Pluralismo estratégico no desenvolvimento tecnológico: a controvérsia dos “patches ck” e o conflito entre interesses empresariais e coletivos no software livre

Miguel Said Vieira[1]

Embora não seja um campo estritamente ligado ao desenvolvimento científico, o desenvolvimento de software livre se presta a analogias ao modelo de Hugh Lacey sobre a interação entre ciência e valores (LACEY; MARICONDA, 2014). Neste trabalho, aponto relações entre esse modelo e uma controvérsia — a dos *patches* “ck” — ocorrida no desenvolvimento do kernel Linux.

Software livre

O fato de que o kernel Linux é um software livre é central para a compreensão desta controvérsia; assim, apresentaremos de início algumas características do software livre que são relevantes para este caso.

Software livre é aquele que valoriza as liberdades específicas de seus usuários e produtores; liberdades que garantem a circulação e utilização desse software sem restrições significativas. Richard M. Stallman, o fundador do movimento do software livre, elencou quatro liberdades como sendo essenciais a esse tipo de software: a liberdade de executá-lo; de estudá-lo; de modificá-lo; e de redistribuí-lo, seja em sua forma original, seja com modificações (STALLMAN, 2010, p. 3).

Esses princípios básicos do software livre influenciam, por sua vez, a sua dinâmica de desenvolvimento: as liberdades citadas favorecem muito o desenvolvimento colaborativo de software. Um dos principais valores que acompanhou essa forma de desenvolvimento colaborativo, desde o seu surgimento, foi o que é visto como uma “meritocracia do código”: a participação nas comunidades de desenvolvimento é em geral formalmente aberta a qualquer pessoa; e embora as lideranças de projetos possam ter o poder de escolher aceitar ou não uma contribuição específica, esse poder é contrabalanceado pelo fato de que as liberdades permitem a qualquer um *forkear* o projeto — isto é, copiar o código e acrescentar uma contribuição rejeitada, eventualmente atraindo desenvolvedores do projeto anterior para o novo (WEBER, 2004, p. 144–5, 180–5). Há portanto uma pressão para que as contribuições sejam avaliadas por seus méritos técnicos (na performance do software, por exemplo) — embora na prática, como veremos, a situação seja bem mais complexa.

1 Doutor em Educação, professor adjunto na Universidade Federal do ABC, miguel.vieira@ufabc.edu.br, <https://impropriedades.wordpress.com>. Este capítulo desenvolve ideias inicialmente apresentadas em minha tese de doutorado (VIEIRA, 2014, cap. 4) e em comunicação elaborada para o ESOCITE (VIEIRA, 2019), e delas contém trechos.

O kernel Linux

Com a popularização da internet e de algumas ferramentas nela baseadas (como as listas de e-mails e os sistemas de controle de versões), tornou-se possível que essa colaboração ocorresse numa escala nunca antes vista, de forma descentralizada, geograficamente dispersa, e agrupando atores de origens variadas (amadores, acadêmicos, profissionais etc.). O desenvolvimento passou a se agrupar em torno de projetos de softwares específicos, como o kernel Linux — o primeiro software livre a demonstrar o potencial amplo do desenvolvimento colaborativo pela internet.

Um kernel é um núcleo de sistema operacional: a "peça" do sistema operacional que faz a interface entre softwares e o hardware (CPU, memória, disco, periféricos etc.). O kernel Linux é utilizado em sistemas como o Android e as diversas variantes de GNU/Linux, e tornou-se o software livre mais disseminado até hoje: em 2023, cerca de 80% dos servidores web (W3TECHS, 2023)[2] e 70% dos celulares (STATCOUNTER, 2023b) utilizam sistemas operacionais baseados em Linux — embora isso ocorra em apenas 3% dos computadores pessoais (STATCOUNTER, 2023a) (detalhe relevante para esta controvérsia, como veremos).

Em função desse grau de disseminação e da complexidade técnica envolvida no Linux, a sua comunidade de desenvolvimento é bastante singular: grande (com milhares de contribuidores ativos), dinâmica (com várias atualizações lançadas a cada ano, desde sua criação), e bastante seletiva (as interações entre as diversas áreas do kernel impõem uma curva de aprendizagem íngreme para começar a contribuir, e a comunidade é sabidamente intimidadora para iniciantes).

