УДК 004.7:004.273

Красовська Є.В., к.т.н.

АРХІТЕКТУРА КОМПЛЕКСУ МОНІТОРИНГУ АКТИВНОСТІ КОРИСТУВАЧІВ КОРПОРАТИВНОЇ КОМП'ЮТЕРНОЇ МЕРЕЖІ

Національний авіаційний університет

Запропоновано підхід до створення та використання програм моніторингу діяльності та активності користувачів в комп'ютерній корпоративній мережі.

Актуальність використання засобів контролю активності користувачів комп'ютерної мережі

Проблеми, які виникають з безпекою передавання інформації при роботі в комп'ютерних мережах (КМ), можна розділити на три основні типи:

- перехоплення інформації – цілісність інформації зберігається, але її конфіденційність порушена;
- модифікація інформації – вихідне повідомлення змінюється або повністю замінюється іншим;
- підміна авторства інформації.

Сучасні швидкісні методи криптографічного перетворення є найбільш ефективним засобом забезпечення конфіденційності даних, їх цілісності й достовірності. Лише їх використання в сукупності з необхідними технічними і організаційними заходами можуть забезпечити захист від потенційних погроз [1–2].

Для надійного захисту інформації та виявлення випадків неправомочних дій проводиться реєстрація роботи системи, яка періодично перевіряє працездатність апаратних і програмних засобів захисту.

Постановка проблеми

Практична реалізація заходів щодо забезпечення конфіденційності сучасних інформаційних систем (ІС) натрапляє в Україні на серйозні труднощі. По-перше, відомості про технічні канали витоку інформації є закритими, тому більшість користувачів позбавлена можливості скласти уявлення про потенційні ризики. По-друге, на шляху криптографії стоять багаточисельні законодавчі перепони й технічні проблеми.

При аналізі проблематики, пов'язаної з інформаційною безпекою (ІБ), необхідно зважати на специфіку даного аспекту безпеки, що полягає в тому, що ІБ є складова частина інформаційних технологій. Тут важливі не стільки окремі рішення (закони, учбові курси, програмно-технічні
вироби), скільки механізми генерації нових рішень, які дозволяють жити в темпі технічного прогресу.

Сучасна технологія програмування не дозволяє створювати безпомилкові програми, що не сприяє швидкому розвитку засобів забезпечення ІБ. Це можливо, але вимагає дотримання певних архітектурних принципів і контролю стану захищеності на всьому життєвому циклі ІС.

Порівняльний огляд існуючих рішень

На противагу світу *Windows*, в *UNIX*-подібних операційних системах (ОС) відсутні системи контролю та реєстрації користувацької активності. Якісні комплекси – велика рідкість та в країнах СНД не представлені та серед програмного забезпечення (ПЗ) немає повнофункціональних програм, які б могли повністю покрити технічні вимоги.

Досить популярною технологією віддаленого доступу до віддаленої робочої станції є *Virtual Network Computing* (*VNC*), яка підтримує роботу з графічним інтерфейсом ОС сімейства *Unix* (*X*11), а також *Windows* і *Mac OS X*.

Команди, що надходять від клавіатури та миші, передаються на віддалений комп'ютер по мережі, звідти ви постійно отримуєте знімки екрану. Таким чином, при достатній швидкості мережевого з'єднання користувач фактично працює за віддаленим комп'ютером.

Характерною особливістю *VNC* є можливість організації кількох «точок підключення» на одному сервері. Такими точками є *X*-екрани *VNC*-сервера, яких може бути від одного до семи. Розподіл роботи декількох клієнтів з різними екранами відбувається через *TCP*-порти.

Важливим класом ПЗ в рамках ІБ є системи розмежування доступу. *SELinux* – це розширення базової моделі безпеки ОС *Linux*, що додає механізм мандатного доступу. За допомогою *SELinux* можна задати правила того, як суб'єкти можуть звертатися до об'єктів системи. Таким

чином, можна обмежити програми, прописавши можливості їх поведінки у вигляді політики, а ОС забезпечить її дотримання. Мандатний доступ в *SELinux* реалізовано в рамках моделі домен-тип [3–4].

Система *SELinux* є дуже гнучкою та потужною системою керування правами доступу, та має можливість реєстрації потенційно зловмисних дій з боку ПЗ чи користувача. Одночасно вона має дуже складний синтаксис конфігураційних файлів та потребує досвіду системного адміністратора, зокрема і в області ПЗ.

