Зміст

[Зміст 3](#_Toc467182946)

[Вступ 4](#_Toc467182947)

[1. Мета курсової роботи 6](#_Toc467182948)

[2. Постановка задачі 7](#_Toc467182949)

[3. Опис етапів реалізації 8](#_Toc467182960)

[4. Опис програмного продукту 12](#_Toc467182961)

[5. Вид екранну монітору, як результат роботи кожного завдання 18](#_Toc467182962)

[6. Інструкція з експлуатації 22](#_Toc467182963)

[Загальні висновки 23](#_Toc467182964)

[Література 24](#_Toc467182964)

**Вступ**

Багатопоточність є природним продовженням багатозадачності, точно також як віртуальні машини, що дозволяють запускати кілька ОС на одному комп'ютері, є логічний розвиток концепції поділу ресурсів. В рамках неформального, але простого визначення, потік - це виконання послідовності машинних інструкцій. У багатопотоковому додатку одночасно працює кілька потоків.

Перш ніж приступати до програмування потоків, слід відповісти на питання, а чи потрібні вони вам. Ми вже знаємо, наскільки добре розвинені в Linux кошти взаємодії між процесами. За допомогою управління процесами в Linux можна вирішити багато завдань, які в інших ОС вирішуються тільки за допомогою потоків. Потоки часто стають джерелами програмних помилок особливого роду. Ці помилки виникають при використанні потоками поділюваних ресурсів системи (наприклад, загального адресного простору) і є окремим випадком більш широкого класу помилок - помилок синхронізації. Якщо завдання розділена між незалежними процесами, то доступом до їх загальних ресурсів управляє операційна система, і ймовірність помилок через конфлікти доступу знижується. Втім, поділ завдання між декількома незалежними процесами само по собі не захистить вас від інших різновидів помилок синхронізації. На користь потоків можна вказати те, що накладні витрати на створення нового потоку в багатопотоковому додатку нижче, ніж накладні витрати на створення нового самостійного процесу. Рівень контролю над потоками в багатопотоковому додатку вище, ніж рівень контролю додатки над дочірніми процесами. Крім того, багатопотокові програми не схильні залишати за собою низки зомбі або «осиротілих» незалежних процесів.

Перша підсистема потоків в Linux з'явилася близько 1996 року і називалася, без зайвих викрутасів, - LinuxThreads. Рудимент цієї підсистеми, який ви знайдете в будь-якій сучасній системі Linux, - файл /usr/include/pthread.h, вказує рік релізу - 1996 і ім'я розробника - Ксав'є Лерой (Xavier Leroy). Бібліотека LinuxThreads була спробою організувати підтримку потоків в Linux в той час, коли ядро ​​системи ще не надавало жодних спеціальних механізмів для роботи з потоками. Пізніше розробку потоків для Linux вели відразу дві конкуруючі групи - NGPT і NPTL. У 2002 році група NGPT фактично приєдналася до NPTL і тепер реалізація потоків NPTL є стандартом Linux. Підсистема потоків Linux прагне відповідати вимогам стандартів POSIX, так що нові багатопотокові програми Linux повинні без проблем компілюватися на нових POSIX-сумісних системах.

1. **Мета курсової роботи**

**Варіант 11**

1. Одержати|отримати| наступну|слідуючу| системну інформацію:

* ім'я операційної системи, номер версії і модифікації ядра;
* інформацію про платформу, на якій працює система.

1. Використовуючи Системний монітор KDE:

* проаналізувати завантаження|загрузку| системи за одну хвилину;
* визначити тактову частоту процесора;
* відобразити|відображувати| дані про простої в роботі процесора.

1. Продемонструвати взаємодію батьківського і дочірнього процесів за допомогою каналу.
2. **Постановка задачі**

Для досягнення поставленої мети необхідно виконати наступні дії:

1. Завантажити утілиту Dialog
   * Ознайомитися з її використанням
   * Створити необхідне меню для управління майбутньою програмою
2. Завантажити системний монітор Ksysguard
3. Розглянути роботу каналів
   * Продемонструвати взаємодію батьківського і дочірнього процесів за допомогою каналу
4. Повторити пройдений матеріал по використанню shell-команд
5. **Опис етапів реалізації**

Для реалізації інтерфейсу програмного продукту була використана утілита Dialog.

