Curso LINUX

AREA 1: Entornos de programación

Entornos de programación

- Unix / Linux dispone de todas las herramientas necesarias como línea de comando
- Pero también dispone de entornos integrados
- Son front-ends a los comandos que ya vimos
- La elección de IDE es un tema importante al plantear el desarrollo

Elección de un IDE

- Un IDE debe elegirse en base a estos parámetros:
  - Soporte para el lenguaje que utilicemos
  - Facilidades para la generación de código
  - Facilidades para la integración con el entorno de versionado
  - Compatibilidad con otras herramientas
  - Sencillez de uso

IDEs linux (C/C++)

- emacs (si, no es broma)
- Anjuta
- Kdevelop
- Eclipse CDT
- Code Dragon
- Geany
- ... y muchos más

No existe el IDE ideal

- Cada IDE se ha desarrollado inicialmente para una cosa
- Los IDEs suelen ampliarse para converger
- La evolución propia de cada IDE hace que sea más o menos atractivo en un momento dado
- Las constumbres de los programadores también influyen en la adopción de un IDE u otro

Anjuta DevStudio

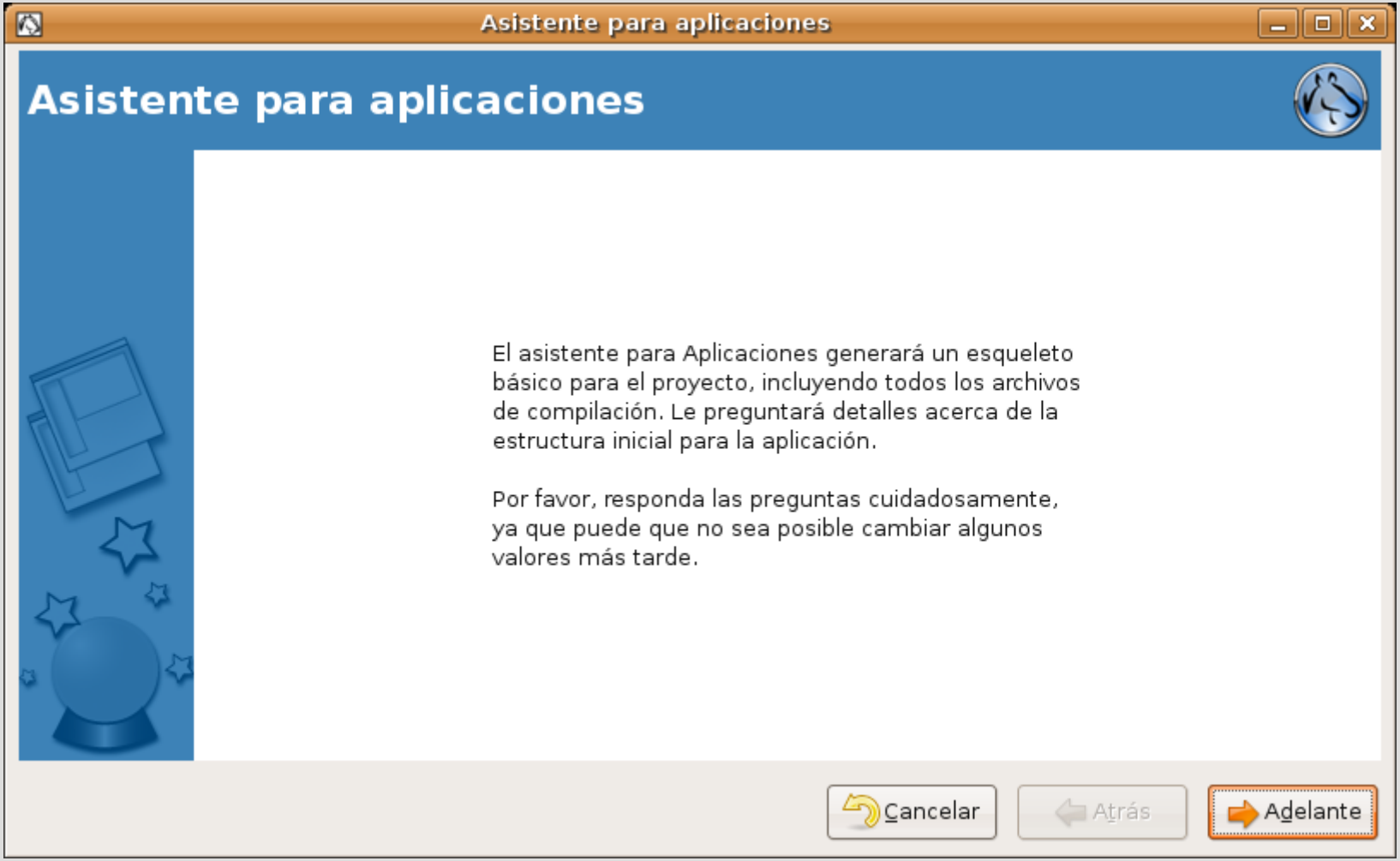
- Anjuta es un IDE para C/C++ y otros lenguajes en Linux.
- Se ha desarrollado para Gtk/Gnome
- Incluye facilidades para gestión de proyecto, wizards y depuración
- Tiene un editor de código con sourcebrowsing y reconocimiento de sintaxis.

Anjuta - Versiones

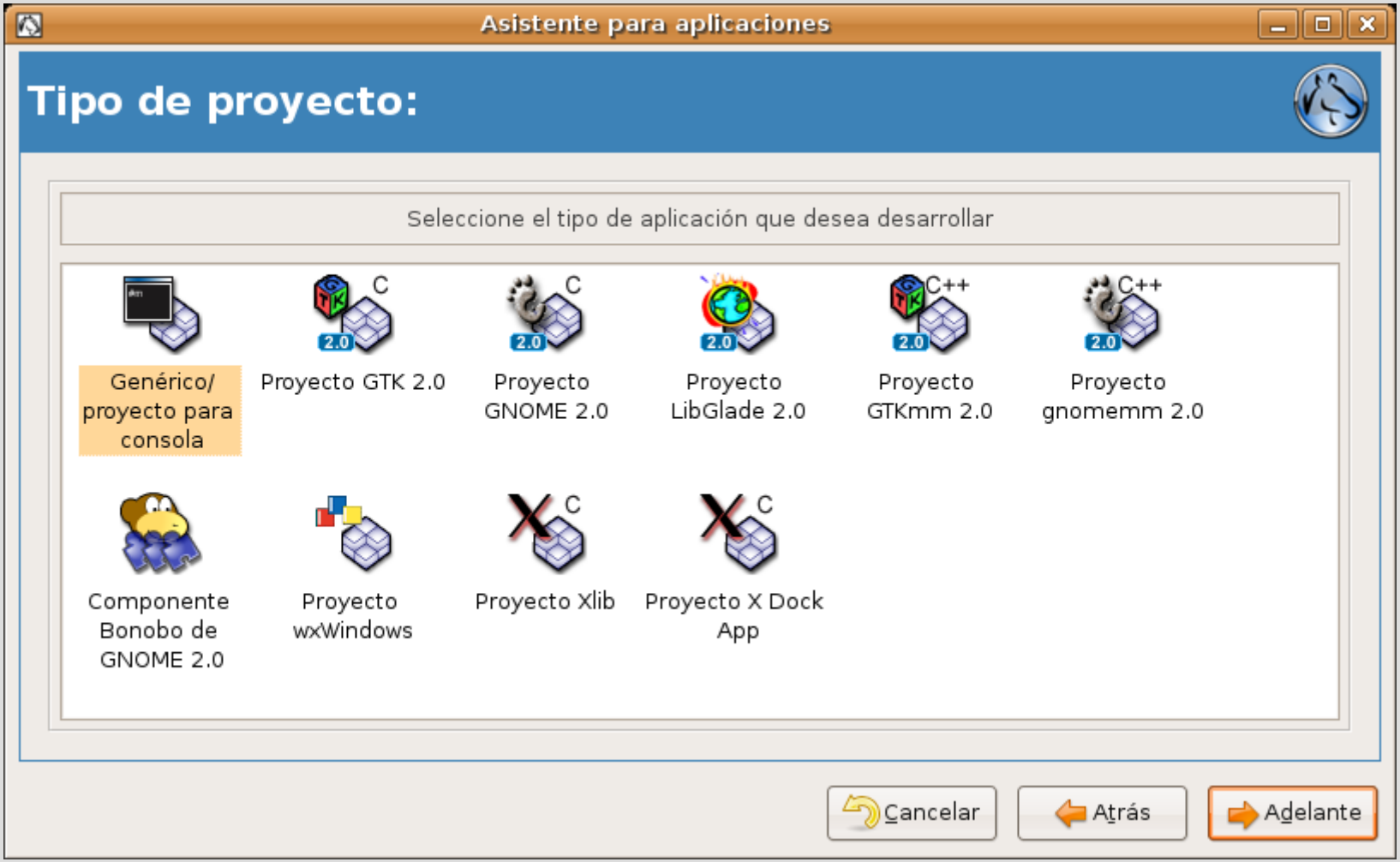
- La versión estable para muchas distribuciones es la 1.2.X
- Las versiones 2.X aportan muchas novedades, pero no está incluida en todas las distribuciones
- El desarrollo es muy activo por lo que se preven nuevas versiones en poco tiempo.

Anjuta – Nuevo proyecto

Anjuta - Nuevo proyecto (2)



Anjuta – Nuevo Proyecto (3)



Anjuta – Nuevo proyecto (4)