Escalonamento de processos

A controvérsia dos *patches* "ck" envolveu o escalonamento de processos — uma função tipicamente desempenhada por um núcleo de sistema operacional, como o Linux. Em um computador típico, há sempre diversos softwares sendo executados em paralelo: o sistema operacional, o navegador de internet, um banco de dados etc.; o processador, porém, executa apenas um deles de cada vez, e vai alternando entre eles. O escalonamento é justamente o mecanismo que determina qual processo deve ser executado a cada ciclo de funcionamento do processador. Os escalonadores ou algoritmos de escalonamento usam diferentes métodos e critérios para determinar essa sequência, de acordo com (combinações de) diferentes finalidades: a maximização do uso do processador, a priorização de certos tipos de processos, a distribuição equânime de ciclos etc.

O escalonamento é uma tarefa bastante complexa, e quando não se realiza de forma adequada pode produzir efeitos indesejáveis, que incluem redução de responsividade ou performance. Por um

2 Nesse levantamento, embora o percentual de 80% seja atribuído à categoria "Unix", os sistemas GNU/Linux representam a esmagadora maioria dos sistemas operacionais identificáveis agrupados sob essa categoria.

lado, se o mouse não se move na tela no mesmo momento em que é manipulado, ou se a reprodução de uma música em um computador é interrompida ou desacelerada, mesmo que momentaneamente, a experiência do usuário é muito degradada; nesse caso, o escalonador não conseguiu manter uma responsividade ou latência adequada para esses softwares. Por outro lado, se o escalonamento dedica muitos ciclos do processador aos processos relacionados à interface gráfica ou à reprodução de áudio, outros softwares sendo executados naquele momento — como um gerenciador de arquivos que esteja realizando uma cópia ou compressão, digamos — podem demorar mais a concluir suas tarefas; nesse caso, houve uma perda na performance desses softwares. Em computadores pessoais, assim, os processos de interface e multimídia em geral demandam uma prioridade maior para garantir responsividade, e em servidores tende-se a priorizar tarefas de consultas a bancos de dados, processamento intenso de dados etc., para obter ganhos de performance nelas — uma vez que a responsividade na interface não é tão relevante assim, e o hardware tende a ser mais poderoso. No entanto, esta é uma descrição bastante simplificada: outros fatores — como o cenário de uso, os acessos a disco e memória RAM — afetam muito os resultados do escalonamento, impossibilitando que a questão seja resolvida apenas com a atribuição de prioridades estáticas para cada tipo de processo.

As contribuições de Con Kolivas e sua saída da comunidade

Embora o kernel Linux tenha sido desenvolvido inicialmente para uso em computadores pessoais, ele tem uma relação estreita com a computação científica e corporativa, por ser em certa medida descendente do Unix, uma família de sistemas operacionais bastante usados nesses setores. Talvez em função disso, os sistemas operacionais baseados no Linux sofreram durante bastante tempo com problemas que afetavam particularmente “usuários comuns” (incluindo os de movimentos sociais, do setor público, do Sul Global etc.), que utilizavam Linux em computadores modestos, para fins pouco especializados, e às vezes não possuíam muita experiência com computação; embora com o passar do tempo parte desses problemas tenham sido superados, eles impactaram negativamente a adoção de Linux em computadores pessoais. A performance inadequada em interface gráfica ou tarefas de multimídia é um exemplo particularmente relevante desses problemas aqui, pois tem relação direta com a questão do escalonamento: processos ligados a essas tarefas precisam ser executados em frequência constante e latência adequada, ou a interface do computador aparenta “travar”, músicas e vídeos ficam picotados e distorcidos etc.