Утиліта dialog – одна з небагатьох, що допомагають створювати доброзичливі до користувача (у плані користувальницького інтерфейсу) скрипти та програми. Вона може конструювати різні діалогові вікна, не потребуючи при цьому в запущеному графічному інтерфейсі - достатньо лише консолі, адже для малювання використовується бібліотека ncurses. Як наслідок, dialog добре пристосований для використання в shell-скриптах, в них він і застосовується в більшості випадків. Короткий список діалогових вікон, які здатний малювати dialog:

* + вікна з кнопками Так / Ні;
  + меню;
  + вікна з полями введення;
  + інформують вікна і просто вікна з текстом;
  + поля з радіокнопка;
  + вікно для вибору файлів;
  + вікна для вибору необхідного часу, дати,
  + спеціальне вікно для запиту пароля.

Для виведення необхідних датчиків був використаний системний монітор Ksysguard.

KSysGuard (KDE System Guard) - розширювана плагінами QT / KDE графічний менеджер завдань і монітор продуктивності для середовища робочого столу KDE (KDE Applications).

KSysGuard має простий і частково настроюється призначений для користувача інтерфейс, використовує архітектуру клієнт-сервер, що дозволяє контролювати локальні і мережеві комп'ютери. Установки і функціонал "за замовчуванням" достатні для базового управління системою і задовольнять потреби більшості користувачів (досвідчені користувачі і системні адміністратори мають інші потреби і інструменти).

Робочий простір KSysGuard організовано у вигляді вкладок, для кожного виду інформації можна створити одну або кілька вкладок. Вкладки можна заповнити екраном з одного або декількох датчиків (перетягуванням з древа датчиків), вкладки можуть бути збережені і завантажені незалежно один від одного.

Глобальні налаштування KSysGuard задають основні параметри для нових вкладок, властивості кожної вкладки і відображення датчиків мають індивідуальні настройки. Значення датчиків відображаються різними кольорами і з певним інтервалом, при достатньому розмірі екрану відображається сітка, діапазон графіка даних "за замовчуванням" встановлюється автоматично (в "властивостях" можна задати самому).

Щоб продемонструвати взаємодію батьківського та дочірнього процесу за допомогою каналів були реалізовані додаткові функції reader та writer. **Термін pipe (труба) надзвичайно органічно увійшов до англомовного комп'ютерного жаргону. Цим словом називається не тільки спосіб передачі виведення однієї команди на введення інший, але і оператор, яким позначається ця дія: | (вертикальна межа). Крім того, те ж слово служить дієсловом, що означає дану дію.**

**Програмним каналом називається використання виведення однієї команди як введення для іншої програми. Наприклад:**

**$ dmesg | less**

**Команда dmesg виводить повідомлення ядра Лінукс про процес завантаження ОС (ті самі, що пробігають по екрану монітора при завантаженні системи). Ці повідомлення не уміщаються на одному екрані, і пролітають так швидко, що прочитати їх неможливо. Тому виведення програми dmesg передають на введення команді less. (Команда less дозволяє виведенню команди dmesg заповнити тільки один екран. Щоб прочитати наступну порцію тексту, потрібно натиснути клавішу пропуску, а щоб повернутися до попередньої порції - клавішу b. Перервати роботу програми можна клавішею q). Оператором такої передачі служить вертикальна межа (|). (Пропуски до і після вертикальної межі ставляться для зручності читання, але можна обійтися і без них). Всі разом і є простий програмний канал.**

**Того ж результату можна досягти, якщо спочатку перенаправити виведення команди dmesg в тимчасовий файл, а потім проглянути вміст цього файлу на екрані монітора.**

**$ dmesg > временний.файл**

**$ временний.файл > less**

**Очевидно, що така схема менш продуктивна: по-перше, необхідно давати дві команди, по-друге тому, що наступна команда може почати працювати тільки після завершення першої.**

**Необхідно пояснити поняття, які я будучи схожим назвав "введенням" і "виведенням" програми.**

**Будь-яка програма командної оболонки (шелла) оперує з трьома потоками даних: стандартним введенням (stdin), стандартним виводом (stdout), і стандартним повідомленням про помилку (stderr).**

