《Linux 操作系统设计实践》

实验报告

实验 5：图形介绍

院　　系：数学与计算机科学学院

专　　业：计算机科学与技术

年　　级：2016 级计算机 5 班

学　　号：031602507

姓　　名：陈俞辛

一、实验环境：Ubuntu　16.04

二、实验内容：

（一）实验代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gtk/gtk.h>

#define STRING_LEN 64
#define FILE_NAME "ToDoList.txt"

GtkWidget *uiList;
GtkWidget *textToDo;
GtkWidget *textDeadLine;
char selectText[STRING_LEN*2] = {0};
const gchar *list_item_data_key="list_item_data";

typedef struct _Node
{
	char name[STRING_LEN];
	char num[STRING_LEN];
	struct _Node *next;
}Node;

Node *head = NULL;   //链表头
Node *tail = NULL;    //链表尾

void DelList()       //清空链表
{
	if(head==NULL)
		return;
	Node *p = head;
	while(p)
	{
		Node *tmp = p->next;
		free(p);
		p = tmp;
	}
	head = NULL;
	tail = NULL;
}

void InitList()       //初始化
{
	DelList();
	head = (Node *)malloc(sizeof(Node));
	head->next = NULL;
	tail = head;
}
```

```
void InsertNode(Node *node)//添加结点
{
	if(head==NULL)
		return;
	tail->next = node;
	tail = node;
	tail->next = NULL;
}

void DelNode(const char *str)//删除结点
{
	if(head==NULL)
		return;
	Node *pl = head, *pr = head->next;
	while(pr)
	{
		char buf[STRING_LEN*2]={0};
		sprintf(buf,"ToDo:%s \t DeadLine:%s",pr->name,pr->num);
		if(strcmp(buf,str)==0)
		{
			if(pr->next==NULL)
			{
				tail = pl;
			}
			pl->next = pr->next;
			free(pr);
			pr = pl->next;
			break;
		}
		else
		{
			pl = pr;
			pr = pr->next;
		}
	}
}

void ShowTip(const char *info)
{
	GtkWidget *dialog =
gtk_message_dialog_new(NULL,GTK_DIALOG_MODAL|GTK_DIALOG_DESTROY_WIT
H_PARENT,GTK_MESSAGE_INFO,GTK_BUTTONS_OK,"%s",info);
	gtk_dialog_run(GTK_DIALOG(dialog));
	gtk_widget_destroy(dialog);
}
```

```
void ReadFile()//从文件读取ToDoList
{
	FILE *fp = fopen(FILE_NAME,"r");
	if(fp==NULL)
		return;
	DelList();
	InitList();
	char ToDoStr[STRING_LEN],DDLStr[STRING_LEN];
	while(fscanf(fp,"%s %s",ToDoStr,DDLStr)==2)
	{
		Node *pnew = (Node *)malloc(sizeof(Node));
		strcpy(pnew->name,ToDoStr);
		strcpy(pnew->num,DDLStr);
		InsertNode(pnew);
	}
	fclose(fp);
}

void WriteFile()//保存数据到ToDoList文件
{
	FILE *fp = fopen(FILE_NAME,"w");
	if(fp==NULL)
		return;
	Node *p = head->next;
	while(p)
	{
		fprintf(fp,"%s %s\n",p->name,p->num);
		p = p->next;
	}
	fclose(fp);
}


void AddNodeToList(const char *ToDoStr, const char *DDLStr)//向uiList中加入元素
{
	char str[STRING_LEN*2] = {0};
	sprintf(str, "ToDo:%s \t DeadLine:%s",ToDoStr,DDLStr);