Asistente para aplicaciones

Información básica

Introduzca la información básica del proyecto

Nombre del proyecto: Hola Mundo

Versión del proyecto: 0.1

Autor del proyecto: yo

Objetivo del proyecto: hola mundo

Lenguaje de programación

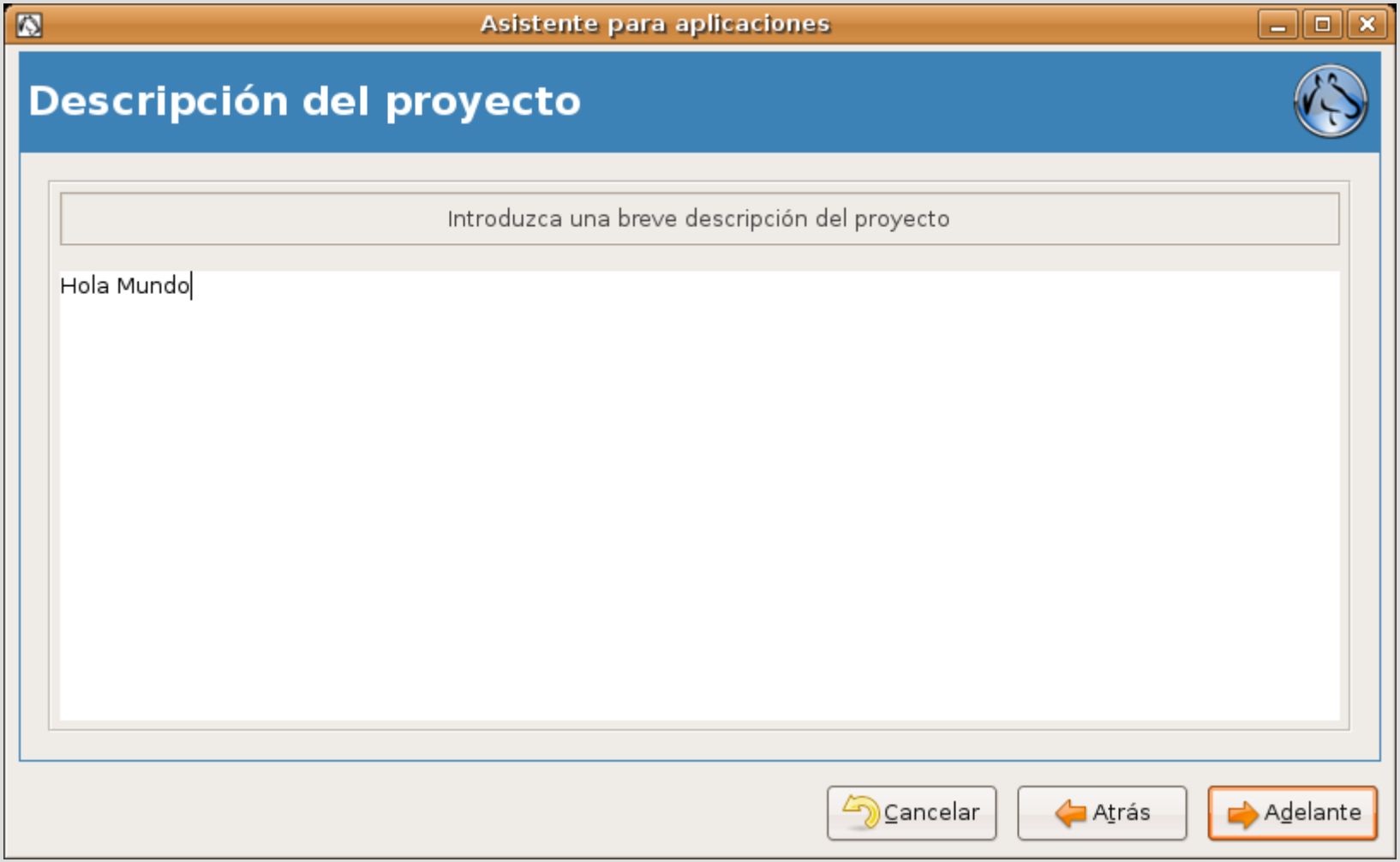
- (•) C
- ( ) C++
- ( ) C y C++

Tipo de objetivo

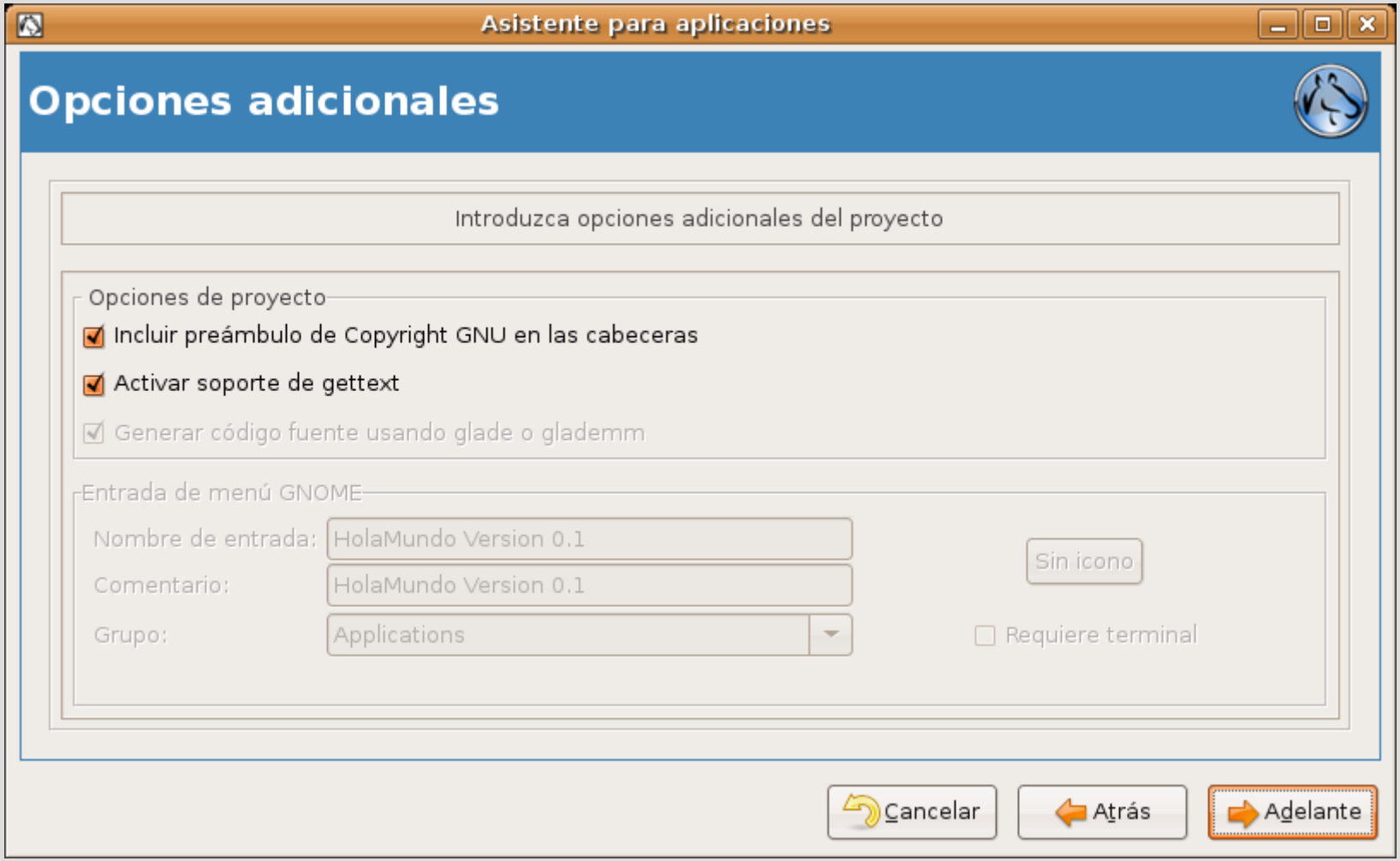
- (•) Objetivo ejecutable
- ( ) Objetivo biblioteca estática
- ( ) Objetivo biblioteca dinámica

Cancelar | Atrás | Adelante

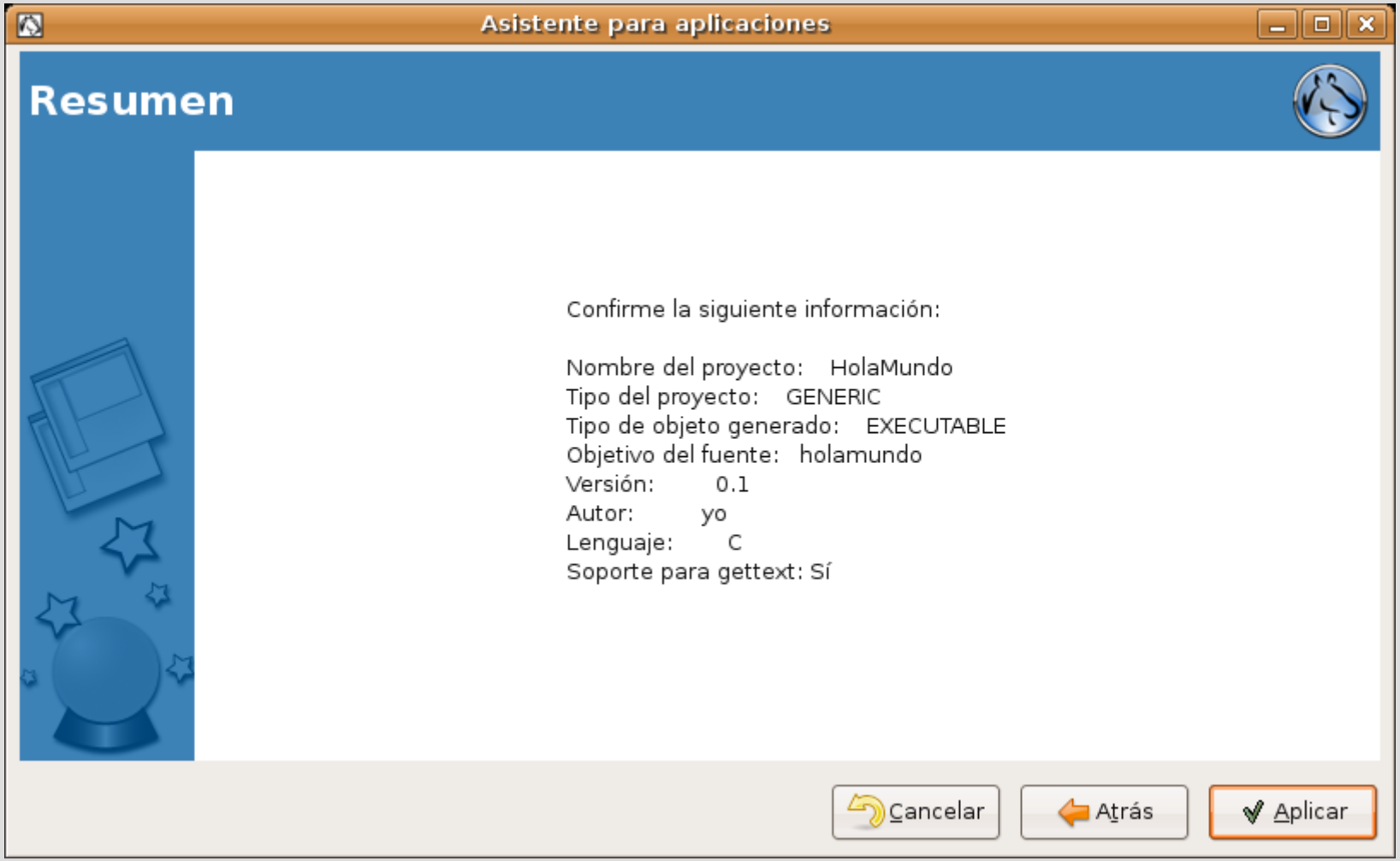
Anjuta – Nuevo proyecto (5)



Anjuta – Nuevo proyecto (6)



Anjuta – Nuevo proyecto (7)



Anjuta - Entorno

Anjuta: /home/espinosa/Projects/HolaMundo/src/main.c (Guardado)

Archivo Editar Ver Proyecto Formato Construir Marcador Depurador CVS Opciones Ayuda

Etiquetas: main [5]

Proyecto

- HolaMundo-0.1
  - source - src
    - main.c

Proyecto | Símbolos | Archivos

main.c

```
/* Created by Anjuta version 1.2.4a */
/*  This file will not be overwritten */

#include <stdio.h>
int main()
{
    printf("Hello world\n");
    return (0);
}
```

```
config.status: executing default-1 commands
config.status: executing default-2 commands
Now type `make' to compile.
Completado .... correctamente
Tiempo total empleado: 16 segundos
```

Build | Find | CVS | Terminal | Stdout | Stderr | Debug | Locals | Stack | Watches | Explorer