**За умовчанням, стандартне введення здійснюється з клавіатури, а стандартний вивід - на екран монітора. Якщо ж ми задіюємо оператора програмних каналів (|), то стандартне виведення першої програми стане стандартним введенням другої, при цьому на екрані монітора він вже не з'явиться.**

**Такий ланцюжок зовсім не обмежується двома програмами, але може продовжуватися скільки завгодно довго.**

**Англійська назва іменованого каналу - named pipe або FIFO (File In, File Out - файл прийшов, файл пішов). Іменовані канали служать в основному для межпроцессного взаємодії, коли різні процеси в системі обмінюються інформацією. Тема це складна і велика, заслуговуюча окремої статті. Тому в даній роботі я тільки коротко торкнуся її.**

**На відміну від анонімного програмного каналу, автоматично створюваного шеллом, іменований канал володіє ім'ям, і створюється явно за допомогою команд mknod або mkfifo. Створимо іменований канал fifo1:**

**$ mkfifo fifo1**

**Тепер запустимо процес, що звертається до даного каналу:**

**$ grep fs < fifo1 (ENTER)**

**Не дивлячись на натиснення клавіші ENTER нічого не відбувається, що не дивно, адже файл fifo1 поки порожній, і команді grep нічого обробляти. Проте консоль виявляється зайнята процесом, що чекає, і розблоковувати її можна тільки перервавши процес (скажімо, натисненням клавіш Ctrl+c).**

**Щоб наповнити іменний канал вмістом, потрібно щоб до нього звернувся другий процес. Для цього ми повинні відкрити другу консоль і запустити яку-небудь команду, передавальну дані у файл fifo1. Наприклад:**

**(Інша консоль)$ ls /etc > fifo1**

**Негайно в першій консолі спрацює команда grep:**

**$ grep fs < fifo1 (ENTER)**

**fstab**

**gettydefs**

**login.defs**

**mke2fs.conf**

**Абсолютно ясно, що користуватися таким незручним механізмом в призначених для користувача цілях ніхто не буде, адже набагато простіше запустити один програмний канал:**

**$ ls /etc | grep fs**

**і отримати той же результат.**

**4.Опис програмного продукту**

**start.sh**

#!/bin/bash

dialog --title "Title stage" --clear \

--msgbox "

Course work

\nfrom Paralle and Multithreading programming

\n student of 333a course

\n by Anton Rynkovoy " 9 50

#USERDATE=`$DIALOG --stdout --title "Календарь" --calendar "Выберите дату..." 00 7 7 1981`

# Запоминаем нажатую кнопку

button=$?

#Обрабатываем нажатую кнопку

case $button in

0)sh password

;;

esac

clear

**password.sh**

#!/bin/bash

until false #Создаем безконечный цыкл

do

p\_file=`mktemp 2>/dev/null` #Создаем временный файл и записываем путь к нему в переменную

trap "rm -f $p\_file" 0 1 2 5 15 #Удаляем файл если получен один из сигналов

#Запускаем диалог

dialog --title "Your password" \

--clear \

--insecure \

--passwordbox "Enter the password:" 10 30 2> $p\_file #Запись в файл

val=$? #Считываем то что нажимаем

DATE=`/bin/date '+%d.%m.%Y'`

case $val in

0)

if [ "$(cat $p\_file)" = $DATE ]

#Если нажали ОК то сравниваем пароль с введенным

then sh tasks

break

fi

;;

1) break;;

255) break;;

esac

done

**task.sh**

#!/bin/bash

# Запись путей временных файлав в переменные

choice=`mktemp 2>/dev/null`

# Удаление временых файлов при получени одного из сигналов

dialog --title "Choose the task" \

--clear \

--menu "Choose one of the task:" 10 105 3 \

"1" "Get system infomation" \

"2" "Launch system monitor KDE" \

"3" "Show work of channels" 2> $choice

# Запоминаем нажатую кнопку

button=$?

