

目次

編集にあたって	7
ソフトウェアについてのコメント	11
本書について	15
序文	19
第 1 部 GNU プロジェクトとフリーソフトウェア	25
第 1 章 GNU プロジェクト	27
第 2 章 GNU 宣言	55
第 3 章 フリーソフトウェアの定義	71
第 4 章 ソフトウェアが所有権者を持つてはならない理由	75
第 5 章 名前にどういう意味があるのか	83
第 6 章 「オープンソース」ではなく「フリーソフトウェア」と呼ぶべき理由	89
第 7 章 大学勤務のプログラマがフリーソフトウェアをリリースする方法	99
第 8 章 フリーソフトウェアの販売	103
第 9 章 フリーソフトウェアはフリードキュメントを必要とする	109
第 10 章 フリーソフトウェアの歌	113
第 2 部 コピーライト、コピーレフト、特許	115
第 11 章 読む権利	117
第 12 章 著作権の誤解 一連の誤り	123
第 13 章 科学は著作権を離れなければならない	139
第 14 章 コピーレフトとは何か	143
第 15 章 コピーレフト：プラグマティックな理想主義	147
第 16 章 ソフトウェア特許の危険	153

第 3 部	自由、社会、ソフトウェア	181
第 17 章	自分のコンピュータを信用できるか	183
第 18 章	ソフトウェアがフリーであるべき理由	189
第 19 章	コンピュータネットワーク時代の著作権と グローバル化	213
第 20 章	フリーソフトウェア：自由と協力	249
第 21 章	避けたほうがよい用語	307
第 4 部	ライセンス	317
	GNU 一般公有使用許諾書	319
	GNU 劣等一般公衆利用許諾契約書	333
	GNU 自由公開文書使用許諾書	353
	訳者あとがき	367
	索引	369

序文

時代には、必ずそれを代表する哲学者がいる。すなわち、時代の想像力を掴んだ作家や芸術家のことである。彼がそのような存在としてすぐに認められることがないわけではないが、時代とのつながりがはっきりとわかるまでには時間がかかることが多い。しかし、認められようがそうでなかろうが、時代の刻印を押すのは、詩のささやきとして、あるいは政治運動の嵐という形で、自らの理想を語った者である。

私たちの時代にも、哲学者がいる。彼は芸術家ではなく、職業的な作家でもない。彼はプログラマーである。リチャード・ストールマンは、MITの研究所で、オペレーティングソフトウェアを構築するプログラマー、アーキテクトとして仕事を始めた。彼は、「コード」によって支配されつつある世界の中で、自由を求める運動を確立したプログラマー、アーキテクトとして、公的なキャリアを積んできた。

「コード」とは、コンピュータを動かすためのテクノロジーである。ソフトウェアとして書かれているか、ハードウェアの中に焼き付けられているかにかかわらず、コードはマシンの動作を指示する命令のコレクションで、最初は言葉によって書かれる。これらのマシン(コンピュータ)は、次第に私たちの生活を規定し、支配するようになってきている。コードは、電話をどのようにつなぐかとか、テレビに何を表示するかといったことを決める。ブロードバンドリンクを介してビデオをコンピュータに流し込めるようにするかどうかを決める。そして、コンピュータが、それぞれのメーカーに報告することを決める。マシンが私たちを動かす、コードがマシンを動かす。

私たちは、このようなコードをどのように制御すべきなのだろうか。どのような理解が必要なのか。コードが実現する力に見合うだけの自由としてはどのようなものが必要なのだろうか。それはどのような力なのか。

これらは、ストールマンが生涯をかけて取り組んできた問題だった。彼は、仕事や言葉を通じて、コードの「フリー」を保つことの重要性を私たちに示してきた。フリーと言っても、コードの作者にお金を払わないという意味ではない。コード

の作者が構築した支配力の透明性を万人に対して保証するとともに、誰もがその力を自分のものとし、自分のニーズに合わせて書き換えられるようにすることである。これが「フリーソフトウェア」である。「フリーソフトウェア」は、コードで組み立てられた世界に対する1つの答えである。

「フリー」。ストールマンは、自らの用語の曖昧さを嘆いている。しかし、嘆く必要はない。パズルは、人々に考えることを強いる。そして、この「フリー」という言葉は、パズルのこのような機能を非常によく果たしている。現代のアメリカ人の耳には、「フリーソフトウェア」は空想的で不可能なことに聞こえるかもしれない。「フリー」なものなど、たかが昼食まで含めてもどこにもありはしない。世界を動かすもっとも重要なマシンのもっとも重要な言葉がどうやったら「フリー」になるのだろうか。健全なる社会がどうやったらそのような理想を追い求めることができるのだろうか。

「フリー」という単語が不協和音を発するのは、単語自体の問題ではなく、私たちの問題である。「フリー」には別の意味があり、「無料」はいくつもある意味の中の1つに過ぎない。ストールマンによれば、「フリー」という言葉のより根源的な意味は、「フリースピーチ（自由討論）」とか「free labor：自由労働（奴隷労働の反対語）」という言葉の中に含まれている。価格がないという意味ではなく、他者による支配が制限されているという意味でのフリーである。フリーソフトウェアとは、透明性が保証されており、変更に対して開かれている支配力である。free law、すなわち「自由社会：free society」の法律が、その規制内容を了解可能にするとともに、変更に対して開かれているのと同じである。ストールマンの「フリーソフトウェア運動」の目的は、できる限り多くのコードを「フリー」にすることにより、透明性を保証し、変更できるようにすることである。

これを実現するためのメカニズムは、GPL と呼ばれるライセンスを通じて構成される「コピーレフト」という恐ろしく巧妙な装置である。「フリーソフトウェア」は、著作権（コピーライト）法の力を使い、自らの公開性、変更可能性を確保するだけでなく、「フリーソフトウェア」を利用、援用する他のソフトウェア（および専門用語で「派生的な仕事」と呼ばれるもの）自体もフリーになることを保証する。フリーソフトウェアプログラムを利用し、改良し、その改良版を広くリリースするなら、その新バージョンも、オリジナルバージョンと同様にフリー

でなければならない。これは義務であり、違反すれば著作権法に抵触することになる。

「フリーソフトウェア」は、自由な社会と同様に敵を持つ。Microsoft は、GPL を「危険な」ライセンスと喧伝し、GPL に対して戦いを仕掛けてきた。しかし、Microsoft が言うところの危険は、ほとんど幻想に過ぎない。変更版もフリーでなければならないという GPL の条件を「強圧的」と非難する者もいる。しかし、条件は強制ではない。数百万ドル（おそらく）を支払わなければ Microsoft Office の変更版の頒布をユーザーに認めようとしないう Microsoft の姿勢が強圧的でないのなら、フリーソフトウェアの変更版もフリーでなければならないと主張することは強圧的ではないだろう。

そして、ストールマンのメッセージを過激だと言う人たちがいる。しかし、過激ということはないはずだ。実際、ストールマンの仕事は、コード以前の世界で私たちの伝統が築き上げてきた自由の概念の単純な翻訳である。「フリーソフトウェア」は、コードに支配された世界が、コード以前の世界を築き上げてきた伝統と同じくらい自由になることを保証するはずである。

たとえば、「自由社会」は、法律によって規制されている。しかし、自由社会が法律によって加えられる規制には、限界がある。法律を秘密にしている社会を自由社会と呼ぶことはできない。規制される人々から規制内容を隠すような体制は、未だかつて存在したことはない。法律が支配する。しかし、法律は、目に見える形でしか支配しない。そして、法律が目に見えると言えるのは、規制されている人々、あるいはその代理人（法律家や議会）が条文を知ることができ、コントロールできるときだけである。

法律のこの条件は、議会の仕事の範疇に留まるものではない。アメリカの法廷における法律の運用について考えてみよう。弁護士は、顧客の利益確保を助けるために顧客に雇われる。利益は、訴訟を通じて追求される場合がある。訴訟が提起されると、弁護士は摘要書を書く。摘要書は、裁判官が書く判決に影響を与える。判決は、特定の訴訟において勝利者を決めたり、特定の法律が憲法に違反していないかどうかを決める。