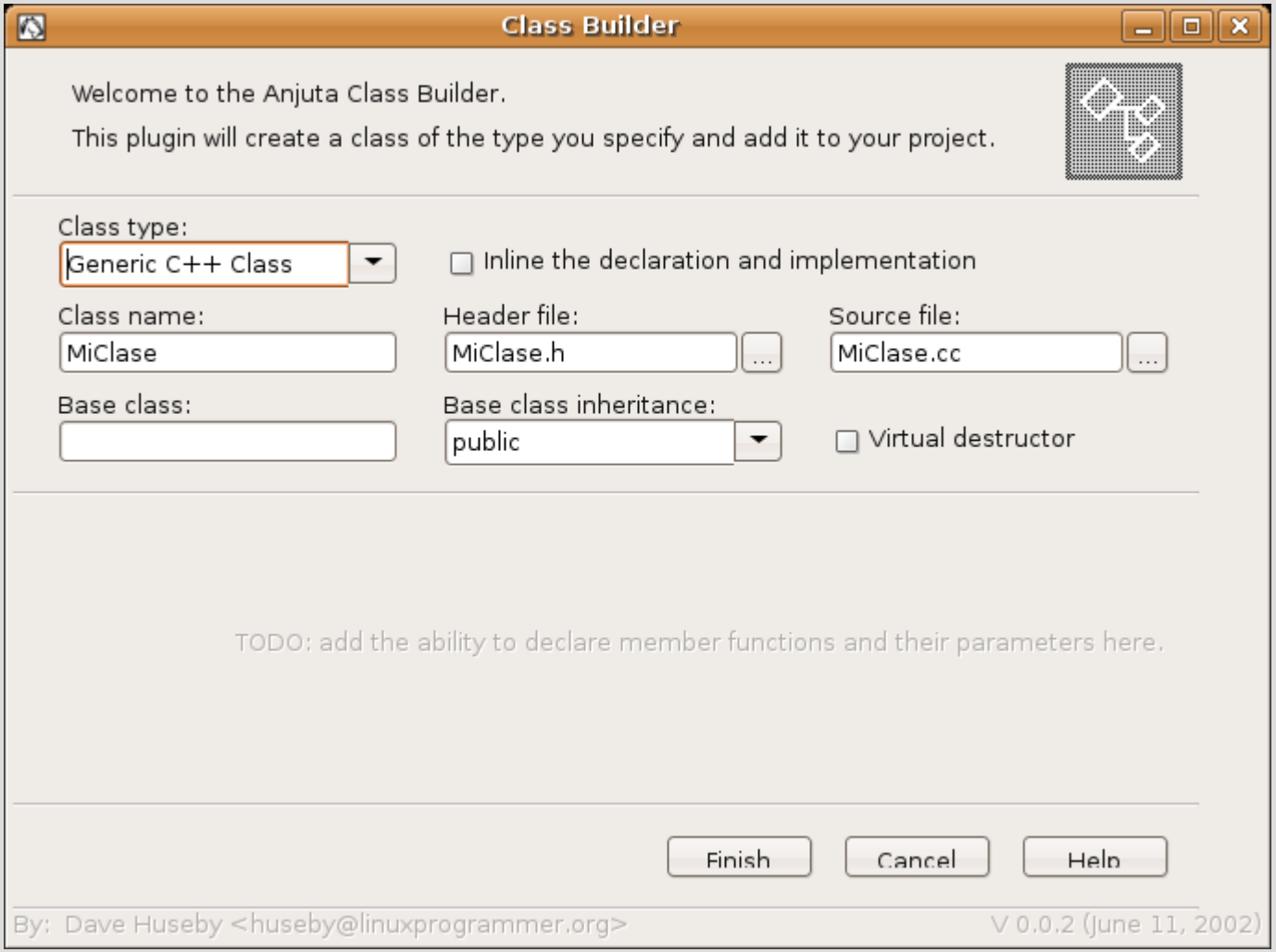
Proyecto: HolaMundo Ampliación: 0 Línea: 0001 Col: 000 INS Tarea: Ninguno Modo: Unix (LF)

Anjuta - Autoconf

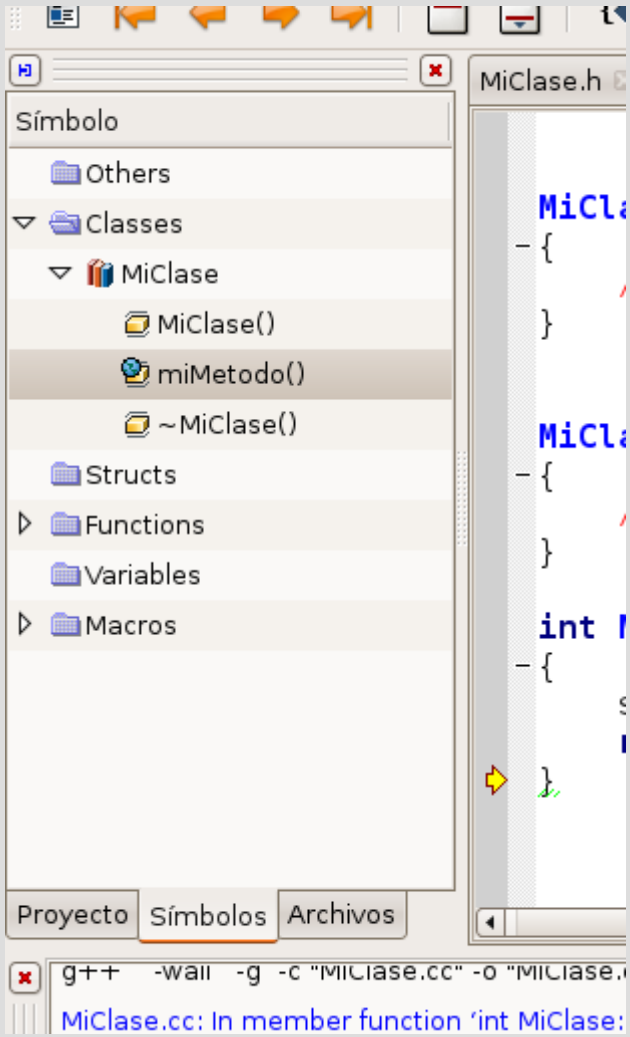
- Anjuta se encarga de preparar todo para el autoconf

```
~/Projects/HolaMundo$ ls
acconfig.h       config.h.in        HolaMundo.pws  mkinstalldirs
acinclude.m4     config.log         INSTALL        NEWS
aclocal.m4       config.status      install-sh     po
AUTHORS          config.sub         libtool        README
autogen.sh       configure          ltmain.sh      setup-gettext
autom4te.cache   configure.in       Makefile       src
ChangeLog        COPYING            Makefile.am    stamp-h
config.guess     HolaMundo.prj      Makefile.in    stamp-h.in
config.h         HolaMundo.prj.bak  missing        TODO
```

Anjuta – Class Builder



Anjuta – Visor Simbolos



Anjuta - Compilación

Anjuta - Construcción

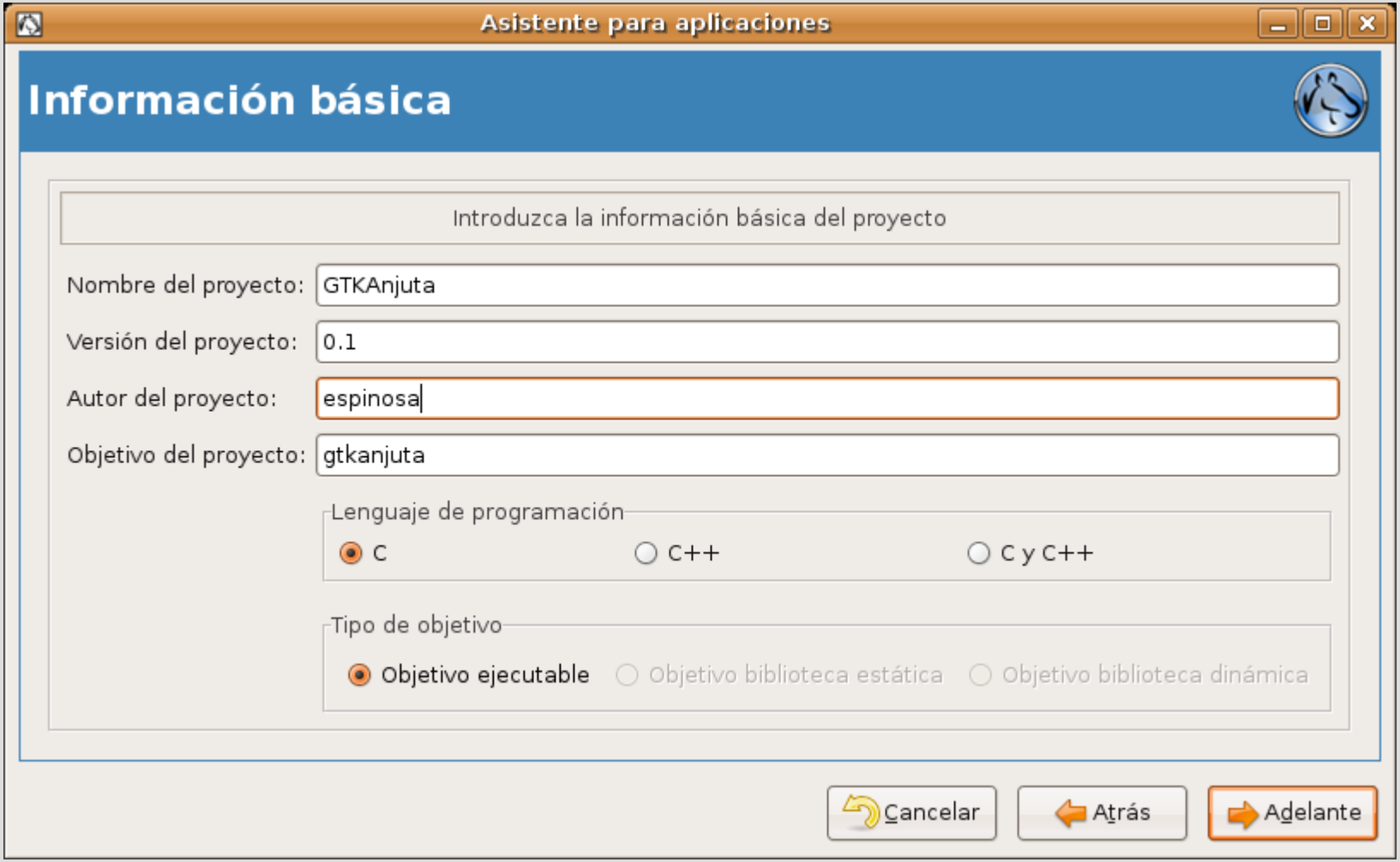
Anjuta - Depuración

- Permite puntos de ruptura
- Ejecución paso a paso
- Permite inspeccionar variables y evaluar expresiones

Anjuta – GTK

- Anjuta dispone de una integración con el generador de interfaces glade
- El generador es invocado en el momento que seleccionamos Proyecto ->Editar GUI de la aplicación
-

Anjuta – Ejemplo GTK



Anjuta – Ejemplo GTK (2)

Asistente para aplicaciones

Opciones adicionales

Introduzca opciones adicionales del proyecto

Opciones de proyecto

- [x] Incluir preámbulo de Copyright GNU en las cabeceras
- [x] Activar soporte de gettext
- [x] Generar código fuente usando glade o glademm

Entrada de menú GNOME

Nombre de entrada: GTKAnjuta Version 0.1

Comentario: GTKAnjuta Version 0.1

Grupo: Applications

Sin icono

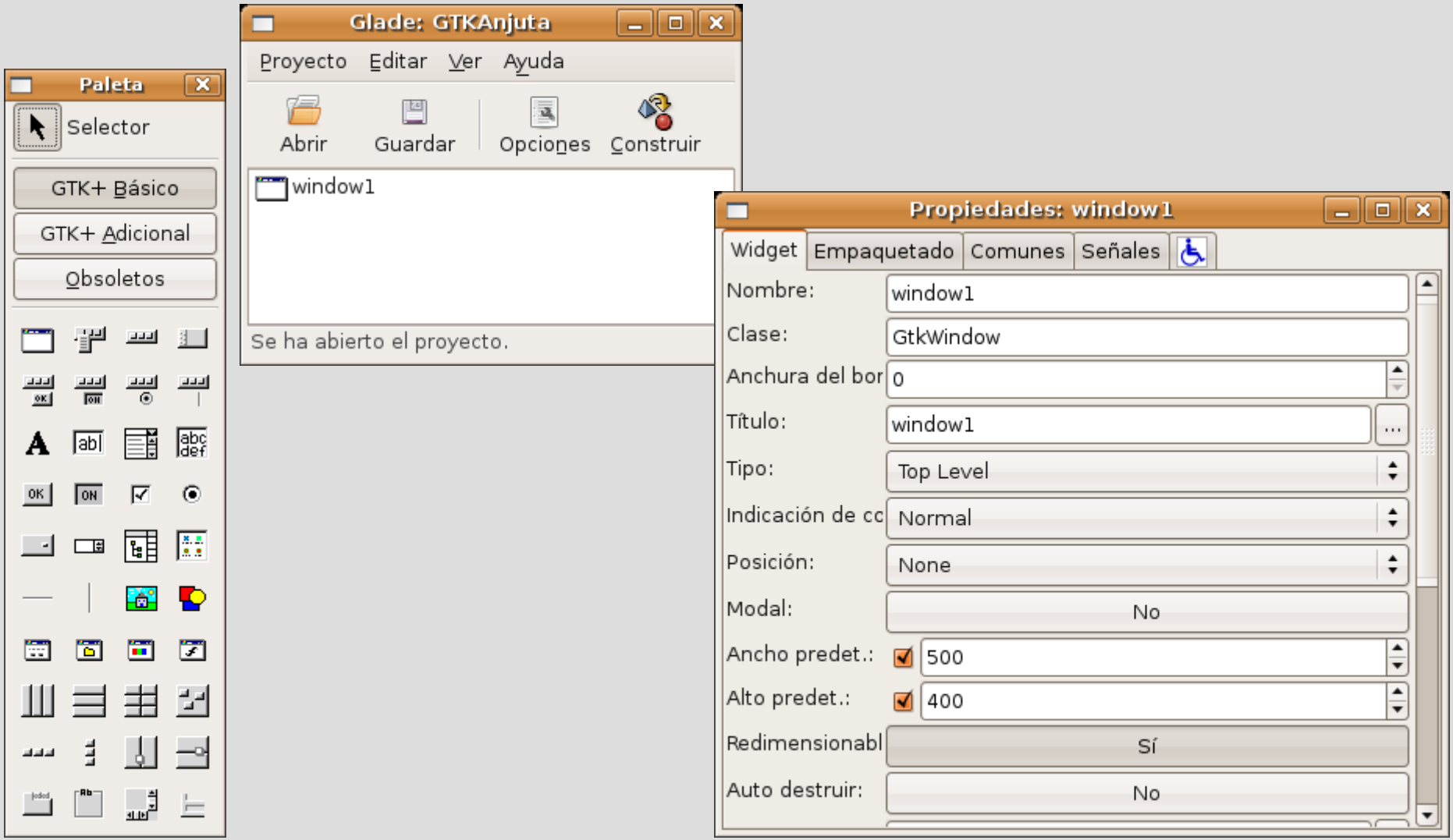
Requiere terminal

Cancelar | Atrás

Confirme la siguiente información:

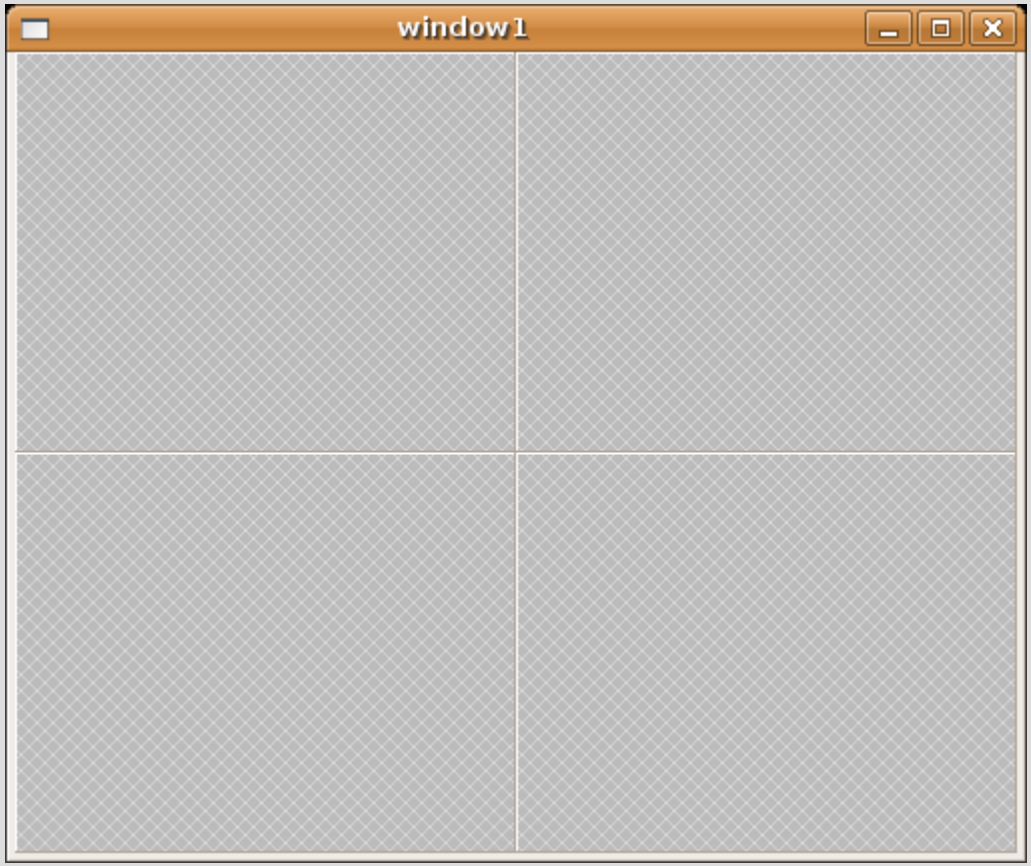
Nombre del proyecto: GTKAnjuta
Tipo del proyecto: GTK 2.0
Tipo de objeto generado: EXECUTABLE
Objetivo del fuente: gtkanjuta
Versión: 0.1
Autor: espinosa
Lenguaje: C
Soporte para gettext: Sí

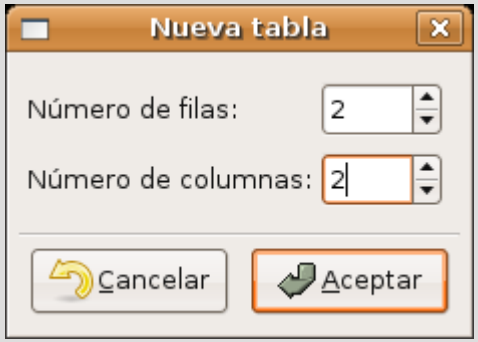
Anjuta – Ejemplo GTK (3)



Anjuta – Ejemplo GTK (4)

- Creamos una tabla 2x2



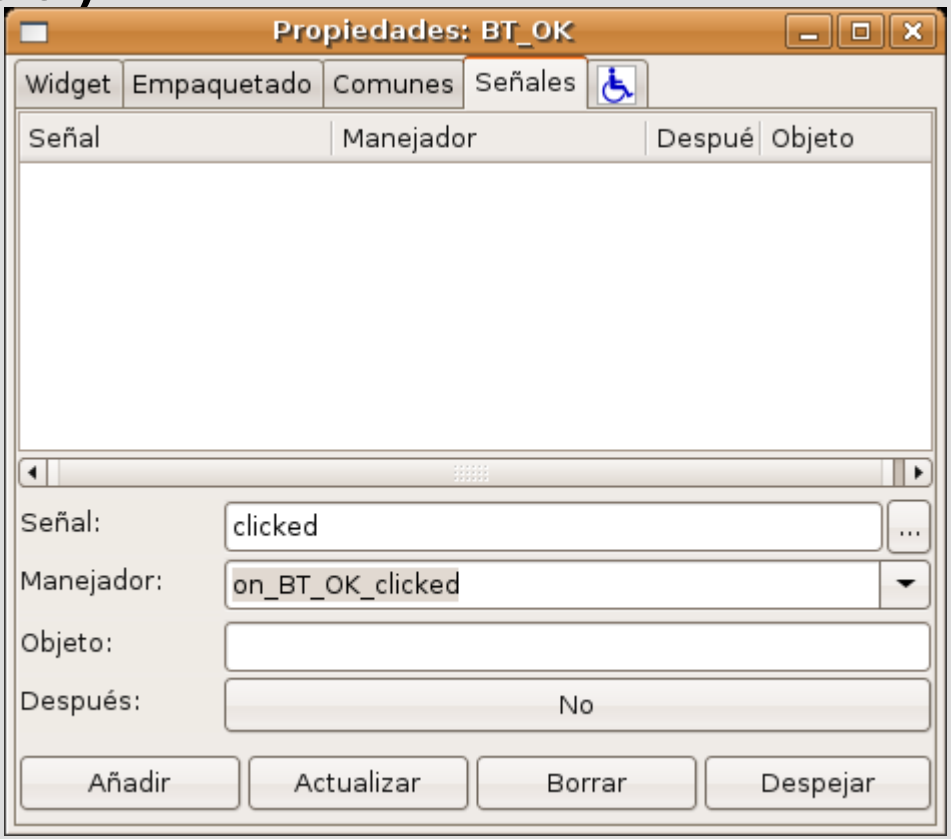


Anjuta – Ejemplo GTK (5)

- Creamos 2 botones, un label y un entry
- BT_OK, BT_EXIT, label y ENTRY
- Propiedad de la ventana homogeneus = si

Anjuta – Ejemplo GTK (6)

- Por cada botón añadimos una señal para clicked (on_BT_OK_clicked, on_BT_CANCEL_clicked)
- Guardar
- Construir
- Salir



Anjuta – Ejemplo GTK (7)

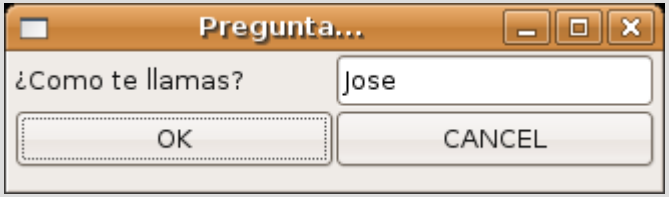
- Editar callbacks.c

```
void
on_BT_CANCEL_clicked (GtkButton       *button,
                      gpointer         user_data)
{
    gtk_main_quit();
}

void
on_BT_OK_clicked     (GtkButton       *button,
                      gpointer         user_data)
{
        GtkWidget *entry = lookup_widget (GTK_WIDGET(button), "ENTRY");
        gchar *text1;
        text1 = gtk_entry_get_text (GTK_ENTRY(entry));
        GtkWidget *msgbox = gtk_message_dialog_new(
                                    NULL,GTK_DIALOG_MODAL,GTK_MESSAGE_INFO,
                                    GTK_BUTTONS_CLOSE,"Hola %s",text1);
        gtk_widget_show_all (msgbox);
}
```

Anjuta – Ejemplo GTK (8)

- Guardar
- Construir -> Construir todo
- Construir -> Ejecutar



Anjuta - Conclusiones

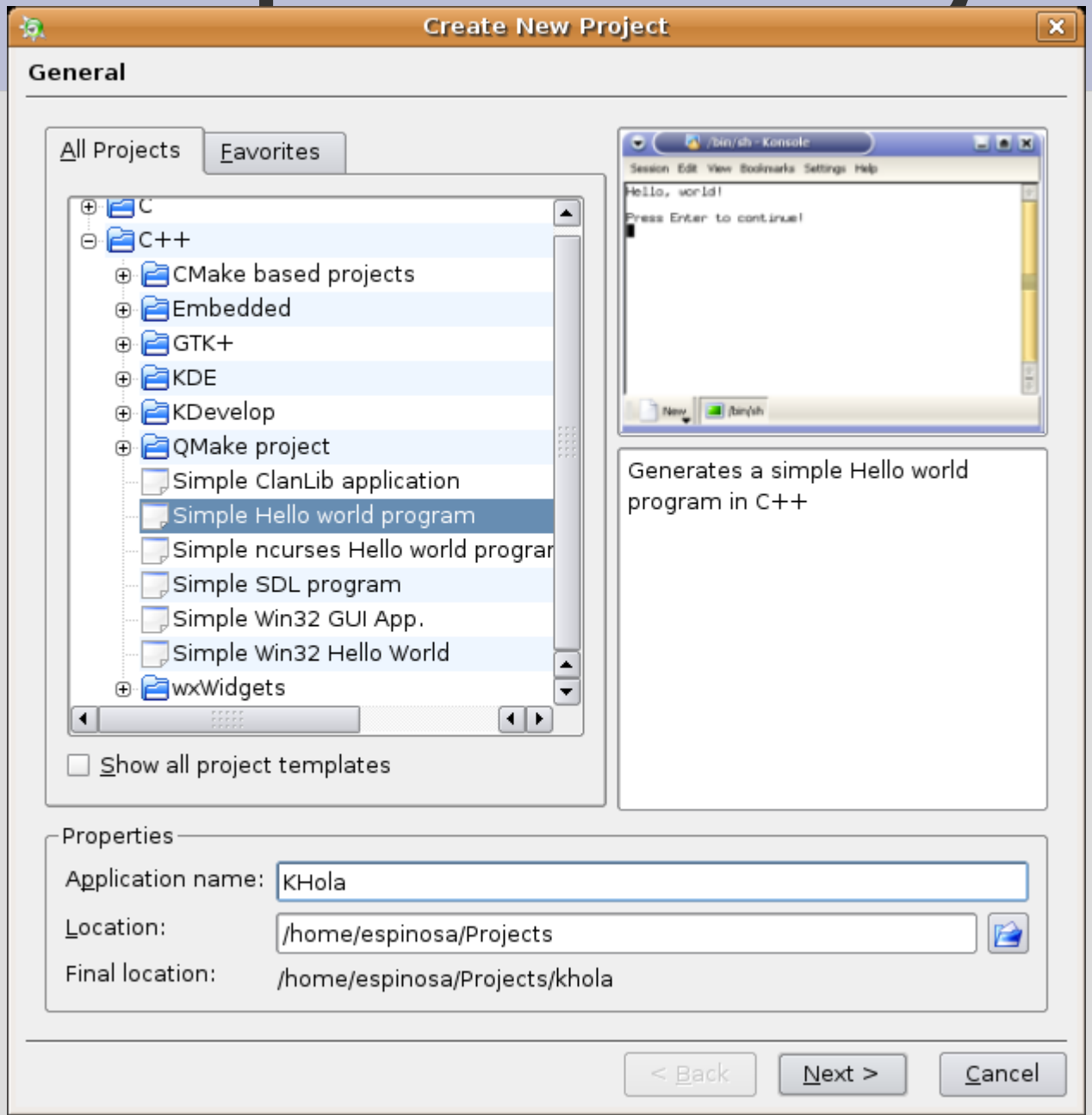
- Buen editor para C / C++ / GTK
- Integración con glade para interfaces
- Integración con autoconf / autotools
- Integración con CVS
- Depuración integrada
- Asistente para expresiones / funciones / métodos
- Integración con ayuda devhelp

- Kdevelop es el entorno más completo de desarrollo genérico de linux
- Surgió en 1998 para crear un entorno de desarrollo sencillo para KDE
- KDE está basado en QT, por eso Kdevelop integró primero esas librerías
- Actualmente soporta hasta 12 lenguajes distintos (inc. C, C++, PHP, Ruby, Java, etc.)

