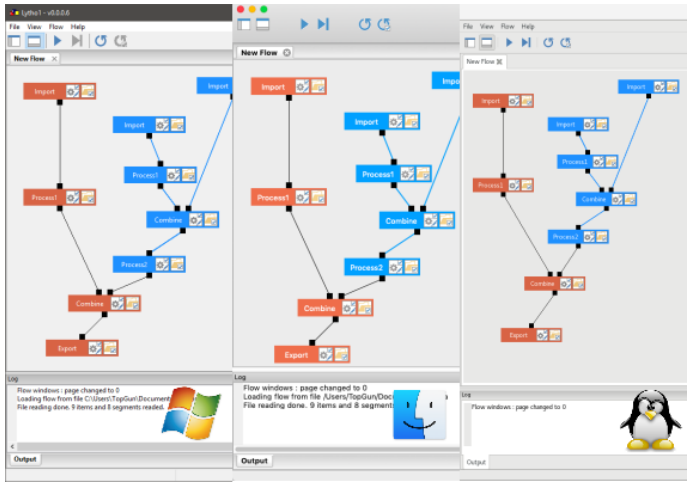
Национальный исследовательский университет «МИЭТ»

Институт интегральной электроники (группы ЭН-34-35, каф. ПКИМС)

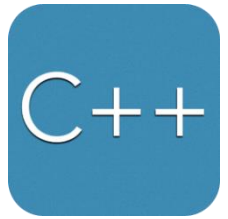
Кроссплатформенная разработка программного обеспечения



Лабораторная работа №2

Нативные API. XWS. Основы GTK

Сборка консольных проектов в Linux (1)



gcc/main.cpp

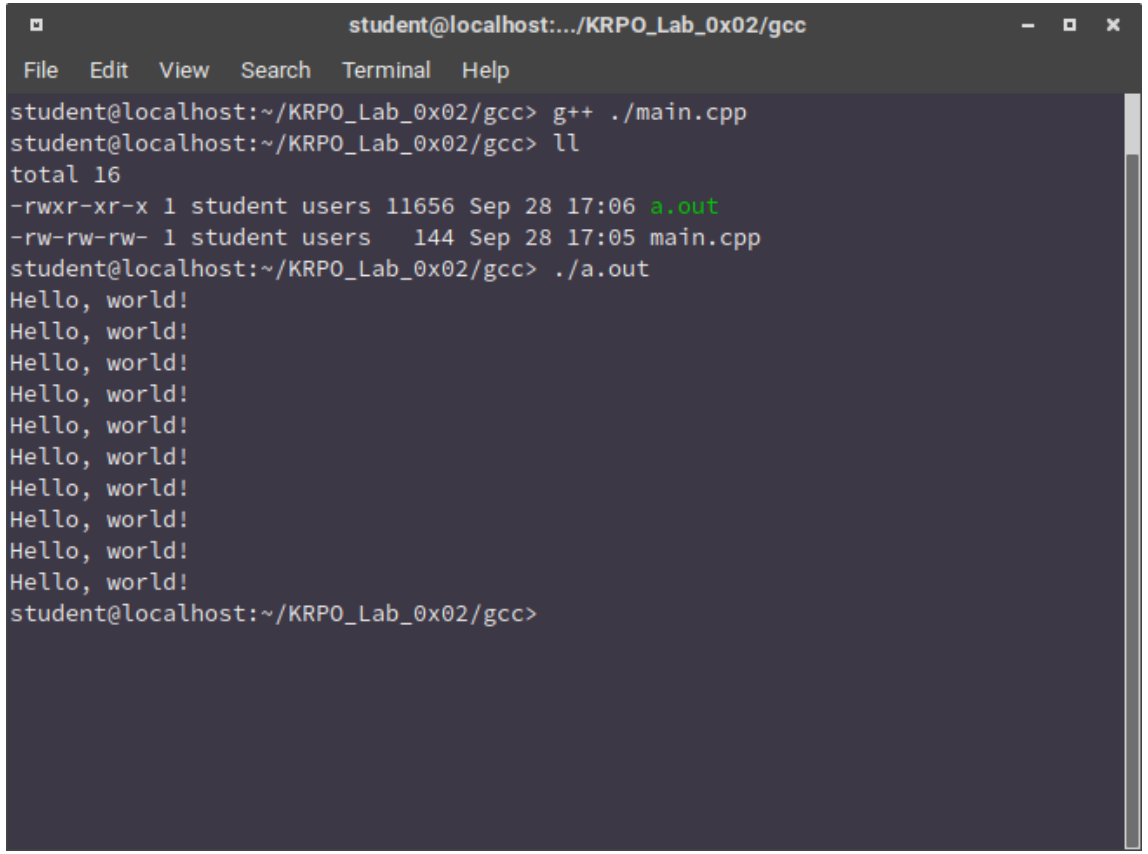
```cpp
#include <stdio.h>

#define N 10

int main(int argc, char *argv[]) {
  for (int i = 0; i < N; ++i)
    printf("Hello, world!\n");
  return 0;
}
```

Компиляция:

```
g++ ./main.cpp
```

Запуск:

```
./a.out
```



Сборка консольных проектов в Linux (2)

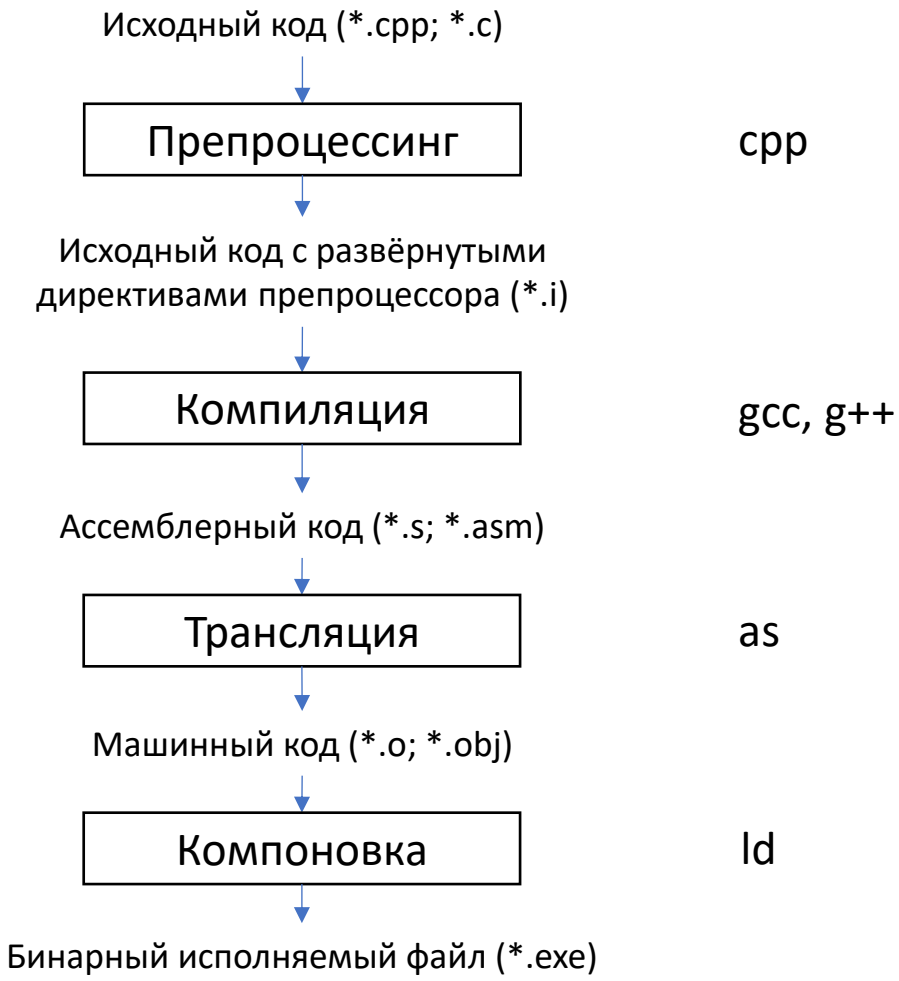
```c
#include <stdio.h>

#define N 10

int main(int argc, char *argv[]) {
  for (int i = 0; i < N; ++i)
    printf("Hello, world!\n");
  return 0;
}
```

Компиляция:

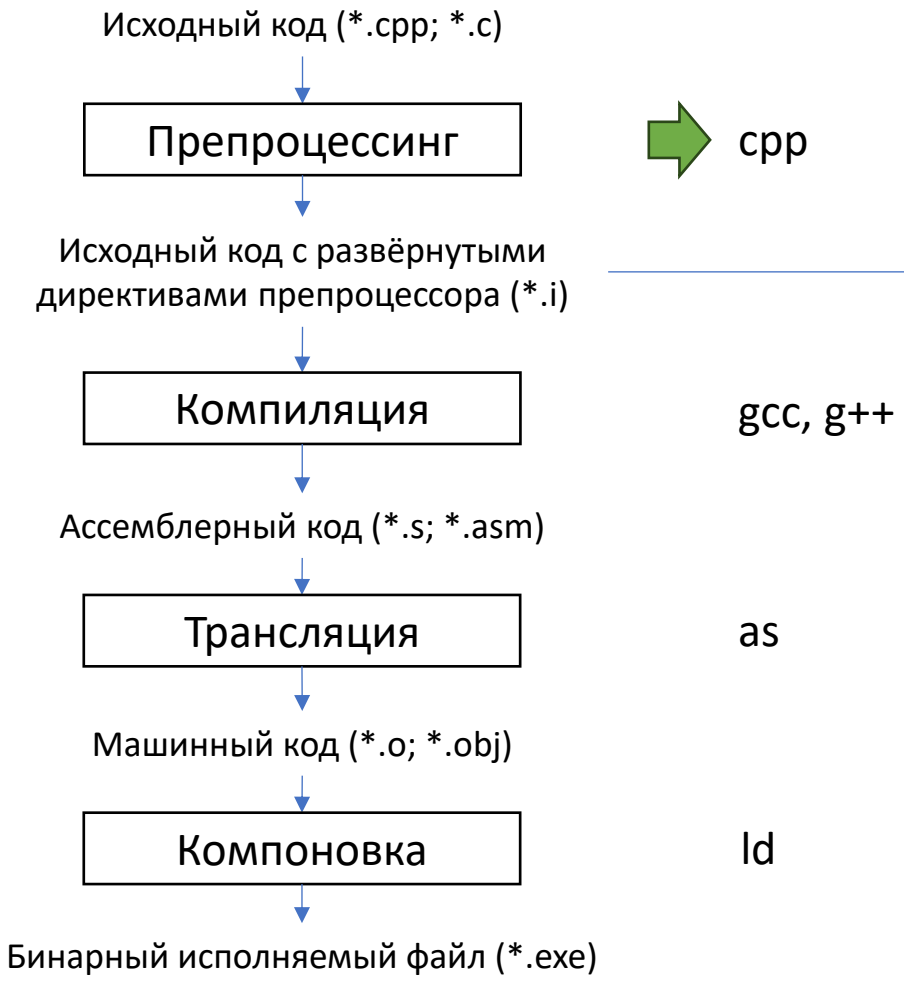
```
g++ -o main ./main.cpp
```

Запуск:

```
./main
```



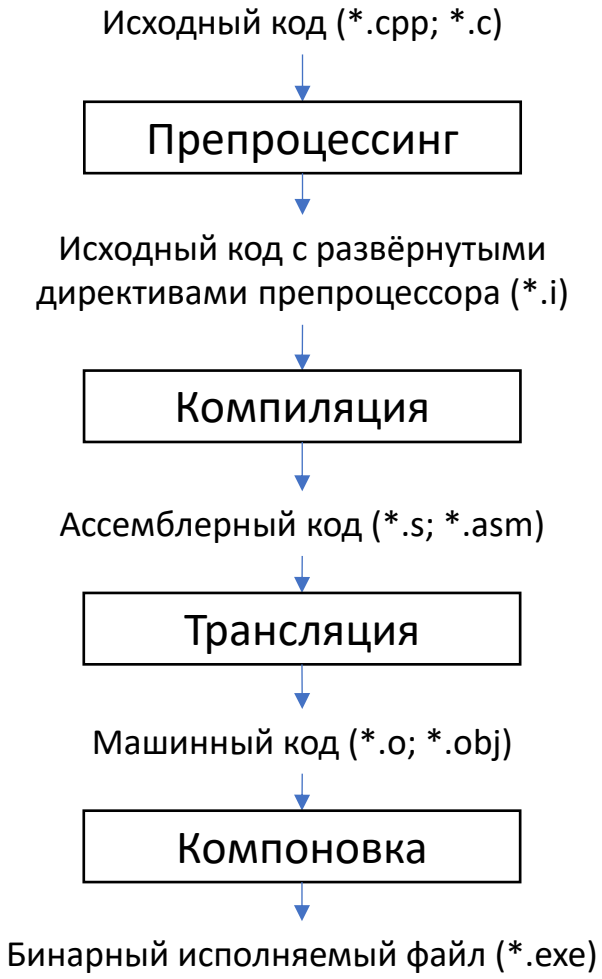
Сборка консольных проектов в Linux (3)




```c
#include <stdio.h>

#define N 10

int main(int argc, char *argv[]) {
  for (int i = 0; i < N; ++i)
    printf("Hello, world!\n");
  return 0;
}
```

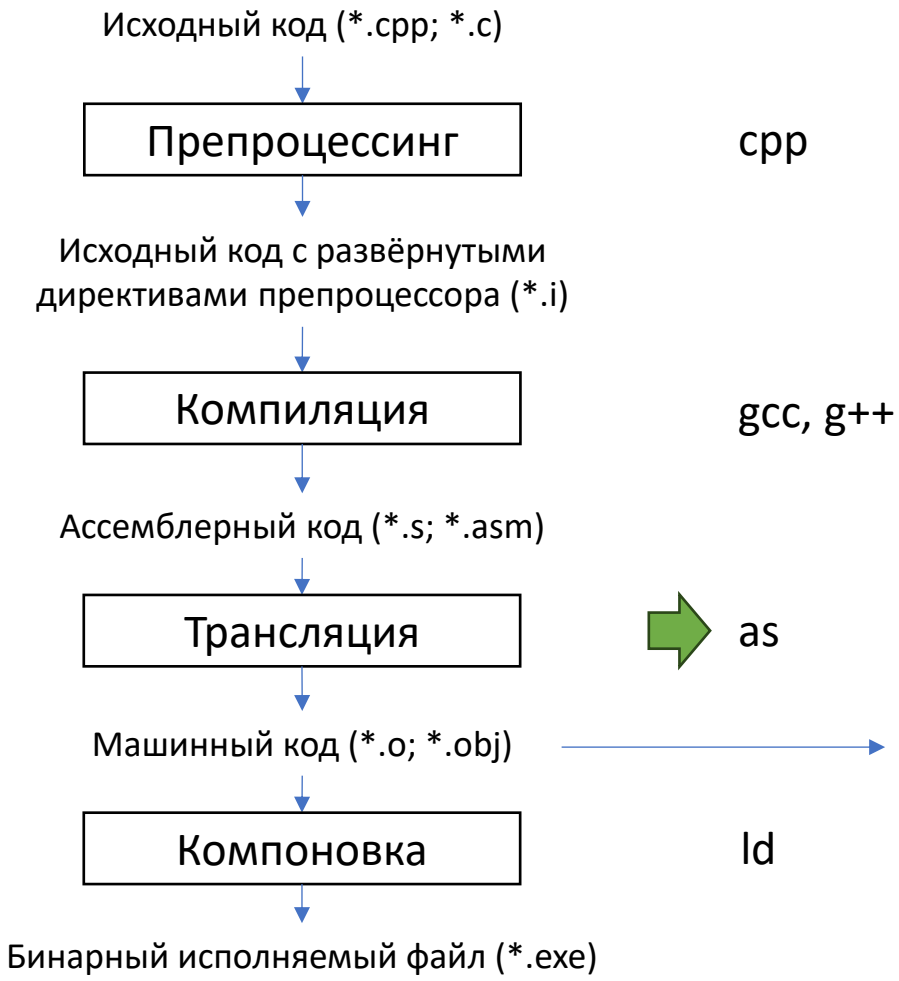
```
870   extern int ftrylockfile (FILE *__stream) throw () ;
871
872   extern void funlockfile (FILE *__stream) throw ();
873   # 858 "/usr/include/stdio.h" 3 4
874   extern int __uflow (FILE *);
875   extern int __overflow (FILE *, int);
876   # 873 "/usr/include/stdio.h" 3 4
877   }
878   # 2 "./main.cpp" 2
879
880   # 5 "./main.cpp"
881   int main(int argc, char *argv[]) {
882     for(int i = 0; i < 10; ++i)
883       printf("Hello, world!\n");
884     return 0;
885   }
886
```