Os chamados *patches* “ck” foram um conjunto de contribuições ao Linux voltadas a melhorar sua funcionalidade em situações como essa; seu autor, Con Kolivas, é um anestesista que envolveu-se com o desenvolvimento do Linux como programador amador, interessado pelos problemas de responsividade que o sistema apresentava. Após desenvolver uma ferramenta (“idlerun”) que tentava lidar com a questão no *userspace* (isto é, na camada de aplicações, subordinada ao sistema ope-

racional), percebeu que qualquer melhoria mais robusta só seria possível no sistema operacional; em seguida, desenvolveu um software para tentar fazer medições comparativas de responsividade, o que pareceu abrir o caminho, na comunidade de desenvolvimento, para mudanças no escalonador do Linux.

Em paralelo, a partir de 2002, Kolivas começa a manter os *patches* "ck", com mudanças pontuais (inicialmente apenas agrupadas por ele, mas depois passando a ter modificações do próprio Kolivas) que buscavam melhorar a responsividade do sistema em PCs. Mas conforme se aproximava o lançamento de uma nova versão do núcleo, a 2.6, o problema não parecia estar próximo de uma solução: em 2003, havia relatos de que o núcleo estava até pior nesse respeito do que em versões anteriores (CORBET, 2003).

No final desse ano, sai uma nova versão do núcleo; o conjunto de *patches* de Kolivas não é incorporado, ainda que uma ou outra de suas propostas sejam aproveitadas por Ingo Molnár, o desenvolvedor responsável pelo escalonador que então era o do *mainline*[3] (e que era conhecido como "O(1)"). Há uma série de relatos (na própria *lkml*) sobre as melhoras que os *patches* "ck" apresentam; mas também há indicações de casos bastante específicos em que ele não se comporta bem. Por outro lado, de forma geral esse também era o caso do O(1), que continuava com problemas em tarefas como a reprodução de áudio. Kolivas afirmaria posteriormente que a não-inclusão nessa versão seria sua primeira grande frustração no seu envolvimento com o núcleo (MILLS, 2007).

Alguns desenvolvedores argumentavam que a questão poderia ser resolvida no médio prazo se o escalonador fosse um elemento modular do núcleo: isto é, se houvesse a possibilidade de usar várias opções de escalonadores, e que essa mudança fosse simples — algo que não ocorria até então: já não era muito simples fazer a manutenção de *patches* como os de Kolivas, e eles nem chegavam a reescrever o escalonador de forma tão radical. Assim, as pessoas particularmente afetadas pelos casos em que o escalonador oficial falha poderiam utilizar outro, e vice-versa. O O(1) era tido como um escalonador "ultraescalável":[4] um dos seus grandes méritos, nesse sentido, era o fato de que sua performance é mantida mesmo em servidores com uma grande quantidade de CPUs; e embora ele tenha representado um avanço na responsividade em relação ao escalonador anterior, os relatos já mencionados mostram que a situação estava longe de ser perfeita para os usuários da outra ponta do espectro — os PCs caseiros, com uma ou poucas CPUs, e que executam tarefas que de-

3 *Mainline* é o nome que se dá ao código "oficial" de um projeto em software livre, sem modificações, ou *patches*.

4 Essa é literalmente a maneira como ele é descrito numa breve seção de documentação do Linux (mais especificamente, um bloco de comentários do código) que registra a evolução histórica dos escalonadores até 2007 (ANDREWS, 2007). Veja-se também uma nota de quando foi lançado o núcleo incluindo esse escalonador: "Em operação, você provavelmente não notará mudanças em cargas baixas, mas maior escalabilidade com grandes números de processos, *especialmente em grandes sistemas SMP* [*symmetric multiprocessor system*; máquinas com múltiplos processadores]" (JONES, 2003; trad. nossa, grifo adicionado). Um resumo parcial da discussão na *lkml* sobre a modularização do escalonador no núcleo encontra-se em Brown (2005).

mandam bastante responsividade. A modularização permitiria aplicar abordagens diferenciadas para esses dois tipos de usuários, e talvez chegar a soluções melhores para ambos. Kolivas chega a desenvolver o código que permitiria modularizar esse aspecto do núcleo, mas ele não é aceito: o argumento vencedor — defendido por Molnár, entre outros — foi o de que a modularização poderia levar à criação de diversos escalonadores medíocres, bons apenas em casos específicos (CORBET, 2004).