この過程におけるすべての文献は、ストールマンが言う意味でフリーである。摘要書は公開で、他者が自由に使うことができる。議論は透明であり（良いと言っ

ているわけではない)、理由書は最初に文書を書いた弁護士の許可を得ずに引用できる。弁護士が書いた理由書は、その後の摘要書で引用できる。他の摘要書、理由書は、それらを複製、統合することができる。アメリカ法にとっての「ソースコード」は、設計上、また原則上、公開で誰もが自由に利用できる。弁護士がしていることを真似よう。何しろ、摘要書の優劣は、過去に発生した事件をうまく再利用して創造的な内容になっているかどうかによって判断されるのだから。ソースはフリーであり、創造性と経済性はそれを基礎としている。

このフリーコードの経済(ここでは、法律の世界のフリーコードのことを意味する)は、弁護士の仕事を奪ったりはしない。弁護士事務所は、誰もが摘要書を自由に引用、複製できるにもかかわらず、優れた摘要書を書く意欲を保っている。弁護士は職人であり、その製品は公開されている。しかし、工芸は慈善事業ではない。弁護士には報酬が与えられる。社会は、無償でそのような仕事を求めようとはしない。それどころか、この経済は、古い仕事に新しい仕事が追加されることによって繁栄している。

これとは異なる法律運用を想像することもできるだろう。摘要書と理由書が秘密にされ、判決は結果を発表するだけで理由を述べない。法律は警察によって管理され、それ以外の人間には公開されない。規則を説明せずに規制が運用されている状態である。

このような社会を想像することはできるが、それを「フリー」だと言うことは考えられないだろう。そのような社会は、より活気に満ち、より効率的に力を配分できるかもしれないし、そうでないかもしれないが、そのような社会をフリーと呼ぶことはできない。自由な社会では、自由の理想は、効率的な運用よりも重視される。公開性と透明性は、その中で法律のシステムを構築するための制約であって、指導者にとって便利であれば付け加えられるようなオプションではない。ソフトウェアコードによって支配される社会は、それ以下であってはならないだろう。

コード開発は、訴訟ではない。訴訟よりも良いものであり、豊かであり、生産的である。しかし、法律は、製品の生産に対する完全な支配が創造性や意欲に影響を与えないことをはっきりと示す例である。ジャズ、小説、建築などと同様に、法律は、以前に行われた仕事を基礎として構築されている。創造性が宿るのは、

いつもこの追加や変更の部分である。そして、自由社会とは、このような意味でもっとも重要な資源の自由を保証する社会のことである。

本書は、リチャード・ストールマンの評論や講演を集め、その微妙な意味や説得力を明確にすることを意図した初めての試みである。評論は、著作権からフリーソフトウェア運動の歴史まで、広い範囲にまたがっている。その中には、デジタルの世界における著作権に懐疑の目を向けさせることになった環境の変化についての示唆に富む論述など、あまりよく知られていないテーマを扱ったものが多く含まれている。これらは、このとでも強力な（他のことは差し置いても、発想、情熱、高潔さにおいて強力な）人物の思想を理解したい人々にとって貴重な資料になるだろう。また、彼の考え方を受け入れ、その上に新たな思想を構築しようとする人々に刺激を与えることだろう。

私は、ストールマン氏のことをよく知らない。好きになるのは大変そうな人物だということを知っている程度である。彼は衝動的で、しばしば短気になる。彼の怒りは、敵と同じように友にも向かう。彼は妥協を知らず、頑固である。この2つのことについては病気のようなものだ。

しかし、私たちの社会がコードの威力と危険性を理解するようになったら、また、コードは法律や政府と同じく透明でフリーなものでなければならないということがわかったら、私たちはこの妥協を知らない頑固なプログラマのことを振り返り、彼が現実化しようとして闘ってきたビジョン——自由と見識の前にコンパイラが跪く世界というビジョンを思い出すだろう。そして、次世代の社会が手にするはずの自由を獲得するために、その言動を通じて最大限の努力を示したのが、彼以外の誰でもないことを知るはずである。

私たちはまだその自由を手にしていない。それどころか、その自由を守ることに失敗するかもしれない。しかし、その成否にかかわらず、本書にはその自由がどのようなものになり得るかが描かれている。そして、これらの言葉と仕事を残した人物の中には、この自由の獲得のためにストールマンと同じように闘おうとするすべての人々を鼓舞してやまないひらめきがある。

ローレンス・レッシグ

スタンフォードロースクール法学教授



第1章

GNU プロジェクト

最初のソフトウェア共有コミュニティ

1971年にMIT人工知能研究所(AIラボ)で働き始めたとき、私は長年にわたって存在してきたソフトウェア共有コミュニティの一員になった。ソフトウェアの共有は、別に私たちのコミュニティのみに限定されていたわけではない。レシピの共有が料理と同じだけの歴史を持つと同じように、コンピュータと同じだけの歴史を持っていた。しかし、私たちの共有の進め方は、他のコミュニティ以上だったと言ってよいだろう。

AIラボは、スタッフのハッカーたちが設計し、Digital PDP-10(当時のメジャーなコンピュータの1つ)のアセンブリ言語で書いた、ITS(Incompatible Timesharing System: 互換性のないTSS)と呼ばれるタイムシェアリングオペレーティングシステムを使っていた。このコミュニティの一員として、またAIラボスタッフのシステムハッカーとしての私の仕事は、このシステムを改良することだった。

私たちは自分たちのソフトウェアを「フリーソフトウェア」とは呼んでいなかったが、それはまだその用語が存在していなかったからで、このソフトウェアはまさにフリーソフトウェアだった。ほかの大学や企業の人たちがプログラムを移植、利用することを望めば、私たちは喜んでそれを認めた。誰かが見慣れない面白そうなプログラムを使っていたら、いつでもソースコードを見せてくれと頼むことができた。そうすれば、それを読み、書き換えることができるし、一部を引っこ抜いて新しいプログラムを書くことができる。

「ハッカー」という用語を「セキュリティ破壊者」の意味で使うのは、マスメディアの誤解である。私たちハッカーは、そのような意味付けを拒否し、「プログラムを書くことを愛し、そのための工夫を楽しむ人々」という意味でハッカーという単語を使い続ける*1。

コミュニティの崩壊

1980 年代初めに AI ラボハッカーコミュニティが崩壊し、PDP-10 コンピュータが製造中止になると、状況は一変した。

1981 年に、Symbolics という AI ラボ出身者によって設立された企業が AI ラボのほぼすべてのハッカーを雇い入れ、メンバーを失ったコミュニティは、自らを維持することができなくなった (Steven Levy, "Hackers", Dell, 1985 : 邦訳『ハッカーズ』工学社、1987 年) は、AI ラボコミュニティの最盛時の姿をくっきりと描くとともに、この間の経緯を明らかにしている。AI ラボが 1982 年に新しい PDP-10 を購入したとき、管理者は、ITS ではなく、Digital のフリーではないタイムシェアリングシステムを新マシンに導入することを決めた。

それからまもなく、Digital は、PDP-10 シリーズの製造を中止した。PDP-10 のアーキテクチャは、60 年代にはエレガントで強力だったが、80 年代に実用化されたより広いアドレス空間に合わせて自然に拡張することは不可能だった。つまり、ITS を構成するほぼすべてのプログラムが時代遅れになったということである。ITS にとっては、これが命取りになった。15 年の仕事が水の泡となったのである。

VAX や 68020 などの当時の最新コンピュータは、それぞれのオペレーティングシステムを持っていたが、それらはどれ 1 つとしてフリーソフトウェアではなかつ

*1 ハッキングのように多様な内容を持つものについて単純な定義を与えるのは難しいが、私はほとんどの「ハック」に共通しているものは、遊び、工夫、探求心だと思う。つまり、ハッキングとは、遊び心に満ちた工夫によって可能なことの限界を探求することである。セキュリティ破壊については「クラック」という用語を使い、セキュリティ破壊とハッキングを区別するだけで、誤解を解くための一助になる。セキュリティ破壊を行う人物は「クラッカー」である。クラッカーの一部には、チェスプレイヤーやゴルファーが含まれているのと同じように、ハッカーが含まれているかもしれないが、ほとんどはそうではない ("On Hacking", RMS, 2002)。