	gchar *string;
	GtkWidget *label=gtk_label_new (str);
	GtkWidget *list_item = gtk_list_item_new ();
	gtk_container_add (GTK_CONTAINER (list_item), label);
	gtk_widget_show (label);
	gtk_container_add (GTK_CONTAINER (uiList), list_item);
	gtk_widget_show (list_item);
	gtk_label_get (GTK_LABEL (label), &string);
	g_object_set_data (G_OBJECT (list_item), list_item_data_key, string);
}
```

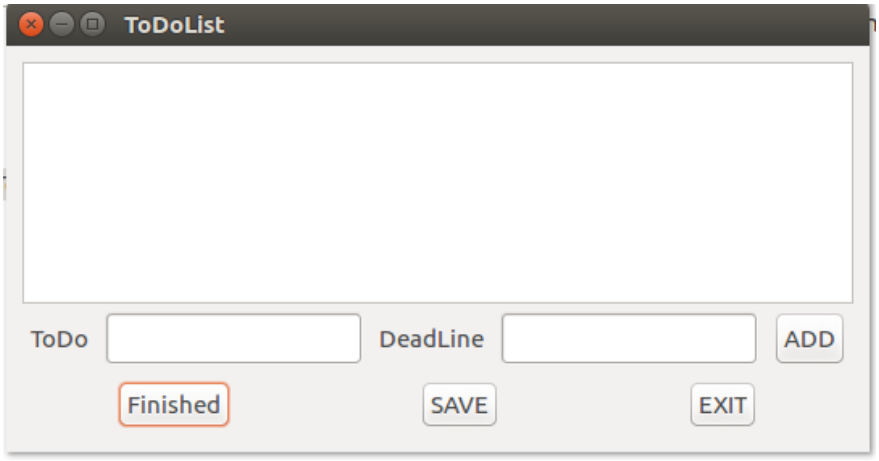
```
void onBtnInsert(GtkWidget *widget, gpointer data)//点击ADD按钮
{
	const char *ToDoStr = gtk_entry_get_text((GtkEntry *)textToDo);
	const char *DDLStr   = gtk_entry_get_text((GtkEntry *)textDeadLine);
	if(strlen(ToDoStr)==0 || strlen(DDLStr)==0)
		return;

	Node *pnew = (Node *)malloc(sizeof(Node));
	sprintf(pnew->name,"%s",ToDoStr);
	sprintf(pnew->num,"%s",DDLStr);
	InsertNode(pnew);

	AddNodeToList(ToDoStr,DDLStr);

	gtk_entry_set_text((GtkEntry *)textToDo,"");
	gtk_entry_set_text((GtkEntry *)textDeadLine,"");
	ShowTip("ADD SUCCESSED");
}
void onBtnDelete(GtkWidget *widget, gpointer data)//点击Finished按钮
{
	gtk_list_clear_items(GTK_LIST(uiList),0,-1);
	DelNode(selectText);
	Node *p = head->next;
	while(p)
	{
		AddNodeToList(p->name,p->num);
		p = p->next;
	}
	ShowTip("Congratulations!");
}
void onBtnSave(GtkWidget *widget, gpointer data)//点击SAVE按钮
{
	WriteFile();
	if(widget!=0 || data!=0)
		ShowTip("SAVE SUCCESSED");
}
void onBtnRead(GtkWidget *widget, gpointer data)//READ按钮
{
	gtk_list_clear_items(GTK_LIST(uiList),0,-1);
	ReadFile();
	Node *p = head->next;
	while(p)
	{
		AddNodeToList(p->name,p->num);
		p = p->next;
	}
	if(widget!=0 || data!=0)
		ShowTip("READ SUCCESSED");
}
void onBtnExit(GtkWidget *widget, gpointer data)//点击EXIT按钮
{
	gtk_main_quit();
	onBtnSave(0,0);
}
```