Kolivas acumula experiência sobre escalonamento fazendo a manutenção de seus *patches*, e em 2007 desenvolve um escalonador alternativo, que ele nomeia de “rotating staircase deadline scheduler”, ou RSDL; não se tratava mais de ajustes ou modificações parciais ao escalonador oficial, mas sim da implementação completa de uma nova abordagem de escalonamento (já conhecida na ciência da computação, mas nunca implementada para o Linux).[5] Ocorre então uma situação parecida com a de 2003, quando ele propusera modificações substanciais ao escalonador oficial: sua proposta é aplaudida por vários dos testadores (e a princípio, o próprio Linus Torvalds — criador do Linux e líder de desenvolvimento — sinaliza estar propenso a incluí-la no *mainline*), mas surgem críticas apontando defeitos em casos bastante específicos. Ele busca resolver alguns dos problemas apontados, corrigindo o código e lançando novas versões do escalonador. Mas uma dessas críticas, em particular, leva a discussões muito acaloradas na *lkml*; Torvalds interveio na briga, anunciando que não incluiria o RSDL no *mainline* por estar incomodado com a maneira pela qual Kolivas havia respondido àquela crítica. Pouco tempo depois, Ingo Molnár, o autor e mantenedor do escalonador oficial até então, escreve e lança um novo escalonador, que ele nomeia de “completely fair scheduler”, ou CFS — baseado na mesma abordagem seguida por Kolivas, mas implementado de forma diferente, e que três meses depois seria incorporado ao *mainline*. Quinze dias após esse anúncio, Kolivas informa à *lkml* que decidira abandonar a comunidade. Ocorria ali, nas palavras do seu principal cronista (CORBET, 2011), uma das maiores perdas da comunidade do núcleo Linux.[6]

Interpretando a controvérsia

Como fica evidente pela apresentação feita até aqui, este caso é complexo; envolve divergências técnicas, pessoais, e centenas de atores — com interesses conflitantes — que se debruçaram sobre as questões do escalonador durante um período de pelo menos cinco anos. Por conta disso, é preciso cuidado para não tirar conclusões simplistas a respeito dessa disputa; mas ainda assim, é possível tentar interpretá-la.

5 Para uma descrição dos princípios de funcionamento do RSDL, ver Corbet (2007b).

6 Jonathan Corbet é o criador daquele que até hoje é o principal site de notícias e análises sobre o desenvolvimento do núcleo Linux, o LWN.net (*Linux Weekly News*). Nessa apresentação ele faz uma lista de casos em que algo deu muito errado no desenvolvimento do núcleo; respondendo a uma pergunta da plateia (aos 42:30 da apresentação referida), ele indica o caso de Kolivas como aquele que, dentre os que ele mencionara, havia provocado a perda mais séria para a comunidade.

O já citado Jonathan Corbet é um dos que tentou fazer essa interpretação. Ele entende que Kolivas tem uma parcela de responsabilidade pelo que ocorreu, por ter conduzido o desenvolvimento de suas contribuições principalmente em uma lista paralela à *lkml*, em que participavam muitos usuários de PCs caseiros. Torvalds também tocara nesse ponto, afirmando que os testes do escalonador feitos ali estavam sujeitos a um forte viés de seleção — as pessoas que utilizavam o código de Kolivas eram as que mais se beneficiavam por ele, e portanto elas não encontravam ou enxergavam defeitos.

Além disso, para Corbet o caso traria algumas lições, como a importância de "desapego" em relação ao próprio código, e a dificuldade de modificar certas partes do núcleo. Outra lição seria a importância de, ao realizar melhorias no núcleo, evitar "quebrá-lo" para terceiros — e aqui Corbet se refere aos defeitos apontados no RSDL, e que geraram a briga que culminou na saída de Kolivas.