#Обрабатываем нажатую кнопку

case $button in

0)

case $(cat $choice) in

1) #Выбрано задание №1

./sysinfo 1> info && dialog --title "System information" \

--textbox ./info 10 70

sh tasks

rm info

;;

2) #Выбрано задание №2

ksysguard sysmonitor.sgrd

sh tasks

;;

3) #Выбрано задание 3№

dialog --inputbox "Enter the string for writing into the channel:" 10 40 2> string

click=$?

case $click in

0) str=`cat string`

rm string

./channel "$str" 1> cinf && dialog --title "Channel work" \

--textbox ./cinf 15 70

rm cinf

sh tasks ;;

1) break ;;

255) break ;;

esac ;;

esac ;;

1) break ;;

255) break ;;

esac

clear

**sysinfo.c**

#include<stdio.h>

#include<stdlib.h>

#include<sys/utsname.h>

int main()

{

struct utsname u;

uname(&u);

printf("Operating System: %s\n",u.sysname);

printf("Version: %s\n",u.release);

printf("Modification: %s\n",u.version);

printf("Information about the platform: %s \n",u.machine);

return 0;

}

**channel.c**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<sys/types.h>

#include<sys/wait.h>

void reader(FILE \*kanal)

{

char buf[1024];

while(!feof(kanal) && fgets(buf,sizeof(buf),kanal)!=NULL)

fputs(buf,stdout);

}

void writer(FILE \*kanal,char \*m)

{

fprintf(kanal,"%s\n",m);

fflush(kanal);

}

int main(int argc,char \*argv[])

{

if (argc!=2)

{

printf("No string for writing\n");

exit(1);

}

int f[2];

pid\_t pid;

pipe(f);

pid=fork();

if (pid!=0)

{

printf("\n\n\n\n!!!Parent channel with id %d is working...\n", pid);

close(f[0]);

FILE \*kanal=fdopen(f[1],"w");

printf("Writing string into the channel: %s",argv[1]);

writer(kanal,argv[1]);

close(f[1]);

wait(&pid);

printf("\nParent process completed its work\n");

}

else

{

sleep(1);

printf("\n!!!Subsidiary channel with is working...\n");

close(f[1]);

FILE \*kanal=fdopen(f[0],"r");

printf("Read the string from the channel: ");

reader(kanal);

close(f[0]);

printf("Subsidiary process completed its work");

}

return 0;