```
void OnListSelectionChanged(GtkWidget *gtklist, gpointer func_data)//选择的uiList项目发生变化触发
{
	GList *dlist = GTK_LIST(gtklist)->selection;
	if (!dlist)
	{
		return;
	}
	else
	{
		if(dlist!=NULL)
		{
			const char *buf = g_object_get_data(G_OBJECT(dlist->data),list_item_data_key);
			sprintf(selectText,"%s",buf);
		}
	}
}

void UI_Init(int argc, char *argv[])//初始化界面
{
	gtk_init(&argc,&argv);
	GtkWidget *window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
	gtk_window_set_default_size(GTK_WINDOW(window),100,250);
	g_signal_connect(G_OBJECT(window),"delete_event",G_CALLBACK(gtk_main_quit),NULL);
	gtk_window_set_title(GTK_WINDOW(window),"ToDoList");
	gtk_window_set_position(GTK_WINDOW(window),GTK_WIN_POS_CENTER);
	gtk_container_set_border_width(GTK_CONTAINER(window),10);
	//--------------------垂直线性布局----------------------
	GtkWidget *vbox=gtk_vbox_new(0,0);
	gtk_container_add(GTK_CONTAINER(window),vbox);
	//-------------------------------------------------------


	GtkWidget *scrolled_window = gtk_scrolled_window_new (NULL, NULL);//可拖动窗口
	gtk_widget_set_size_request (scrolled_window, 250, 150);
	gtk_container_add (GTK_CONTAINER (vbox), scrolled_window);


	uiList = gtk_list_new();//放于可拖动窗口中的uiList
	gtk_scrolled_window_add_with_viewport (GTK_SCROLLED_WINDOW (scrolled_window),uiList);
	g_signal_connect (uiList, "selection-changed", G_CALLBACK (OnListSelectionChanged), NULL);
	//------------------水平布局---------------------------
	GtkWidget *hbox1 = gtk_hbox_new(0,0);
	gtk_box_pack_start(GTK_BOX(vbox),hbox1,FALSE,FALSE,5);

	GtkWidget *lab_name = gtk_label_new("ToDo");
	gtk_box_pack_start(GTK_BOX(hbox1),lab_name,1,0,5);

	textToDo = gtk_entry_new();
	gtk_box_pack_start(GTK_BOX(hbox1),textToDo,1,0,5);
```

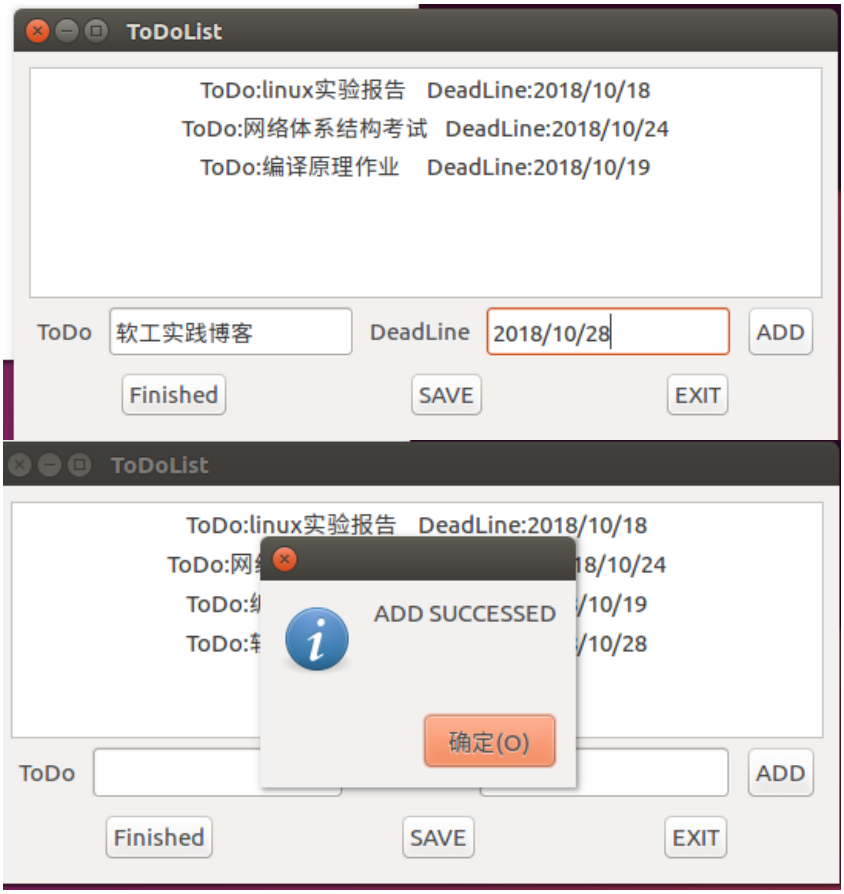
```
    GtkWidget *lab_num = gtk_label_new("DeadLine");
    gtk_box_pack_start(GTK_BOX(hbox1),lab_num,1,0,5);