Mas em um texto escrito logo após a saída de Kolivas, Corbet menciona algo que pode dar um sentido diferente — e bastante relevante — a essa ideia de que mudanças que busquem melhorias não devem "quebrar" o núcleo para terceiros. Comentando um *patch* de Kolivas na área de gerenciamento de memória — que foi abandonado por conta de sua saída da comunidade —, ele disse:

> Fazer ajustes pontuais na heurística que guia o gerenciamento de memória é um processo difícil; uma mudança que faz com que uma carga de trabalho [*workload*] rode melhor pode, imprevisivelmente, destruir a performance em outro lugar. E esse "outro lugar" pode nunca aparecer *até que alguma grande instituição financeira qualquer tente implantar* [*deploy*] *uma nova versão do núcleo. Os mantenedores principais do núcleo já viram isso acontecer numa frequência suficiente para se tornarem bastante conservadores* com mudanças no gerenciamento de memória. (CORBET, 2007a; trad. minha, grifo adicionado)

Em outras palavras, Corbet relata que os desenvolvedores temem degradar a performance das empresas que usam o Linux em grandes servidores, e que isso leva a um conservadorismo em relação a mudanças. Embora Corbet não desenvolva o tema, isso parece ter uma razão bastante clara: boa parte dos desenvolvedores do núcleo é contratada por empresas como essas, ou — principalmente — por grandes empresas de tecnologia da informação (como Sun, IBM, Red Hat) que prestam serviços a elas. Nesses casos, uma pequena degradação de performance pode implicar em perdas de contratos milionários; e para um desenvolvedor profissional, até mesmo a perda de seu emprego. O percentual desses desenvolvedores "profissionais" na comunidade do núcleo tem aumentado progressivamente (e é notório que participar ativamente dessa comunidade abre muitas portas de emprego).

E esse ponto se relaciona às razões para a recusa do RSDL, em 2007; naquela ocasião, Torvalds afirmara que Kolivas não era um mantenedor em quem ele confiava: "Foi aí que os *patches*

SD[7] caíram. Eles não tinham um mantenedor que eu pudesse confiar que de fato se importaria com questões além das dele mesmo" (TORVALDS, 2007; trad. nossa). Mas é difícil determinar objetivamente o quanto dessa desconfiança se devia estritamente à personalidade de Kolivas — isto é, como ele respondeu às críticas —, como alegara Linus, e o quanto se devia ao fato de que ele era um programador amador, que — ao contrário do próprio Linus, de Ingo Molnár (contratado da Red Hat, a maior empresa de tecnologia da informação no mundo dedicada exclusivamente a software livre) e de tantos outros desenvolvedores da comunidade — não era remunerado para dedicar seu tempo integral ao desenvolvimento do núcleo.

É razoável supor que este último motivo tenha influenciado a decisão contra o escalonador de Kolivas (entre outros motivos, porque Linus admitiu explicitamente que o fator principal em sua decisão não era a qualidade do código do escalonador); e isso é até compreensível, se imaginamos a perspectiva individual de Linus — alguém que enfrenta diariamente os desafios de coordenar um trabalho colaborativo desse porte e complexidade. Entretanto, também não se pode negar que esse tipo de critério desfavorece fortemente aqueles que, como Kolivas, estavam mais preocupados com as pessoas que, embora sejam maiores em número, são quem têm menos poder de influência sobre o desenvolvimento do núcleo: os usuários caseiros de PCs. A adoção desse critério faz com que o argumento (utilizado por Linus e Corbet) de que Kolivas baseava seu trabalho em uma comunidade fechada, com viés de seleção, possa ser igualmente aplicado à comunidade de desenvolvimento do núcleo: uma comunidade formada em sua maioria por programadores que utilizam PCs superiores à média, e que estão preocupados em garantir a performance do núcleo em servidores.

Relações com a tese do pluralismo estratégico

À primeira vista, pode parecer inadequado analisar esse caso à luz do modelo de Lacey sobre as interações entre ciência e valores: o desenvolvimento de software, ainda mais em contexto não acadêmico, é tipicamente aplicação tecnológica; e com efeito, embora nesta controvérsia houvesse esforços para comparar empiricamente os resultados dos algoritmos propostos, eles não eram particularmente sistemáticos nem precisos: o foco era muito mais produzir software funcional, do que construir conhecimento fundamentado.