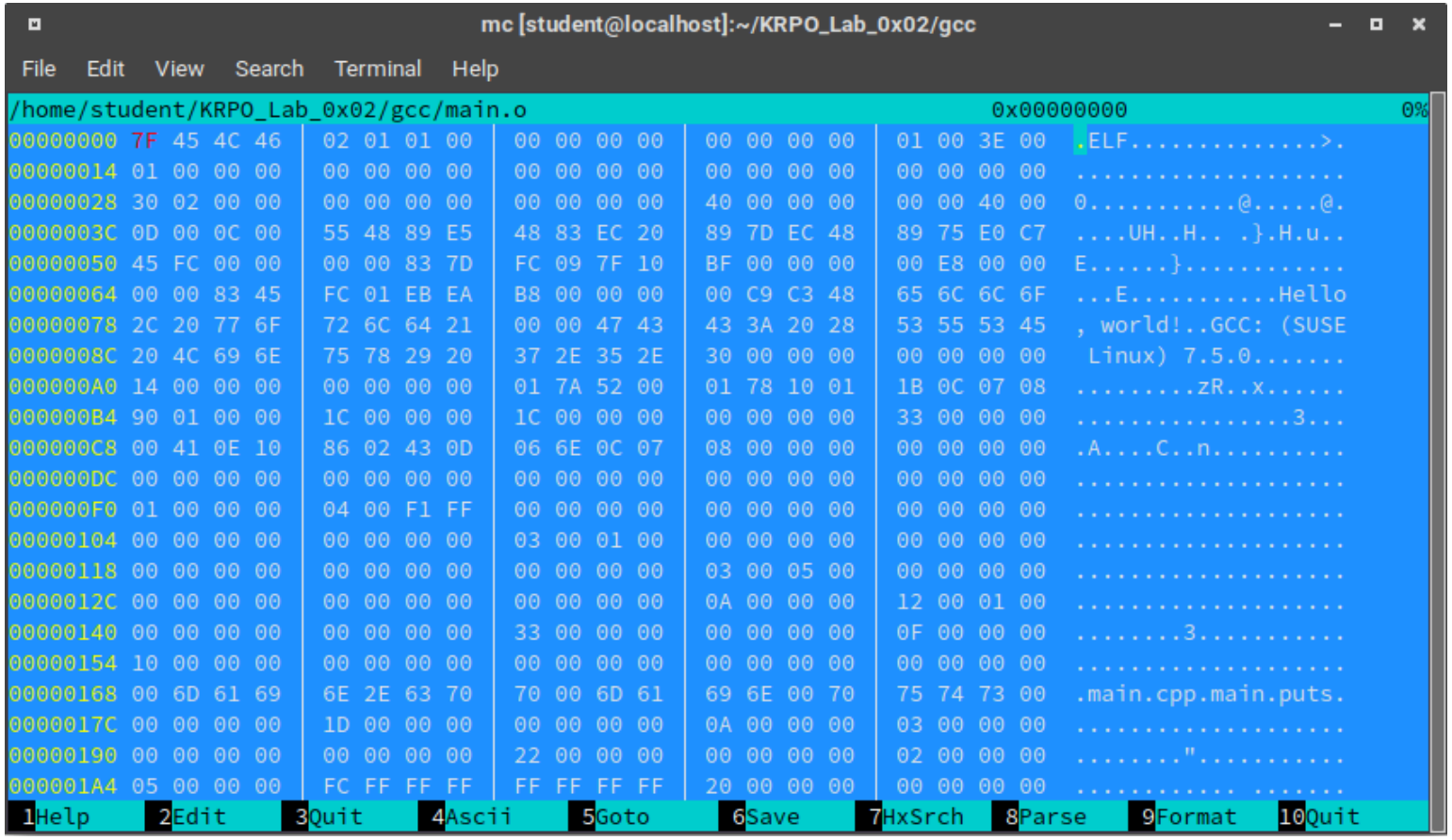
Сборка консольных проектов в Linux (4)



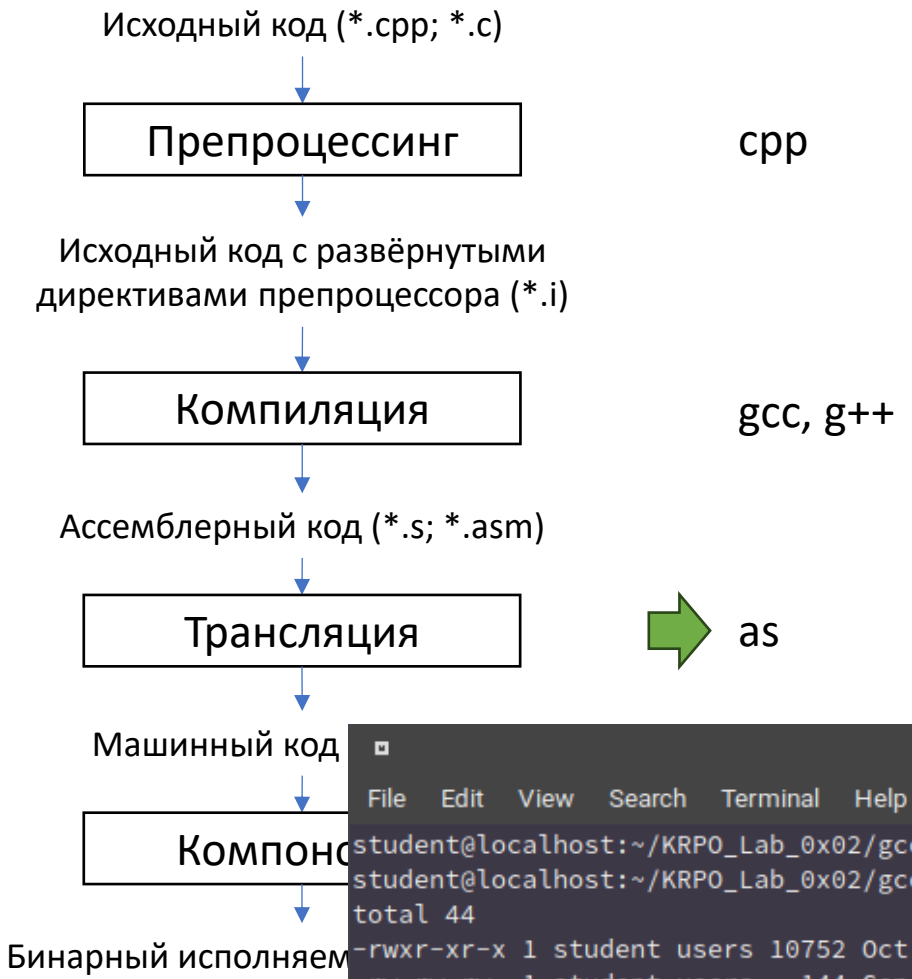

```
.file           "main.cpp"
.text
.section        .rodata
.LC0:
.string         "Hello, world!"
.text
.globl          main
.type           main, @function
main:
.LFB0:

.cfi_startproc
pushq           %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq            %rsp, %rbp
.cfi_def_cfa_register 6
subq            $32, %rsp
movl            %edi, -20(%rbp)
movq            %rsi, -32(%rbp)
movl            $0, -4(%rbp)
.L3:
cmpl            $9, -4(%rbp)
jg              .L2
movl            $.LC0, %edi
call            puts
addl            $1, -4(%rbp)
jmp             .L3
.L2:
movl            $0, %eax
leave
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE0:
.size           main, .-main
.ident          "GCC: (SUSE Linux) 7.5.0"
.section        .note.GNU-stack,"",@progbits
```

Сборка консольных проектов в Linux (5)





Сборка консольных проектов в Linux (6)



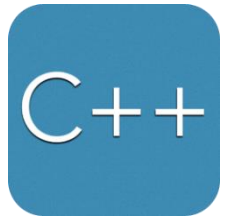

```
student@localhost:.../KRPO_Lab_0x02/gcc
File  Edit  View  Search  Terminal  Help
student@localhost:~/KRPO_Lab_0x02/gcc> ld -o main ./main.o
ld: warning: cannot find entry symbol _start; defaulting to 00000000004000b0
ld: ./main.o: in function `main':
main.cpp:(.text+0x22): undefined reference to `puts'
student@localhost:~/KRPO_Lab_0x02/gcc>
```

```
student@localhost:.../KRPO_Lab_0x02/gcc
File  Edit  View  Search  Terminal  Help
student@localhost:~/KRPO_Lab_0x02/gcc> ld --dynamic-linker /lib64/ld-linux-x86-64.so.2 /usr/lib64/crt1.o /usr/lib64/crti.o /usr/lib64/cr
student@localhost:~/KRPO_Lab_0x02/gcc> ll
total 44
-rwxr-xr-x 1 student users 10752 Oct  1 15:42 main
-rw-rw-rw- 1 student users   144 Sep 28 17:05 main.cpp
-rw-r--r-- 1 student users 17539 Sep 28 17:11 main.i
-rw-r--r-- 1 student users  1392 Sep 28 17:20 main.o
-rw-r--r-- 1 student users   577 Sep 28 17:15 main.s
student@localhost:~/KRPO_Lab_0x02/gcc> ./main
Hello, world!
```

