TCD - Shell Gráfico

Graduação em Ciência da Computação - UFU
Disciplina: Sistemas Operacionais (GBC045)
Professor: Rivalino Matias
Nomes do aluno: Luis B. S. Bulhões, Paulo V. C. Silva, Pedro H. R. Ribeiro

Sumário

1 Introdução

O objetivo deste trabalho é criar um programa denominado *shell* gráfico. Para desenvolver o programa, são requisitados conhecimentos de GUI (Graphical User Interface) e bibliotecas que permitirão a criação e a manipulação dos elementos que compõem a interface gráfica. A GUI é uma forma de interface de usuário que permite a iteração entre dispositivos eletrônicos e seus usuários através de ícones gráficos e áudio [1]. Um dos principais motivos do ganho de popularidade de tal interface se deve pelo fato de que os usuários possuíam dificuldades para utilizar as antigas interfaces de comando de linha (CLI), que muitas vezes era algo complicado de se aprender [2]. Na Figura 1 pode ser visualizado um exemplo de uma interface de linha de comando e pode-se observar que, para

que o usuário usufrua do sistema, é necessário que ele compreenda os comandos utilizados para operar o computador, criando uma curva de aprendizado que muitas vezes pode ser uma barreira para novos usuários. A empresa Xerox criou a primeira interface gráfica, mas esta era utilizada apenas internamente na empresa. Na mesma época, a Apple introduziu no mercado um dos primeiros computadores pessoais com interface gráfica, o Apple Lisa [3]. A Figura 2 mostra como era a interface gráfica do Apple Lisa.

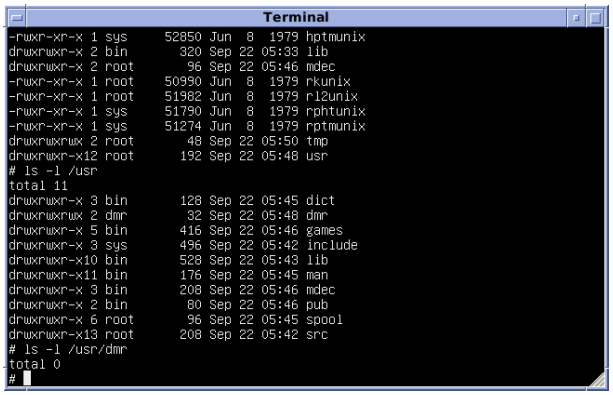

Figura 1: Interface de linha de comando do Unix.

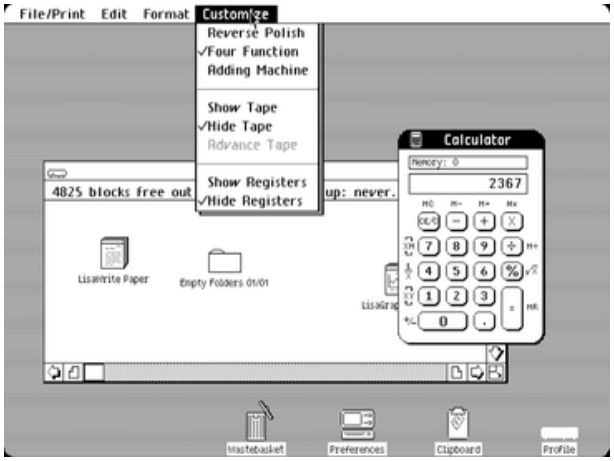

Figura 2: Screenshot do Lisa Office System 3.1, o primeiro sistema operacional com interface gráfica comercializado com sucesso de vendas.

Nos sistemas operacionais Linux, uma das interfaces gráficas utilizadas é a GNOME (GNU Network Object Model Environment) [4], que é o ambiente de desktop principal das distribuições Linux. A GNOME é desenvolvida pelo The GNOME Project, que visa criar frameworks para desenvolvimento de software e coordenação de ações para internacionalização, localização e acessibilidade dos softwares desenvolvidos [5]. O desenvolvimento de programas com interfaces que utilizem recursos disponíveis na GNOME é dado através do GTK [6], um widget toolkit gratuito, open-source e com alta portabilidade.

Já para sistemas Microsoft Windows, que possuem uma interface gráfica própria denominada de Windows, é possível criar programas que utilizam a interface do Windows através da Windows API. A Windows API permite a criação desde interfaces até serviços de rede, logo é possível perceber que esta é uma ferramenta poderosa para criação de programas voltados ao Windows. A Figura 3 ilustra a primeira interface gráfica que a

Microsoft desenvolveu para o Windows. Vale citar que as vendas deste sistema operacional foram um sucesso, levando a continuação da linha Windows da Microsoft.

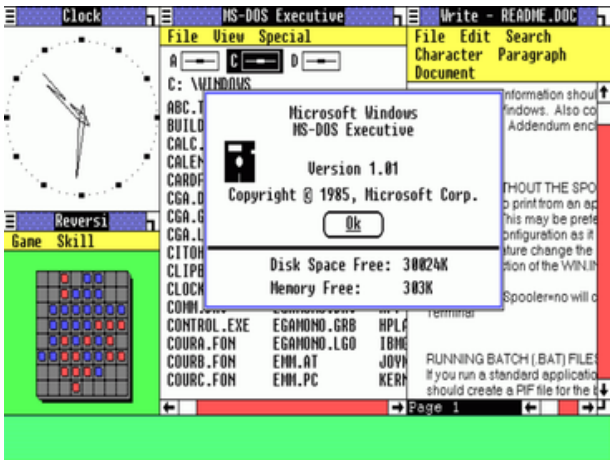

Figura 3: Screenshot do Microsoft Windows 1.01, o primeiro sistema operacional da linha Windows da Microsoft.