Іншою системою розмежування доступу є базова система контролю побудована на правилах (*RSBAC*)). *SBAC* – це надбудова над ядром ОС *GNU*/*Linux*, запропонована в [5], яка дозволяє створити на базі дистрибутиву *Linux* захищену ОС. Певні системні виклики доповнюються кодом, який виконує звернення до центрального компонента *RSBAC*, що приймає рішення про допустимість системного виклику. *RSBAC* є одночасно досить гнучкою та багатогранною системою, окрім того, вона не настільки громіздка, як *Securty-Enhanced Linux*, тому може використовуватись більш широко. Це інструмент обмеження, що пов'язано з обмеженими властивостями щодо пасивного спостереження за активністю.

Найпростішою та найдоступнішою для рядового користувача є порівняно нова система розмежування прав доступу *AppArmor*, яка дозволяє майже повністю переключити себе в режим стеження, але певний серверний додаток вимагає більш складної конфігурації та контроль мережевих з'єднань в цій системі реалізовано на зародковому рівні, а, отже, для цього доведеться використовувати стороннє ПЗ. *AppArmor* є найкращим вибором у випадку, коли передусім стоїть завдання контролю користувацької активності у файловому просторі.

Проаналізувавши існуючі комплекси, стає зрозуміло, що їх використання у якості системи спостереження вимагає, по-перше, глибокої реконфігурації та відключення непотрібних модулів, по-друге, кожна з них не є універсальною та потребує використання додаткового ПЗ. Іншою особливістю систем розподілення є необхідність виконання в режимі ядра, що накладає суттєві обмеження на вибір дистрибутиву та версії ядра.

Мета роботи

Аналіз актуальності використання засобів контролю користувачів, проектування архітектури комплексу моніторингу активності користувачів КМ.

Архітектура розроблюваного комплексу

Комплекс має розподілену, клієнт-серверну структуру, основану на архітектурі *Multi-Agent System* (рис. 1). На робочих станціях знаходяться клієнтські модулі, основною функцією яких є збір інформації. Кожен такий модуль складається з службової програми-демона та додаткових модулів, які й займаються стеженням. Основна функція програми-демона – збір отриманої інформації та її відправлення на сервер.

Серверна частина працює на виділеному сервері. Основні її задачі: одночасне отримання інформації від усіх клієнтів та забезпечення її архівації. Серверна частина також має модульну структуру, що дозволяє, наприклад, проводити запис інформації у базу даних, або ж організувати додаткове передавання на центральний сервер. Це дозволяє зберігати отриману інформацію в єдиному місці, що, по-перше, спрощує адміністрування та обслуговування комплексу, а, по-друге, збереження даних в одному місці збільшує їх доступність та спрощує аналіз.

Система розділена на клієнтську та серверну частини. Серверна частина встановлюється на виділений апаратний сервер та є центром збереження даних, що отримуються від агентів.

Мережева взаємодія, з точки зору збереження цілісності та недоторканості даних, вимагає по-перше, авторизації, а по-друге, шифрування самого сеансу зв'язку. Для цього використовується реалізація криптографічного протоколу *SSL* (у вигляді *OpenSSL*).

Головний серверний модуль, який запускається першим, виконує запуск та конфігурування інших, додаткових модулів, приймає з'єднання, та передає дані далі, до модулю зберігання.

Модуль зберігання – допоміжна система, що отримує та консолідує дані, дозволяє їх зберігання у відповідному форматі. Наразі використовується простий формат – інформація зберігається у звичайних текстових файлах.

Клієнтська частина має модульну будову (з можливістю подальшого розширення), складається з наступних компонентів:

1. Модуля мережевої взаємодії, який є центральною частиною, виконує передусім запуск та конфігурацію інших додаткових модулів, а також забезпечує встановлення з'єднання, передавання зібраної інформації та перемикан-

ня в локальний режим роботи в разі зникнення мережевого з'єднання.

2. Завантажувального модуля стеження за файловою системою, який використовує механізм перехоплення системних викликів, протоколює події доступу до об'єктів згідно конфігурації.

3. Модуля контролю мережі. Він дозволяє отримувати інформацію про мережеві з'єднання, не прибігаючи до використання драйверів, за допомогою технології *Netfilter* та *Conntrack*.