Сборка консольных проектов в Linux (7)

```
student@localhost:.../KRPO_Lab_0x02/gcc
File  Edit  View  Search  Terminal  Help
student@localhost:~/KRPO_Lab_0x02/gcc> ld --dynamic-linker /lib64/ld-linux-x86-64.so.2 /usr/lib64/crt1.o /usr/lib64/crti.o /usr/lib64/crtn.o -lc -o main ./main.o
student@localhost:~/KRPO_Lab_0x02/gcc> ll
total 44
-rwxr-xr-x 1 student users 10752 Oct  1 15:42 main
-rw-rw-rw- 1 student users   144 Sep 28 17:05 main.cpp
-rw-r--r-- 1 student users 17539 Sep 28 17:11 main.i
-rw-r--r-- 1 student users  1392 Sep 28 17:20 main.o
-rw-r--r-- 1 student users   577 Sep 28 17:15 main.s
student@localhost:~/KRPO_Lab_0x02/gcc> ./main
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
student@localhost:~/KRPO_Lab_0x02/gcc>
```

Сборка многофайловых консольных проектов в Linux (1)



make/main.cpp

Заголовочный файл hello.hpp

```
#pragma once

void say_hello();
```

Файл main.cpp

```
#include "hello.hpp"

int main(int argc, char *argv[]) {
  say_hello();
  return 0;
}
```

Файл реализации hello.cpp

```
#pragma once
#include <stdio.h>
#include "hello.hpp"

void say_hello() {
  printf("Hello, world!\n");
}
```

Сборка многофайловых консольных проектов в Linux (2)

Компиляция:

```
g++ -c -o hello.o ./hello.cpp
g++ -c -o main.o ./main.cpp
```

Сборка:

```
g++ -o main ./hello.o ./main.o
```

Скрипт compile.sh

```
#!/usr/bin/bash

g++ -c -o hello.o ./hello.cpp
g++ -c -o main.o ./main.cpp
g++ -o main ./hello.o ./main.o
```

```
#!/usr/bin/bash

g++ -c -o hello.o ./hello.cpp
if [ ! -f ./hello.o ]; then
  echo -e "\e[31mError on hello.cpp!\e[0m"
  exit 1
fi

g++ -c -o main.o ./main.cpp
if [ ! -f ./hello.o ]; then
  echo -e "\e[31mError on main.cpp!\e[0m"
  exit 2
fi

g++ -o main ./hello.o ./main.o
if [ ! -f ./main ]; then
  echo -e "\e[31mError on link!\e[0m"
  exit 3
fi
```

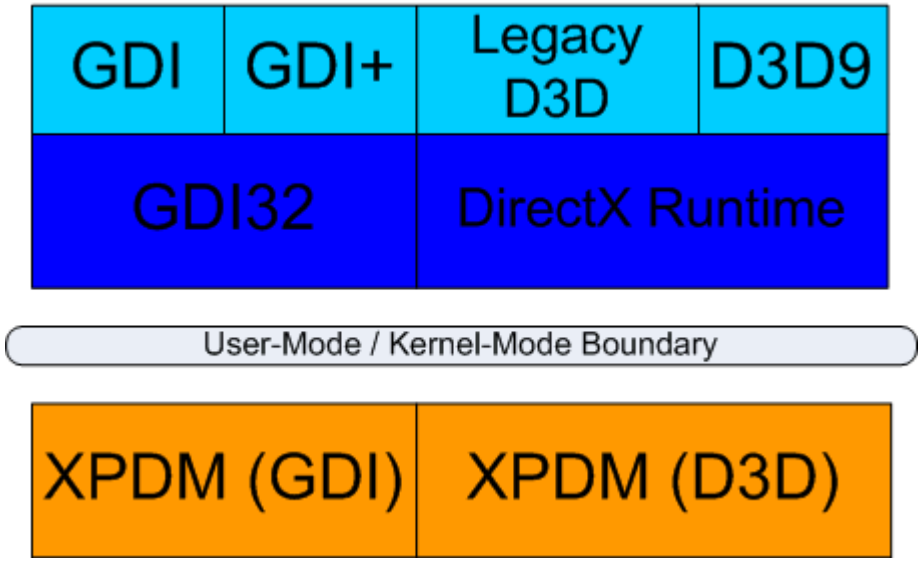
Утилита make, синтаксис Makefile (1)

```
[переменные]

<цель>:<зависимости>
→     <команды>

<цель>:<зависимости>
→     <команды>

<цель>:<зависимости>
→     <команды>

...
```

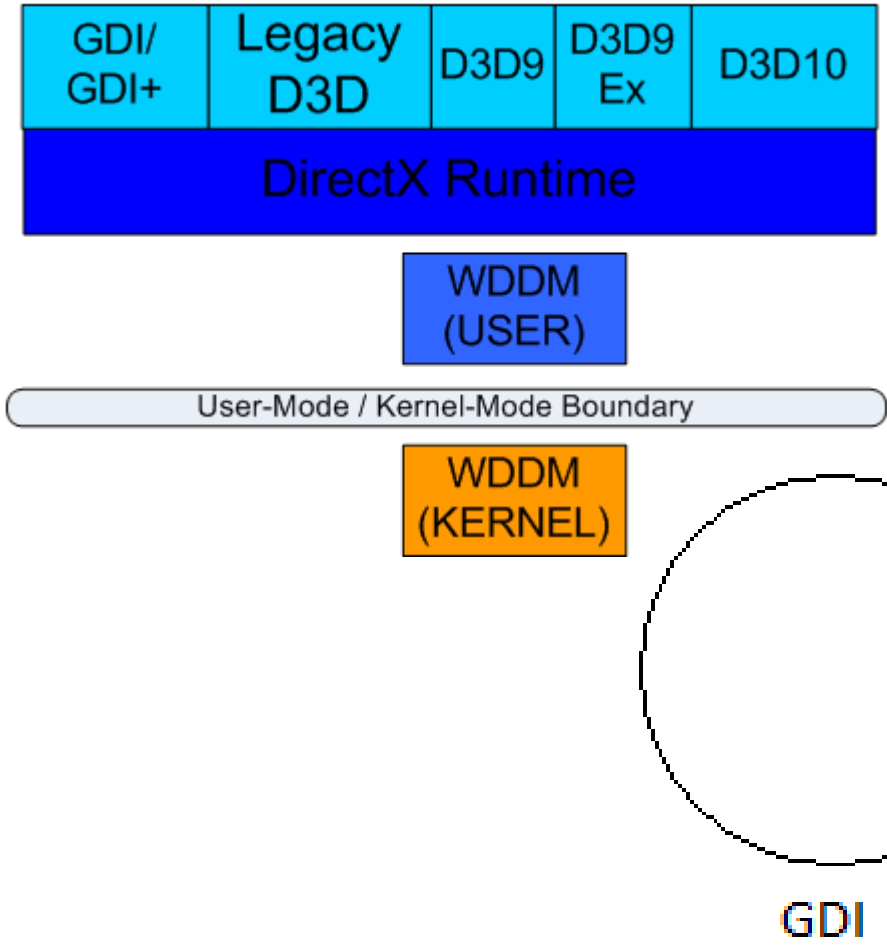
```
all:            main

main:           hello.o main.o
                g++ hello.o main.o -o main

hello.o:        hello.cpp hello.h
                g++ –Wall –c hello.cpp

main.o:         main.cpp
                g++ –Wall –c main.cpp

clean:
                rm -f *.o
                rm -f ./main
```