た。実行可能コードを入手するだけのためにも、NDA (nondisclosure agreement : 非開示契約) を結ばなければならなかった。

つまり、コンピュータを使うときの第一歩が、隣人を助けないということを約束することだったのである。協力し合うコミュニティは禁止された。私有ソフトウェアの所有者が作った規則は、「隣人と共有したら、あなたは著作権侵害者である。変更を希望するなら、私たちにそうしてくれるよう懇願せよ」というものだった。

私有^{*2}ソフトウェア制度 (ユーザーにソフトウェアの共有や変更を認めない制度) が反社会的で、非倫理的で、単純に間違っているという思想は、一部の読者には驚きかもしれない。しかし、一般の人々を分断し、ユーザーを無援の状態に放置することを基礎とする制度に対して、ほかに何と行うことができるだろうか。私の考え方に驚いた読者は、この私有ソフトウェア制度を当然のものと考えているか、私有ソフトウェア企業が提示している約款を判断基準にしているのだろうか。ソフトウェア企業は、この問題に対する考え方は1つしかないという人々が思い込むように、古くから精力的に活動してきている。

ソフトウェア企業がそれぞれの「権利」を「強制する」とか「海賊版の流通を防止する」と言うとき、彼らが実際に「言っている」ことは副次的なことである。これらの言明の本当のメッセージは、彼らが当然と考えている実際には書かれていない仮定にある。一般の人々は、それを無批判的に受け入れるものと考えられているのである。では、それらの仮定について考えてみよう。

1つ目の仮定は、ソフトウェア企業がソフトウェアを所有する疑問の余地のない自然権を持っており、すべてのユーザーにその権力を行使できるということである (もしこれが自然権なら、公共のためにいかに有害であっても、私たちはそれに反対することはできないところである)。興味深いことに、合衆国憲法と法律への伝統は、この考え方を付けている。つまり、著作権は自然権ではなく、政策的に設けられた人為的な独占権であって、コピーをするというユーザーの自然権

*2 【訳注】言語は proprietary で、辞書には「所有者の」「独占的な」「著作権を持つ」等の訳語が書かれているが、本書では「私有」という訳語を選んだ。それは、GNU の「共有」の思想との対照がもっとも際立つと考えたからである。もっとも、「共有」の原語である public の反対語は private であり、「私有財産」も private property と表現されることが多い。

に制約を課するものなのだ。

もう 1 つの書かれていない仮定は、ソフトウェアで重要なことは、それによってどのような仕事ができるようになるかだけだということである。コンピュータユーザーは、どのような社会を持つことを認められているかということについて考えてはならないものとされている。

第 3 の仮定は、企業にユーザーへの支配権を差し出さなければ、使えるソフトウェア（あるいは、あれこれの特定の仕事をやるプログラム）は作られないというものである。この仮定は、フリーソフトウェア運動が、ユーザーを縛らずに大量の役に立つソフトウェアを提供できることを実証するまで、もっともらしく見えていたかもしれない。

私たちがこれらの仮定を鵜呑みにすることなく、ユーザーを第 1 に考えながら、通常の常識的な倫理に基づいてこれらの問題を判断するなら、まったく異なる結論に達する。コンピュータユーザーは、それぞれのニーズに合わせてプログラムを書き換える自由やソフトウェアを共有する自由を持たなければならない。なぜなら、他人を助けることは、社会の基礎だからである。

倫理的に厳しい選択

コミュニティが消えてから、以前と同じように生きることは不可能になった。私は倫理的に厳しい選択を迫られた。

簡単な選択肢は、NDA に署名し、仲間のハッカーを助けないことを約束して、私有ソフトウェアの世界に入ることだった。その場合、おそらく NDA のもとでリリースされるソフトウェアを開発し、他人にも仲間に対する裏切りを迫ることになるだろう。

こういうやり方でも、稼ぐことはできただろうし、コードを書くことを楽しむこともおそらくできただろう。しかし、生涯を終えようとするときに、人々を分断する壁を築くために過ごしてきた年月を振り返り、世界を悪くするために人生を費やしてきたのかと感じるだろうことはわかっていた。

私はすでに NDA を押し付けられる側になることは経験していた。それは、私と AI ラボがある人物にプリンタ制御プログラムのソースコードの提供を拒否さ

れたときのことである（このプログラムがある機能を持っていないがために、このプリンタを使うのはとてもモライラすることだった）。そのため、私はNDAを罪のないものだとはとても思えなかったのである。私は、彼がソースコードの共有を拒否したときに非常に怒った。逆の立場になって、他のすべての人たちに対して同じことをすることは不可能だった。

簡単だが面白くない選択肢として、コンピュータの世界から離れるというものもあった。そうすれば、私の技能が悪用されることはないが、無駄になることに変わりはない。私自身はコンピュータユーザーを分断し、制約を押し付ける罪を免れることができるが、同じ問題は間違いなく発生する。

そこで、私は、プログラマが善に貢献するための道を探した。コミュニティを復活させるために書けるプログラムはないか、自問自答した。

答えははっきりしていた。まず最初に必要なものは、オペレーティングシステムだった。オペレーティングシステムがあればさまざまなことができるが、なければコンピュータを動かすことさえままならない。フリーオペレーティングシステムがあれば、協力し合うハッカーたちのコミュニティを再び築くことができるだろう。コミュニティに加わるように誘うこともできるだろう。そして、まず友達をなくすことを強いられずに、誰もがコンピュータを自由に使えるようになるはずだ。

私は、オペレーティングシステム開発者として、この仕事をするための適切な技能を持っていた。成功を当然のこととして見込むことはできないものの、私は自分がこの仕事をするために選ばれた者だと確信した。私は、Unix 互換システムを作ることにした。そうすれば、移植性を確保できるし、Unix ユーザーが簡単に移って来れる。GNU という名前は、ハッカーの伝統に従い、“GNU’s Not Unix” という再帰的な（入れ子状の）頭字語として選ばれたものである。

オペレーティングシステムとは、単なるカーネルのことではない。他のプログラムを動かせれば充分というわけではないのだ。1970年代、オペレーティングシステムの名に値するシステムは、コマンドプロセッサ、アセンブラ、コンパイラ、インタープリタ、デバッガ、テキストエディタ、メーラなどのプログラムを含んでいた。ITS 然り、Multics 然り、VMS 然り、Unix 然り。GNU オペレーティングシステムも、それらを持つものにしよう。

その後、私はヒレル*3のものだとされる次の言葉を聞いた。

「私が自分のために動かなければ、誰が私のために動いてくれるというのだ。私が私だけのために動くのだとすれば、私は一体何者なのだ。今やらなければ、いつやる？」

GNU プロジェクトを立ち上げることにしたのも、同じような考え方からだった。

私は無神論者なので、宗教界の指導者には従わないが、彼らの中の誰かが言ったことを尊敬することはある。

自由という意味でのフリー

「フリーソフトウェア」という用語は、誤解されることがある。価格とは無関係だ。フリーとは自由のことである。そこで、フリーソフトウェアの定義を下しておくことにしよう。以下の条件を満たすプログラムは、そのユーザーにとって、フリーソフトウェアである。

ユーザーが任意の目的のためにプログラムを実行する自由を持っている。

ユーザーが自らのニーズに合わせてプログラムを書き換える自由を持っている（この自由を現実的に有効なものにするためには、ソースコードにアクセスできなければならない。なぜなら、ソースコードを持たないプログラムに変更を加えるのは、極度に難しいことだからである）。

ユーザーが無料、あるいは有料でコピーを再頒布する自由を持っている。

ユーザーがプログラムの変更バージョンを頒布する自由を持っており、コミュニティがそのユーザーによる改良から利益を受け取ることができる。

「フリー」が価格のことではなく、自由であることを意味しているので、コピーを売ることとフリーソフトウェアであることの間には何らの矛盾もない。実際、コピーを販売する自由は、非常に重要である。CD-ROM 化されたフリーソフト

*3 【訳注】キリストと同時代のユダヤ教ラビ。

ウェアのコレクションはコミュニティにとって重要であり、それを販売することは、フリーソフトウェアの開発資金を稼ぐための重要な手段である。そのため、このようなコレクションに自由に組み込めないプログラムは、フリーソフトウェアではない。

