었으며 이와 같은 초기 공개 SW들은 지금도 널리 활용되고 있음
   * (리눅스 커널) 1991년 공개된 이후 공개SW 활성화를 가능하게 한 대표적인 사례로 초기에는 RedHat, Suse 등 리눅스를 기반으로 하는 공개SW 기업들을 탄생시켰고 2000년대에는 구글에서 리눅스 커널을 기반으로 Android를 개발하여 모바일 플랫폼 시장을 주도하고 있음
   * 미래 자동차 인포테인먼트(Infortainment) 시장에서도 <그림 1>에서 보는 바와 같이 리눅스의 활용이 폭발적으로 증가할 것으로 예상되고 있을 정도로 리눅스 커널의 중요성은 커지고 있음
2. 새로운 공개SW 활용 분야
   * 신 SW기술

최근 SW산업은 인공지능, 빅데이터, 클라우드와 같은 신 SW기술들이 주도하고 있으며 특히 공개SW 형식으로 개발된 텐서플로우(Tensorflow), R 그리고 오픈스택(Open Stack) 등이 새로운 SW 분야의 기술 발전을 이끌어 가고 있음

* + 텐서플로우

구글에서 2009년도부터 개발하였던 DistBelief의 후속 프로젝트인 텐서플로우는 구글 브레인팀의 주도로 개발된 기계학습 오픈소스 SW 라이브러리이며 2015년 Apache 사용권으로 공개된 이후 2018년 현재 기계학습 분야에서 가장 활발한 공개SW 프로젝트임

* + 활용
* 최신 기계학습 기술을 CPU, GPU, TPU 등의 다양한 컴퓨팅 환경에서 이용할 수 있도록 개발되었고 퀄컴, 엔비디아, 인텔, 우버, 에어비엔비 그리고 AMD 등 대표적인 IT 기업들에서 활용되고 있으며 텐서플로우를 활용하는 기업들은 지속적으로 늘어날 것으로 예상됨
* 가상화 기술을 기반으로 클라우드 업무 분산을 위한 엣지(Edge) 컴퓨팅, 통신분야에서 네트워크 가상화, 금 융 분 야에서 빅데이터 분 석과 고 성능 컴퓨팅 자원이 필요한 과학 분야 등 다양한 분야에서 활용이 가능하며 레드햇, 시스코, HP, 델, IBM, Huawei 등 500개 이상의 기업들이 오픈스택을 활용함
  + 깃허브 동향

텐서플로우는 대표적인 공개SW 저장소인 깃허브(Github)에서 2017년 가장 많이 포크(Fork)된 프로젝트로서, 약 2.4만 번 포크되었으며 참여한 개발자 수는 약 7,300명으로 5위를 차지했고 관련 프로젝트인 텐서플로우 모델(Tensorflow Models)은 약 8천 번 포크 되어 포크 순위로 5위를 차지함.

* + 오픈 스택4

2010년 미국 NASA와 Rackspace가 공동 추진한 클라우드 플랫폼 개발 프로젝트를 기반으로 하고 있으며 2012년 오픈 스택 재단이 창립되어 Apache 사용권이 적용되어 관리되고 있으며 클라우드 폼 구축에 필요한 서버, 저장장치(Storage), 네트워킹 자원을 가상화하기 위한 다양한 세부 프로젝트들이 진행되고 있음

* + 시장전망

초연결 기반의 제4차 산업혁명의 다양한 서비스들을 제공하기 위해서는 컴퓨팅 자원 활용의 효율화를 위한 클라우드 플랫폼 구축이 필수적이기에 오픈 스택을 이용하여 사설 및 공용 클라우드 자체 구축 방안이 주목받고 있기 때문에 관련 시장은 <그림 3>과 같이 급성장할 것으로 예상됨

* + R5

1992년부터 뉴질랜드 오클랜드 대학에서 개발되기 시작하여 2000년에 안정적인 빅데이터 분석이 가능해진 통계 프로그래밍 언어 겸 공개SW로서, 현재 R 재단에서 GPL 사용권을 적용하여 관리하고 있으며 하둡(Hadoop)과도 연동이 가능하고 MS 윈도우, 애플 맥, 리눅스 그리고 유닉스 등 다양한 환경에서 동작 가능함

* + 성장

200만 명 이상의 사용자를 가지고 있는 R은 사용자들이 주도적으로 R 컨소시엄을 조직하여 전세계 사용자 및 관리자 그리고 개발자를 지원하고 있으며 다양한판(Version)의 R 관련 코드와 문서를 내려받기 가능한 CRAN(Comprehensive R Archive Network)의 R 패키지 수는 <그림 4>처럼 급속히 늘어나고 있음