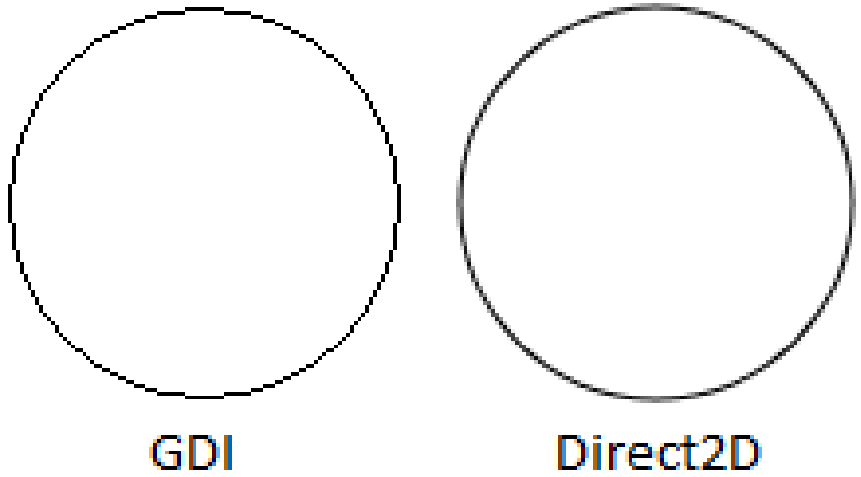
Архитектура графической подсистемы Windows (Windows GDI)