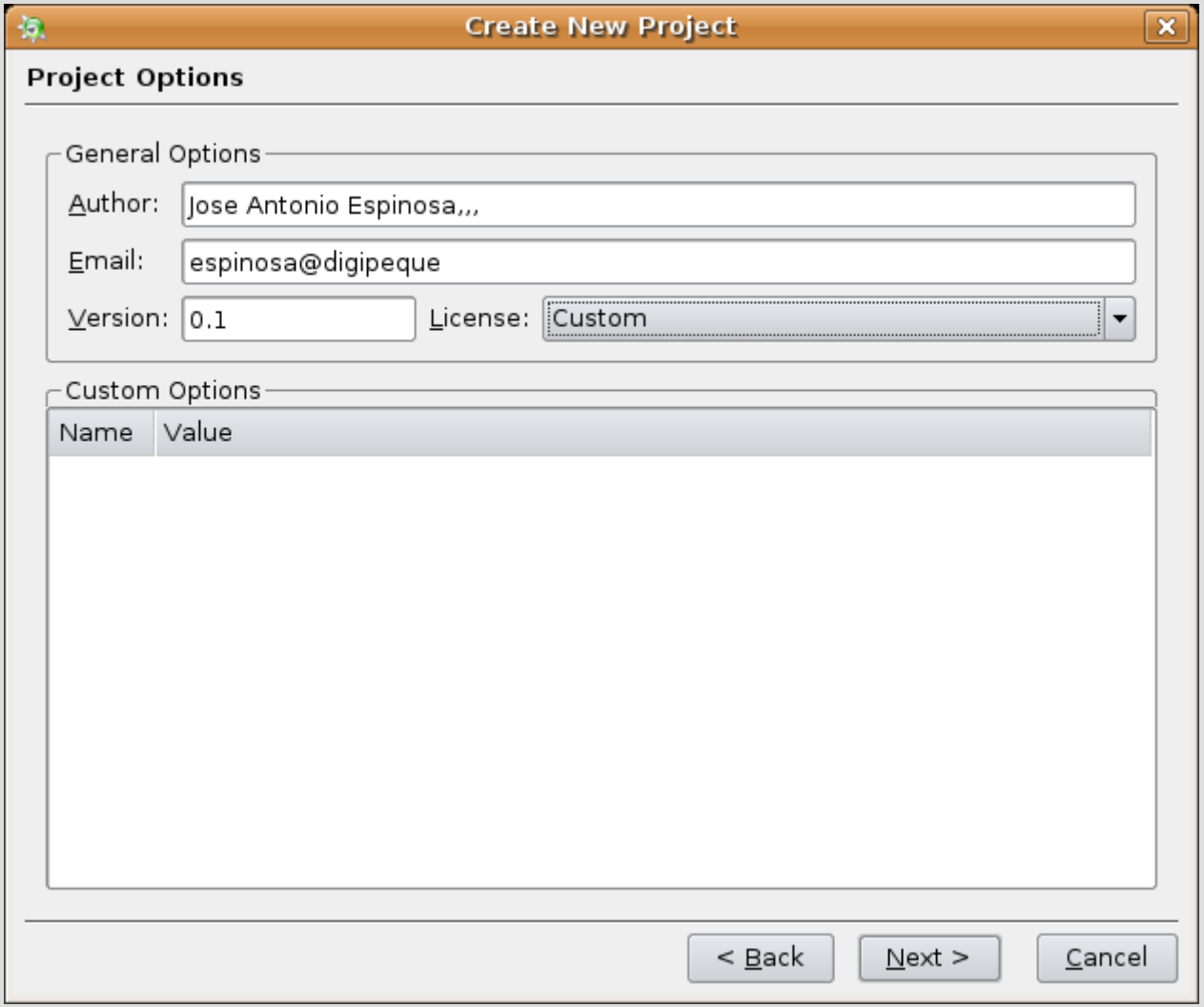
Kdevelop - Versiones

- Versión estable 3.5.0
  - KDE >= 3.4, < 4
  - Qt >= 3.3, < 4
  - 2007-10-16
- Versión en desarrollo 4.0
  - KDE >= 4
  - QT >= 4
  - Sin fecha de lanzamiento

Kdevelop – Nuevo Proyecto



Kdevelop – Nuevo proyecto (2)



Kdevelop – Nuevo proyecto (3)

- Permite elegir el sistema de control de versiones
  - CVS
  - SVN
  - ...
- Permite elegir plantillas para la construcción de los archivos fuente

Kdevelop - Entorno

khola - KDevelop

File Edit View Project Build Debug Scripts Bookmarks Window Tools Settings Help

(no function)

File Selector | Variables | File List | File Tree | Bookmarks | Classes | File Groups

khola.kdevelop
- src
  - main(int

khola.cpp

```
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <iostream>
#include <cstdlib>

using namespace std;

int main(int argc, char *argv[])
{
  cout << "Hello, world!" << endl;

  return EXIT_SUCCESS;
}
```

Documentation | Code Snippets | Automake Manager

Application | Diff | Messages | Valgrind | Find in Files | Replace

Line: 1 Col: 1 INS NORM

Kdevelop - Autoconf

khola - KDevelop

File Edit View Project Build Debug Scripts Bookmarks Window Tools Settings Help

(no function)

File Selector | Variables | File Tree | File List | Bookmarks | Classes | File Groups

:a/Projects/khola/

- src
- templates
- AUTHORS
- ChangeLog
- config.guess
- config.sub
- configure.in
- COPYING
- depcomp
- Doxyfile
- INSTALL
- install-sh
- khola.kdevelop
- ltmain.sh
- Makefile.am
- Makefile.cvs
- missing
- mkinstalldirs
- NEWS
- README
- TODO

khola.cpp

```
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <iostream>
#include <cstdlib>

using namespace std;

int main(int argc, char *argv[])
{
  cout << "Hello, world!" << endl;

  return EXIT_SUCCESS;
}
```

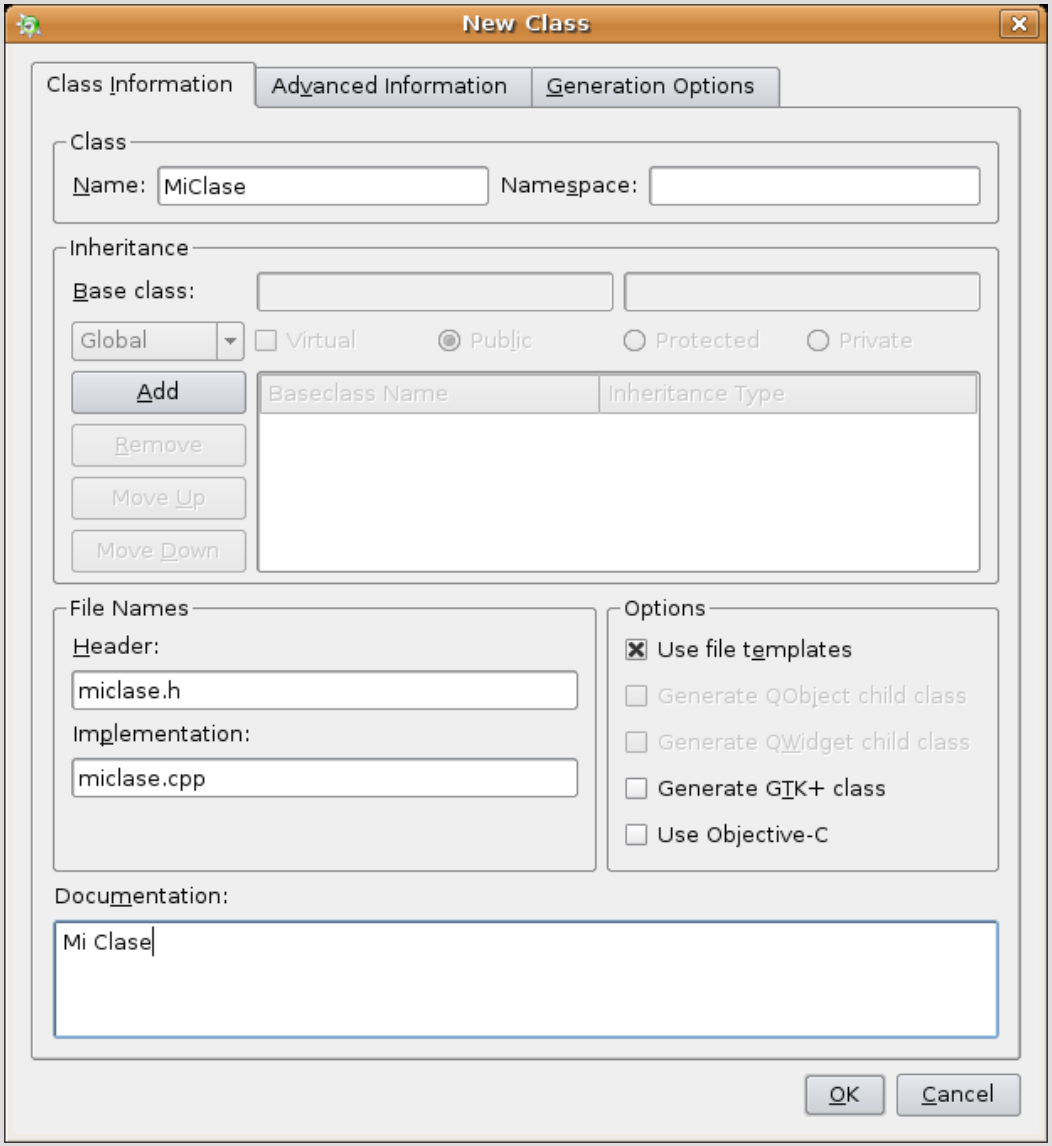
Documentation | Code Snippets | Automake Manager