Por outro lado, é possível identificar aspectos do caso que se assemelham, como que em escala reduzida, a conceitos propostos por Lacey. Embora de maneira não monolítica, a comunidade de desenvolvimento do Linux apresenta fortes vinculações com o valores do progresso tecnológico (particularmente na vertente *open source* do movimento de software livre, que enfatiza os valores da eficiência e da velocidade — em detrimento do valor da autonomia, que é priorizado na vertente

7 Outra sigla pela qual era conhecido o escalonador RSDL.

original do movimento),[8] e em termos materiais é cada vez mais vinculada ao capital e ao mercado: a larga maioria dos participantes ativos na comunidade são empregados por grandes corporações, como Google e IBM (STEWART; KHAN; GERMAN, 2020). E ainda que não se possa diferenciar na controvérsia analisada a adoção de estratégias descontextualizadoras / sensíveis ao contexto, é possível enxergar elementos de *estratégias* conflitantes — havia uma diferença clara entre, por exemplo, quais fontes de dados eram consideradas relevantes (Torvalds enxergava viés de seleção entre usuários de PCs, mas não entre desenvolvedores profissionais), e o que era ou não um problema digno de investigação no Linux: para contribuidores amadores, sua baixa responsividade em PCs era um problema grave; para desenvolvedores profissionais, não só ela era pouco notada (por terem acesso a computadores mais avançados), como seria um *trade off* aceitável caso o escalonamento garantisse eficiência em servidores (como aqueles das grandes empresas financeiras e industriais, que indiretamente remuneram os desenvolvedores profissionais).

Por fim, o ponto mais crucial de contato é com a tese do pluralismo estratégico: a escolha por não modularizar o algoritmo de escalonamento literalmente determinou a exclusividade de uma das abordagens que estavam em discussão (aquela mais associada ao capital e ao mercado), e na prática inviabilizou o desenvolvimento e testagem da outra (mais associada a usos não mercadológicos do software). A modularização realmente implicaria em "sacrifícios" (como a dispersão do trabalho entre mais de um escalonador), mas eles estavam longe de serem absolutos, ou de justificarem inequivocamente essa escolha.

Tomadas em conjunto, estas relações sugerem que o modelo de Lacey talvez também seja relevante para analisar áreas mais diretamente ligadas ao desenvolvimento tecnológico (que à pesquisa científica); ainda seria preciso, porém, investigar eventuais adaptações necessárias para tanto. Como exemplo, no desenvolvimento de software livre (ou de outras tecnologias), em geral não estará em jogo a avaliação de uma teoria científica, algo central para o modelo de Lacey; no entanto, o momento da revisão de código — mais especificamente de *pull requests*, as alterações propostas por contribuidores — parece ser bastante análogo ao M_3 do modelo de Lacey (etapa de avaliação cognitiva), pois é a etapa em que são testadas e avaliadas as propostas tecnológicas em elaboração. Haveria, porém, uma diferença significativa: se no M_3 do modelo original trata-se de avaliação cognitiva (baseada em valores cognitivos: adequação empírica, poder de explicação etc.), na revisão de código podemos dizer que trata-se de avaliação funcional: o software *funciona* melhor com a alteração proposta? Essa avaliação funcional é baseada em valores que poderíamos chamar de "funcionais": eliminação de *bugs*, velocidade, responsividade, economia de recursos. Adaptando o modelo dessa forma, seria possível também incluir considerações sobre contextualização (outro tema caro ao raci-

8 Sobre as duas vertentes do movimento software livre, ver Evangelista (2010).

ocínio de Lacey), já que alguns desses valores “funcionais” requerem que aspectos de contexto sejam considerados; a responsividade, por exemplo, depende de cenários de uso, tipos de computadores e usuários: uma melhora no PC de um telecentro na periferia pode ser ao mesmo tempo uma piora no servidor de um banco.

Referências bibliográficas[9]

ANDREWS, Jeremy. *Linux: The Original Process Scheduler*, 15 ago. 2007. Disponível em: <http://wayback.archive.org/web/20080229144843/http://kerneltrap.org/Linux/The_Original_Process_Scheduler>.