    textDeadLine = gtk_entry_new();
    gtk_box_pack_start(GTK_BOX(hbox1),textDeadLine,1,0,5);

    GtkWidget *btn_add = gtk_button_new_with_label("ADD");
    gtk_box_pack_start(GTK_BOX(hbox1),btn_add,1,0,5);
    g_signal_connect(G_OBJECT(btn_add),"clicked",G_CALLBACK(onBtnInsert),NULL);

    //-------放入Finished、SAVE、EXIT3个按钮--------
    GtkWidget *hbox2 = gtk_hbox_new(0,0);
    gtk_box_pack_start(GTK_BOX(vbox),hbox2,FALSE,FALSE,5);

    GtkWidget *btn_delete = gtk_button_new_with_label("Finished");
    gtk_box_pack_start(GTK_BOX(hbox2),btn_delete,1,0,5);
    g_signal_connect(G_OBJECT(btn_delete),"clicked",G_CALLBACK(onBtnDelete),NULL
);

    GtkWidget *btn_save = gtk_button_new_with_label("SAVE");
    gtk_box_pack_start(GTK_BOX(hbox2),btn_save,1,0,5);
    g_signal_connect(G_OBJECT(btn_save),"clicked",G_CALLBACK(onBtnSave),NULL);


    GtkWidget *btn_exit = gtk_button_new_with_label("EXIT");
    gtk_box_pack_start(GTK_BOX(hbox2),btn_exit,1,0,5);
    g_signal_connect(G_OBJECT(btn_exit),"clicked",G_CALLBACK(onBtnExit),NULL);
    //-----------------------------------------------------
    onBtnRead(0,0);//从文件读取数据放入链表
    gtk_widget_show_all(window);//显示所有窗体
    gtk_main();
}