「フリー」という用語は曖昧なので、人々はずっと代わりになる言葉を探してきたが、適切なものを見つけた人はまだいない。英語は他の言語よりも多くの単語とニュアンスを持っているが、自由としての「フリー」を意味する単純で曖昧さのない単語を持っていない。もっとも近い意味を持つ単語は、「unfettered」である。「liberated」、「freedom」、「open」などの言葉は、誤った意味を兼ね備えていたり、他の欠点を持っていたりする。

GNUソフトウェアとGNUシステム

システム全体を開発するのは、非常に大規模なプロジェクトである。目標を手が届くところに置くために、私は可能な限り既存のフリーソフトウェアを利用、修正することにした。たとえば、私はごく初期の段階で、テキストフォーマットとして主に $\text{T}_\text{E}\text{X}$ を使うことにした。数年後、GNU のために新しいウィンドウシステムを書く代わりに、X Window System を使うことにした。

このような決定のために、GNUシステムとすべてのGNUソフトウェアのコレクションは同じではない。GNUシステムには、GNUソフトウェアではないプログラム、他の人々や他のプロジェクトがそれぞれの目的のために開発したプログラムが含まれている。しかし、それらはフリーソフトウェアなので、私たちはそれを使うことができるのだ。

プロジェクトの立ち上げ

私は、1984年1月にMITの仕事を辞め、GNUソフトウェアの開発を開始した。MITがフリーソフトウェアとしてのGNUの頒布に干渉できないようにするために、MITを辞めることは必要だった。私がスタッフとして残っていたら、MITは私の仕事の所有権を主張したり、独自の頒布条件を課したり、私の仕事を私有ソ

ソフトウェアパッケージに化けさせたりすることさえできていただろう。新しいソフトウェアコミュニティの創出という当初の目的にまったく添わない結果を招くために、大仕事をするつもりはなかった。

もっとも、ウィンストン教授（当時の MIT AI ラボ所長）は、ラボの設備をそのまま使えるようにはからってくれた。

第一歩

GNU プロジェクトを立ち上げる直前に、私は Free University Compiler Kit 別名 VUCK（オランダ語の「フリー」は、V から始まる）というものの話を聞いた。これは、C や Pascal など、複数の言語を処理し、複数のターゲットマシンをサポートするコンパイラだった。私は GNU でそれを使えるかどうかを尋ねるために、作者に手紙を書いた。

返事は嘲笑的なものだった。大学はフリー（自由）だが、コンパイラは違う（フリーコンパイラではない）というのである。そこで、GNU プロジェクトの最初のプログラムは、マルチ言語マルチプラットフォームコンパイラにすることにした。

コンパイラ全体を自分で書かなくても済むとよいと思い、私はローレンスリバーモア研究所で開発されたマルチプラットフォームコンパイラ、Pastel のソースコードを入手した。Pastel は、システムプログラミング言語になるように Pascal を拡張した言語で書かれており、その拡張 Pascal をサポートしていた。私は、C フロントエンドを追加し、Motorola 68000 コンピュータへの移植作業を開始した。しかし、このコンパイラが何 M バイトものスタック領域を必要とするのに対し、68000 の Unix システムが 64K バイトのスタック領域しか認めていないことを知り、移植を諦めなければならなかった。

その後、Pastel コンパイラは、入力ファイル全体を走査して構文木を構築し、構文木全体を「命令」のチェーンに変換し、出力ファイル全体を生成しており、その間、メモリを一切解放していないことがわかった。これがわかった時点で、私は 0 から新しいコンパイラを書かなければならないという結論に達した。その新コンパイラは、今、GCC と呼ばれているものである。Pastel コンパイラのコードは一切使われていないが、自分で書いた C フロントエンドは何とか修正して活用

した。しかし、その仕事に取り掛かったのは、数年後のことである。まず、私は GNU Emacs を書いた。

GNU Emacs

私は 1984 年 9 月に GNU Emacs の仕事を始めた。そして、1985 年始めには、使えるものになり始めていた。これで、私は Unix システムを使って編集の仕事ができるようになった。私は、vi や ed の使い方を学ぶ気にはなれなかったので、そのときまでは Unix 以外のマシンで編集をしていたのである。

この頃から、人々は GNU Emacs を使いたがるようになったが、そのことは、ソフトウェアをどのように流通させるかという問題を引き起こした。もちろん、自分が使っていた MIT のコンピュータの anonymous FTP サーバから入手できるようにはした（そのため、このコンピュータ、prep.ai.mit.edu は、GNU のメイン ftp サイトになった。数年後、このマシンが廃棄処分になったときには、新しい ftp サーバに同じ名前を移した）。しかし、その当時、GNU に関心を持つ人々の多くは Internet に接続しておらず、ftp ではコピーを入手できなかった。そこで、そのような人々にどう言ったらよいか問題になった。

「Internet に接続できてコピーを作れる友達を探してください」と言ってもよかつたし、オリジナルの PDP-10 Emacs のときと同じように「私にテープと SASE（切手を貼った返信用封筒）を郵送していただければ、Emacs をコピーして送り返します」と言ってもよかつた。しかし、私は無職だったし、フリーソフトウェアで稼ぐ方法を探していた。そこで、私は、150 ドルで希望者全員にテープを郵送すると発表した。このようにして私はフリーソフトウェア流通ビジネスを始め、現在、Linux ベースの GNU システムを販売している企業の先駆者になったのである。

すべてのユーザーに対してフリーなソフトウェア

作者の手を離れたときにフリーソフトウェアであったとしても、そのプログラムは、コピーを持っているすべての人々にとってフリーソフトウェアであるとは限らない。たとえば、PDS（パブリックドメインソフトウェア、すなわち著作権が放棄されているソフトウェア）はフリーソフトウェアだが、誰もがそれに変更を加えた私有バージョンを作ることができる。同様に、多くのフリープログラムは、著作権を放棄してはいないものの、変更が加えられた私有バージョンを認めるような単純で寛大なライセンスのもとで流通されている。

この問題を非常によく示している例が X Window System である。MIT で開発され、寛大なライセンスのもとでフリーソフトウェアとしてリリースされた X Window は、すぐにさまざまなコンピュータメーカーに採用された。これらの企業は、私有している Unix システムにバイナリ形式のみで X を追加し、同じ NDA を適用した。これらの X は、Unix と同様にフリーソフトウェアではなくなってしまったのである。

X Window System の開発者たちは、これを問題だとは考えなかった。むしろ、こうなることを予想し、意図していたのである。彼らの目標は自由ではなく、「多くのユーザーの獲得」と定義される「成功」だった。彼らは、ユーザーが自由を持つかどうかには注意を払わなかった。ユーザーを増やすことしか考えていなかったのである。

これは、ずいぶん逆説的な話である。自由の量を計る方法が 2 通りあり、「このプログラムはフリーか」という問いに対してそれらが異なる答えを出しているのである。MIT リリースの頒布条件が与える自由に基づいて判断するなら、X はフリーソフトウェアだったと言える。しかし、X の平均的なユーザーの自由度を計るなら、X は私有ソフトウェアだと言わなければならない。ほとんどの X ユーザーは、フリーバージョンではなく、Unix システムに付属している私有バージョンを実行していたのである。

コピーレフトとGNU GPL

GNU の目標は、単にポピュラーになることではなく、ユーザーに自由を与えることだった。そこで、私たちは、GNU ソフトウェアが私有ソフトウェアに化けるのを防ぐ頒布条件を必要としていた。そして、私たちが今使っている方法は、**コピーレフト** (copyleft) というものである。

コピーレフトは著作権 (copyright) 法を利用しているが、通常とは逆の目的に奉仕するように、逆立ちさせた使い方をしている。著作権法は、ソフトウェアを私有化するための手段ではなく、ソフトウェアの自由を守る手段になったのである。

コピーレフトという考え方の中心は、プログラムを実行し、プログラムをコピーし、プログラムを書き換え、書き換えられたバージョンを流通させることをすべての人に認める一方で、独自の制限事項の追加を禁止することにある。つまり、「フリーソフトウェア」がよって立つ由縁であるところの自由が、コピーを持つすべての人に対して保証される。自由は、絶対に奪うことのできない権利になったのである。