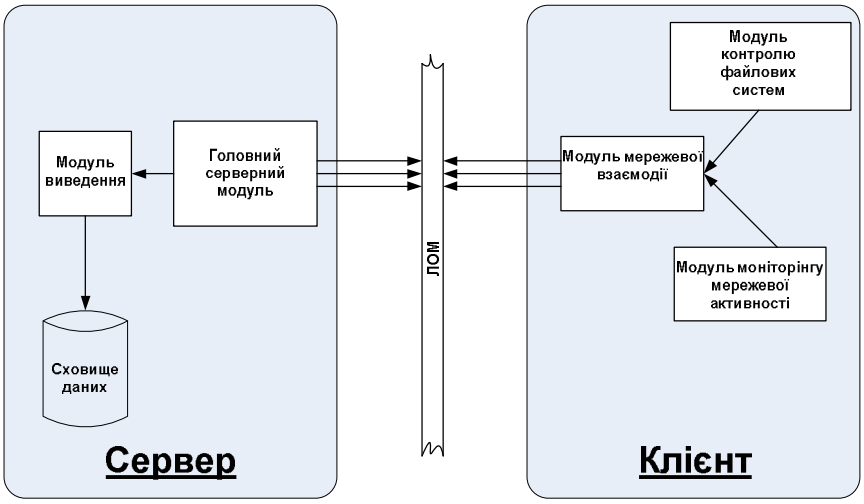

Рис. 1. Архітектура комплексу моніторингу дій користувачів

Віртуальна файлова система (*Virtual File System, VFS*), або віртуальний файловий комутатор (*Virtual File Switch*, *VFS*) – це підсистема ядра, що реалізує інтерфейс користувача програм до файлової системи (ФС). Всі ФС залежать від *VFS*, що дозволяє їм спільно функціонувати.

Підсистема *VFS* дозволяє системним викликам працювати незалежно від ФС та фізичного середовища носія. Загальний інтерфейс можливий тільки завдяки тому, що в ядрі реалізовано узагальнюючий рівень, що приховує низькорівневий інтерфейс ФС: *VFS* реалізує загальну файлову модель, що представляє загальні функції та особливості роботи потенційно можливих ФС.

Узагальнюючий рівень працює шляхом визначення базових інтерфейсів і структур даних, необхідних для підтримки всіх ФС, яка формує концепції роботи з ФС. Всі деталі – в коді самої ФС. По відношенню до інших частин ядра всі ФС підтримують однакові функції.

Реалізація підсистеми *VFS* має риси об'єктно-орієнтованого підходу. Загальна модель представлена у вигляді структур даних, схожих на об'єкти: структури даних містять покажчики на елементи даних і на функції, що працюють з цими даними. Існує чотири типи об'єктів *VFS*:

1. Об'єкт суперблоки (*superblock*), що представляє одну змонтовану ФС.

2. Об'єкт файловий індекс (*inode*), що представляє певний файл.

3. Об'єкт елемент каталогу (*dentry*), що представляє певний елемент каталогу.

4. Об'єкт файл (*file*), що представляє відкритий файл, пов'язаний з процесом.

Кожен з об'єктів *VFS* містить об'єкт операцій, що описують методи, які ядро може застосовувати для основних об'єктів:

1. Об'єкт *super_operations* (операції з суперблоку). Містить методи, які ядро може викликати для певної ФС.

2. Об'єкт *inode_operations* (операції з файловими індексами). Містить методи, які ядро може викликати для певного файлу.

3. Об'єкт *dentry_operations* (операції з елементами каталогів). Містить методи, які ядро може застосовувати для певного елемента каталогів.

4. Об'єкт *file_operations* (операції з файлами). Містить операції, які ядро може викликати для відкритого файлу.

Об'єкти операцій реалізовані у вигляді структур, що містять покажчики на функції, які оперують об'єктом *VFS*.

Об'єкт суперблоки описує певну ФС і реалізується для кожної ФС. Представляється за допомогою структури *super_block*.

Об'єкт *inode* містить інформацію, яка необхідна ядру для маніпуляцій з файлами та каталогами. Для ФС *Unix* об'єкт *inode* підсистеми

VFS зчитується з дискових індексів. Для кожного файлу в системі існує його індекс, однак об'єкт файлового індексу в пам'яті створюється лише тоді, коли до цього файлу здійснюється доступ.