2 GTK

2.1 Informações Gerais

Para realizar este trabalho, a biblioteca do GTK foi escolhida para criar a interface gráfica. Esta seção possui como objetivo aprofundar sobre a biblioteca utilizada, com o intuito de facilitar o entendimento do leitor sobre como funciona o GTK e exemplos básicos de códigos. O GTK, como citado anteriormente, é uma biblioteca de criação de interface, logo ele é utilizado para criar diversas aplicações que utilizam interface gráfica. A versão utilizado do GTK neste projeto é a versão 2 (gtk2.0). Vale ressaltar que a cada nova versão, mudanças nas funções da biblioteca podem ocorrer, surgindo problemas de compatibilidades.

O GTK possui dependências de outras bibliotecas, que são as seguintes:

- Glib [7] - Conjunto de bibliotecas de sistemas de baixo nível escritas em C. Desenvolvida em grande parte pela GNOME;
- Pango [8] - É o que cuida do texto e fonte para o GTK;
- ATK [9] - Biblioteca que provê APIs para implementação de suporte de acessibilidade;
- GDK [10] - Toolkit do GTK que possui diversas funcionalidades como desenho de primitivas, imagens bitmap, cursor, fontes, eventos de janelas, dentre outras;
- GdkPixbuf - [11] - Biblioteca de carregamento de imagens;
- Cairo [12] - Renderiza a maioria dos elementos gráficos de controle.

2.2 Instrução de compilação

Para compilar um código que utiliza a biblioteca GTK, é necessário usar os seguintes parâmetros durante a compilação:

- gcc -o <nome_executavel> <arquivo>.c ‘pkg-config –libs –cflags gtk+-2.0‘

Como pode ser notado, para a compilação do código fonte, será utilizada uma ferramenta chamada pkg-config, que auxilia a compilação de aplicações e bibliotecas. Além disso, é preciso definir qual a versão do GTK utilizada no projeto.

2.3 Exemplos de códigos

Nesta subseção serão apresentados alguns exemplos de códigos utilizando o GTK, demonstrando como são criados os elementos de interface e como são manipulados.

2.3.1 Observação importante

A função gtk_init() deve ser chamada antes de qualquer função do GTK, pois ela permite a inicialização do GTK. Além disso, a função gtk_main(), que sempre será a última função do GTK chamada no código, permite que o programa fique em um loop esperando eventos ocorrerem.

2.3.2 Criação de janela

Figura 4: Janela criada utilizando o código mostrado abaixo.

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

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

  GtkWidget *window;
```

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

    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_widget_show(window);

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

    gtk_main();

    return 0;
}
```

Como pode ser visto, na Figura 4 temos uma janela simples, que primeiramente é definida com GtkWidget *window, que permite a criação de uma widget. GtkWidget é a classe base que todos os widgets do GTK derivam, que maneja o ciclo de vida, estados e estilos de uma widget.
A função gtk_window_new(GTK_WINDOW_TOPLEVEL) é utilizada para criar uma janela e o argumento passado na função define o tipo de janela a ser criada. Neste caso, será criada uma janela toplevel, que terá borda e uma barra de título (titlebar).
Para mostrar o widget criado, é necessária utilizar a função get_widget_show(window).
Por fim, a função g_signal_connect() conecta uma função de callback a um sinal para um objeto em particular. Com os parâmetros definidos neste programa, ao fechar a janela, será emitido um sinal para o termino da aplicação. Isso é necessário, pois a janela não possui tratamento de termino por default.
Observações finais:

- Caso queira modificar o tamanho da janela, basta colocar a seguinte função: gtk_window_set_default_size(GTK_WINDOW(widget_window), largura_janela, altura_janela);
- Para a centralização da janela, deve-se utilizar função e os seguintes parâmetros: gtk_window_set_position(GTK_WINDOW(widget_window), GTK_WIN_POS_CENTER);

2.3.3 Criação de botões e eventos

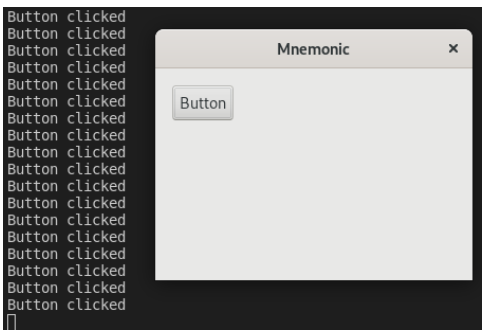

Figura 5: Janela com um botão que a cada click imprime no terminal a mensagem "Button clicked".