コピーレフトの効力を保つためには、書き換えられたバージョンもフリーでなければならない。こうすれば、私たちのコミュニティは、私たちの仕事を基礎とする仕事が公表されたときに、必ずその仕事を利用できる。プログラマとしての仕事を持つプログラマがボランティアで GNU ソフトウェアを改良したときに、その雇い主が「君が加えた変更は、プログラムの私有バージョンを作るためにうちの会社で使うつもりだから、共有に付するわけにはいかない」などと言えないようにするのがコピーレフトである。

プログラムのすべてのユーザーに対して自由を保証したければ、変更点もフリーにするよう要求することは非常に重要である。X Window System を私有化した企業は、通常、それぞれのシステムやハードウェアに X Window System を移植するために、何らかの変更を加えている。これらの変更点は、X の規模の大きさから比べればごく少量だが、ごくわずかというわけでもない。変更を加えたことが、ユーザーの自由を否定する言い訳になるのであれば、誰もがその言い訳を利用するようになるだろう。

フリープログラムとフリーではないコードを結合するときにも、同様の問題が発

生ずる。そのような結合は、確実にフリーではないコードを作るだろう。フリーではない部分で欠けている自由は、全体として欠けている自由でもある。そのような結合を認めれば、船を沈めるに足る穴を開けることになるだろう。コピーレフトは、絶対にこの穴を塞ぐ必要がある。コピーレフトの対象プログラムに何かを追加、結合した場合、結合後の大きなバージョンもまた、フリーでコピーレフトの対象になっていなければならない。

私たちがほとんどの GNU ソフトウェアのために使っているコピーレフトの具体的な条文は、GNU 一般公開使用許諾書 (GNU GPL) である。私たちは、特定の状況で使われる別の種類のコピーレフトも持っている。たとえば、GNU マニュアルもコピーレフトの対象だが、マニュアルには GNU GPL の複雑さは不要なので、ずっと単純な種類のコピーレフトを使っている。

1984 年か 85 年のことだが、ドン・ホプキンス (非常に想像力豊かな人物) が私に手紙を送ってきた。その封筒には、いくつかの面白いことが書かれていたが、「Copyleft--all rights reversed.」(コピーレフト 全ての権利が逆になっている) というのもその 1 つだった。私は、このコピーレフトという言葉を当時構想していた流通概念の名前として使うことにした。

フリーソフトウェア財団

Emacs を使うことに対する関心が高まるにつれて、GNU プロジェクトには、私以外の人々が関わるようになってきた。そこで、私たちは再び資金集めを追求すべきだと考え、1985 年にフリーソフトウェア開発のための非課税慈善団体として、フリーソフトウェア財団 (FSF) を設立した。FSF は、Emacs のテープ頒布ビジネスも引き継ぎ、その後、テープに他のフリーソフトウェア (GNU および非 GNU) を追加し、フリーマニュアルの販売も開始して、事業を拡大した。

FSF は寄付も受け付けているが、その収入の大半は、フリーソフトウェアのコピーや他の関連サービスの販売から得ている。現在、FSF はソースコードの CD-ROM、バイナリを収めた CD-ROM、美しく印刷されたマニュアル (どれも、再頒布、変更の自由を伴う) およびデラックスディストリビューション (顧客が選んだプラットフォームのためにソフトウェアの全コレクションを構築する) を

販売している。

FSF のスタッフは、たくさんの GNU ソフトウェアパッケージを開発し、保守してきた。特筆すべきは、C ライブラリとシェルである。GNU C ライブラリは、GNU/Linux システムで動作するすべてのプログラムが Linux とやり取りするために使っているものである。このライブラリは、FSF のスタッフ、ローランド・マクグラスによって開発された。ほとんどの GNU/Linux システムで使われているシェルは BASH (Bourne Again Shell) だが、これは FSF のスタッフであるブライアン・フォックスによって開発された。

私たちがこれらのプログラムの開発に資金提供したのは、GNU プロジェクトが単なるツールや開発環境ではないからである。私たちの目標は、完全なオペレーティングシステムを作ることであり、これらのプログラムはその目標のために必要だったのだ。

Bourne Again Shell は、Unix で通常に使われている Bourne Shell というシェルの名前を借りたジョークである。

フリーソフトウェアのサポート

フリーソフトウェアの哲学は、広く普及しているビジネスのある特定の形態に反対するが、ビジネスそのものに反対するわけではない。ビジネスがユーザーの自由を尊重するとき、私たちはそのビジネスの成功を望む。

Emacs のコピーの販売は、フリーソフトウェアビジネスの 1 つの形態を実証するものである。FSF がその事業を引き継いだとき、私は生計を立てるための別の手段を見つけなければならなくなった。そして、私が開発したフリーソフトウェアに関連したサービスの販売を思いついた。たとえば、GNU Emacs のプログラム方法や GCC のカスタマイズ方法といったテーマについて教えることやソフトウェア開発である。ちなみに、ソフトウェア開発のほとんどは GCC の新プラットフォームへの移植だった。

今日、これらのフリーソフトウェアビジネスは、数社によって展開されている。CD-ROM でフリーソフトウェアコレクションを販売する企業、ユーザーの質問への回答から、バグフィックス、大きな新機能の開発に至るまでのサポートサー

ビスを販売する企業がある。新しいフリーソフトウェア製品の開発を事業の軸に据えるフリーソフトウェア企業さえ生まれ始めている。

しかし、注意していただきたい。「オープンソース」という言葉を掲げているいくつかの企業は、実際には、フリーソフトウェアと併用される非フリーソフトウェアに事業の基礎を置いている。これらは、フリーソフトウェア会社ではなく、ユーザーを自由から引き離そうとする製品を作っている私有ソフトウェア会社である。彼らはそれを「付加価値」と称するが、それは彼らが私たちに受け入れさせようとする価値にほかならない。つまり、自由よりも便宜性を高く評価する価値観である。自由により高い価値を認めるなら、そのようなものは「自由を抜き取った」製品と呼ぶべきだろう。

技術的な目標

GNU の第 1 の目標は、フリーソフトウェアになることだった。GNU が Unix と比べて技術的に優れていなかったとしても、ユーザーの協力を認めるという社会的なメリット、ユーザーの自由を尊重するという倫理的なメリットはある。

しかし、標準として認められている優れた作業習慣に従うのは当然のことである。たとえば、適当にサイズの上限を決めるのではなく、動的にデータ構造を確保するとか、意味がある場合には、すべての 8 ビットコードを処理するといったことである。

さらに、私たちは 16 ビットマシンをサポートしないことにして (GNU システムが完成する頃には、32 ビットマシンが標準になることは明らかだったので)、Unix の小さなメモリサイズに対するこだわりを捨てた。また、1M バイトを超えない限り、メモリの利用を抑制しようとはしなかった。非常に大きなファイルを処理することがそれほど重視されないプログラムでは、入力ファイル全体をメモリに読み込み、入出力の心配をせずにその内容を走査することをプログラマたちに勧めた。

これらの決定により、多くの GNU プログラムは、信頼性とスピードの面で Unix の同等のプログラムよりも優れたものになった。

寄付されたコンピュータが抱えていた問題

GNU プロジェクトの評価が上がってくるにつれて、プロジェクトには Unix マシンが寄付されるようになってきた。GNU コンポーネントのもっとも簡単な開発方法は、Unix 上で Unix のコンポーネントを1つずつ取り替えていくことだったので、これは非常に役に立った。しかし、このことはある倫理的な問題を引き起こした。それは、私たちが Unix のコピーを持つことは正しいかどうかということである。

Unix は私有ソフトウェアだった（今もそうである）が、GNU プロジェクトの哲学は、私有ソフトウェアを使わないとしていた。しかし、自己防衛のための暴力が正当化されると同じ理由から、他人が私有パッケージを使うのをやめるのを助けるフリーバージョンを開発するために必要不可欠なのであれば、私有パッケージを使うことは正当化されるという結論を下した。