}

1. **Вид екрану монітора, як результат роботи кожного завдання курсової роботи**

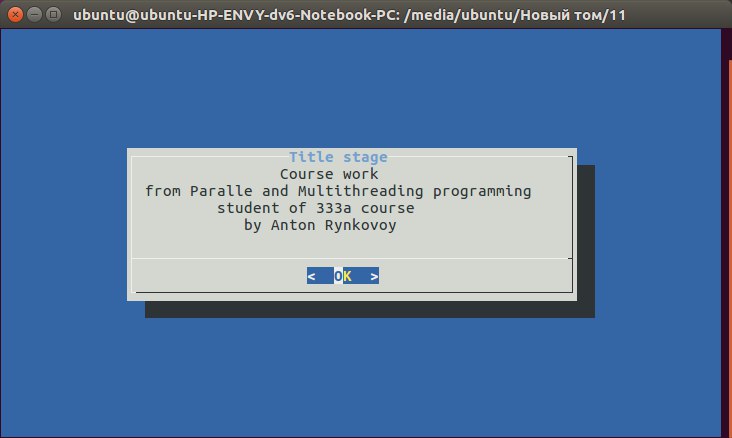


Рис. 5.1 – Титульне вікно

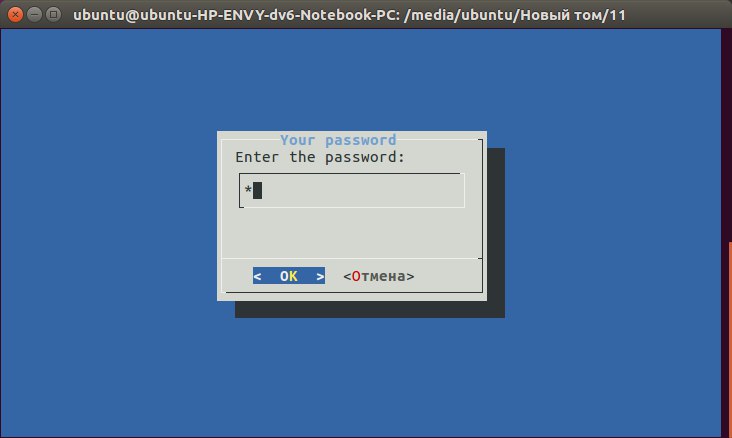


Рис. 5.2 – Вікно для вводу пароля

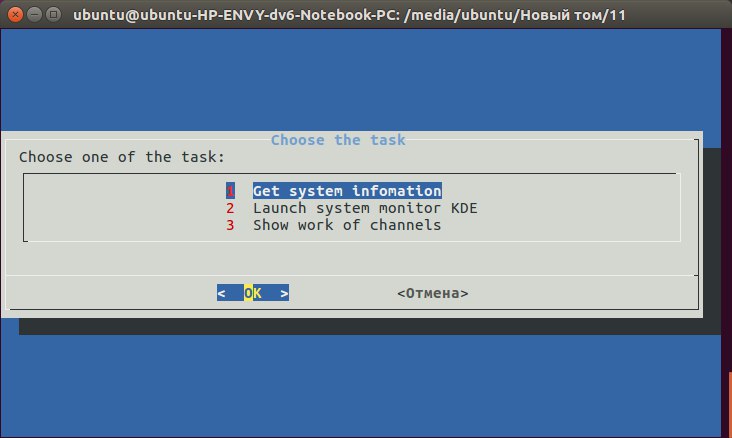


Рис. 5.3 – Вікно пунктів меню із завданнями

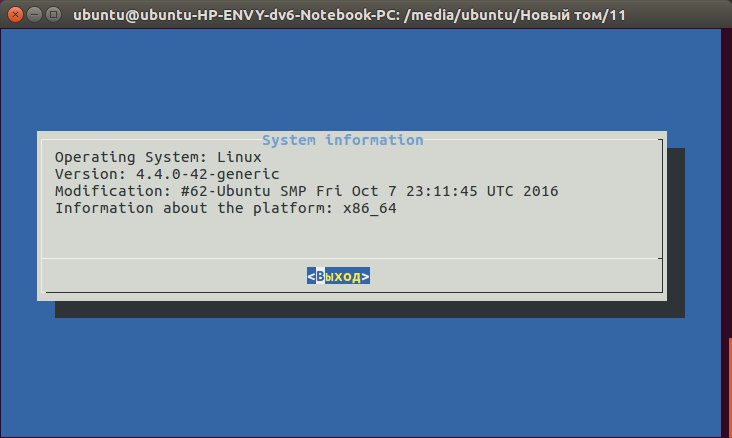


Рис. 5.4 – Вікно з системною інформацією

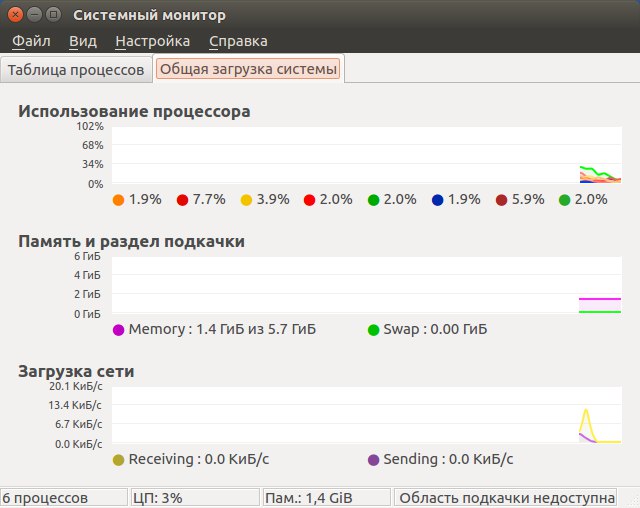


Рис. 5.5 – Вікно системного монітору KDE

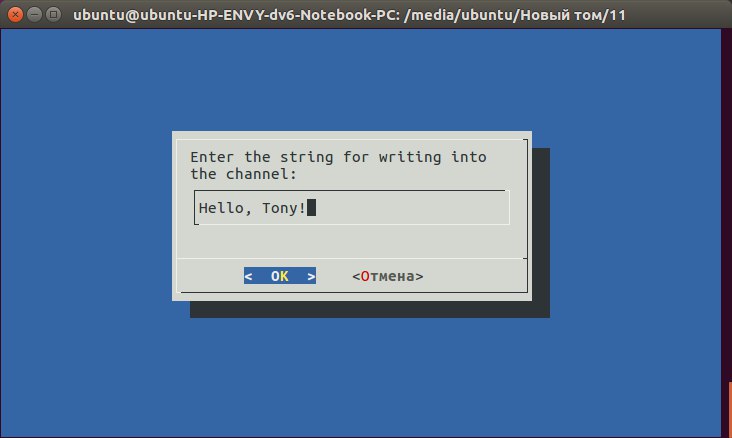


Рис. 5.6 – Вікно вводу строки у канал

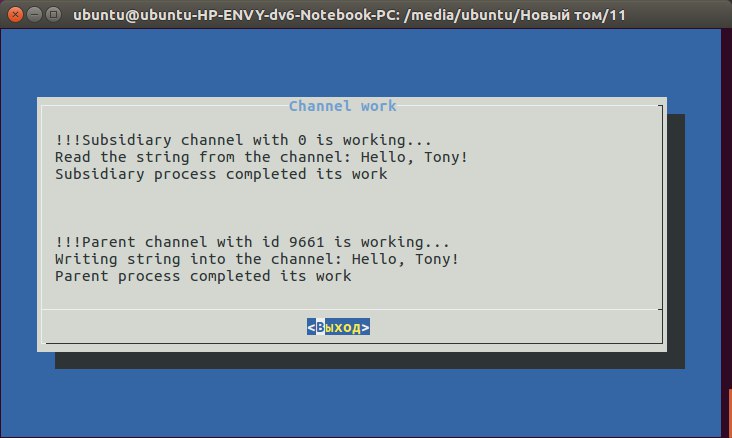


Рис. 5.6 – Вікно демонстрації роботи каналу

1. **Інструкція з експлуатації**

При відкритті програми (запуску файлу з назвою start ) ми побачимо вікно вітання.

В головному меню програми є три пункти:

1. Одержати наступну теоретичну інформацію

2. Виклик системного монітору

3. Взаємодія батьківського і дочірнього процесів за допомогою каналів

При натисканні першого пункту відкривається діалогове меню в якому можна подивитися:

* ім'я операційної системи, номер версії і модифікації ядра;
* інформацію про платформу, на якій працює система.

При відкритті другого пункту активується команда ksysguard, котра викликає системний монітор Лінукс, а в слід за нею відкривається заздалегідь заготовлений файл у котрому збережені потрібні нам датчики.

При натисканні на третій пункт програми відкривається вікно з повідомленнями по яким ми розуміємо, що канали успішно передали інформацію у процесах.

