УДК 004.42
Н.А. Христинець, О.І. Міскевич
Луцький національний технічний університет

SHELL-ПРОГРАМУВАННЯ ПРОЦЕСІВ ФОНОВОГО РЕЖИМУ В ОС LINUX

Христинець Н.А., Міскевич О.І. Shell-програмування процесів фонового режиму в ОС Linux. *В роботі проведено дослідження способів побудови архітектури мережених сервісів з використанням shell-інтерпретатора, що дозволяє керувати взаємодією між системою та користувачем.*
Ключові слова*: процес, фоновий режим, потоки, програмування, інтерпретатор, shell, Linux*

Христинец Н.А., Мискевич О.И. Shell-программирование процессов фонового режима в ОС Linux. *В работе проведено исследование способов построения архитектуры сетевых сервисов с использованием shell-интерпретатора, который позволяет управлять взаимодействием между системой и пользователем.*
Ключевые слова*: процесс, фоновый режим, потоки, программирование, интерпретатор, shell, Linux*

Hrystynets N., Miskevych O. Shell-programming processes of the background in OS Linux. *This paper studied the ways of constructing the architecture of network services using shell-interpreter which allows you to manage the interaction between the system and the user.*
Keywords: process*, background, streams, programming interpreter, shell, Linux*

Постановка проблеми загальному вигляді і її зв'язок з важливими науковими та практичними завданнями. Як відомо, для ефективної роботи комп'ютера потрібно постійно слідкувати за станом операційної системи, встановленої на ньому. Проте, навіть у тих випадках, коли ОС функціонує належним чином, її налаштування не завжди відповідають оптимальним. Тому час від часу для деяких користувачів постає питання оптимізації або тонкого налаштування системи. Отже, питання оптимізації системи є досить актуальною в наш час. Надзвичайно велика кількість сервісів, що виконуються в системі, надає ймовірності того, що якийсь з них почне працювати невірно.

Аналіз останніх досліджень, у яких запчатковано вирішення проблеми. У багатьох статтях різними авторами (Bozidar Levi, W. Richard. Stevens, С. Могильний та ін.) згадуються і навіть описуються способи побудови архітектури мережевих сервісів (демонів). При цьому мало у кого з авторів є реальний досвід створення та оптимізації демонів, що працюють з десятками тисяч одночасних з'єднань, або керують, наприклад, гігабітним трафіком.

Цілі статті. Метою статті є програмна організація способу побудови архітектури мереженого сервісу для ОС Linux за допомогою вбудованого інтерпретатора shell.

Виклад основного матеріалу дослідження. Крім вирішення проблем відслідковування сервісу екстенсивними методами в сучасних операційних системах (як то Linux) ця задача виконується в фоновому режимі, і при запуску системи стартує цілий пакет таких задач. Управління автоматизованими задачами, демони управління живленням і переметрами CPU, демони для друку, для ведення системних журналів – це далеко не весь перелік необхідних сервісів сучасних ОС. Так як компаніям-виробникам дистрибутивів важко дізнатись, що точно треба буде для роботи конкретного десктопу, вона намагається діяти по принципу надлишковості (тобто, зазвичай, кількість запропонованих сервісів значно перевищує необхідний мінімум). В результаті такого підходу комп`ютер може отримати демона управління живленням і частотою CPU для ноутбука або демоном для Bluetooth. Задіяність таких програм при роботі конкретного споживача може бути обмежена, а от об'єми пам`яті та ресурси процесора вони викорисровують.

Існує багато способів знайти і вимкнути непотрібні сервіси.

Найзручніше скористатися програмою *chkconfig*, що дозволяє редагувати необхідні сервіси. Linux, так само як і Unix, використовує різні рівні запуску для різноманітних сервісів. Для прикладу, рівень запуску 1 зазвичай використовується для аварійного завантаження в режимі однокористувацької системи. Це означає, що в ньому немає ні мережі, ні графічного робочого столу.

Перед тим як почати вимикати непотрібні сервіси, потрібно створити за допомогою нехитрої команди список сервісів, що були присутні до оптимізації (це звичайна застережливість на випадок некоректних подальших змін). Для цього слугує команда *chkconfig -A > services.save,* що перенаправить вивід команди *chkconfig* в файл з ім`ям *services.save*. Переглянувши його в текстовому редакторі, можна побачити, що він містить в точності те саме, що й вивід стандартної команди.

Отже, щоб вимкнути демон, потрібно виконати команду *chkconfig назва_демону off,* (ввімкнути – *chkconfig назва_демону on*)

Всі зміни будуть виконані лише після рестарту системи.

Якщо щось пішло не так, команда *chkconfig -s < services.save* – і вся підбірка сервісів, що була до цього, почне працювати.