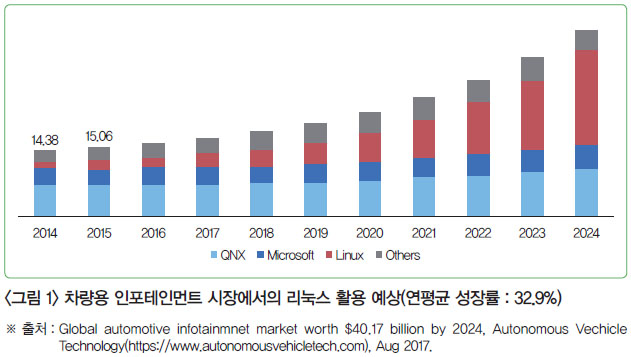
1. 공개SW의 상업적 활용 현황
   * 공개SW 전략적 활용국 내에 공 개SW 도입 될 당시에는 무 료 사용이 가능하다는 점 때문에 독점적인 일부 상용SW를 대체하기 위해 공개SW를 활용하였지만 해외 선진 SW기업들은 공개SW를 전략적으로 활용하여 시장 선점, 기술 우위 유지 등의 상업적으로 활용하는 사례가 늘어나고 있음
   * 국내

공개된 소스코드를 기반으로 무료로 활용할 수 있었기에 운영체제, 데이터베이스 등의 일부 분야에서 SW기술력을 확보하고 이를 통해 독점 상용SW를 대체하였지만 무료라는 인식이 과도하여 상업적 활용을 저해하는 이유가 되었고 이는 SW 선진국에 비해 SW경쟁력을 약화되는 부작용이 있었음

* + 구글

모바일 플랫폼 구축에 있어서 리눅스 커널을 기반으로 안드로이드 플랫폼을 개발하여 개발기간을 단축하였고 API를 공개하여 HTC, 삼성, LG, Huawei 등 주요 스마트폰 제조업체들이 낮은 가격으로 모바일 플랫폼을 사용하게 하여 모바일 플랫폼 시장의 주도권을 확보하였고 이를 통해 광고 수익 및 구글 지도, 구글 플레이 등의 서비스 사용료로 수익을 창출함

* + 공개SW 기업 증가 공개SW 시장이 성숙되는 초기에는 일부 업체들이 리눅스 커널과 MySQL 등을 활용하여 일부 상업화에 성공하였으나 현재는 클라우드 주도로 성장하는 SW시장을 기반으로 분산처리, 기계학습, 저장장치 관리 등 다양한 분야로 확산되어 <그림 5>처럼 많은 공개SW 기업들이 등장함

1. 오픈소스의 전망

* 해외

세계 OSS 시장 규모는 2015년 약 619억 달러로 추정되며 연평균 18.8%로 급성장하고 있다. OSS가 다양한 신 SW 산업 분야(클라우드, 빅데이터, 보안, 가상현실 등)로 활용 범위가 확대되면서 애플, 마이크로소프트와 같은 글로벌 SW 기업과 다양한 중소 SW 기업이 자사 SW를 공개하고 OSS 관련 커뮤니티를 지원하고 있다. 특히 미국을 중심으로 OSS 선순환 생태계가 구축되어 OSS 활동 참여가 SW 혁신의 동력으로 작용하고, 결과적으로 SW 산업 경쟁력 강화를 견인한다.

* 국내

최근 IT 업계의 화두로 자리 잡은 빅데이터, 사물인터넷(IoT), 머신 러닝, 클라우드는 리눅스, 하둡, 오픈스택과 같은 오픈소스 SW를 기반으로 하고 있다. 운영체제, DB, 클라우드 컴퓨팅 등 많은 분야에서 오픈소스의 활용도가 급속도로 높아지고 있는 것이다.  
시장분석기관 가트너는 내년까지 IT 분야 글로벌 기업들의 오픈소스 SW 채택율이 99%까지 늘어날 것으로 전망했다.  
모바일 기술 분야에서 대표적인 사례는 구글의 안드로이드 OS다. 안드로이드 OS는 오픈소스 전략을 통해 모바일 디바이스 운영체제(OS) 시장의 80%를 점유하고 있다.  
오픈소스 시대를 맞으며 폐쇄적 기업이라는 이미지를 가지고 있었던 마이크로소프트(MS)도 오픈소스 생태계에 적극 뛰어들고 있다. MS는 그동안 소스코드를 공개하지 않는 폐쇄적 정책으로 소프트웨어를 개발해왔다.  
MS는 지난 10년 간 다수 오픈소스 프로젝트에 참여한 데 이어 최근 리눅스 재단에도 가입했다. "리눅스는 암적인 존재"라며 오픈소스 기술을 비난했던 모습에서 벗어나 오픈소스 생태계를 적극 포용하는 전략을 취하고 있다.  
기업뿐만 아니라 각국 정부도 오픈소스SW 도입에 나서고 있다. 2015년부터 오픈소스 기술 친화 정책을 펼치고 있는 미국 정부는 직접 오픈소스 기술을 만들고 배포하는 데도 앞장서고 있다. 최근에는 개발을 주도한 오픈소스 SW 기술을 한 눈에 살펴볼 수 있는 웹사이트 code.gov도 공개했다.

국내 오픈소스 SW 시장 규모는 2013년 460억원에서 2014년 548억원으로 약 19.1% 성장한 것으로 추산된다. 이후 매년 12.3%씩 성장해 2018년에는 약 820억원 규모 시장을 형성할 것으로 예측된다. 이는 전체 IT 시장보다 2배 이상 빠른 성장세다.