**Загальні висновки**

У ході виконання курсової роботи, згідно із поставленим завданням, було розроблено додаток із графічним інтерфейсом, виконаним за допомогою утілити Dialog, що дозволяє реалізувати наступне:

* вікно
* титул вікон
* поле для вводу тексту
* поле виводу тексту
* поле для вводу паролю
* діалогові кнопки

Був зроблений аналіз завантаження системи. Було визначено тактову частоту процесора. Реалізована демонстрація взаємодії батьківського і дочірнього процесу за допомогою каналу.

Таким чином, мета, поставлена на курсове проектування, була виконана у повному обсязі.

**Література**

1. Гордеев А. В.,. Молчанов А. Ю Системное программное обеспечение: учебник для вузов. – СПб.: Питер, 2003. – 736 с.
2. Шеховцов В. А. Операційні системи. –К.: Видавнича група BHV, 2005. – 576 с.
3. Митчелл М., Оулдем Д., А. Самьюэл А. Программирование для Linux. Профессиональный подход.: Пер. с англ. – М.:Издательский дом «Вильямс», 2004. – 288 с.
4. Роббинс А. Linux. Программирование в примерах. Пер. с англ. – М.: КУДИЦ-ОБРАЗ, 2005. – 656 с.
5. Галатенко В. А. Программирование в стандарте POSIX: Курс лекций:: Учеб. пособие –М.: Интернет-университет информ. технологий, 2004. – 560 с.
6. Секунов Н. Ю. Программирование на С++ в Linux. –СПб.:БХВ, 2004. – 368 с.
7. Таненбаум Э. Современные операционные системы. 2-е изд. \_ СПб.: Питер, 2002. – 1040 с.
8. Робачевский А. М. Операционная система UNIX. –СПб.: БХВ-Петербург, 2002. – 528 с.
9. Колисниченко Д. Н. Linux. Установка. Настройка. Использование. – 2-е изд., перераб. и доп. - –СПб.: Наука и Техника, 2004. – 640 с.