Є ще один метод вимкнути непотрібне. Цей метод – програма *sysvconfig*. Sysvconfig – це утиліта, що має певний консольний інтерфейс і запускається з терміналу. Її по замовчуванні в дистрибутиві в основному немає, але її можна встановити командою *sudo aptitude install sysvconfig*.

Будь демон повинен приймати і обробляти мережеві з'єднання. Так як стек транспортних протоколів TCP/IP виріс із UNIX, а складовими таких ОС є файли, то можемо зробити висновок, що мережеві з'єднання є також файли, причому файли особливого типу, які можна відкривати, закривати, читати і писати стандартними функціями ОС для роботи з файлами.

Отже, насамперед, будь-який демон викликає системні функції:

socket ( ), *bind ( )*, а потім *listen ( )* і в підсумку отримує файл спеціального типу. Параметри цих функцій і подальші дії демона дуже сильно залежать від застосовуваного транспортного протоколу (TCP, UDP, ICMP, RPD).

Усе, що може відбуватися з файлом спеціального типу «той, що слухає сокет» - це періодично виникаючі події типу «запит вхідного з'єднання». Демон може прийняти таке з'єднання функцією *accept (),* яка створить новий файл, на цей раз вже типу «відкритий мережевий сокет». Імовірно, демон повинен прочитати з цього з'єднання запит, обробити його і відправити назад результат. При цьому, мережевий сокет – це вже більш-менш нормальний файл: хоч він і був створений не самим стандартним чином, принаймні з нього можна намагатися читати і писати дані. Але є й істотні відмінності від звичайних файлів, розташованих на файловій системі:

– всі події відбуваються дійсно асинхронно і з невідомої тривалістю за часом і будь-яка операція в гіршому випадку може зайняти десятки хвилин;

– з'єднання, на відміну від файлів, можуть закриватися «самі собою» в будь-який, найнесподіваніший момент;

– ОС не завжди повідомляє про що закрилося з'єднання, недіючі сокети можуть висіти по півгодини;

– з'єднання на клієнті і на сервері закриваються в різний час. Якщо клієнт спробує створити нове з'єднання і «довідправити» дані, можливо дублювання даних, а при неправильно написаному клієнті – і їх втрата. Також можлива наявність на сервері декількох відкритих з'єднань від одного клієнта;

– дані розцінюються як потік байтів і можуть буквально приходити порціями по 1 байту. Тому вважати їх, наприклад, рядками UTF- 8 не можна;

– ніяких буферів крім тих, які надав сам демон, в мережі немає. Тому запис в сокет навіть 1 байта може заблокувати демон на десятки хвилин;

– будь-які помилки можуть траплятися в будь-якому місці, демон повинен коректно обробляти їх всі.

Найпростіший спосіб не дати сервісам впливати один на одного – це запустити для кожного з них окремий процес (тобто окрему копію програми). Недоліки цього методу очевидні – запуск окремого процесу це дуже ресурсномістка операція. Процес у всіх ОС є одиницею обліку системних ресурсів – пам'яті, відкритих файлів, прав доступу, квот і так далі. Якщо створюється демон віддаленого доступу до операційної системи на зразок Shell або FTP - користувач просто зобов'язаний запускати окремий процес від імені кожного зареєстрованого користувача, щоб правильно враховувати права доступу до файлів. Аналогічно, на сервері shared-хостингу одночасно, на одному «фізичному» порту крутяться сотні сайтів різних користувачів – і процеси потрібні apache для того, щоб сайти одних користувачів хостингу не могли потрапити до даних інших користувачів. Використання процесів не дуже впливає на продуктивність apache, про що свідчать результати бенчмарк з урахуванням аналізу обробки графіків запитів

Для відносно невеликої кількості одночасних з'єднань багатопотокова архітектура – найкращий вибір. Але якщо сполук дійсно багато, скажімо десять тисяч, перемикання між потоками починає займати занадто багато часу. Але навіть це – не головний недолік багатопотокової архітектури.

Головний аспект полягає в тому, що потоки не незалежні і можуть один одного блокувати. Якщо два потоки почнуть одночасно виконувати операції $a = b + c$ і $b = c + a$, вони заблокують один одного назавжди. Така ситуація називається клінчем, пошук і дозвіл клінчів – окрема тема

паралельного програмування. Але й без клінчів потоки, якщо вони не знімають блокування швидко, можуть зупиняти один одного на досить тривалі проміжки часу.

Крім того, атомарні операції фізично реалізуються за допомогою монопольного захоплення шини ОЗП і самого ОЗП. Робота безпосередньо з пам'яттю, а не з кешем, дуже повільна сама по собі, а в даному випадку ще й викликає інвалідацію відповідних кеш-ліній всіх ядер всіх інших процесорів сервера. Тобто, навіть у кращому випадку, за відсутності блокувань, кожна атомарна операція виконується досить довго і погіршує продуктивність інших потоків.