Об'єкт *dentry* представляється структурою *dentry*. Він не відповідає ніякій реальній структурі даних на фізичному носії. Після знаходження шляху до об'єкта, система може зберегти знайдені об'єкт *dentry* в кеші для полегшення пошуку згодом.

Об'єкт файл – це подання відкритого файлу, що зберігається в ОП. Для одного файлу може існувати кілька об'єктів *file*, оскільки одночасно до одного файлу може звертатися декілька процесів. Файловий об'єкт представляється структурою *file*. Об'єкт *file* не відповідає ніякій реальній структурі на фізичному носії. Він містить покажчик на пов'язаний з ним об'єкт *dentry*, який містить покажчик на пов'язаний з ним об'єкт *inod*e, що представляє фізичний файл.

Кожна точка монтування представлена структурою *vfsmount*, яка використовується для представлення конкретної ФС, або точки монтування. Структура містить інформацію про точку монтування, включаючи положення і прапори, з якими ця точка була монтована.

Для того, щоб створити ФС, треба визначити необхідні об'єкти *VFS* та операції з ними. Назвемо нову ФС *LinFS*.

Загальний порядок дій такий:

1. Для роботи з файлами та каталогами необхідно визначити функцію створення об'єкта *inode*.

2. Щоб ядро змогло виконувати всі визначені у ФС операції, ФС треба зареєструвати в ядрі.

3. Для ФС необхідно реалізувати функції створення файлів і каталогів.

4. Реалізуються функції операцій з файлами і суперблоку.

5. Після цього ФС можна монтувати.

Мережевий стек по своїй конструкції має багаторівневу архітектуру, що повторює структуру самих протоколів.

Рівень сокетів представляє собою стандартний *API* до мережевої підсистеми. Він надає інтерфейс користувача до різних мережевих протоколів. Рівень сокетів реалізує стандартизований спосіб управління з'єднаннями і передаванням даних.

Хоча основна частина *Linux* незалежна від архітектури, на якій працює ОС, в деяких елементах для забезпечення нормальної роботи та підвищення ефективності необхідно враховувати архітектуру.

Крім переносимості та ефективності, ядро *Linux* має ряд інших функцій. *Linux* є випробувальним майданчиком для нових протоколів і їх удосконалень, підтримує велику кількість мережевих протоколів, включаючи традиційний *TCP*/*IP* і його високошвидкісні розширення. Слід зазначити, що ядро *Linux* є динамічним (підтримує додавання та видалення програмних компонентів без зупинки системи). Ще одне недавнє удосконалення *Linux* – можливість її використання в якості ОС для інших ОС (т.з. гіпервізора).

Ядро *Linux* можна, у свою чергу, розділити на рівні. Вгорі розташовується інтерфейс системних викликів, який реалізує базові функції, наприклад, читання і запис. Нижче за інтерфейс системних викликів розташовується архітектурно-незалежний код ядра. Цей код є загальним для всієї процесорної архітектури, що підтримує *Linux*. Ще нижче розташовується архітектурно-залежний код, який залежить від процесора і платформи для конкретної архітектури.

Ядро *UNIX*-подібної системи, якою і є *Linux*, по суті являє собою диспетчер ресурсів. Незалежно від того, що є керованим ресурсом – процес, пам'ять або апаратний пристрій, – ядро організовує і упорядковує доступ до ресурсу безлічі конкуруючих користувачів.

Управління процесами сконцентроване на виконання процесів. У ядрі ці процеси називаються потоками; вони відповідають окремим віртуалізованим об'єктам процесора (код потоку, дані, стік, процесорні регістри).

Ядро надає інтерфейс програмування додатків (*API*) через *SCI* для створення нового процесу (породження копії, запуску на виконання, виклику функцій, зупинки процесу, взаємодії і синхронізації між процесами.

Ще одне завдання управління процесами – спільне використання процесора активними потоками. У ядрі реалізовано новаторський алгоритм планувальника, час роботи якого не залежить від числа потоків, що претендують на ресурси процесора. Назва цього планувальника – *O*(1) – підкреслює, що на диспетчеризацію одного потоку витрачається стільки ж часу, як і на безліч потоків.