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

void print_msg(GtkWidget *widget, gpointer window) {

  g_printf("Button clicked\n");
}

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

  GtkWidget *window;
  GtkWidget *button;
  GtkWidget *halign;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(window), "Mnemonic");
  gtk_window_set_default_size(GTK_WINDOW(window), 300, 200);
  gtk_container_set_border_width(GTK_CONTAINER(window), 15);

  button = gtk_button_new_with_mnemonic("_Button");

  g_signal_connect(button, "clicked",
      G_CALLBACK(print_msg), NULL);

  halign = gtk_alignment_new(0, 0, 0, 0);
  gtk_container_add(GTK_CONTAINER(halign), button);
  gtk_container_add(GTK_CONTAINER(window), halign);

  gtk_widget_show_all(window);

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

  gtk_main();

  return 0;
}
```

- gtk_button_new_with_mnemonic("_Button"): cria-se um mnemônico a um botão (podem ser usados também com labels e itens de menus), logo podemos interagir com o botão de maneira mouseless. Para isso, basta aperta Alt+B (o segundo botão a ser apertado sempre será o carácter que vier após o _);
- g_signal_connect(button, "clicked", G_CALLBACK(print_msg), NULL): será conectado um sinal de clique a função print_msg que irá imprimir no terminal uma mensagem mostrando que o botão foi clicado;
- halign = gtk_alignment_new(0, 0, 0, 0): é um contêiner básico que possibilita o

alinhamento de seu filho em uma janela. O primeiro e segundo argumento são para as coordenadas de posição (X e Y) que o botão estará na janela, já o terceiro e quarto argumento serão para mudar o tamanho do widget.

- gtk_container_add(GTK_CONTAINER(halign), button): é utilizado para inserir o botão no contêiner criado de alinhamento. E, para adicionar o contêiner na janela, basta fazer o seguinte - gtk_container_add(GTK_CONTAINER(window), halign)

2.3.4 Manipulação de Toolbar

Toolbars permitem um acesso rápido para comandos que são normalmente muito utilizados. A Figura 6 ilustra uma toolbar simples.

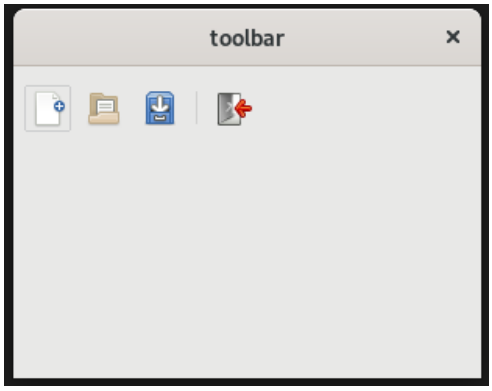

Figura 6: Toolbar criada com ícones interativos.

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

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

  GtkWidget *window;
  GtkWidget *vbox;

  GtkWidget *toolbar;
  GtkToolItem *newTb;
  GtkToolItem *openTb;
  GtkToolItem *saveTb;
  GtkToolItem *sep;
  GtkToolItem *exitTb;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 300, 200);
  gtk_window_set_title(GTK_WINDOW(window), "toolbar");

  vbox = gtk_vbox_new(FALSE, 0);
```

```
    gtk_container_add(GTK_CONTAINER(window), vbox);

    toolbar = gtk_toolbar_new();
    gtk_toolbar_set_style(GTK_TOOLBAR(toolbar), GTK_TOOLBAR_ICONS);

    newTb = gtk_tool_button_new_from_stock(GTK_STOCK_NEW);
    gtk_toolbar_insert(GTK_TOOLBAR(toolbar), newTb, -1);

    openTb = gtk_tool_button_new_from_stock(GTK_STOCK_OPEN);
    gtk_toolbar_insert(GTK_TOOLBAR(toolbar), openTb, -1);

    saveTb = gtk_tool_button_new_from_stock(GTK_STOCK_SAVE);
    gtk_toolbar_insert(GTK_TOOLBAR(toolbar), saveTb, -1);

    sep = gtk_separator_tool_item_new();
    gtk_toolbar_insert(GTK_TOOLBAR(toolbar), sep, -1);

    exitTb = gtk_tool_button_new_from_stock(GTK_STOCK_QUIT);
    gtk_toolbar_insert(GTK_TOOLBAR(toolbar), exitTb, -1);

    gtk_box_pack_start(GTK_BOX(vbox), toolbar, FALSE, FALSE, 5);

    g_signal_connect(G_OBJECT(exitTb), "clicked",
            G_CALLBACK(gtk_main_quit), NULL);

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

    gtk_widget_show_all(window);

    gtk_main();

    return 0;
}
```

- gtk_toolbar_new(): cria um widget denominado toolbar;
- gtk_toolbar_set_style(GTK_TOOLBAR(toolbar), GTK_TOOLBAR_ICONS): é usada para alterar a visualização da toolbar, de modo a ilustrar os ícones e/ou textos. Neste exemplo, só haverá o display de ícones, pois somente utilizou-se como parâmetro o GTK_TOOLBAR_ICONS;
- A função gtk_tool_button_new_from_stock(GTK_STOCK_NEW): cria um GtkToolButton que contém uma imagem e um texto do stock;
- gtk_toolbar_insert(GTK_TOOLBAR(toolbar), new, -1): insere um novo GtkToolItem em uma toolbar em uma posição especifica (terceiro parâmetro). Caso a posição for negativa, o item é adicionado no final da toolbar (caso exista um ou mais ícones inseridos anteriormente, o novo ícone ficará na frente deles);

- sep = gtk_separator_tool_item_new(): cria um novo GtkSeparatorToolItem e gtk_toolbar_insert(GTK_TOOLBAR(toolbar), sep, -1) coloca o separador na toolbar (o separador pode ser visto na Figura 6 entre o penúltimo e último ícone).