- Змінні трапляються за таких умов:
- Загальна черга нових сполук ;
- Загальна черга доступу до бази даних або подібних ресурсів ;
- Загальна черга запитів на виділення пам'яті (так-так, malloc ( ) і new ( ) можуть викликати блокування) ;
- Загальний журнал (log -файл) і загальні об'єкти підрахунку статистики.

Це тільки найочевидніші.

У деяких випадках існують способи обійтися без загальних змінних. Наприклад, від блокувань черги нових сполук можна відмовитися, якщо наділити один з потоків функцією «диспетчера», який буде таким чином роздавати завдання заздалегідь. Іноді вдається застосувати спеціальні «неблокуючі» структури даних. Але вцілому проблема взаємних блокувань в багатопотокової архітектури не вирішена.

Написання демона можливе на будь-якій мові (звичайно, мається на увазі мови високого рівня), наприклад, C, Perl (і навіть PHP). Написати демон у вигляді shell-скрипта досить проблематично, але вирішення такого роду завдання є моментом вдосконалення розуміння як архітектури ОС в цілому, так і специфіки використання інтерпретатора, а також можливостей застосування великої кількості консольних утиліт.

Визначимо загальні принципи роботи демона.

Найперше, слід пам'ятати, що демон – це звичайна програма, що виконується у фоновому режимі. Але так як демон буде запускатися з init.d, то на нього накладаються певні обмеження :

1. Демон має зберегти свій PID у файл, для того щоб потім можна було його коректно зупинити.

2. Необхідно виконати ряд підготовчих операцій для початку роботи у фоновому режимі.

У запропонованій у цій статті моделі демон функціонуватиме за наступним алгоритмом:

1) Відділення від керуючого терміналу і перехід у фоновий режим.

2) Поділ на дві частини: батько (моніторинг) і нащадок (функціонал демона).

3) Моніторинг стану процесу демона.

4) Обробка команди оновлення конфігу.

5) Обробка помилок.

Запропоноване програмне вирішення проблеми складається з скрипта, який рекомендується запустити в фоновому режимі:

```
#!/bin/sh
set -euf
while true;do
logger -p local2.notice -f /etc/rsyslog/dmzfifo_access
done
```

Даний скрипт за допомогою програми *Logger* слухає *fifo* пайп *dmzfifo_access* і відправляє вміст з пайпа в *syslog*. Даний скрипт повинен знаходитись в фоновому режимі і постійно слухати пайп.

Наступним кроком є написаний скрипт для *init.d:*

```
#!/bin/sh
#
# processname: dmzfifo_accessd
DMZFIFO_BIN=/etc/rsyslog/dmzfifo_access.sh
KIND="DMZFIFO_ACCESSD"
```

```
. /etc/rc.d/init.d/functions
        start() {
                echo -n $"Starting $KIND services: "
                daemon /etc/rsyslog/dmzfifo_access.sh
                echo
        }
        stop() {
                echo -n $"Shutting down $KIND services: "
                killproc /etc/rsyslog/dmzfifo_access.sh
                echo
        }
        restart() {
                echo -n $"Restarting $KIND services: "
                killproc /etc/rsyslog/dmzfifo_access.sh
                daemon /etc/rsyslog/dmzfifo_access.sh
                echo
        }
        case "$1" in
        start)
                start
                ;;
        stop)
                stop
                ;;
        restart)
                restart
                ;;
        *)
                echo $"Usage: $0 {start|stop|restart}"
                exit 1
        esac
        exit $?
```

Дане вирішення проблеми організації роботи процесів в фоновому режимі може бути вдосконалене, залежно від потреб користувача.

Висновки. Навіть якщо користувачі запрограмованих процесів фонового режиму не зареєстровані в операційній системі, але до програмного сервісу висуваються підвищені вимоги з безпеки, то в загальному виділення окремого процесу під кожного користувача є дуже розумним архітектурним рішенням. Це не дозволить користувачам отримувати або блокувати доступ до даних інших користувачів, навіть якщо вони знайдуть баг у написаному демоні, що дозволяє читати чужі дані або просто руйнуючий процес. Нарешті, у кожного процесу мається власний адресний простір, і різні процеси не заважають один одному користуватися пам'яттю. Тобто, особливості написання Shell-додатків для процесів фонового режиму в ОС Linux, є сучасним вирішенням проблем організації роботи процесів.

1. Томас Р., Йейтс Дж. Операционная система UNIX. Руководство для пользователей. – М.: Радио и связь, 1986. -352 с.
2. Банахан М., Раттер Э. Введение в операционную систему UNIX. -М.: Радио и связь, 1986. -341 с.
3. Тихомиров В.П., Давидов М.И. Операционная система UNIX: Инструментальные средства программирования. -М.: Финансы и статистика, 1988. -206 с.
4. Баурн С. Операционная система UNIX. -М.: Мир, 1986. -462 с.
5. http://www.citforum.ru/ftp/pub/os/shell-win.zip - Інтерпрітатор командного мови Shell