Application | Diff | Messages | Valgrind | Find in Files | Replace

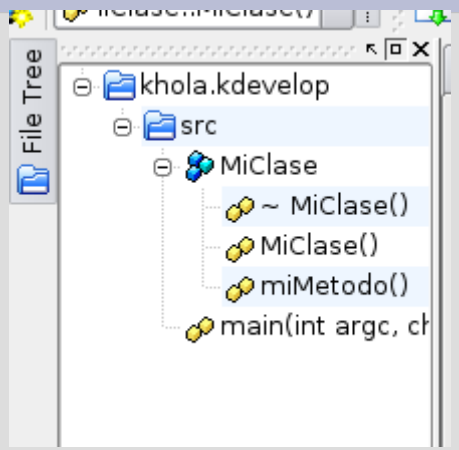
Line: 1 Col: 1 INS NORM

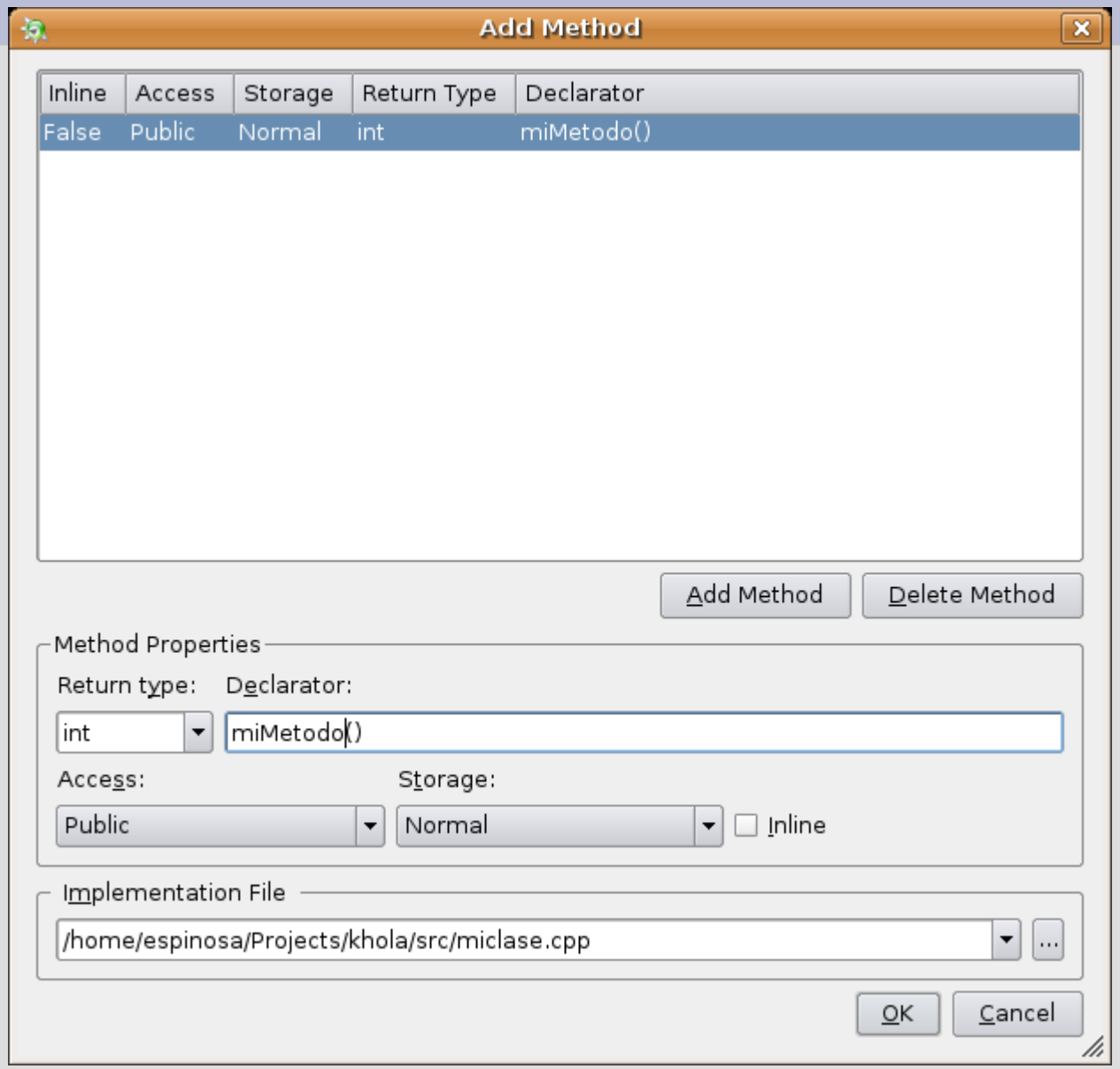
Kdevelop – Class builder

- Project
  - New Class



Kdevelop – Visor clases





Kdevelop - Construcción

- Permite compilar selectivamente
- Indica errores de sintaxis diréctamente sin compilar
- Permite invocar el configure
- Permite invocar las autotools
- Permite instalar el programa
- Permite generar la documentación (formato doxygen)

Kdevelop - depuración

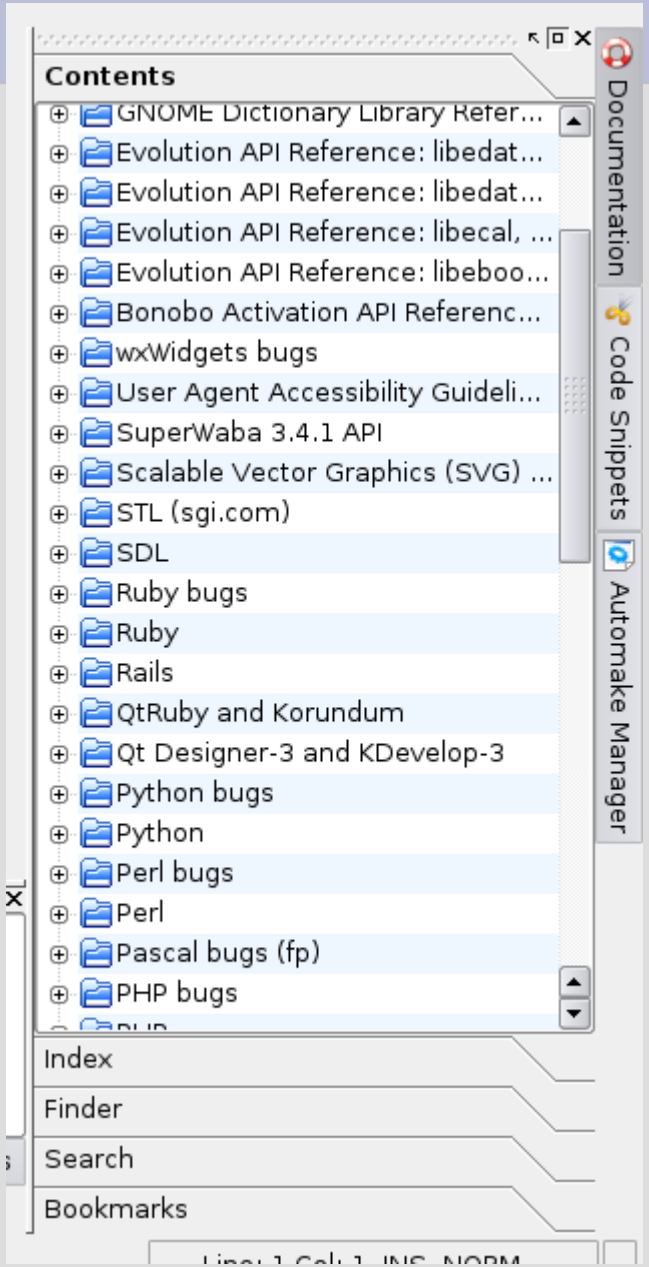
- Permite puntos de ruptura
- Permite depuración paso a paso
- Permite añadir watches y evaluar expresiones
- Permite unirse a un proceso en marcha
- Resultados por ventana del editor
- Inspección de memoria
- Uso de valgrind y Kcachegrid para profiling

Kdevelop - Gráfico

- Para su integración con QT incrusta el QT-designer como parte de su UI
- Permite crear e importar proyectos gtk
- Pero la integración con glade es muy pobre (hay que salir del entorno para editar con glade).

Kdevelop - Ayuda

- Ayuda muy completa
- Fácil de localizar en la barra lateral
- Ampliable con más paquetes
- Permite búsquedas e incluir marcas



Kdevelop - Conclusiones

- El mejor IDE para desarrollar en KDE
- Un buen editor C/C++ y otros lenguajes
- Buena integración con QT
- Más sistemas de control de versiones
- Integración con autoconf / autotools
- Depuración integrada mejorada
- Integración con profilers
- Ayuda integrada

Eclipse CDT

- Eclipse es un entorno de programación diseñado en Java
- Permite plugins de todo tipo
- CDT significa C/C++ development tooling
- Consiste en una serie de plugins y editores para el lenguaje C/C++
- Es multiplataforma (tantas como java) y tiene una comunidad muy amplia alrededor.

Eclipse CDT - Versiones

- Versión actual 4.0.2
  - 30 / 11 / 2007
  - Eclipse 3.3
  - Parte de eclipse europa
- CDT 3.1
  - 15 / 2 /2007
  - Eclipse 3.2
  - Parte de eclipse callisto

Eclipse CDT – Nuevo proyecto

C++ Project

Create C++ project of selected type

Project name: holamundo

☑ Use default location

Location: /home/espinosa/desarrollo/holamundo Browse...

Project types:

- Executable
  - Hello World C++ Project
  - Hello World ANSI C Project
  - Empty Project
- Shared Library
- Static Library
- Makefile project
  - Hello World C++ Project

Toolchain:

- Linux GCC

☑ Show project types and toolchains only if they are supported on the platform

< Back Next > Finish Cancel

Eclipse CDT – Nuevo proyecto

Basic Settings

Basic properties of a project

Author	Jose Antonio Espinosa
Copyright notice	Your copyright notice
Hello world greeting	Hola Mundo
Source	src

< Back | Next > | Finish | Cancel

Eclipse CDT – Nuevo proyecto

Select Configurations

Select platforms and configurations you wish to deploy on

Project type: Executable
Tool-chains: Linux GCC
Configurations:

- [x] Debug
- [x] Release

Select all
Deselect all
Advanced settings...

Use "Advanced settings" button to edit project's properties.

Additional configurations can be added after project creation.
Use "Manage configurations" buttons either on toolbar or on property pages.

< Back | Next > | Finish | Cancel

Eclipse CDT – Entorno

File Edit Refactor Navigate Search Run Project Window Help

C/C++

Project Explorer

- holamundo
  - Binaries
  - Includes
  - src
    - holamundo.cpp
  - Debug
- test2

holamundo.cpp

```cpp
//===========================================================================
// Name        : holamundo.cpp
// Author      : Jose Antonio Espinosa
// Version     :
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
//===========================================================================

#include <iostream>
using namespace std;

int main() {
    cout << "Hola Mundo" << endl; // prints Hola Mundo
    return 0;
}
```

Outl Mak

- iostream
- std
- main() : int

Problems Tasks Console Properties Debug

C-Build [holamundo]

```
Building target: holamundo
Invoking: GCC C++ Linker
g++  -o"holamundo"  ./src/holamundo.o
Finished building target: holamundo
```

Writable Smart Insert 1 : 1

Eclipse CDT – Class Builder

Create a new C++ class.

Source Folder: holamundo/src [Browse...]

Namespace: [Browse...]

Class Name: MiClase

Base Classes:

Name	Access	Virtual

[Add...] [Remove] [Up]

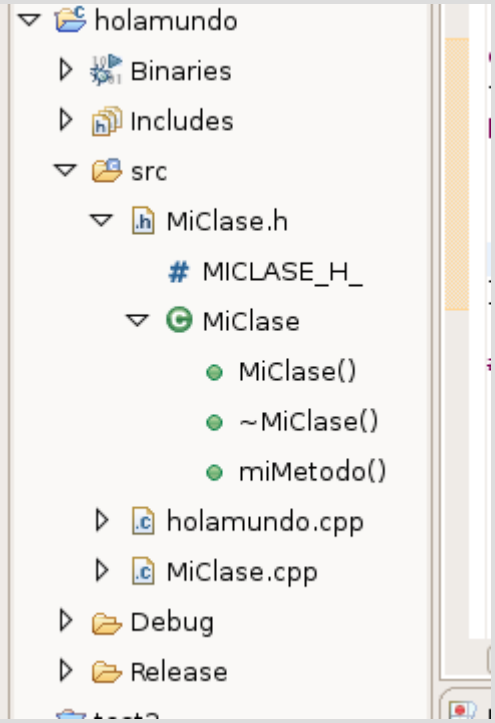
Method Stubs:

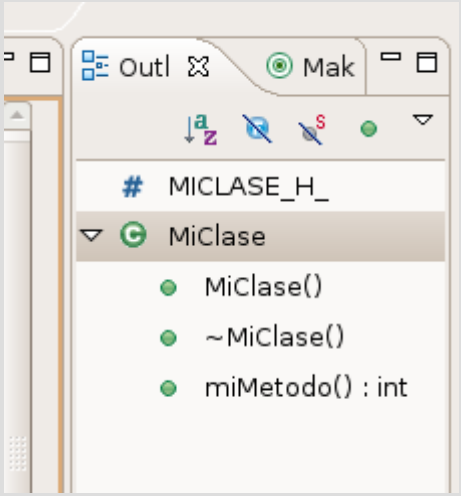
Name	Access	Virtual	Inline
Constructor	public	no	no
Destructor	public	yes	no

Use Default Header: MiClase.h [Browse...]