2.3.5 Janela com caixas de input

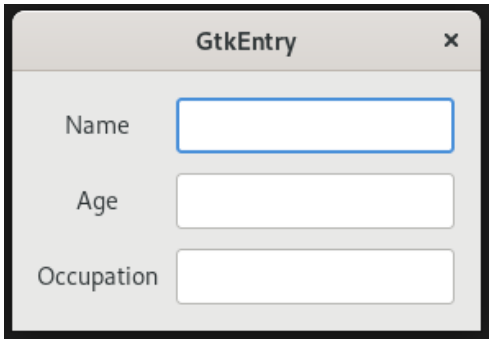

Figura 7: Janela com caixas de inputs.

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

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

  GtkWidget *window;
  GtkWidget *table;

  GtkWidget *label1;
  GtkWidget *label2;
  GtkWidget *label3;

  GtkWidget *entry1;
  GtkWidget *entry2;
  GtkWidget *entry3;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_title(GTK_WINDOW(window), "GtkEntry");
  gtk_container_set_border_width(GTK_CONTAINER(window), 10);

  table = gtk_table_new(3, 2, FALSE);
  gtk_container_add(GTK_CONTAINER(window), table);

  label1 = gtk_label_new("Name");
  label2 = gtk_label_new("Age");
  label3 = gtk_label_new("Occupation");

  gtk_table_attach(GTK_TABLE(table), label1, 0, 1, 0, 1,
```

```
        GTK_FILL | GTK_SHRINK, GTK_FILL | GTK_SHRINK, 5, 5);
    gtk_table_attach(GTK_TABLE(table), label2, 0, 1, 1, 2,
        GTK_FILL | GTK_SHRINK, GTK_FILL | GTK_SHRINK, 5, 5);
    gtk_table_attach(GTK_TABLE(table), label3, 0, 1, 2, 3,
        GTK_FILL | GTK_SHRINK, GTK_FILL | GTK_SHRINK, 5, 5);

    entry1 = gtk_entry_new();
    entry2 = gtk_entry_new();
    entry3 = gtk_entry_new();

    gtk_table_attach(GTK_TABLE(table), entry1, 1, 2, 0, 1,
        GTK_FILL | GTK_SHRINK, GTK_FILL | GTK_SHRINK, 5, 5);
    gtk_table_attach(GTK_TABLE(table), entry2, 1, 2, 1, 2,
        GTK_FILL | GTK_SHRINK, GTK_FILL | GTK_SHRINK, 5, 5);
    gtk_table_attach(GTK_TABLE(table), entry3, 1, 2, 2, 3,
        GTK_FILL | GTK_SHRINK, GTK_FILL | GTK_SHRINK, 5, 5);

    gtk_widget_show_all(window);

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

    gtk_main();

    return 0;
}
```

- gtk_table_new(3, 2, FALSE): cria uma GtkTable widget com 3 linhas e 2 colunas. O terceiro parâmetro é usado para determinar se as células da tabela serão redimensionadas (TRUE) ou não (FALSE), proporcionalmente de acordo com o maior widget na tabela;
- gtk_label_new(): coloca uma label e o parâmetro será uma string que aparecerá na janela.
- gtk_entry_new(): cria uma caixa de input para o usuário digitar;
- Para inserir tanto a label como a caixa de input, utiliza-se a seguinte função: gtk_table_attach(). Esta função adiciona um filho a uma table anteriormente criada.

3 Desenvolvimento

3.1 Código fonte - link

O código fonte do trabalho está disponível em um repositório do GitHub. O link de acesso é mostrado abaixo:

https://github.com/pedro-hr-resende/ShellGrafico

3.2 Janela

3.2.1 Criação da janela e plano de fundo

Para criar a janela principal do *shell* gráfico, primeiramente será criado um widget chamado janela. Em seguida, cria-se uma janela para ser fullscreen:

```
GtkWidget *janela;
janela = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_fullscreen(GTK_WINDOW(janela));
```

Com a janela criada, pode-se colocar o plano de fundo. Para isso, cria-se um layout que será importante para a inserção dos próximos elementos da interface. O layout será feito da seguinte maneira:

```
layout = gtk_layout_new(NULL, NULL);
gtk_container_add(GTK_CONTAINER(janela), layout);
gtk_widget_show(layout);
```

O layout resultará em uma janela totalmente branca. Agora pode-se colocar o plano de fundo:

```
wallpaper = gtk_image_new_from_file("image.jpg");
gtk_layout_put(GTK_LAYOUT(layout), wallpaper, 0, 0);
```

Figura 8: anela em tela cheia com uma imagem de fundo.

3.2.2 Adicionando outros widgets à janela

Nesta etapa, os outros widgets serão adicionado à janela e, para isso, os widgets serão inseridos no layout. Isso é feito com a seguinte função:

```
gtk_layout_put(GTK_LAYOUT(layout), new_widget, x, y);
```

Os parâmetros x e y serão as coordenadas que o programador definirá para o widget. Vale pontuar que, para cada tipo de monitor, tais valores devem ser modificados, pois ocorrerá o posicionamento de widgets em regiões onde não foram designadas.

3.3 Toolbar

A Figura 9 mostra o barra na área de baixo da janela.

Para criar a toolbar da Figura 9, é necessário definir um widget que será a toolbar. Em seguida, pode-se criar uma toolbar e definir o estilo da toolbar. Para este projeto, decidiu-se que a toolbar aceitará apenas ícones. Segue abaixo o código relacionado a criação da toolbar:

Figura 9: Barra com os ícones de programas, menu de iniciar e a visualização de hora e data.