しかし、必要悪であろうが、悪は悪である。現在はフリーオペレーティングシステムで置き換えたので、私たちは Unix のコピーを持っていない。あるマシンのオペレーティングシステムをフリーオペレーティングシステムに置き換えられない場合、私たちはマシンのほうを取り替える。

GNU タスクリスト

GNU プロジェクトが前進し、見つかった、あるいは開発が終わったシステムコンポーネントの数が増えてくると、残された穴のリストを作ると役に立つだろうということになった。私たちは、そのリストを使って、足りない部分を開発してくれるプログラマを募った。このリストは、GNU タスクリストと呼ばれるようになった。私たちは、足りない Unix コンポーネントのほか、本当に完全なシステムが備えるべき役に立つソフトウェアとドキュメントをリストに加えた。

現在、GNU タスクリストに残っている Unix コンポーネントはほとんどない。あまり重要ではない一部を除き、それらの仕事は終わった。しかし、リストには、人によっては「アプリケーション」と呼ぶようなプロジェクトが満載されている。ごく狭い範囲のユーザー以外にも魅力的なプログラムは、オペレーティングシ

テムに追加すべき有益なプログラムである。

タスクリストには、ゲームさえ含まれている。それはリストを作った当初からのことである。Unix にはゲームが含まれているので、当然ながら GNU もそれらを含まなければならない。しかし、ゲームの場合、互換性が問題になることはないので、Unix ゲームのリストに忠実に従っているわけではない。私たちのリストには、ユーザーが好みそうな別のゲームセットが含まれている。

GNU LGPL

GNU C ライブラリは、フリーライブラリに対する私有ソフトウェアのリンクを許可する GNU ライブラリ一般公開使用許諾書 (LGPL) という特別な種類のコピーレフトを使っている。このような例外を設けたのはなぜなのだろうか。

これは、原則の問題ではない。私有ソフトウェア製品が私たちのコードを組み込む権利を認める原則など存在しない (私たちとの共有を拒否することがわかっているプロジェクトに貢献する理由があるのか)。C ライブラリやその他のライブラリで LGPL を使っているのは、戦略の問題である。

C ライブラリは、一般的な仕事をする。すべての私有システムやコンパイラには、C ライブラリが付属している。そのため、フリーソフトウェアでなければ私たちの C ライブラリを使えないようにしても、フリーソフトウェアには何の利益もない。単に、私たちのライブラリを使う気になれないものにするだけである。

これには例外となるシステムが 1 つある。GNU システム (これには GNU/Linux も含まれる) では、GNU C ライブラリが唯一の C ライブラリである。そのため、GNU システムで私有プログラムをコンパイルできるかどうかは、GNU C ライブラリの頒布条件によって決まる。GNU システムで私有アプリケーションの実行を認める倫理的な理由はないが、戦略的に考えて、それを禁止すると、フリーアプリケーションの開発を奨励するよりも、GNU システムを使いたくない気分を助長することになる。

C ライブラリでは LGPL を使うことが戦略的に優れているというのは、そういうことである。他のライブラリについての戦略は、ケースバイケースで検討しなければならない。ある種のプログラムを書きやすくするような特別な仕事をこな

すライブラリは、GPL のもとでリリースし、使用をフリープログラムのみに制限すれば、他のフリーソフトウェア開発者を助け、私有ソフトウェアに対する優位性を与えることになる。

たとえば、BASH のコマンドライン編集機能のために開発された GNU Readline ライブラリ*4 について考えてみよう。Readline は、LGPL ではなく、通常の GNU GPL のもとでリリースされている。おそらく、このことによって Readline の使用数は減るだろうが、私たちにとって、それはまったく損失ではない。その一方で、Readline を使うために、少なくとも 1 つの役に立つアプリケーションがフリーソフトウェアとして作られた。これこそ、コミュニティにとって本物の利益である。

私有ソフトウェアの開発者は金銭的な面で優位に立つが、フリーソフトウェアの開発者は相互のために優位性を生まなければならない。いずれ、私有ソフトウェアに対応するものがない GPL によって保護されるライブラリの大規模なコレクションを作りたいものだ。これらのライブラリは、新しいフリーソフトウェアの部品として使える役に立つモジュールを提供し、フリーソフトウェアの更なる開発を促進する大きな力になるだろう。

痒いところを掻く？

エリック・レイモンドは、「ソフトウェアの優れた仕事は、どれもプログラマが自分で痒いと思ったところを掻くところから始まる」と言っている。そういうこともあるかもしれないが、GNU ソフトウェアの多くの主要要素は、完全なフリーオペレーティングシステムを作るために開発されている。これらは、衝動ではなく、ビジョンとプランによって作られている。

たとえば、私たちが GNU C ライブラリを開発したのは、Unix 風のシステムでは C ライブラリが必要だからであり、BASH を開発したのは、Unix 風のシステムではシェルが必要だからである。GNU tar を開発したのは、Unix 風システムが tar プログラムを必要とするからである。同じことが私自身のプログラム、GNU C コンパイラ、GNU Emacs、GDB、GNU Make にも当てはまる。

*4 GNU Readline ライブラリは、入力したコマンドラインを編集できるようにしたいアプリケーションのための関数群を提供する。

一部の GNU プログラムは、私たちの自由に対する個別の脅威に対処するために開発された。私たちが compress プログラムの代わりに gzip を開発したのは、LZW の特許^{*5}のために compress がコミュニティから失われたためである。私たちは LessTif の開発者を見つけ、さらに最近では GNOME と Harmony をスタートさせたが、それも特定の私有ライブラリに起因する問題に対処するためである（後述の「非フリーライブラリ」を参照）。また、ユーザーがプライバシーと自由のどちらかを選択しなければならないようなことのないよう、ポピュラーな非フリーの暗号ソフトウェアに代わる GNU Privacy Guard を開発している。

もちろん、これらのプログラムを書いている人々は、仕事に興味を持つようになったし、多くの人々がそれぞれのニーズや関心に基づいてプログラムにさまざまな機能を追加している。しかし、興味はプログラムが存在する理由ではないのである。

予想外の開発作業

GNU プロジェクトを立ち上げた頃、私は GNU システム全体を開発し、全体としての形でリリースすることを想像していたが、実際にはそのようにはならなかった。

GNU システムの個々のコンポーネントは Unix システムで実装されたので、完全な GNU システムが完成するはるか以前から、それらのコンポーネントは Unix システムでも実行できた。これらのプログラムの一部はポピュラーになり、ユーザーはそれらの拡張や移植を始めた。移植先は、Unix のさまざまな互換性のないバージョンだったり、まったく別のシステムだったりした。

その過程でこれらのプログラムははるかに強力になり、GNU プロジェクトに資金や貢献者を集めることになった。しかし、GNU 開発者の時間は、足りないコンポーネントを次々に書いていくことではなく、これらの移植版の維持や既存コンポーネントへの機能の追加に割かれていったので、おそらくこの動きは最小限の稼動するシステムの完成を数年遅らせる効果も持っていたはずである。

*5 データの圧縮のために Lempel-Ziv-Welch アルゴリズムが使われている。

GNU Hurd

1990 年までに、GNU システムはほとんど完成していた。主要コンポーネントで欠けていたものは、カーネルだけだった。私たちは、Mach の上で動作するサーバプロセスのコレクションという形でカーネルを実装することにしていた。Mach は、カーネギーメロン大学、ついでユタ大学で開発されたマイクロカーネルである。GNU Hurd は、Mach の上で動作するサーバコレクション（「GNU の群れ」）であり、Unix カーネルのさまざまな仕事を行う。開発の開始は遅れたが、それは約束通りに Mach がフリーソフトウェアとしてリリースされるのを待っていたからである。

このような設計を選んだ理由の 1 つは、仕事の中でもっとも難しいと思われた部分を避けるためだった。それは、ソースレベルデバッグなしでカーネルプログラムをデバッグすることである。この部分の仕事は Mach の中ですでに終わっているはずなので、Hurd サーバはユーザープログラムとして GDB でデバッグできると思っていた。しかし、それが可能になるまでには非常に長い時間がかかった。また、互いにメッセージを送り合うマルチスレッドサーバは、デバッグが非常に難しいことがわかった。Hurd を安定動作させる仕事は、何年も遅れている。