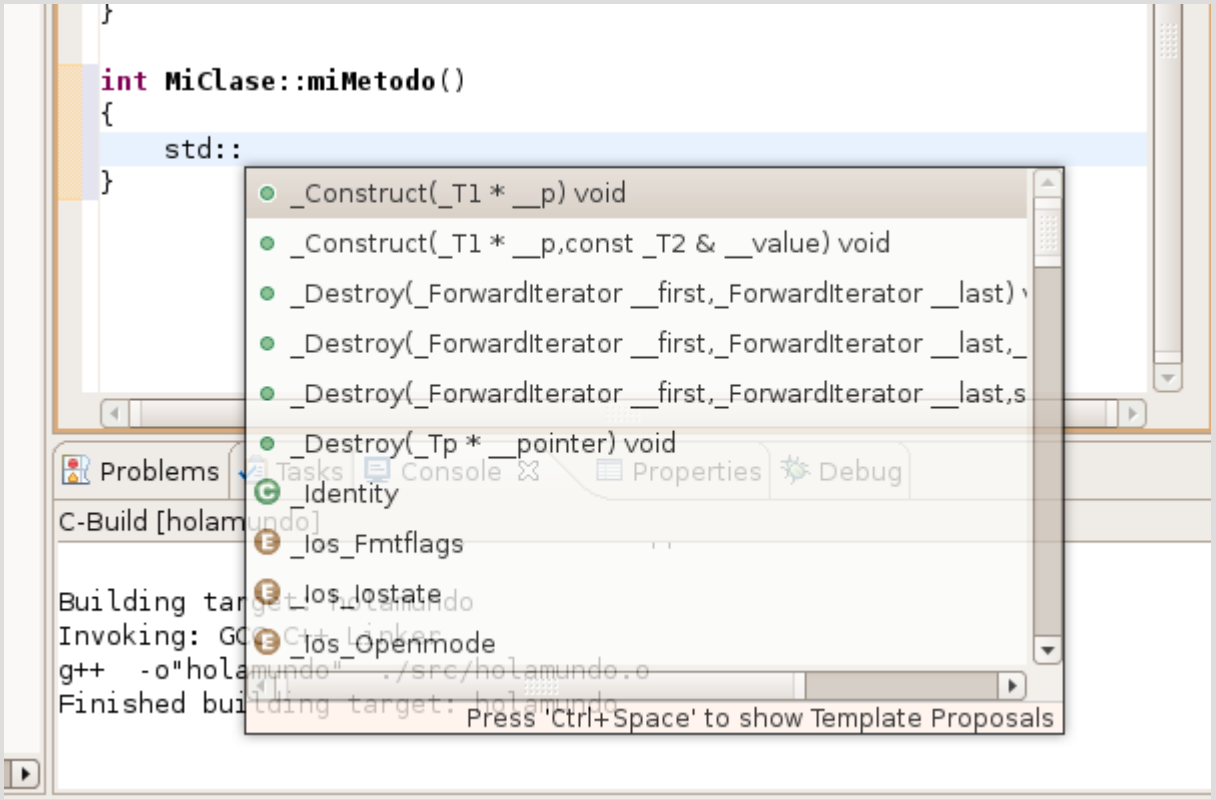
[Finish] [Cancel]

Eclipse CDT – Visor de clases





Eclipse CDT – asistente código



Eclipse CDT - Construcción

- No usa autoconf
- Sistema compatible con visual studio o con makefile
- Permite generar tantas configuraciones como necesitemos (Debug, Release, etc.)
- Intenta ser portable a más plataformas

Eclipse CDT - Depuración

File Edit Refactor Navigate Search Run Project Window Help

Debug

Debug

gdb/mi (13/02/08 9:28) (Suspended)
Thread [0] (Suspended)
1 main() /home/espinosa/desarrollo/holamundo/src/
gdb (13/02/08 9:28)
/home/espinosa/desarrollo/holamundo/Debug/holamundo

Variables Breakpoints Registers Modules

holamundo
ld-linux-x86-64.so.2
libstdc++.so.6
libm.so.6
libgcc_s.so.1
libc.so.6

holamundo.cpp MiClase.cpp MiClase.h

```
#include "MiClase.h"
using namespace std;

int main() {
    cout << "Hola Mundo" << endl; // prints Hola Mundo
```

Outline

iostream
MiClase.h
std
main() : int

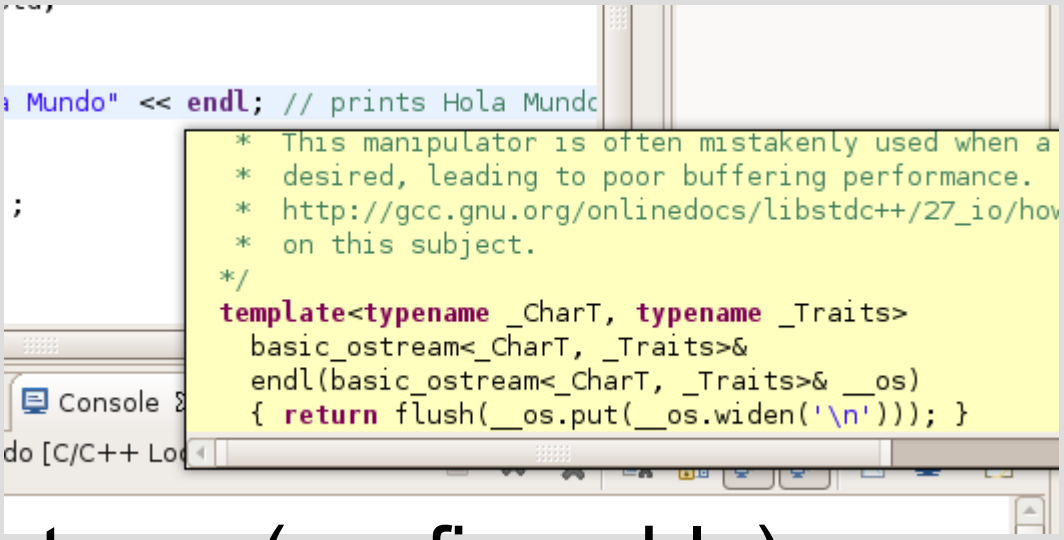
Console Tasks Problems Memory

holamundo [C/C++ Local Application] gdb (13/02/08 9:28)

```
Stopped due to shared library event
Stopped due to shared library event
```

Eclipse CDT - Ayuda

- Ayuda contextual



- Ayuda externa (configurable)

Eclipse CDT - Conclusiones

- Muy buen entorno multilenguaje
- Editor de código muy depurado
- Multiplataforma
- Pobre integración con autoconf
- Futura integración con QT, actualmente no se integra con ningún sistema gráfico

Curso LINUX

AREA 1: Multitarea - multihilo

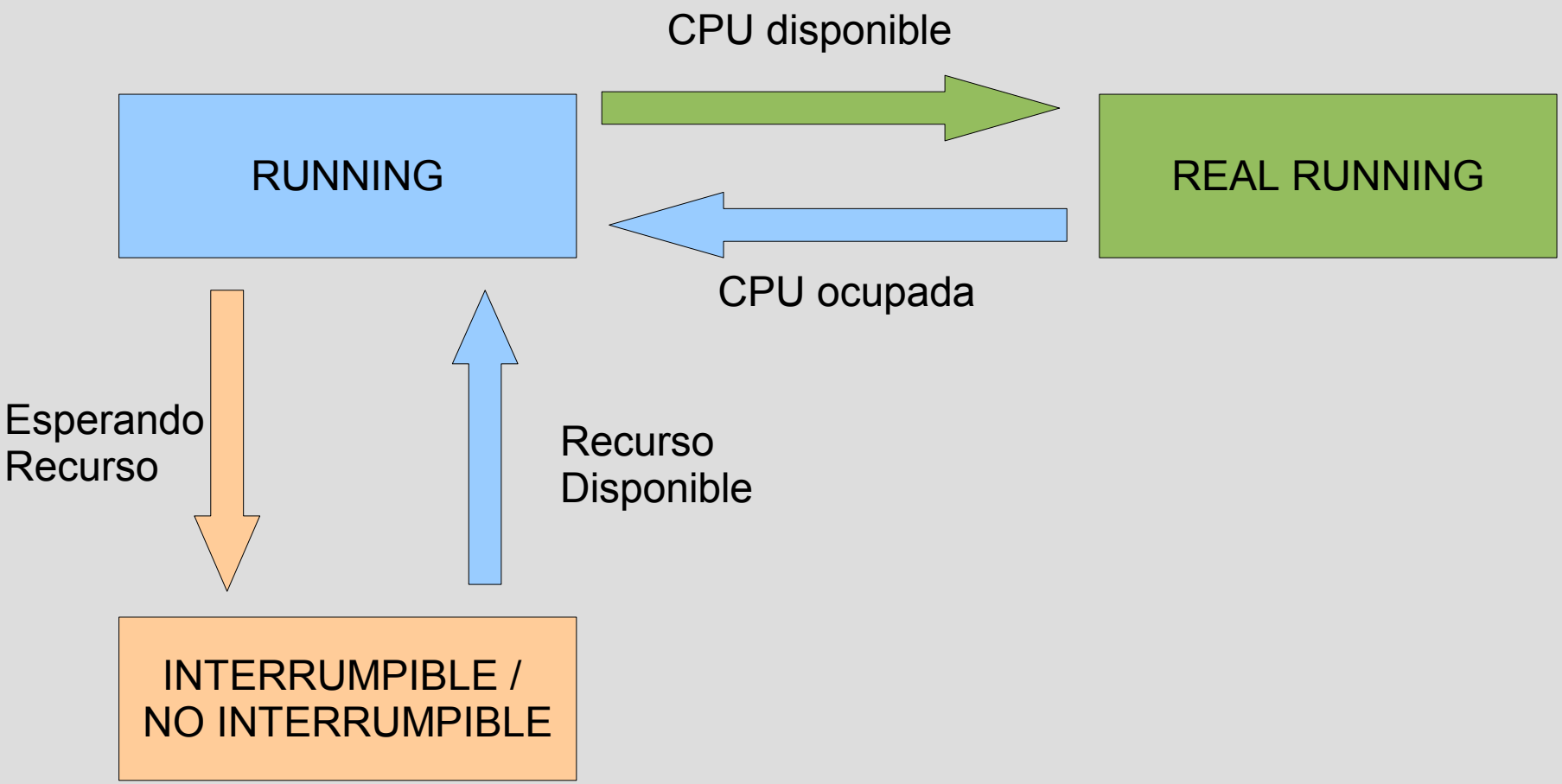
Multitarea

- Multitarea es un método donde varios procesos comparten recursos del ordenador
- Estrategias de multitarea:
  - Multiprogramming
  - Tiempo compartido cooperativo
  - Tiempo compartido preferente (preemptive)
  - Real

Linux Multitasking

- Una tarea en linux puede estar en uno de estos estados:
  - TASK_RUNNING
  - TASK_INTERRUPTIBLE
  - TASK_UNINTERRUPTIBLE
  - TASK_ZOMBIE
  - TASK_STOPPED

Transición de estados



Partición de tiempo

- En Linux, las “timeslice” suelen ser de 10ms
- Cada timeslice se genera una interrupción IRQ0
- Se interrumpe la ejecución del proceso en marcha y se realizan operaciones internas
- El Scheduler (planificador) es el encargado de decidir que tarea se ejecutará la próxima vez.