BROWN, Zack. *Kernel Traffic #296 For 12Feb2005*, 12 fev. 2005. Disponível em: <http://wayback.archive.org/web/20070527085358/http://kerneltraffic.org/kernel-traffic/kt20050212_296.html#3>.

CORBET, Jonathan. *Fixing interactive response in 2.6*, 29 jul. 2003. Disponível em: <http://lwn.net/Articles/41618/>.

CORBET, Jonathan. *How kernel development goes wrong and why you should be a part of it anyway*, 6 fev. 2011. Disponível em: <http://www.youtube.com/watch?v=MzCIBZONf5M>.

CORBET, Jonathan. *Schedulers, pluggable and realtime*, 3 nov. 2004. Disponível em: <https://lwn.net/Articles/109458/>. Acesso em:.

CORBET, Jonathan. *Still waiting for swap prefetch*, 25 jul. 2007a. Disponível em: <https://lwn.net/Articles/242765/>.

CORBET, Jonathan. *The Rotating Staircase Deadline Scheduler*, 6 mar. 2007b. Disponível em: <http://lwn.net/Articles/224865/>.

EVANGELISTA, Rafael de Almeida. *Traidores do movimento: política, cultura, ideologia e trabalho no Software Livre*. 2010. Tese de doutorado – Unicamp. Disponível em: <https://hdl.handle.net/20.500.12733/1611403>.

JONES, Dave. *Linux-Kernel Archive: 2.5 “what to expect”*, 14 abr. 2003. Disponível em: <http://lkml.iu.edu//hypermail/linux/kernel/0307.1/1082.html>.

LACEY, Hugh; MARICONDA, Pablo Rubén. “O modelo das interações entre as atividades científicas e os valores”. *Scientiae Studia*, v. 12, n. 4, p. 643–668, dez. 2014. Disponível em: <https://doi.org/10.1590/S1678-31662014000500002>.

MILLS, Ashton. *Interview with Con Kolivas*. 2007. Disponível em: <https://web.archive.org/web/20150524163528/http://apcmag.com/interview_with_con_kolivas_part_1_computing_is_boring.htm/>.

STALLMAN, Richard M. *Free Software, Free Society: Selected Essays of Richard M. Stallman*. 2. ed. [S.l.]: Free Software Foundation, 2010.

STATCOUNTER. *Desktop Operating System Market Share Worldwide*, fev. 2023a. Disponível em: <https://gs.statcounter.com/os-market-share/desktop/worldwide>.

9 As URLs foram acessadas pela última vez em fevereiro de 2023.

STATCOUNTER. *Mobile Operating System Market Share Worldwide*, fev. 2023b. Disponível em: <https://gs.statcounter.com/os-market-share/mobile/worldwide>.

STEWART, Kate; KHAN, Shuah; GERMAN, Daniel M. *Linux Kernel History Report 2020*. [S.l.]: The Linux Foundation, 2020. Disponível em: <https://project.linuxfoundation.org/hubfs/Reports/2020_kernel_history_report_082720.pdf?hsLang=en>.

TORVALDS, Linus. *Re: Linus 2.6.23-rc1*, 27 jul. 2007. Disponível em: <https://lkml.org/lkml/2007/7/27/426>.

VIEIRA, Miguel Said. *Os bens comuns intelectuais e a mercantilização*. 2014. 365 f. Tese de doutorado – Faculdade de Educação da Universidade de São Paulo, São Paulo. Disponível em: <http://www.teses.usp.br/teses/disponiveis/48/48134/tde-01102014-104738/>.

VIEIRA, Miguel Said. “Software livre, mas para quem? O conflito entre interesses empresariais e coletivos na controvérsia dos ‘patches ck’ ”. In: ESOCITE.BR, 2019, Belo Horizonte. *Anais*... Belo Horizonte: [s.n.], 2019. p. 20. Disponível em: <https://www.esocite8.cefetmg.br/anais-2/>.

W3TECHS. *Usage Statistics and Market Share of Operating Systems for Websites, February 2023*. Disponível em: <https://w3techs.com/technologies/overview/operating_system>.

WEBER, Steven. *The Success of Open Source*. Cambridge, MA: Harvard University Press, 2004.