```
GtkWidget *toolbar;
toolbar = gtk_toolbar_new();
gtk_toolbar_set_style(GTK_TOOLBAR(toolbar), GTK_TOOLBAR_ICONS);
```

3.3.1 Menu popup

A figura abaixo ilustra o menu "Iniciar"do *shell* gráfico criado pelo grupo. No menu existem duas opções de eventos que podem ocorrer: término do *shell* gráfico e a abertura das configurações do sistema.

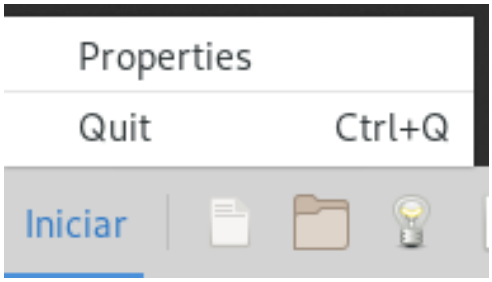

Figura 10: Menu Iniciar do *shell* gráfico.

Para criar o menu da Figura 10, será necessário dois widgets, que serão o menubar (GtkMenuBar) e o menu (GtkMenu). O GtkMenu é um contêiner que terá itens de menu. Neste caso serão o "Properties"e "Quit", que aparecem na Figura 10. Segue o código referente a criação do menu "Iniciar":

```
menubar = gtk_menu_bar_new();
fileMenu = gtk_menu_new();
```

Para criar a menubar e o menu utiliza-se as seguintes funções:

```
gtk_layout_put(GTK_LAYOUT(layout), menubar, x, y);
gtk_widget_set_size_request(menubar, 55, 45);
```

Para inserir a menubar no layout nas coordenadas x e y e depois definir o tamanho da menubar, serão utilizadas as seguintes funções:

```
accel_group = gtk_accel_group_new();
gtk_window_add_accel_group(GTK_WINDOW(janela), accel_group);
```

onde accel_group se refere a accelerator group - será um grupo de keyboards accelerator (atalhos de teclados melhorando a usabilidade e acessibilidade), que criará o atalho Ctrl+Q, que pode ser visto na Figura 10.

Para inserir os textos e ícones no menu Iniciar, utiliza-se as seguintes funções:

```
fileMi = gtk_menu_item_new_with_mnemonic("_Iniciar");

newMi = gtk_image_menu_item_new_from_stock(GTK_STOCK_PROPERTIES,
NULL);
sep = gtk_separator_menu_item_new();
quitMi = gtk_image_menu_item_new_from_stock(GTK_STOCK_QUIT, NULL);
```

onde gtk_menu_item_new_with_mnemonic("_Iniciar") permite criar o item menu Iniciar, que pode ser ativado com Alt+I, já que será um mnemônico (explicado na seção "2.3.3 Criação de botões e eventos"). Além disso, tem-se a criação dos itens "Properties"e "Quit"com a função gtk_image_menu_item_new_from_stock(). Note que há um separador (sep) que ira criar a linha entre Properties e Quit.

```
gdk_color_parse("#D3D3D3", &cor);
gtk_widget_modify_bg(menubar, GTK_STATE_NORMAL, &cor);
gtk_widget_add_accelerator(quitMi, "activate", accel_group,
    GDK_q, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);

gtk_menu_item_set_submenu(GTK_MENU_ITEM(fileMi), fileMenu);
gtk_menu_shell_append(GTK_MENU_SHELL(fileMenu), newMi);
gtk_menu_shell_append(GTK_MENU_SHELL(fileMenu), sep);
gtk_menu_shell_append(GTK_MENU_SHELL(fileMenu), quitMi);
gtk_menu_shell_append(GTK_MENU_SHELL(menubar), fileMi);
g_signal_connect(G_OBJECT(newMi), "activate",
  G_CALLBACK(abrirConfiguracao), NULL);

g_signal_connect(G_OBJECT(quitMi), "activate",
    G_CALLBACK(gtk_main_quit), NULL);
```

Concluindo o menu Iniciar, tem-se a gdk_color_parse, que terá como parâmetro uma cor definida em hexadecimal ou em nome da própria cor em inglês (blue/orange/...). Um detalhe a ser mencionado é que a variável cor que está sendo passada é um GdkColor, que é usado para descrever uma cor [13] e a mudança do background do menubar será mudada com a função gtk_widget_modify_bg(). Com a definição da cor do menubar, agora basta colocar o Properties e Quit com o gtk_menu_shell_append(), que recebe como parâmetros GTK_MENU_SHELL(menu) e os elementos que serão inseridos (Properties, Quit e o separador entre os dois). Para ocorrer um evento/funcionalidade ao Properties e Quit será usado g_signal_connect(), onde serão passados como parâmetros o elementos do menu, o tipo de ação e qual a função será o callback para o elemento em questão.

3.3.2 Ícones e suas funcionalidades

A Figura 11 ilustra os ícones inseridos na toolbar do *shell* gráfico. O primeiro ícone abre o editor de texto para um novo arquivo; o segundo abre o navegador de arquivos; o terceiro abre o navegador firefox (navegador default de sistemas operacionais Linux); o quarto ícone abre a janela de configuração do sistema e, por fim, o quinto abre o terminal.