Task Switching

- El intercambio de tareas es necesario cuando:
  - Termina el timeslice y necesitamos dar acceso a otra tarea
  - Cuando una tarea decide acceder a un recurso y se queda en espera, hay que dar paso a otra
  - Cuando una tarea espera en una tubería (pipe) hay que dar paso a otra que podría escribir en el pipe.

FORK

- Cuando se necesita crear una tarea nueva, se utiliza fork
- fork copia todos los datos del proceso padre con las siguientes diferencias:
  - EL PID (identificador de proceso)
  - el resultado de la función fork es 0 en el hijo y PID en el padre
  - Las páginas de datos del hijo se copian en solo lectura

Procesos en linux

- Podemos observar los procesos con el comando **ps**
- Podemos ver el estado de ejecución general de la máquina con el comado **top**
- Podemos enviar señales a los procesos con el comando **kill**
- Si un proceso no puede tratar una señal, simplemente muere (de ahí el nombrecito)

ps

- **`ps`** muestra información sobre los procesos en ejecución
- **`ps`** : muestra solo los procesos del usuario
- **`ps ax`** : muestra todos los procesos
- **`ps axjf`** : muestra los árboles de proceso
- Existen múltiples opciones (**`man ps`**)

ps axu

espinosa@digipeque: ~

Archivo Editar Ver Terminal Solapas Ayuda

```
espinosa@digipeque:~$ ps axu
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   1564   532 ?        S    08:53   0:00 init [2]
root         2  0.0  0.0      0     0 ?        SN   08:53   0:00 [ksoftirqd/0]
root         3  0.0  0.0      0     0 ?        S    08:53   0:00 [watchdog/0]
root         4  0.0  0.0      0     0 ?        S<   08:53   0:00 [events/0]
root         5  0.0  0.0      0     0 ?        S<   08:53   0:00 [khelper]
root         6  0.0  0.0      0     0 ?        S<   08:53   0:00 [kthread]
root         8  0.0  0.0      0     0 ?        S<   08:53   0:00 [kblockd/0]
root         9  0.0  0.0      0     0 ?        S<   08:53   0:00 [kacpid]
root       148  0.0  0.0      0     0 ?        S    08:53   0:00 [pdflush]
root       149  0.0  0.0      0     0 ?        S    08:53   0:00 [pdflush]
root       151  0.0  0.0      0     0 ?        S<   08:53   0:00 [aio/0]
root       150  0.0  0.0      0     0 ?        S    08:53   0:00 [kswapd0]
root       738  0.0  0.0      0     0 ?        S<   08:53   0:00 [kseriod]
root      1828  0.0  0.0      0     0 ?        S<   08:53   0:00 [ata/0]
root      1829  0.0  0.0      0     0 ?        S<   08:53   0:00 [ata_hotplug/0]
root      1832  0.0  0.0      0     0 ?        S<   08:53   0:00 [scsi_eh_0]
root      1833  0.0  0.0      0     0 ?        S<   08:53   0:00 [scsi_eh_1]
root      1834  0.0  0.0      0     0 ?        S<   08:53   0:00 [scsi_eh_2]
root      1835  0.0  0.0      0     0 ?        S<   08:53   0:00 [scsi_eh_3]
root      1856  0.0  0.0      0     0 ?        S<   08:53   0:00 [khubd]
root      1874  0.0  0.0      0     0 ?        S    08:53   0:00 [khpsbpkt]
root      1904  0.0  0.0      0     0 ?        S    08:53   0:00 [knodemgrd_0]
```

ps : Códigos de estado

- **D** Uninterruptible sleep (usually IO)
- **R** Running or runnable (on run queue)
- **S** Interruptible sleep (waiting for an event to complete)
- **T** Stopped, either by a job control signal or because it is being traced.
- **W** paging (not valid since the 2.6.xx kernel)
- **X** dead (should never be seen)
- **Z** Defunct ("zombie") process, terminated but not reaped by its parent.

top

espinosa@digipeque: ~

Archivo Editar Ver Terminal Solapas Ayuda

```
top - 11:41:56 up  2:48,  3 users,  load average: 0.00, 0.01, 0.14
Tasks: 115 total,   3 running, 112 sleeping,   0 stopped,   0 zombie
Cpu(s): 29.1% us,  1.8% sy,  2.0% ni, 61.9% id,  4.6% wa,  0.5% hi,  0.1% si
Mem:   2068332k total,  2005840k used,    62492k free,   222456k buffers
Swap:  1228964k total,        0k used,  1228964k free,  1352744k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
13399 root      15   0  561m  39m  11m S  3.9  1.9  43:49.34 Xorg
    1 root      16   0  1564  532  460 S  0.0  0.0   0:00.97 init
    2 root      34  19     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
    3 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    4 root      10  -5     0    0    0 S  0.0  0.0   0:00.18 events/0
    5 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 khelper
    6 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 kthread
    8 root      10  -5     0    0    0 S  0.0  0.0   0:00.71 kblockd/0
    9 root      20  -5     0    0    0 S  0.0  0.0   0:00.00 kacpid
  148 root      20   0     0    0    0 S  0.0  0.0   0:00.00 pdflush
  149 root      15   0     0    0    0 S  0.0  0.0   0:00.12 pdflush
  151 root      19  -5     0    0    0 S  0.0  0.0   0:00.00 aio/0
  150 root      15   0     0    0    0 S  0.0  0.0   0:00.38 kswapd0
  738 root      10  -5     0    0    0 S  0.0  0.0   0:00.06 kseriod
 1828 root      11  -5     0    0    0 S  0.0  0.0   0:00.00 ata/0
 1829 root      11  -5     0    0    0 S  0.0  0.0   0:00.00 ata_hotplug/0
 1832 root      11  -5     0    0    0 S  0.0  0.0   0:00.00 scsi_eh_0
```

Práctica

- Crear un programa en c que:
  - Cree un proceso hijo
  - El proceso hijo se quede en un bucle infinito o duerma durante cierto tiempo
  - El proceso padre volverá a lanzar al hijo si detecta que este ha muerto. (wait o waitpid)
- Monitorizar el proceso con ps
- Matar el proceso hijo (kill -9) y observar cómo se levanta

Threads

- El intercambio de procesos conlleva una carga pesada al tener que realizar un “cambio de contexto” de todos los datos del proceso que abandona y el nuevo
- Los threads (hilos) son elementos de ejecución independiente con datos compartidos, que, por tanto, no requieren de un cambio de contexto completo.

Multithreading

- El modelo de ejecución multihilo permite a varios hilos existir dentro del mismo contexto de un único proceso.
- Los hilos comparten todos los recursos del proceso en el que existen
- Esto permite, entre otras cosas, permitir paralelismo dentro de un único proceso.

Implementaciones de hilos

- Light weight kernel threads (BSD)
- POSIX Threads (pthreads)
- Apple Multiprocessing Services
- GNU portable Threads (GNU Pth)
- Windows Threads

Programación de hilos

- Librería pthread

```
#include <pthread.h>

int pthread_create(pthread_t *restrict thread,
          const pthread_attr_t *restrict attr,
          void *(*start_routine)(void*), void *restrict arg);

int pthread_join(pthread_t thread, void **value_ptr);
pthread_t pthread_self(void);


compilar con -lpthread
```

Ejemplo

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void *print_message_function( void *ptr );

main()
{
     pthread_t thread1, thread2;
     char *message1 = "Thread 1";
     char *message2 = "Thread 2";
     int  iret1, iret2;

     iret1 = pthread_create( &thread1, NULL, print_message_function, (void*) message1);
     iret2 = pthread_create( &thread2, NULL, print_message_function, (void*) message2);

     pthread_join( thread1, NULL);
     pthread_join( thread2, NULL);

     printf("Thread 1 returns: %d\n",iret1);
     printf("Thread 2 returns: %d\n",iret2);
     exit(0);
}

void *print_message_function( void *ptr )
{
     char *message;
     message = (char *) ptr;
     printf("%s \n", message);
}
```

Ejemplo práctico

- Crear un programa que lance 3 threads con funciones que impriman cosas distintas

Sincronización de threads

- Mutex
- Join
- Variables de condición

Mutex

- Creados para evitar condiciones de carrera e interbloqueos
- Sirven para proteger el acceso a elementos compartidos
- Funciones:

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Efecto de un mutex

Without Mutex	With Mutex

`int counter=0;`
 	
`/* Function C */`
`void functionC()`
`{`
 	
`  counter++`
 	
`}`
	`/* Note scope of variable and mutex are the same */`
	`pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;`
	`int counter=0;`
	
	`/* Function C */`
	`void functionC()`
	`{`
	`  pthread_mutex_lock( &mutex1 );`
	`  counter++`
	`  pthread_mutex_unlock( &mutex1 );`
	`}`

Possible execution sequence

Thread 1	Thread 2	Thread 1	Thread 2
counter = 0	counter = 0	counter = 0	counter = 0
counter = 1	counter = 1	counter = 1	Thread 2 locked out. Thread 1 has exclusive use of variable `counter`
			counter = 2

Joins

- Un join se ejecuta cuando se quiere esperar que un thread termine.
- Una rutina puede lanzar múltiples threads y esperar que terminen para obtener los resultados.
- Se espera que los threads terminen mediante un join.
- Similar al wait o waitpid de los fork

Ejemplo

- Crear un programa que lance 3 threads, que cada uno espere un tiempo aleatorio, imprima un mensaje y termine
- El programa principal deberá esperar hasta que terminen todos e imprimir un mensaje

Variables de condición

- Tipo `pthread_cond_t`
- Se usan con las funciones adecuadas para esperar y continuar
- Se permite a los threads suspender la ejecución y renunciar a la CPU hasta que se de una condición.
- Siempre asociadas a mutex para evitar interbloqueos

Funciones

- Crear / Destruir
  - `pthread_cond_init`
  - `pthread_cond_t cond = PTHREAD_COND_INITIALIZER;`
  - `pthread_cond_destroy`
- Esperar una condición:
  - `pthread_cond_wait`
  - `pthread_cond_timedwait`
- Despertar threads
  - `pthread_cond_signal`
  - `pthread_cond_broadcast`

Ejemplo

```
void *functionCount1()
{
   for(;;)
   {
      pthread_mutex_lock( &condition_mutex );
      while( count >= COUNT_HALT1 && count <= COUNT_HALT2 )
      {
         pthread_cond_wait( &condition_cond, &condition_mutex );
      }
      pthread_mutex_unlock( &condition_mutex );

      pthread_mutex_lock( &count_mutex );
      count++;
      printf("Contador functionCount1: %d\n",count);
      pthread_mutex_unlock( &count_mutex );

      if(count >= COUNT_DONE) return(NULL);
    }
}
```

Ejemplo (2)

```
void *functionCount2()
{
    for(;;)
    {
       pthread_mutex_lock( &condition_mutex );
       if( count < COUNT_HALT1 || count > COUNT_HALT2 )
       {
          pthread_cond_signal( &condition_cond );
       }
       pthread_mutex_unlock( &condition_mutex );

       pthread_mutex_lock( &count_mutex );
       count++;
       printf("Contador functionCount2: %d\n",count);
       pthread_mutex_unlock( &count_mutex );

       if(count >= COUNT_DONE) return(NULL);
    }
```

Ejemplo (3)

```
int  count = 0;
#define COUNT_DONE  10
#define COUNT_HALT1  3
#define COUNT_HALT2  6

main()
{
   pthread_t thread1, thread2;

   pthread_create( &thread1, NULL, &functionCount1, NULL);
   pthread_create( &thread2, NULL, &functionCount2, NULL);
   pthread_join( thread1, NULL);
   pthread_join( thread2, NULL);

   exit(0);
}
```

¿Qué saldría por pantalla?

Referencias

- http://www.kplug.org/glade_tutorial/glade2_tu
- http://anjuta.sourceforge.net/documentations/
- http://www.kdevelop.org/
- http://www.kdevelop.org/index.html?filename=
- http://www.eclipse.org/cdt/
- http://tldp.org/HOWTO/KernelAnalysis-HOWT
- www.yolinux.com/TUTORIALS/LinuxTutorial PosixThreads.html