Microsoft Windows xp

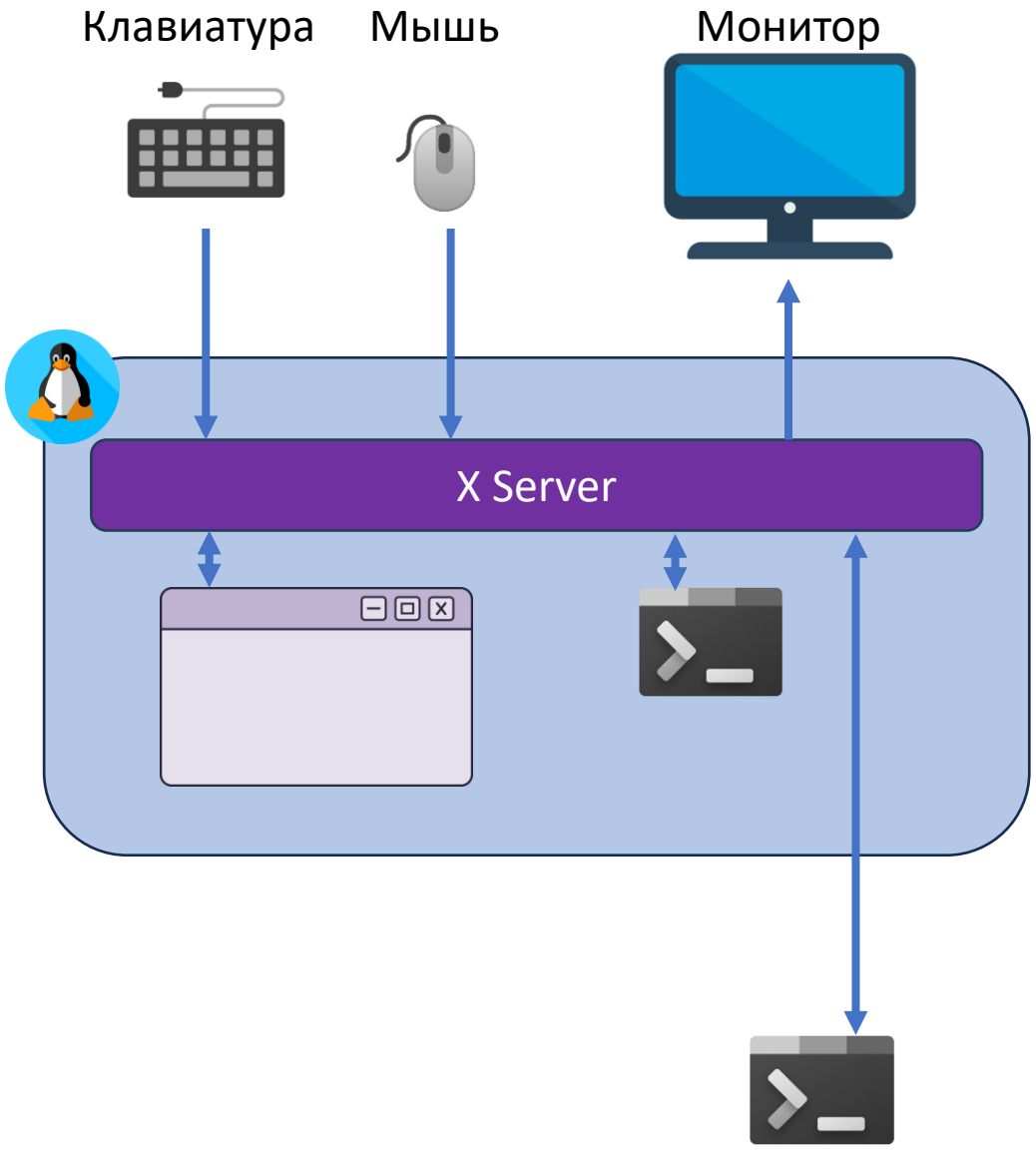


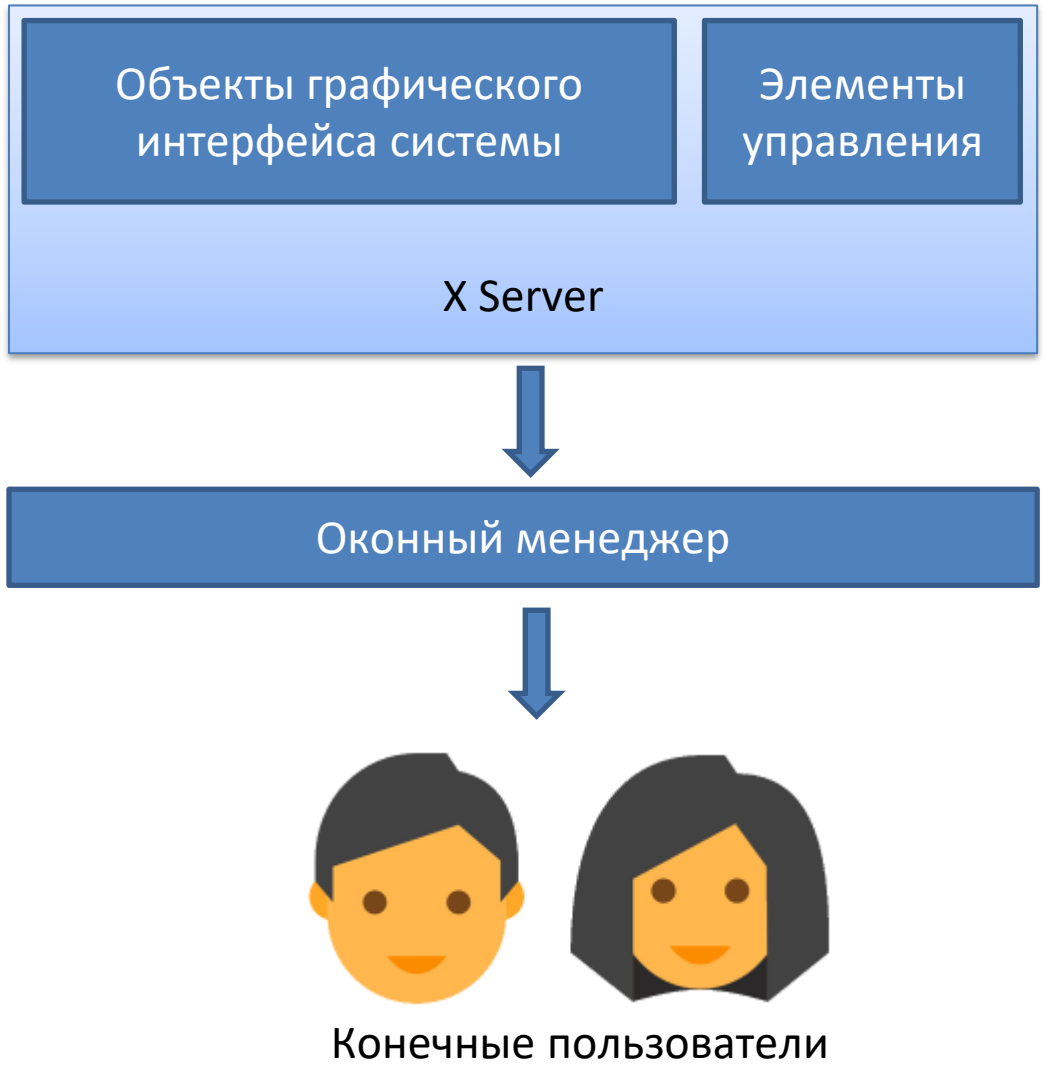
Windows Vista





Архитектура X Window System



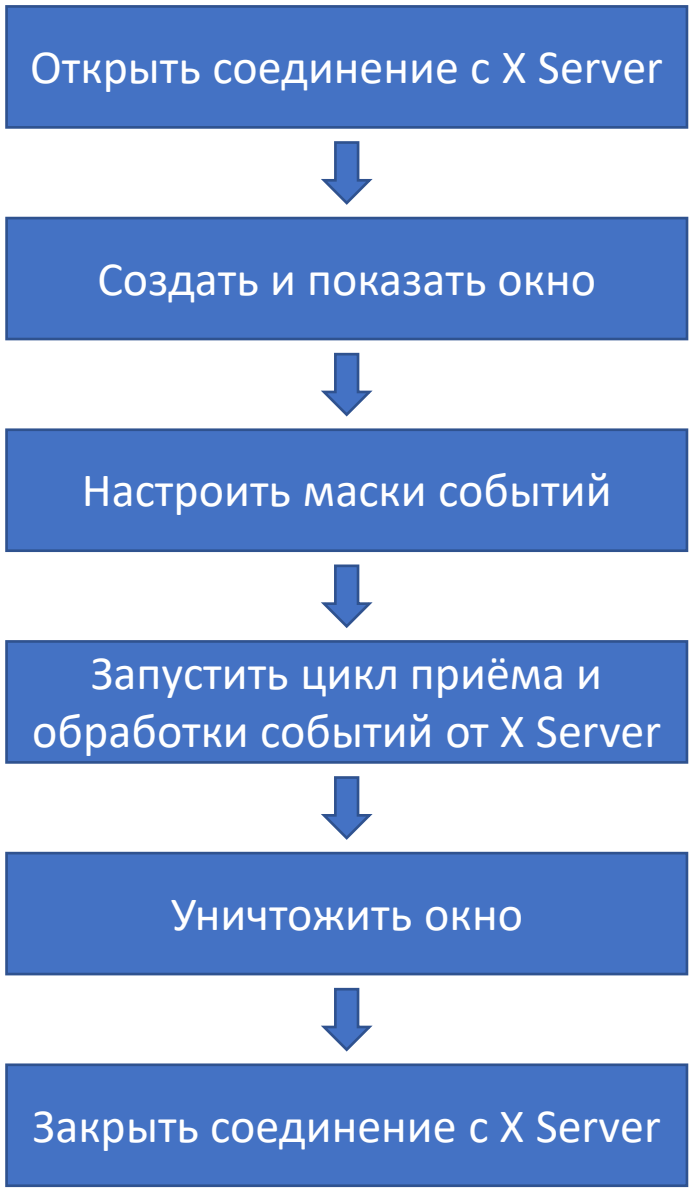


Кодирование сообщений

```
#define WM_NULL             0x0000
#define WM_CREATE           0x0001
#define WM_DESTROY          0x0002
#define WM_MOVE             0x0003
#define WM_SIZE             0x0005
#define WM_ACTIVATE         0x0006
#define WM_SETFOCUS         0x0007
#define WM_KILLFOCUS        0x0008
#define WM_ENABLE           0x000A
#define WM_SETREDRAW        0x000B
#define WM_SETTEXT          0x000C
#define WM_GETTEXT          0x000D
#define WM_GETTEXTLENGTH    0x000E
#define WM_PAINT            0x000F
#define WM_CLOSE            0x0010
#define WM_QUIT             0x0012
#define WM_ERASEBKGND       0x0014
#define WM_SYSCOLORCHANGE   0x0015
#define WM_SHOWWINDOW       0x0018
...
```

```
#define X_CreateWindow              1
#define X_ChangeWindowAttributes    2
#define X_GetWindowAttributes       3
#define X_DestroyWindow             4
#define X_DestroySubwindows         5
#define X_ChangeSaveSet             6
#define X_ReparentWindow            7
#define X_MapWindow                 8
#define X_MapSubwindows             9
#define X_UnmapWindow              10
#define X_UnmapSubwindows          11
#define X_ConfigureWindow          12
#define X_CirculateWindow          13
#define X_GetGeometry              14
#define X_QueryTree                15
#define X_InternAtom               16
#define X_GetAtomName              17
#define X_ChangeProperty           18
#define X_DeleteProperty           19
...
```

Архитектура приложения с использованием Xlib



We gonna create a window, but we need to get the window's background color first.
another one, and even on the same machine, from an execution of the program to the

```
int blackColor = BlackPixel(dpy, DefaultScreen(dpy));
int whiteColor = WhitePixel(dpy, DefaultScreen(dpy));
```

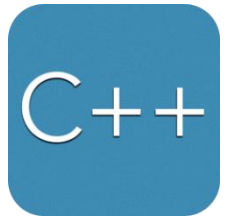
As we yet can see, most of the Xlib calls take the "display" (the value returned by X

There is still someting magic, (the DefaultScreen() stuff), but we gonna keep it for

```
// Create a "Graphics Context"

GC gc = XCreateGC(dpy, w, 0, NIL);
```

Yet another magic stuff. But mastering them is the

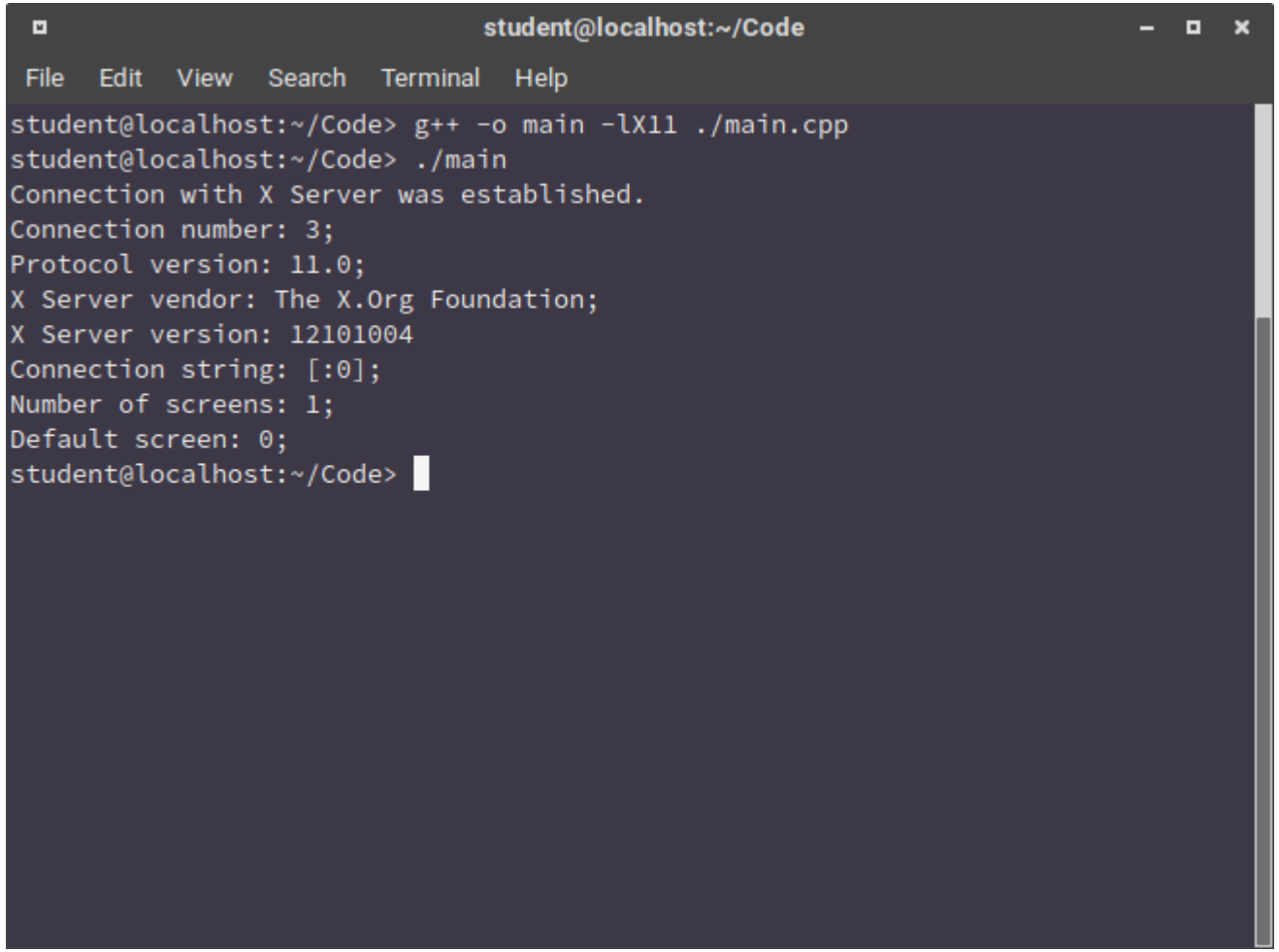
Получение информации о подключении к X Server