int main(int argc, char *argv[])
{
    InitList();                 //初始化链表
    UI_Init(argc, argv);//初始化UI
    DelList();                  //释放链表
    return 0;
}
```

（二）实验结果

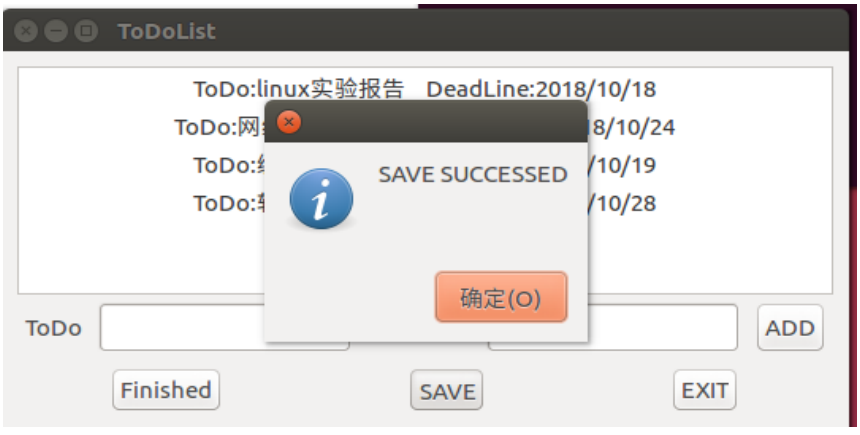
1.原始界面，此时还没有任何的待办事项



2.加入待办事项，添加成功会提示信息

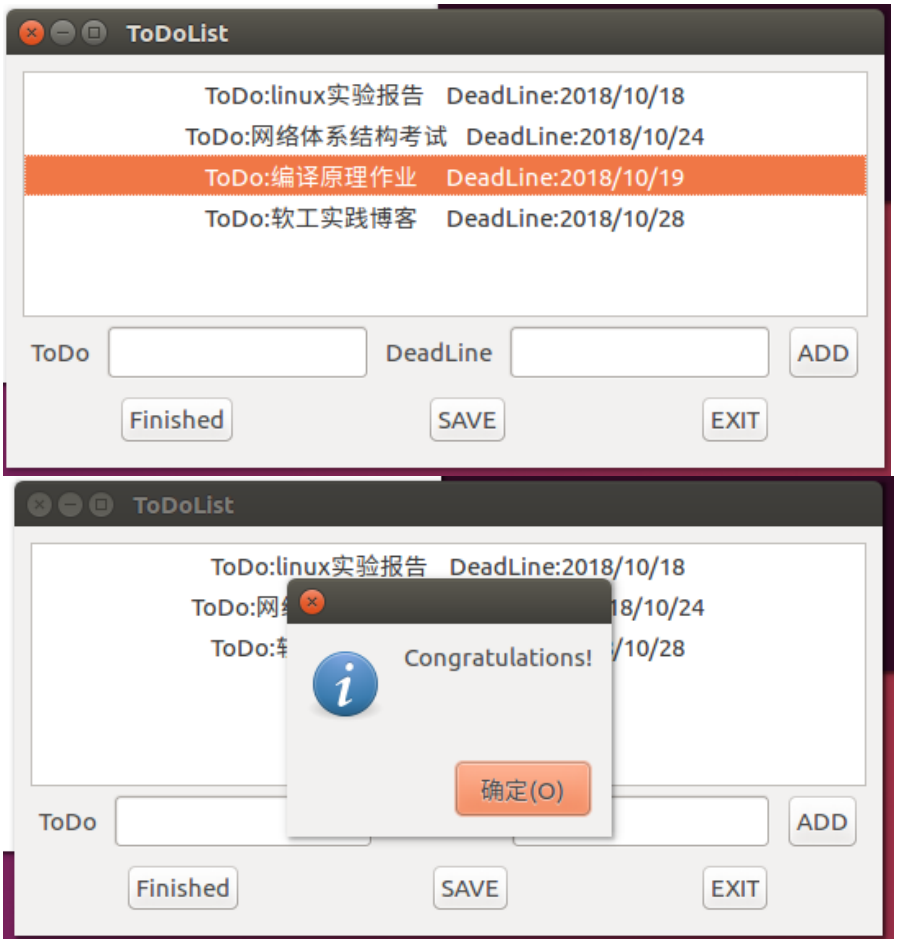


3.加入待办事项之后保存至ToDoList.txt文件，保存成功会有提示信息



4.

可以选中已完成的待办事项可以从中ToDoList中删除，删除成功会有提示信息



5.保存的ToDoList可以在ToDoList.txt文件中查看

6.其他无法截图展示的功能有：点击 EXIT 按钮退出后会自动将ToDoList的内容保存至ToDoList.txt文件；每次打开程序，会自动从ToDoList.txt中读取内容并展示。

三、 实验总结

（一）编程思想

程序实现的功能主要有以下几个：添加待办事项、删除已完成的待办事项、将待办事项保存至 txt 文件、打开程序读取 txt 文件中待办事项。

考虑到删除已完成的待办事项的时候，未必是按序删除，因此采用链表的数据结构来存储数据。维持一个链表，表中各个结点保存各个待办事项。

添加待办事项——向链表内添加一个结点，点击 ADD 按钮时触发

删除已完成的待办事项——从链表中将对应的结点删除，点击 Finished 按钮时触发

将待办事项保存至 txt 文件——遍历链表将各节点的数据存入文件，点击 SAVE 或者 EXIT 按钮时触发

打开程序读取 txt 文件中待办事项——读取文件将数据存入链表，打开程序时自动触发

（二）相关知识

本次实验用到大量 GTK 函数，这些列出一些重要的。

1、gtk_window_new（GTK_WINDOW_TOPLEVEL）

函数创建一个窗口并返回这个窗口的控件指针。

参数 GTK_WINDOW_TOPLEVEL 指明窗口的类型为最上层的主窗口，它最常用。还可以取另一个值 GTK_WINDOW_POPUP 指明窗口的类型为弹出式的无边框的窗口。

2、g_signal_connect（）

使用这个宏为窗口或控件加回调函数。

g_signal_connect 宏有 4 个参数，分别是：

连接的对象，就是要连接信号的控件的指针(注意：必须是已创建完的控件的指针)，需要用 G_OBJECT 宏来转换；

信号名称，就是要连接的信号名称，为字符串形式，用双引号引起来。不同的控件拥有的信号名称是不一样的；

回调函数，指信号发生时调用的函数，这里只用到函数名称，需要用 G_CALLBACK 宏来转换一下；

传递给回调函数的参数，它的值类型应该为 gpointer。如果不是这一类型可以强制转换，如果没有参数则为 NULL。这里只能传递一个参数，如果有多个参数，可以先将多个参数定义为一个结构，再将此结构作为参数传递过来

3、改变窗口外观的几个函数

设定窗口的标题：