Alix

GNU カーネルは、もともと Hurd という名前になるはずではなかった。最初の名前は、当時の私の恋人だった女性の名前を取った Alix だった。彼女は Unix のシステム管理者で、自分の名前が Unix システムの命名パターンに当てはまっていると saying。彼女は、「誰かがきつと私と同じ名前のカーネルを作るはずよ」と友人に冗談を言った。私は何も言わなかったが、Alix という名前のカーネルを作って彼女を驚かせることに心を決めていた。

しかし、事態はそのようには展開しなかった。カーネルのメインプログラマのマイケル・ブッシュネル（現在はトーマス）は、Hurd という名前を好み、Alix はカーネルの特定の部分（システムコールをトラップし、Hurd サーバにメッセージを送ってそれを処理する）を指すものとして再定義された。

最終的に、Alix と私は別れ、彼女は名前を変えた。それとは無関係に、Hurd の設計は、C ライブラリがサーバに直接メッセージを送るように変更された。そのため、Alix コンポーネントは設計から消えてしまった。

しかし、これらのことが起きる前に、彼女の友達の 1 人が Hurd のソースコードに Alix という名前があることに気づき、彼女にそのことを教えた。だから、Alix という名前は機能を果たしたのである。

Linux と GNU/Linux

GNU Hurd は、まだ稼働できる状態にはなっていないが、幸いにも別のカーネルが現れた。リーナス・トーバルズは、1991 年に Unix 互換カーネルを開発し、Linux と名付けた。そして、1992 年頃、Linux とまだ完全には完成していない GNU システムを結合することによって、完全なフリーオペレーティングシステムが完成した（もちろん、両者の結合は、それ自体大きな仕事だった）。現在、実際に GNU システムの 1 バージョンを実行できるのは、Linuxのおかげである。

私たちは、GNU システムとカーネルとしての Linux の結合という構成を表現するために、このバージョンのシステムを GNU/Linux と呼んでいる。

将来の試練

私たちは、広い範囲のフリーソフトウェアを開発できるという自らの能力を証明した。だからといって、私たちが無敵で安泰だというわけではない。フリーソフトウェアの将来を不確実なものにするいくつかの試練が待ち構えている。それらの試練に遭ったときには、不屈の努力と忍耐が必要だろう。場合によっては、それが数年続くこともある。人々が自由を尊重し、自由を奪われることを拒否するときに示す決意が必要とされる。

以下の 4 つの節では、これらの試練について論じる。

非開示主義のハードウェア

ハードウェアメーカーは、次第にハードウェアの仕様を非開示にする傾向にある。これは、フリーのドライバを書いて Linux や XFree86^{*6}が新ハードウェアをサポートできるようにすることを困難にする。今日は、完全なフリーシステムがあるが、明日のコンピュータをサポートできなければ、明日にはフリーシステムがなくなるかもしれない。

この問題への対処方法は2つある。プログラマは、リバースエンジニアリングによって、ハードウェアのサポート方法を突き止めることができる。プログラマ以外の人々は、フリーソフトウェアがサポートしているハードウェアを選ぶことができる。フリーシステムが増えれば、仕様の非開示主義は自分の首を絞める結果になるだろう。

リバースエンジニアリングは大仕事である。そのような仕事に着手するだけの意志を持ったプログラマを集めることができるだろうか？ イエス。フリーソフトウェアは主義主張の問題で、非フリードライバは許容できないという強い意志を築き上げることができれば可能である。そして、フリードライバを使えるようにするために、プログラマ以外の人々は余分なお金や余分な時間を使ってくれるだろうか？ 自由を確保するという意志が浸透すれば、それに対する答えもイエスである。

非フリーライブラリ

フリーオペレーティングシステム上で動作する非フリーライブラリは、フリーソフトウェア開発者を陥れる罠である。エサは魅力的な機能だが、そのライブラリを使ってしまうと、プログラムはフリーオペレーティングシステムの一部ではあり得なくなってしまうので、プログラムは罠に落ちたということになる（厳密に言えば、そのプログラムをフリーオペレーティングシステムの一部にすることはできるが、問題のライブラリがなければそのプログラムは動作しないだろう）。

*6 XFree86 は、表示ハードウェア（マウス、キーボードなど）とのインターフェイスとなるデスクトップ環境を提供するプログラムである。多くの異なるプラットフォームで動作する。

さらに悪いことに、私有ライブラリを使っているプログラムがポピュラーになったら、疑いを知らない他のプログラマたちも罠に陥れることになる場合がある。

この問題の最初の事例は、80年代の Motif^{*7} ツールキットだった。当時はまだ、フリーオペレーティングシステムはなかったが、Motif がその後どのような問題を起こすかは明らかだった。GNU プロジェクトは、Motif に 2 通りの反応を返した。個々のフリーソフトウェアプロジェクトに Motif だけではなく、フリー X ツールキットもサポートするように呼びかけることと、Motif に代わるフリーシステムを書く人を募ることである。この仕事には長い時間がかかった。Hungry Programmers によって開発された LessTif がほとんどの Motif アプリケーションをサポートできるだけの力を付けたのは、1997 年のことである。

1996 年から 1998 年にかけて、Qt という別の非フリー GUI (グラフィカルユーザーインターフェイス) ツールキットが、大規模なフリーソフトウェアコレクションの KDE デスクトップ環境で使われていた。

フリー GNU/Linux システムは、そのライブラリを使うわけにはいかなかった。KDE を使うことができなかった。しかし、フリーソフトウェアに対する厳密なこだわりのない一部の市販 GNU/Linux ディストリビューションがそれぞれのシステムに KDE を追加した。機能は強化されたが自由が弱体化されたシステムを作ったのである。KDE グループは、多くのプログラマに対して Qt を使うように積極的に働きかけており、数百万の新しい「Linux ユーザー」は、そこに問題が潜んでいるということを教えられていなかった。この問題は深刻に感じられた。

フリーソフトウェアコミュニティは、GNOME と Harmony という 2 通りの方法でこの問題に対処した。

GNOME (GNU Network Object Model Environment) は、GNU のデスクトップ環境プロジェクトである。1997 年にミゲル・デ・イカーザが開発に着手し、Red Hat Software の支援のもとで開発された GNOME は、KDE と同様のデスクトップ機能を提供するが、フリーソフトウェアだけを使っているところが異なっていた。C++だけではなく、さまざまな言語をサポートするなど、技術的な長所も持っていた。しかし、GNOME の最大の目的は、自由 すなわち非フリーのソフト

*7 Motif は、X Window の上で動作するグラフィカルユーザーインターフェイスおよびウィンドウマネージャである。

ウェアを使うことを要求しないことである。

Harmony は、Qt を使わなくても KDE ソフトウェアを実行できるようにするために設計された、互換性のある代替ライブラリである。

1998 年 11 月、Qt の開発者たちは、実際に遂行されれば Qt をフリーソフトウェアにするはずのライセンス契約の変更を発表した。確かなことはわからないが、このような方針変更がなされた理由の一部は、Qt がフリーではなかった時代に起こした問題に対するコミュニティの断固とした反応にあったと思う（ただし、新ライセンスは不便で不公平なものなので、まだ Qt の使用は避けることが望ましい）*8。

次の魅力的な非フリーライブラリに対しては、どのように対応すべきだろうか。コミュニティ全体が畏から身を遠ざけることの必要性を理解してくれるだろうか。それとも、多くのメンバーが利便性のために自由を放棄し、大きな問題を引き起こしてしまうのだろうか。私たちの将来は、私たちの哲学にかかっている。

ソフトウェア特許

私たちが直面している最悪の脅威は、ソフトウェア特許である。ソフトウェア特許は、20 年もの間、フリーソフトウェアがアルゴリズムや機能を利用できないようにすることができる。LZW 圧縮アルゴリズム特許は 1983 年に発効となり、私たちはまだ正しく圧縮された GIF を作るフリーソフトウェアをリリースすることができないでいる。1998 年には、特許がらみの訴訟が現実的な脅威となったので、ディストリビューションから MP3 圧縮オーディオを作成するフリープログラムが取り除かれた。