Xlib/xinfo.cpp

```
#include <X11/Xlib.h>
#include <stdio.h>
#include <stdlib.h>

int main(int, char *[]) {
  Display *display = XOpenDisplay(NULL);
  if (!display) {
    fprintf(stderr, "Can’t open display.\n");
    return EXIT_FAILURE;
  };

  printf("Connection with X Server was established.\n");
  printf("Connection number: %d;\n",    ConnectionNumber(display));
  printf("Protocol version: %d.%d;\n", ProtocolVersion(display), ProtocolRevision(display));
  printf("X Server vendor: %s;\n",      ServerVendor(display));
  printf("X Server version: %d\n",      VendorRelease(display));
  printf("Connection string: [%s];\n", DisplayString(display));
  printf("Number of screens: %d;\n",    ScreenCount(display));
  printf("Default screen: %d;\n",       DefaultScreen(display));

  XCloseDisplay(display);
  return EXIT_SUCCESS;
}
```

Результат запуска программы



Соединение с X Server : XOpenDisplay

Синтаксис

```
Display *XOpenDisplay(char *display_name)
```

Описание

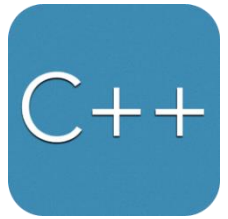
Позволяет подключиться к X Server, адрес которого задаётся в следующем формате:

```
hostname:number.screen_number
```

Здесь:

`hostname` – сетевое имя или IP машины, к которой необходимо подключиться
`number` – номер display сервера на указанной машине
`screen_number` – номер экрана, к которому мы подключаемся

Простое приложение с использованием Xlib



main_00.cpp

```
#include <X11/Xlib.h>
#include <stdio.h>
#include <stdlib.h>

int main(int, char *[]) {
  Display *display = XOpenDisplay(NULL);
  if (!display) {
    printf("Can't open display.\n");
    return EXIT_FAILURE;
  }

  Window window = XCreateSimpleWindow(display, XDefaultRootWindow(display), 100, 100, 200, 200, 4, 0, 0);
  XMapWindow(display, window);

  XSelectInput(display, window, NoEventMask);
  XEvent event;
  for (;;) {
    XNextEvent(display, &event);
  }

  XDestroyWindow(display, window);
  XCloseDisplay(display);
  return EXIT_SUCCESS;
}
```

Создание окна: XCreateSimleWindow и XCreateWindow

Синтаксис

```
Window XCreateSimpleWindow(
       Display      *display,
       Window        parent,
       int           x,
       int           y,
       unsigned int  width,
       unsigned int  height,
       unsigned int  border_width,
       unsigned long border,
      unsigned long background);
```

Синтаксис

```
Window XCreateWindow(
       Display               *display,
       Window                 parent,
       int                    x,
       int                    y,
       unsigned int           width,
       unsigned int           height,
       unsigned int           border_width,
       int                    depth,
       unsigned int           class,
       Visual                *visual,
       unsigned long          valuemask,
       XSetWindowAttributes  *attributes);
```

Обработка событий

```c
#include <X11/Xlib.h>
#include <stdio.h>
#include <stdlib.h>

int main(int, char *[]) {
  Display *display = XOpenDisplay(NULL);
  if (!display) {
    printf("Can't open display.\n");
    return EXIT_FAILURE;
  }

  Window window = XCreateSimpleWindow(display, XDefaultRootWindow(display), 100, 100, 200, 200, 4, 0, 0);
  XMapWindow(display, window);

  XSelectInput(display, window, NoEventMask);
  XEvent event;
  for (;;) {
    XNextEvent(display, &event);
  }

  XDestroyWindow(display, window);
  XCloseDisplay(display);
  return EXIT_SUCCESS;
}
```

Маски событий

Какие события хотим получать:

`NoEventMask`	Никакие
`KeyPressMask`	Нажата клавиша
`KeyReleaseMask`	Отпущена клавиша
`ButtonPressMask`	Нажата кнопка мыши
`ButtonReleaseMask`	Отпущена кнопка мыши
`EnterWindowMask`	Курсор вошёл в зону окна
`LeaveWindowMask`	Курсор вышел из зоны окна
`PointerMotionMask`	Курсор движется над поверхностью окна
`Button1MotionMask`	Курсор движется, пока нажата кнопка 1
`Button2MotionMask`	Курсор движется, пока нажата кнопка 2
`Button3MotionMask`	Курсор движется, пока нажата кнопка 3
`Button4MotionMask`	Курсор движется, пока нажата кнопка 4
`Button5MotionMask`	Курсор движется, пока нажата кнопка 5
`ButtonMotionMask`	Курсор движется, когда нажата любая кнопка
`ExposureMask`	Нужно перерисовать окно
`StructureNotifyMask`	Поменялась информация о структуре окна

…

Обработка событий от X Server

```
XSelectInput(p_display, window, NoEventMask);

XEvent event;
for (;;) {
  XNextEvent(p_display, &event);
}
```

Заменяем на

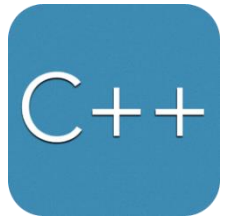
```
KeyPressMask | ButtonPressMask
```

Добавляем код

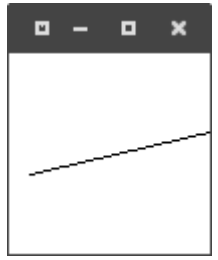
```
printf("%d\n", event.type);
```

Использование графического контекста

```
...

XSelectInput(display, window, ExposureMask);

GC gc = XCreateGC(display, window, 0, NULL);

XEvent event;
for (;;) {
  XNextEvent(display, &event);

  if (event.type == Expose) {
    XClearWindow(display, window);
    XDrawLine(display, window, gc, 10, 60, 180, 20);
    XFlush(display);
  }
}

XFreeGC(display, gc);

...
```



Xlib/main_01.cpp



Рисование примитивов: точки

```
int XDrawPoint(Display  *display,
               Drawable  d,
               GC        gc,
               int       x,
               int       y);

int XDrawPoints(Display *display,
                Drawable d,
                GC g     c,
                XPoint  *points,
                int      npoints,
                int      mode);
```

```
typedef struct {
  short x,
        y;
} XPoint;
```

CoordModeOrigin
CoordModePrevious

Рисование примитивов: линии

```
int XDrawLine(Display *display,
              Drawable d,
              GC       gc,
              int      x1,
              int      y1,
              int      x2,
              int      y2);

int XDrawLines(Display *display,
               Drawable d,
               GC       gc,
               XPoint  *points,
               int      npoints,
               int      mode);

int XDrawSegments(Display  *display,
                  Drawable  d,
                  GC        gc,
                  XSegment *segments,
                  int       nsegments);
```

```
typedef struct {
  short x1,
        y1,
        x2,
        y2;
} XSegment;
```

Рисование примитивов: замкнутые фигуры (1)

```
int XDrawRectangle(Display     *display,
                   Drawable     d,
                   GC           gc,
                   int          x,
                   int          y,
                   unsigned int width,
                   unsigned int height);

int XDrawRectangles(Display    *display,
                    Drawable    d,
                    GC          gc,
                    XRectangle rectangles[],
                    int        nrectangles);
```

```
typedef struct {
  short          x,
                 y;
  unsigned short width,
                 height;
} XPoint;
```

Рисование примитивов: замкнутые фигуры (2)

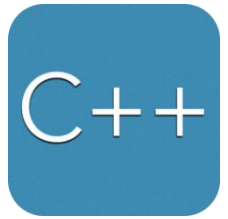
```
int XDrawArc(Display     *display,
             Drawable     d,
             GC           gc,
             int          x,
             int          y,
             unsigned int width,
             unsigned int height,
             int          angle1,
             int          angle2);

int XDrawArcs(Display *display,
              Drawable d,
              GC       gc,
              Xarc    *arcs,
              int      narcs);
```

```
typedef struct {
  short          x,
                 y;
  unsigned short width,
                 height;
  short          angle1,
                 angle2;
} XArc;
```