Інший важливий ресурс, яким управляє ядро, – це пам'ять. Для підвищення ефективності, враховуючи механізм роботи апаратних засобів з віртуальною пам'яттю, пам'ять організовується у вигляді т. з. сторінок (у більшості архітектур розміром 4 Кб). У *Linux* є засоби для управ-

ління наявною пам'яттю, а також апаратними механізмами для встановлення відповідності між фізичною і віртуальною пам'яттю. Проте управління пам'яттю – це значно більше, ніж просто управління буферами по 4 Кб. *Linux* надає абстракції над цими буферами, що дозволяє динамічно розширювати і скорочувати схему в залежності від потреб системи.

Ще один аспект ядра *Linux* – віртуальна ФС (*VFS*), яка надає загальну абстракцію інтерфейсу до ФС.

На верхньому рівні *VFS* розташовується єдина *API*-абстракція таких функцій, як відкриття, закриття, читання та запис файлів. На нижньому рівні *VFS* знаходяться абстракції ФС, які визначають реалізацію функцій верхнього рівня.

Нижче рівня ФС знаходиться кеш буферів, що надає загальний набір функцій до рівня ФС та оптимізує доступ до фізичних пристроїв за рахунок короткострокового зберігання даних. Нижче кешу буферів знаходяться драйвери пристроїв, що реалізують інтерфейси для конкретних фізичних пристроїв.

Найбільш важливі складові ядра – це блоки управління пам'яттю і процесами. Блок управління пам'яттю забезпечує розподіл ділянок пам'яті та *swap*-ділянок між процесами, складовими ядра і для кеш-буфера. Блок управління процесами створює нові процеси і забезпечує багатозадачність шляхом перемикання завдань.

На самому нижньому рівні ядро містить драйвери пристроїв для кожного типа підтримуваного обладнання.

В клієнтській частині для моніторингу мережевих з'єднань використовується бібліотека *libnetlink*, що є частиною проекту *Netfilter*.

З системної точки зору підсистема являє собою набір функцій, розташованих в ядрі, за допомогою яких *firewall*'и можуть отримувати доступ до пакетів і, ґрунтуючись на правилах, вирішувати, як з ними надходити далі. *Netfilter* містить 5 основних функцій.

1. *NF_IP_PRE_ROUTING* – функція спрацьовує як тільки отримано пакет, навіть якщо він проходить.

2. *NF_IP_LOCAL_IN* – спрацьовує, коли пакет адресовано, перед надходженням пакету до мережевого стеку.

3. *NF_IP_FORWARD* – якщо пакет необхідно змаршрутизувати з одного інтерфейсу на інший.

4. *NF_IP_POST_ROUTING* – для вихідних пакетів з мережевого стека.

5. *NF_IP_LOCAL_OUT* – для всіх вихідних пакетів.

Після виклику функції і проведення перевірок над пакетом, потрібно винести вердикт, що робити з цим пакетом далі. Існує 5 варіантів:

1. *NF_ACCEPT*: пропускає пакет.
2. *NF_DROP*: відкидає пакет.
3. *NF_REPEAT*: повторний виклик функції.
4. *NF_STOLEN*: забирає пакет.
5. *NF_QUEUE*: ставить пакет в чергу, як правило, для передавання в користувальницький простір.

Взаємодія із системою управління БД починається із процесу аутентифікації. Аутентифікація *PostgreSQL* на рівні хоста відрізняється високою гнучкістю і великою різноманітністю параметрів.

OpenSSL – це система захисту та сертифікації даних, система безпечних сокетів. *OpenSSL* використовується практично всіма мережевими серверами для захисту переданої інформацією.

Для організації можна запропонувати наступне рішення: на сервері створюється сертифікат організації; генерується запит на сертифікацію і відправляється до довіреного центру сертифікації (який буде відомий всім клієнтам і персоналу даної організації); виходить сертифікат організації, який можна використовувати при створенні сертифікатів клієнтів. Останні створюються так: клієнт посилає запит на видачу сертифіката; сервер створює сертифікат клієнта і підписує його сертифікатом організації; клієнт отримує сертифікат клієнта і сертифікат організації; після перевірки достовірності ключа організації перевіряється достовірність сертифіката клієнта. Після такої операції клієнт буде точно впевнений, що отримав сертифікат від даної організації, і може його використовувати для роботи з нею.