```
editorTexto = gtk_tool_button_new_from_stock(GTK_STOCK_FILE);
gtk_toolbar_insert(GTK_TOOLBAR(toolbar), editorTexto, -1);
```

Figura 11: Ícones da toolbar.

```
    exploradorArquivos = gtk_tool_button_new_from_stock
    (GTK_STOCK_DIRECTORY);
    gtk_toolbar_insert(GTK_TOOLBAR(toolbar), exploradorArquivos, -1);

    web = gtk_tool_button_new_from_stock(GTK_STOCK_INFO);
    gtk_toolbar_insert(GTK_TOOLBAR(toolbar), web, -1);

    configuracao = gtk_tool_button_new_from_stock(GTK_STOCK_PROPERTIES);
    gtk_toolbar_insert(GTK_TOOLBAR(toolbar), configuracao, -1);

    shell_exec = gtk_tool_button_new_from_stock(GTK_STOCK_INDENT);
    gtk_toolbar_insert(GTK_TOOLBAR(toolbar), shell_exec, -1);
```

A inserção dos ícones será bem simples, primeiro define-se um GtkToolItem, depois usar a função gtk_tool_button_new_from_stock() que criará um novo GtkToolButton contendo a imagem e texto de um item stock, que é definido como argumento da função. Agora basta inserir na toolbar usando a função gtk_toolbar_insert() que tem como parâmetros a toolbar que receberá os ícones, o GtkToolButton criado e uma posição que será posto, caso seja negativo o ícone é adicionado no final da toolbar.

3.3.3 fork-exec

Como pode ser visto anteriormente, tantos os ícones da toolbar quanto os itens do menu possuem funcionalidades e, cada vez que se clica em qualquer um deles, uma função será executada. Isso se deve por conta do Callback estabelecido e cada função irá utilizar as system calls fork() e exec(). O fork-exec funciona da seguinte maneira: o processo criará um novo processo filho através da system call fork() [14], feito isso, o processo filho executará uma das funções de exec() [15](há 6 tipos de funções de exec). A função de exec usada neste projeto é a execlp(), que aceita como parâmetros strings com o nome dos programas ou comandos a serem executados e argumentos para o que será executado. As funções exec irão mudar o contexto do processo que irá executá-las.

```
void abrirEditorTexto(GtkWidget *widget, gpointer nomeWidget){
    pid_t pid;

    pid = fork();

    if(pid == 0)
      execlp("gedit", NULL);
}

void abrirExploradorArquivos(GtkWidget *widget, gpointer nomeWidget){
```

```
  pid_t pid;

  pid = fork();

  if(pid == 0)
    execlp("nautilus", NULL);
}

void abrirNavegador(GtkWidget *widget, gpointer nomeWidget){

  pid_t pid;

  pid = fork();

  if(pid == 0)
    execlp("firefox", NULL);
}


void exec_shell(GtkWidget *widget, gpointer nomeWidget){
  pid_t pid;

  pid = fork();

  if(pid == 0)
    execlp("gnome-terminal", NULL);
}


void abrirConfiguracao(GtkWidget *widget, gpointer nomeWidget){
  pid_t pid;

  pid = fork();

  if(pid == 0)
    execlp("gnome-control-center", NULL);
}
```

3.3.4 Display de data e hora

A criação do relógio mostrando a hora e a data será da seguinte maneira:

```
relogio = gtk_drawing_area_new();

gtk_layout_put(GTK_LAYOUT(layout),relogio, 1155, 723);
    gtk_widget_set_size_request(relogio, 200, 30);

gdk_color_parse("#D3D3D3", &cor);
    gtk_widget_modify_bg(relogio, GTK_STATE_NORMAL, &cor);
```

```
g_signal_connect(relogio, "expose-event", G_CALLBACK(on_expose_event)

g_timeout_add(1000, (GSourceFunc) time_handler, (gpointer) janela);
      time_handler(relogio);
```

onde gtk_drawing_area_new é um widget que permite desenhar sobre ela. Depois de definir o widget, ele será colocado na posição vista na Figura 9 e as suas dimensões serão definidas. O próximo passo é mudar a coloração do background do widget. Por fim será feito o evento de desenhar e redesenhar o relógio. Para desenhar será utilizada a função de Callback on_expose_event() que conecta para o sinal de expose-event, no qual é emitido quando o widget será redesenhado. Para atualizar o tempo, será usado o g_timeout_add() onde define-se um temporizador que, quando esgotado, irá executar uma função em um determinado widget. Neste caso, colocamos um tempo de 1 segundo (1000 milissegundos), para a função time_handler e a widget janela que contém a widget relógio (há um grau de parentesco, widget janela possui a widget layout, que por sua vez possui a widget relógio). A função time_handler se encontra abaixo:

```
gboolean time_handler(GtkWidget *widget){

if(widget->window == NULL){
  return FALSE;
}

GDateTime *now = g_date_time_new_now_local();
gchar *my_time = g_date_time_format(now, "%d/%m/%Y  |  %H:%M:%S");

g_sprintf(buf, "%s", my_time);

g_free(my_time);
g_date_time_unref(now);

gtk_widget_queue_draw(widget);

return TRUE;

}
```

Pega-se o horário e a data atuais do sistema com a função g_date_time_new_now_local(). Depois é formatado o output da função anterior para pegar a data e horário. Entre a data e o horário terá uma | apenas para melhorar a visualização das informações. A função g_sprintf armazena a variável my_time na variável global buf e para ativar o sinal de expose-event será a função gtk_widget_queue_draw(). As funções g_free() e g_date_time_unref() são usadas apenas para diminuir o consumo de memória, por conta que após ser armazenado o conteúdo da variável my_time, não teremos mais uso para as variáveis my_time e now. O if que se encontra no início da função é usado para que quando o widget janela for destruído, pode ocorrer que a função temporizada seja chamada e evite que a função trabalhe em uma janela destruída. Segue a função on_expose_event() que desenhará a data e o horário:

```
gchar buf[256];
```

```
gboolean on_expose_event(GtkWidget *widget,
GdkEventExpose *event, gpointer nomeWidget){

cairo_t *cr;

cr = gdk_cairo_create(widget->window);

cairo_move_to(cr, 30, 30);
cairo_set_font_size(cr, 14);
cairo_show_text(cr, buf);

cairo_destroy(cr);

return FALSE;

}
```

Como comentado anteriormente, Cairo é utilizado para renderizar a maioria dos elementos gráficos de controle, logo ele é necessário para o desenhar a data e o horário.

3.4 Área de trabalho

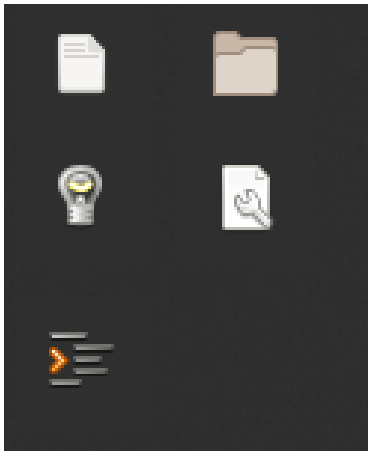

Figura 12: Ícones presentes na área de trabalho.

Para a criação dos ícones foi utilizado uma toolbar para cada um dos ícones e uma cor de background que se assemelha a cor do papel de parede usado. Ao serem pressionados, os ícones executam as mesmas funções estabelecidas para os ícones da toolbar, localizada na região inferior da tela. O código referente a criação dos ícones:

```
// Icones na area de trabalho

// Criar a toolbar icone 1
GtkWidget *desktop1;
GtkToolItem *editorTexto2;
```

```
desktop1 = gtk_toolbar_new();
gtk_toolbar_set_style(GTK_TOOLBAR(desktop1), GTK_TOOLBAR_ICONS);

editorTexto2 = gtk_tool_button_new_from_stock(GTK_STOCK_FILE);
gtk_toolbar_insert(GTK_TOOLBAR(desktop1), editorTexto2, -1);

// Modificacao da cor da toolbar

gdk_color_parse("#2F2F2F", &cor);
gtk_widget_modify_bg(desktop1, GTK_STATE_NORMAL, &cor);

// Acoes para cada botao criado acima

gtk_layout_put(GTK_LAYOUT(layout), desktop1, 10, 75);
gtk_widget_set_size_request(desktop1, 45, 45);

g_signal_connect(G_OBJECT(editorTexto2), "clicked",
G_CALLBACK(abrirEditorTexto), NULL);

// Criar a toolbar icone 2
GtkWidget *desktop2;
GtkToolItem *exploradorArquivos2;

desktop2 = gtk_toolbar_new();
gtk_toolbar_set_style(GTK_TOOLBAR(desktop2), GTK_TOOLBAR_ICONS);

exploradorArquivos2 = gtk_tool_button_new_from_stock
(GTK_STOCK_DIRECTORY);
gtk_toolbar_insert(GTK_TOOLBAR(desktop2), exploradorArquivos2, -1);

// Modificacao da cor da toolbar

gdk_color_parse("#2F2F2F", &cor);
gtk_widget_modify_bg(desktop2, GTK_STATE_NORMAL, &cor);


gtk_layout_put(GTK_LAYOUT(layout), desktop2, 65, 75);
gtk_widget_set_size_request(desktop2, 45, 45);

g_signal_connect(G_OBJECT(exploradorArquivos2),
"clicked", G_CALLBACK(abrirExploradorArquivos), NULL);


// Criar a toolbar icone 3
GtkWidget *desktop3;
GtkToolItem *web2;

desktop3 = gtk_toolbar_new();
```

```
gtk_toolbar_set_style(GTK_TOOLBAR(desktop3), GTK_TOOLBAR_ICONS);

web2 = gtk_tool_button_new_from_stock(GTK_STOCK_INFO);
gtk_toolbar_insert(GTK_TOOLBAR(desktop3), web2, -1);