```
gtk_window_set_title(window,const gchar* title);
```

设定窗口的默认宽高：

```
gtk_window_set_default_size(window,int width,int height);
```

设定窗口的位置：

```
gtk_window_set_position(window,GtkWindowPosition position);
```

其中 position 可以取如下值：

GTK_WIN_POS_NONE 不固定
GTK_WIN_POS_CENTER 居中
GTK_WIN_POS_MOUSE 出现在鼠标位置
GTK_WIN_POS_CENTER_ALWAYS 窗口改变尺寸仍居中
GTK_WIN_POS_CENTER_ON_PARENT 居于父窗口的中部

4、gtk_container_add ()

功能是将另一控件加入到容器中来。

它的第一参数是 GtkContainer 型的指针，这就需要将窗口控件指针用宏 GTK_CONTAINER 转换一下，即 GTK_CONTAINER(window)。它的第二参数是要容纳的控件的指针，即 button。

5、向盒状容器添加并排列控件

用 gtk_box_pack_*系列函数向盒状容器添加并排列控件，这样的函数一共有 2 个，分别是： gtk_box_pack_start、 gtk_box_pack_end。

gtk_box_pack_start、 gtk_box_pack_end 分别表示按顺序从前到后依次排列控件和从后到前依次排列控件。这两个函数都有 5 个参数，依次是 GTK_BOX(box)，要容纳控件的容器对象； button，被容纳控件的指针；是否扩展，是否添充和与前一控件的间隔。例如本次实验中用到的：

```
gtk_box_pack_start(GTK_BOX(vbox),hbox1,FALSE,FALSE,5);
```

6、gtk_widget_show_all（）

原本每一个控件都要用函数 gtk_widget_show 来显示，而这个函数显示容器中所有控件。

它的参数是一个容器控件的指针，例如本次实验中用到的：

```
gtk_widget_show_all(window);
```

（三）收获及展望

一开始看到实验题目觉得很困难，后来仔细阅读了老师提供的资料之后，做了一个简单的图形化界面。学会了如何布局，如何添加控件并为其绑定相应的动作等等。当然因为时间有限，程序还有很多不足，例如没办法设置优先级等。如果接下来有机会希望能够完善这个程序。