特許には対処方法がある。特許が無効である証拠を探すこともできるし、同じ仕事をする別の方法を探することもできる。しかし、これらの方法が機能するのは限られたときだけであり、両方とも失敗したときには、特許のためにすべてのフリーソフトウェアがユーザーの望む機能を失うことを余儀なくされる場合がある。このようなときにはどうしたらよいのだろうか。

*8 2000 年 9 月、Qt は GNU GPL のもとで再リリースされ、この問題は全面的に解決された。

自由のためにフリーソフトウェアを評価する私たちは、いずれにしてもフリーソフトウェアの側に立つだろう。私たちは、特許の対象となっている機能なしで仕事ができるように工夫する。しかし、フリーソフトウェアに技術的な優位性を期待するユーザーは、特許によって優位性が失われたらそれを欠陥と呼ぶだろう。開発の「伽藍」*9 モデルの実践的な有効性や、一部のフリーソフトウェアの信頼性や能力について語ることは意味のあることだが、私たちはそこに留まっているわけにはいかない。私たちは、自由と原則について語らなければならないのである。

フリードキュメント

私たちのフリーオペレーティングシステムの最大の弱点は、ソフトウェアの中にはない。それは、システムに組み込める優れたフリーマニュアルがないことである。ドキュメントは、あらゆるソフトウェアパッケージに必要な不可欠な部分である。重要なフリーソフトウェアパッケージに優れたフリーマニュアルが含まれていなければ、それは大きな欠陥と言わなければならない。現在の私たちはそのような穴を無数に抱え込んでいる。

フリードキュメントは、フリーソフトウェアと同様に、価格の問題ではなく、自由の問題である。フリーマニュアルの基準は、フリーソフトウェアとほぼ同じところにある。重要なのは、すべてのユーザーに自由を与えるかどうかだ。プログラムのすべてのコピーにマニュアルを同梱できるようにするために、オンラインと紙の両方の形態で再頒布（商業目的の販売を含む）が認められていなければならない。

変更に対する許可も重要である。一般原則として、私はあらゆる種類の論文や本を変更する権利が認められることが重要だとは考えていない。たとえば、このような論文に変更を加えることを許可する必要があるとは思わない。この論文には、私たちの活動と思想が書かれているのである。

しかし、フリーソフトウェアのドキュメントについては、書き換えの自由が重要な意味を持つ理由がある。人々がソフトウェアを書き換える権利を行使し、機

*9 おそらく、私は「バザール」モデルについて書こうとしていたのだろう。「バザール」モデルは、その頃としては新しく、最初は論争的となっていた代替モデルである。

能を追加、変更したとき、彼らが仕事に忠実なら、マニュアルも書き換えるだろう。それにより、変更後のプログラムに合った正確で使えるドキュメントを提供できるようになるわけである。プログラマが職務を忠実に遂行し、仕事を完成させることを認めないマニュアルは、私たちのコミュニティのニーズを満たさない。

ここで変更方法に何らかの歯止めをかけても、問題は起きないだろう。たとえば、オリジナルの著者の著作権情報、頒布条件、著者リストを書き換えないことを要求することはかまわない。変更版に書き換えられているということの記載を求めることもかまわないし、技術以外のトピックを扱っている節については、節全体の削除や変更を禁止することまで認められるはずだ。これらの制限が問題とならないのは、職務に忠実なプログラマが変更後のプログラムに合わせてマニュアルを書き換えることを禁止していないからである。言い換えれば、それらの制限は、フリーソフトウェアコミュニティがマニュアルをフル活用することを妨げない。

しかし、マニュアルの「技術的な」内容はすべて書き換え可能でなければならないし、通常のすべてのメディア、すべてのチャンネルを通じて結果を頒布できなければならない。そうでなければ、制限はコミュニティにとって有害であり、そのマニュアルはフリーではないということになる。私たちは、別のマニュアルを用意しなければならない。

フリーソフトウェアの開発者たちは、あらゆるタイプのフリーマニュアルを書くという意識と決意を持つようになるだろうか。繰り返すが、私たちの将来は、私たちの哲学にかかっている。

私たちは自由について話し合わなければならない

Debian GNU/Linux や Red Hat Linux などの GNU/Linux システムのユーザーは数千万人いると推計されている。フリーソフトウェアは、純粋に実践的な理由からそれだけのユーザーが追随するだけの現実的なメリットを生むところまで成長してきたのだ。

このことがもたらす良い結果は明らかである。フリーソフトウェアの開発に対する関心が高まり、フリーソフトウェアビジネスの顧客が増え、私有ソフトウエ

ア製品ではなく市販フリーソフトウェアを開発する企業の士気が高まる。

しかし、フリーソフトウェアに対する関心は、それが基礎としている哲学の認知度よりも速いペースで高まっている。今まで示してきた試練や脅威に私たちが対抗できるかどうかは、自由を求める固い意志にかかっている。コミュニティにその意志を確実に持たせるためには、コミュニティに入ってくる新しいユーザーにこの思想を普及させなければならない。

しかし、私たちはそれに失敗している。人々は、コミュニティの倫理を教えようとするよりも、コミュニティに新しいユーザーを引き付けようとするにはるかに熱心になっている。私たちは両方を必要としており、両方のバランスをとらなければならない。

「オープンソース」

コミュニティの一部が「フリーソフトウェア」という用語の使用を廃して「オープンソースソフトウェア」という言葉を使い出した 1998 年には、新しいユーザーに自由について教えることがさらに困難になった。

この用語を推した人々の一部は、「フリー」が「無料」と間違えられるのを防ごうと思っていた。これは正しい目標である。しかし、フリーソフトウェア運動と GNU プロジェクトの動因であった精神と原則を脇に置いて、経営者やビジネスユーザーにアピールすることを目指した人々もいた。経営者やビジネスユーザーの多くは、自由、コミュニティ、原則よりも利益を重視するイデオロギーを持っている。「オープンソース」というレトリックは、高品質で強力なソフトウェアを創造する可能性にスポットライトを当てるが、自由、コミュニティ、原則の思想を霞ませてしまう効果を持っている。

「Linux」関連の雑誌は、このことをはっきり示す例である。これらの雑誌には、GNU/Linux と併用できる私有ソフトウェアの広告が満載されている。次の Motif や Qt が現れたとき、これらの雑誌は、プログラマたちに離れよと警告を発するのだろうか、それともそれらの広告を掲載するのだろうか。

ビジネス界からの支援は、さまざまな面でコミュニティに貢献できる。そのことに関してはみな平等であり、意味がある。しかし、自由と正義についての言及

を減らすことによってビジネス界からの支援を勝ち取っても、惨めな結果を招く可能性がある。それは、ソフトウェアの普及と思想の浸透のアンバランスをさらに悪化させるだろう。

「フリーソフトウェア」と「オープンソース」は、多かれ少なかれ同じタイプのソフトウェアを指すが、ソフトウェアとその価値観についての思想は異なる。GNU プロジェクトは、単なるテクノロジーではなく、自由が重要なのだという思想を表現するために、「フリーソフトウェア」という用語を使い続ける。

トライしよう

「『トライ』というものはない」というヨーダの哲学はかっこよく聞こえるかもしれないが、私には無関係である。私は、自分にその仕事ができるのかどうか不安に思いながら、そしてできたとしても十分に目標を達したのか確信を持っていないまま、ほとんどの仕事をしてきた。しかし、敵と私の町の間には私しかいなかった。ので、何はともあれトライしてきた。自分自身驚いているが、私はときどき成功を収めた。

逆に、ときどきは失敗し、私の町は敵に奪われた。そして、さらに脅威を受けている別の町を見つけ、新しい戦いに備えた。私は脅威を探し、敵と町の間にも身を置き、他のハッカーを私の陣営に迎えてきた。

今は、多くの場合、私は、たった一人ではない。味方のハッカーたちが防衛線を築いているのを眺め、今のところは町が生き残れそうに感じられるのは、喜びであり、気持ちが休まることである。しかし、危険は年々増大している。そして、今や Microsoft が、私たちのコミュニティを明確にターゲットに据えた。自由に未来があることを当たり前だと思うことはできない。思ってはならないのだ。自由を保ちたいければ、自由を守る備えを持たなければならない。