// Modificacao da cor da toolbar

gdk_color_parse("#2F2F2F", &cor);
gtk_widget_modify_bg(desktop3, GTK_STATE_NORMAL, &cor);

gtk_layout_put(GTK_LAYOUT(layout), desktop3, 10, 120);
gtk_widget_set_size_request(desktop3, 45, 45);

g_signal_connect(G_OBJECT(web2), "clicked",
G_CALLBACK(abrirNavegador), NULL);

// Criar a toolbar icone 4
GtkWidget *desktop4;
GtkToolItem *configuracao2;

desktop4 = gtk_toolbar_new();
gtk_toolbar_set_style(GTK_TOOLBAR(desktop4), GTK_TOOLBAR_ICONS);

configuracao2 = gtk_tool_button_new_from_stock(GTK_STOCK_PROPERTIES);
gtk_toolbar_insert(GTK_TOOLBAR(desktop4), configuracao2, -1);

// Modificacao da cor da toolbar

gdk_color_parse("#2F2F2F", &cor);
gtk_widget_modify_bg(desktop4, GTK_STATE_NORMAL, &cor);

gtk_layout_put(GTK_LAYOUT(layout), desktop4, 65, 120);
gtk_widget_set_size_request(desktop4, 45, 45);

g_signal_connect(G_OBJECT(configuracao2), "clicked",
G_CALLBACK(abrirConfiguracao), NULL);

// Criar a toolbar icone 5
GtkWidget *desktop5;
GtkToolItem *shell_exec2;

desktop5 = gtk_toolbar_new();
gtk_toolbar_set_style(GTK_TOOLBAR(desktop5), GTK_TOOLBAR_ICONS);

shell_exec2 = gtk_tool_button_new_from_stock(GTK_STOCK_INDENT);
gtk_toolbar_insert(GTK_TOOLBAR(desktop5), shell_exec2, -1);

// Modificacao da cor da toolbar
```

```
gdk_color_parse("#2F2F2F", &cor);
gtk_widget_modify_bg(desktop5, GTK_STATE_NORMAL, &cor);

// Acoes para cada botao criado acima

gtk_layout_put(GTK_LAYOUT(layout), desktop5, 10, 175);
gtk_widget_set_size_request(desktop5, 45, 45);

g_signal_connect(G_OBJECT(shell_exec2), "clicked",
G_CALLBACK(exec_shell), NULL);
```

3.5 Eventbox na região superior da janela

Há uma eventbox oculta (da mesma cor que o plano de fundo) na região superior da janela. Esta eventbox será usada para apenas uma única funcionalidade: encerrar o processo da shell gráfico, através de um menu de popup. Entende-se que o ato de encerrar o processo do shell está associado a desligar o computado, porém, para este trabalho, desligar a máquina não era o objetivo. A Figura 13 ilustra um exemplo do menu popup.

Figura 13: Menu popup que pode ser aberto ao clicar com o botão direito do mouse na área superior da janela.

```
barraSuperior1 = gtk_event_box_new();

gtk_layout_put(GTK_LAYOUT(layout), barraSuperior1, 0, 0);
    gtk_widget_set_size_request(barraSuperior1, 1366, 45);

    menuPopup = gtk_menu_new();

    encerrar = gtk_menu_item_new_with_label("Desligar");
gtk_widget_show(encerrar);
    gtk_menu_shell_append(GTK_MENU_SHELL(menuPopup), encerrar);

g_signal_connect(G_OBJECT(encerrar), "activate", G_CALLBACK(gtk_main_


g_signal_connect_swapped(G_OBJECT(barraSuperior1), "button-press-even

gdk_color_parse("#2F2F2F", &cor);
    gtk_widget_modify_bg(barraSuperior1, GTK_STATE_NORMAL, &cor);
```

A função gtk_event_box_new() cria uma nova eventbox que será adicionada ao layout e é redimensionada para ocupar a largura da tela. Depois será criado um menu (igual ao menu do Iniciar) e será adicionado a este menu um menu item, cuja label será Desligar. Este item possui a mesma função que o Quit do menu Iniciar.

Referências

[1] Wikipedia. Graphical user interface. https://en.wikipedia.org/wiki/Graphical_user_interface.

[2] Microsoft. Gui versus cli. https://blogs.technet.microsoft.com/mscom/2007/03/12/the-gui-versus-the-command-line-which-is-better-part-1/.

[3] Wikipedia. Apple lisa. https://en.wikipedia.org/wiki/Apple_Lisa.

[4] Gnome. https://www.gnome.org/.

[5] Wikipedia. Wikipedia gnome. https://en.wikipedia.org/wiki/GNOME.

[6] gtk. https://www.gtk.org/.

[7] glib. https://docs.gtk.org/glib/.

[8] pango. https://pango.gnome.org/.

[9] The accessibility toolkit. https://docs.gtk.org/atk/.

[10] Gimp drawing kit. https://docs.gtk.org/gdk3/.

[11] gdkpixbuf. https://docs.gtk.org/gdk-pixbuf/.

[12] cairo. https://cairographics.org/.

[13] gdkcolor. https://docs.gtk.org/gdk3/struct.Color.html.

[14] system call fork. https://man7.org/linux/man-pages/man2/fork.2.html.

[15] exec. https://man7.org/linux/man-pages/man3/exec.3.html.