Мережева підсистема ОС *GNU*/*Linux* має багаторівневу структуру. На найвищому рівні знаходиться інтерфейс системного виклику, що дозволяє прикладним програмам взаємодіяти з ядром. Наступним йде протокол-незалежний рівень, що надає загальний спосіб роботи з нижчестоящими протоколами транспортного рівня. Далі слідують фактичні протоколи, до яких у системі *Linux* відносяться вбудовані протоколи *TCP*, *UDP* і, звичайно ж, *IP*. Наступний – ще один незалежний рівень, який забезпечує загальний інтерфейс до окремих доступних драйве-

рів пристроїв і від них, супроводжуваний в кінці самими цими драйверами.

Рівень сокетів є протокол-незалежним інтерфейсом, який надає набір стандартних функцій для підтримки низки різних протоколів. Цей рівень не тільки підтримує звичайні *TCP*- і *UDP*-протоколи, але також і *IP*, *raw Ethernet* і інші транспортні протоколи.

Взаємодія через мережевий стек відбувається за допомогою сокета. Структура сокета в *Linux – struct sock*. Ця велика структура містить всі необхідні стани окремого сокета, включаючи певний протокол, який використовується сокетом, і операції, які можна над ним робити.

Мережева підсистема знає про доступні протоколи із спеціальної структури, яка визначає її можливості. Кожен протокол містить структуру під назвою *proto*, яка визначає окремі операції сокета, які можуть виконуватися з рівня сокетів на транспортний рівень.

Розділ мережевих протоколів визначає окремі доступні мережеві протоколи. Після проходження процедури авторизації, клієнт отримує змогу надсилати потік даних, а сервер отримує, буферизує їх та передає по ланцюжку необхідним плагінам. У даній архітектурі використовується додатковий модуль, який виконує збереження отриманої інформації в БД.

Клієнтська частина займається збиранням даних про активність користувачів КС, тому доцільно наділити її підтримкою додаткових модулів – плагінів, для розширення підтримки тих додатків, контроль за якими не зводиться до реєстрації файлової або мережевої активності. Робота з ФС потребує інтеграції в механізм віртуальної ФС ОС *GNU*/*Linux*, тому цими операціями займається модуль, оформлений у вигляді драйвера. Таке розподілення архітектури, а також використання системи *Conntrack* для моніторингу мережевих з'єднань, на відміну від перехоплення системного виклику *socket*(), дозволяє, по-перше, зменшити навантаження на систему, завдяки участі в обробленні лише потрібної мережевої активності, а по-друге, покращить стабільність системи за рахунок мінімізації втручання в механізми ядра ОС.

Висновки

1. Досліджено актуальність розроблення комплексу контролю діяльності користувачів.
2. Запропоновано архітектурне рішення комплексу для моніторингу активності користувачів комп'ютерної мережі.
3. З урахуванням специфіки мережевої структури комплексу, доведено доцільність його побудови на основі архітектури *Multi-Agent System* (система з розподіленими клієнтськими та централізованою серверною частиною) та розподілу функціонального навантаження між клієнтськими модулями, які безпосередньо займаються збиранням інформації, та серверною підсистемою, відповідальною за авторизацію клієнтів, отримання даних та забезпечення їх архівації.
4. Для зберігання даних доцільно використати СУБД з відкритим кодом *PostgreSQL*, яка надає всі можливості для побудови високонавантаженого централізованого сховища інформації. Окрім використання СУБД, серверна частина дозволяє за допомогою системи плагінів зберігати інформацію в будь-яких інших форматах, або ж передавати далі, наприклад, на головний сервер організації.

Список літератури

1. Мельников В. Защита информации в компьютерних системах / В. Мельников. – М.: Финанси и статистика, 1997. – 368 с.

2. Герасименко В.А. Основи защиты информации / В.А. Герасименко, А.А. Малюк. – М.: МОПОРФ, МИФИ, 1997. – 537 с.

3. Немет Э. Руководство администратора *Linux* / Э. Немет, Г. Снайдер, Т. Хейн. – [2-е изд.]; пер. с англ. – М.: ООО "Вильямс", 2008. – 1072 с.

4. Вейрле К. *Linux*: сетевая архитектура. Структура и реализация сетевых протоколов в ядре / К. Вейрле, Ф. Пэльке. – М.: КУДИЦ-Образ, 2006. – 656 с.

5. Лав Р. Разработка ядра *Linux* / Р. Лав. – Пер. с англ. – М.: ООО "Вильямс", 2008. – 448 с.