Рисование примитивов: залитые цветом фигуры

```
int XFillRectangle(Display *display,
                   Drawable d,
                   GC gc,
                   int x, int y,
                   unsigned int width, unsigned int height);

int XFillRectangles(Display *display, Drawable d, GC gc,
                    XRectangle rectangles[],
                    int nrectangles);

int XFillArc(Display *display,
             Drawable d,
             GC gc,
             int x, int y, unsigned int width, unsigned int height,
             int angle1, int angle2);

int XFillArcs(Display *display, Drawable d, GC gc,
              XArc *arcs, int narcs);
```

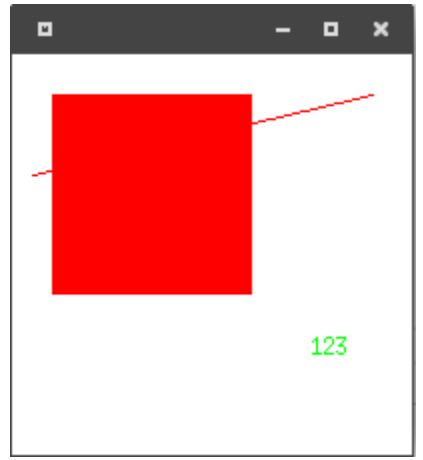
Использование цветов: стандартные цвета (1)



Xlib/main_02.cpp

```
XColor   red;
```

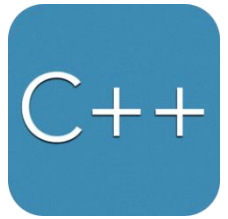
```
Colormap cmap = DefaultColormap(display, screen);
Status   rc   = XAllocNamedColor(display, cmap, "red", &red, &red);

if (!rc) {
  printf("'red' color allocation failure.\n");
  return EXIT_FAILURE;
}

...

XSetForeground(display, gc, red.pixel);
XDrawLine(display, window, gc, 10, 60, 180, 20);
```



Использование цветов: стандартные цвета (2)

```
XColor col;
col.red   = 65535;
col.green = 32767;
col.blue  = 0;

Colormap cmap = DefaultColormap(display, screen);
Status rc = XAllocColor(display, cmap, &col);

if (!rc) {
  printf("Color allocation failure.\n");
  return EXIT_FAILURE;
}

...

XSetForeground(display, gc, col.pixel);
XDrawLine(display, window, gc, 10, 60, 180, 20);
```

Простое приложение на GTK



GTK2/gtk_00.cpp

```
#include <gtk/gtk.h>

int main(int argc, char *argv[]) {
  gtk_init(&argc, &argv);

  GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);

  g_signal_connect( window,
                    "destroy",
                    G_CALLBACK(gtk_main_quit),
                    NULL);

  gtk_widget_show(window);

  gtk_main();

  return 0;
}
```

Компиляция

```
g++ -pthread -I/usr/include/gtk-2.0 -I/usr/lib64/gtk-2.0/include
-I/usr/include/atk-1.0 -I/usr/include/cairo
-I/usr/include/gdk-pixbuf-2.0 -I/usr/include/pango-1.0
-I/usr/include/glib-2.0 -I/usr/lib64/glib-2.0/include
-I/usr/include/harfbuzz -I/usr/include/freetype2
-I/usr/include/pixman-1 -I/usr/include/libpng15
-I/usr/include/libdrm  -lgtk-x11-2.0 -lgdk-x11-2.0 -latk-1.0
-lgio-2.0 -lpangoft2-1.0 -lpangocairo-1.0 -lgdk_pixbuf-2.0 -lcairo
-lpango-1.0 -lfontconfig -lgobject-2.0 -lglib-2.0
-lfreetype   -o gtk_00 ./gtk_00.cpp
```


```
pkg-config --libs gtk+-2.0
```

```
g++ `pkg-config --cflags --libs gtk+-2.0` -o gtk_00 ./gtk_00.cpp
```

Удобство библиотек

```
gtk_init(&argc, &argv);

GtkWidget *window =
  gtk_window_new(GTK_WINDOW_TOPLEVEL);

g_signal_connect(window,
                 "destroy",
                 G_CALLBACK(gtk_main_quit),
                 NULL);

gtk_widget_show(window);
```

```
Display *display = XOpenDisplay(NULL);
if (!display) {
  printf("Can't open display.\n");
  return EXIT_FAILURE;
}

Window window = XCreateSimpleWindow(display,
                    XDefaultRootWindow(display),
                    100, 100, 200, 200, 4, 0, 0);
XMapWindow(display, window);

XSelectInput(display, window, NoEventMask);
XEvent event;
for (;;) {
  XNextEvent(display, &event);
}

XDestroyWindow(display, window);
XCloseDisplay(display);
```

Функции для работы с окнами (2)

```
void
gtk_window_fullscreen (GtkWindow *window);

void
gtk_window_unfullscreen (GtkWindow *window);

void
gtk_window_set_title (GtkWindow *window,
                      const gchar *title);

void
gtk_window_set_default_size (GtkWindow *window,
                             gint width,
                             gint height);

void
gtk_window_set_opacity (GtkWindow *window,
                        gdouble opacity);
```

Обработка событий

```c
#include <gtk/gtk.h>

void on_destroy() {
  gtk_main_quit();
}

int main(int argc, char *argv[]) {
  gtk_init(&argc, &argv);

  GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);

  g_signal_connect( window, "destroy",
                    G_CALLBACK(on_destroy), NULL);

  gtk_widget_show(window);

  gtk_main();

  return 0;
}
```

Добавление элементов управления (2)

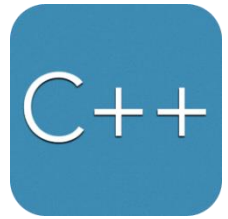
```
g_signal_connect( window,
                  "destroy",
                  G_CALLBACK(on_destroy),
                  NULL);

GtkWidget *button = gtk_button_new_with_label("Push");
gtk_container_add(GTK_CONTAINER(window), button);

gtk_widget_show_all(window);

gtk_main();
```

Работа с упаковщиками


```c
int main(int argc, char *argv[]) {
  gtk_init(&argc, &argv);

  GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);

  gtk_window_set_default_size(GTK_WINDOW(window), 400, 200);

  g_signal_connect( window, "destroy", G_CALLBACK(on_destroy), NULL);

  GtkWidget *button = gtk_button_new_with_label("Push");
  gtk_container_add(GTK_CONTAINER(window), button);
```

```c
  GtkWidget *fixed = gtk_fixed_new();
  gtk_container_add(GTK_CONTAINER(window), fixed);
  gtk_fixed_put(GTK_FIXED(fixed), button, 150, 50);
```

Рисование с gtk: библиотека cairo (1)



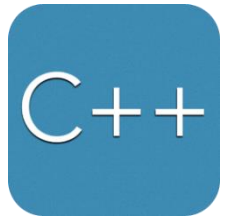
GTK2/gtk_01.cpp

```
g_signal_connect(window, "destroy",
                 G_CALLBACK(gtk_main_quit), NULL);

GtkWidget *draw = gtk_drawing_area_new();

gtk_container_add(GTK_CONTAINER(window), draw);

g_signal_connect(G_OBJECT(draw),
                 "expose",
                 G_CALLBACK(on_draw),
                 NULL);
```

Рисование с gtk: библиотека cairo (2)

```
gboolean on_draw(GtkWidget *widget, GdkEventExpose *event, gpointer data) {

  cairo_t *c = gdk_cairo_create(gtk_widget_get_window(widget));
  cairo_move_to(c, 30, 30);
  cairo_show_text(c, "Text");

  cairo_set_line_width(c, 1.0);
  cairo_set_source_rgb(c, 0, 0, 0);
  cairo_rectangle(c, 10, 10, 100, 100);
  cairo_stroke(c);

  cairo_destroy(c);

  g_print("draw\n");
  return TRUE;
}
```

Пример создания оконного меню



GTK2/gtk_02.cpp

```
vbox = gtk_vbox_new(FALSE, 0);
gtk_container_add(GTK_CONTAINER(window), vbox);


GtkWidget *menubar = gtk_menu_bar_new();
GtkWidget *menu_file = gtk_menu_new();


GtkWidget *mi_file = gtk_menu_item_new_with_label("File");
GtkWidget *mi_quit = gtk_menu_item_new_with_label("Quit");


gtk_menu_item_set_submenu(GTK_MENU_ITEM(mi_file), menu_file);
gtk_menu_shell_append(GTK_MENU_SHELL(menu_file), mi_quit);
gtk_menu_shell_append(GTK_MENU_SHELL(menubar), mi_file);


gtk_box_pack_start(GTK_BOX(vbox), menubar, FALSE, FALSE, 0);
```