Faculdade de Engenharia da Universidade do Porto

Waffle Lib / ChocoTab

Projeto Final LCOM - 2018/19 - MIEIC

Turma 3 Grupo 1

Professor das aulas Laboratoriais: Pedro Miguel Moreira da Silva

Autores

José Silva, up201705591 (up201705591@fe.up.pt)

Mário Mesquita, up201705723 (up201705723@fe.up.pt)

Porto, 7 de janeiro de 2019

Índice

Instruções de utilização **4**

Estado do projeto **11**

TIMER 12

KBD 12

Mouse 12

Video Card 12

RTC 13

UART 13

Organização / Estrutura do código **14**

Periféricos 14

Timer 14

Módulo ‘timer’ 14

Módulo ‘i8254’ 15

Módulo ‘timer_user’ 15

KBC 16

Módulo ‘kbc’ 16

Módulo ‘i8042’ 17

Módulo ‘keyboard’ 17

Módulo ‘mouse’ 18

Video Card 19

Módulo ‘vbe’ 19

RTC 20

Módulo ‘rtc’ 20

Serial Port 21

Módulo ‘serial_port’ 21

Waffle Lib 22

Window 22

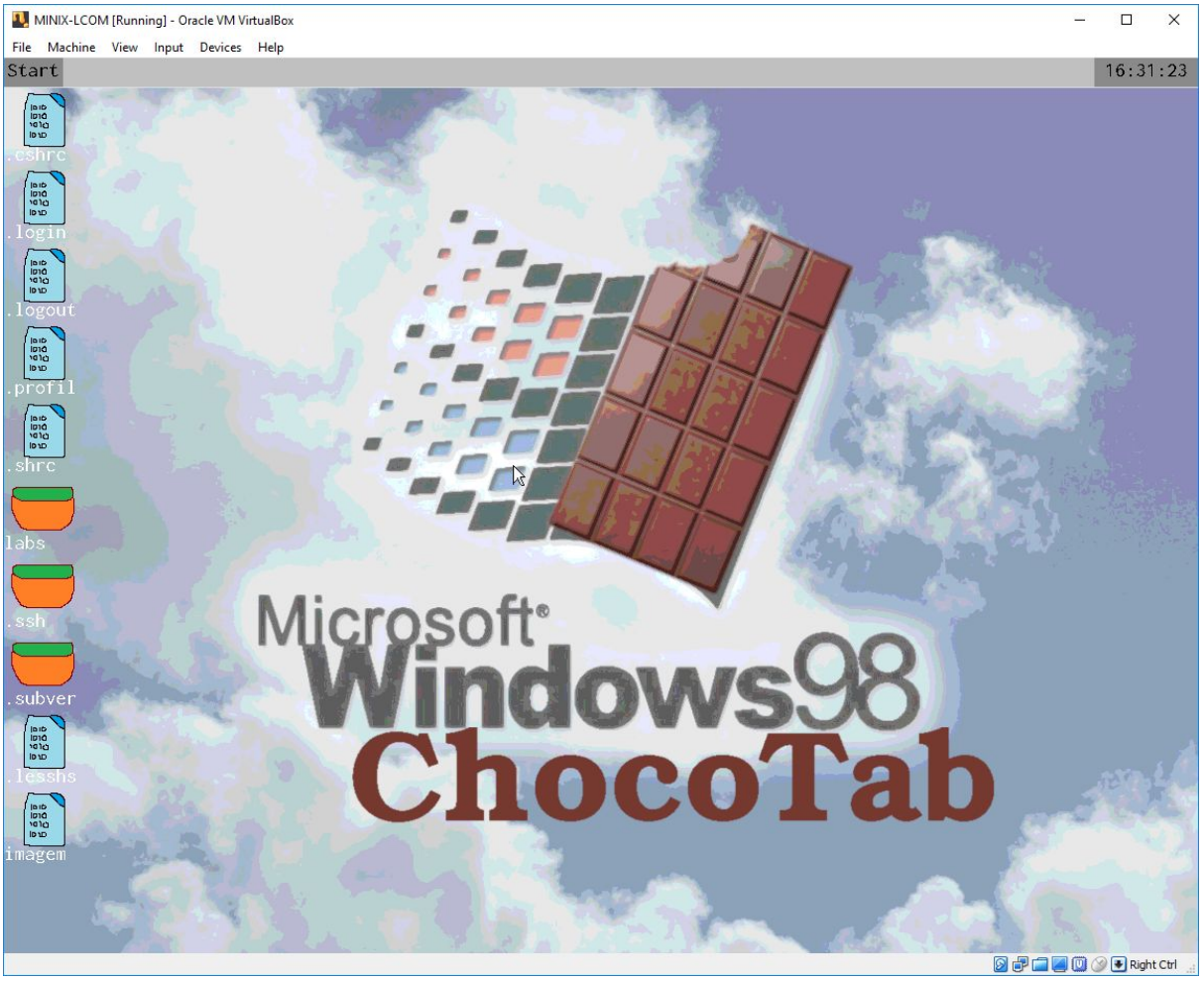
Módulo ‘window’ 22

Módulo ‘context_menu’ 25

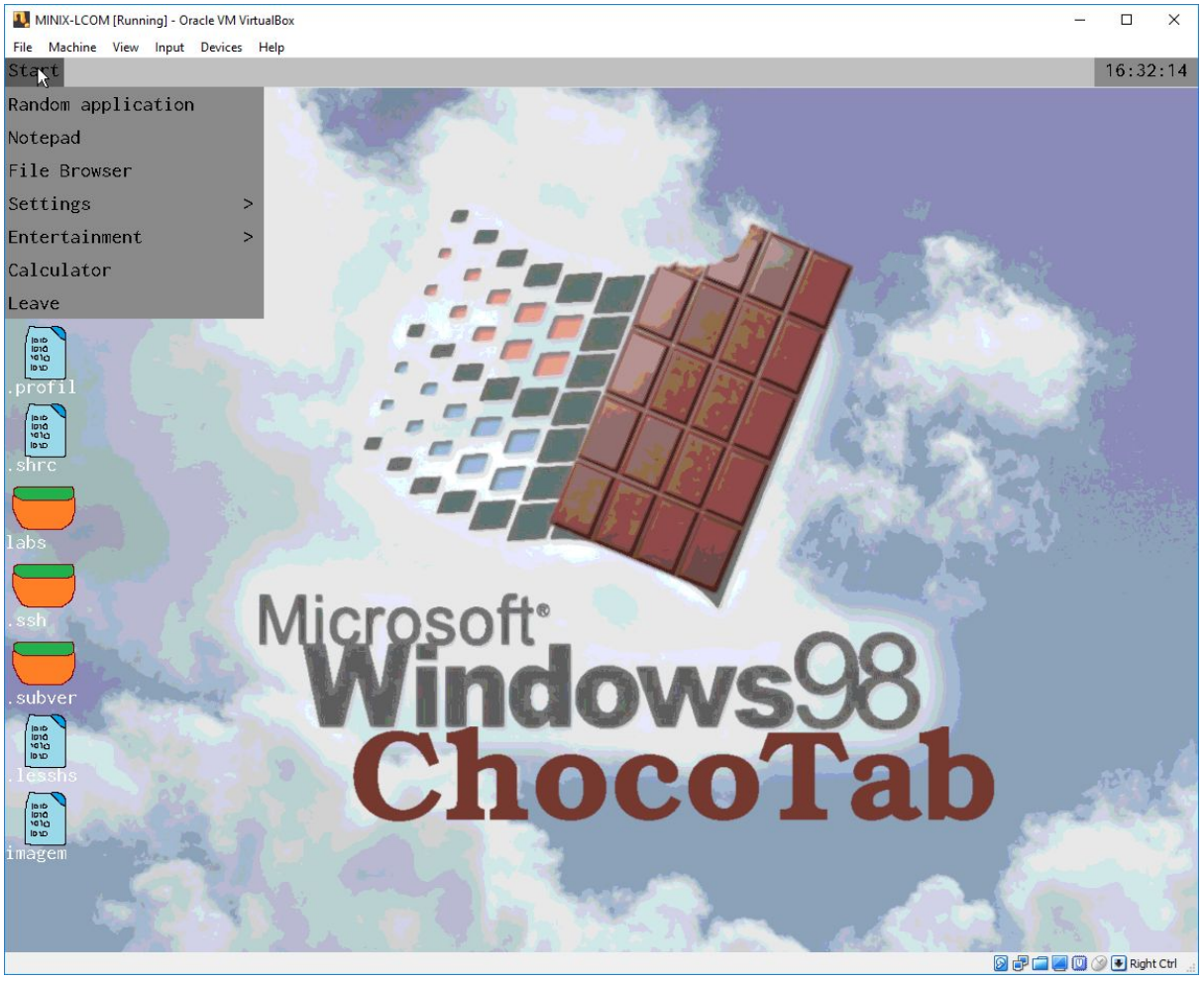
Módulo ‘elements’ 26
Módulo ‘state_machine’ 27
Módulo ‘taskbar’ 28
Aplicações 30
Módulo ‘login’ 30
Módulo ‘background_chooser’ 31
Módulo ‘system_info’ 32
Módulo ‘calculator’ 33
Módulo ‘file_browser’ 34
Módulo ‘painter’ 35
Módulo ‘multi_painter’ 36
Módulo ‘guess_painter’ 37
Módulo ‘notepad’ 38
Módulo ‘chatter’ 39
Módulo ‘example’ 40
Módulo ‘image_render’ 41
Screensaver 42
Módulo ‘screensaver’ 42
Font 43
Módulo ‘letters’ 43
Comunicação 44
Módulo ‘com_protocol’ 44
Outros 44
Módulo ‘util’ 44
Módulo ‘queue’ 44
Módulo ‘proj’ 45
Detalhes de implementação **46**
Conclusões **51**
Instruções de instalação **52**

Instruções de utilização

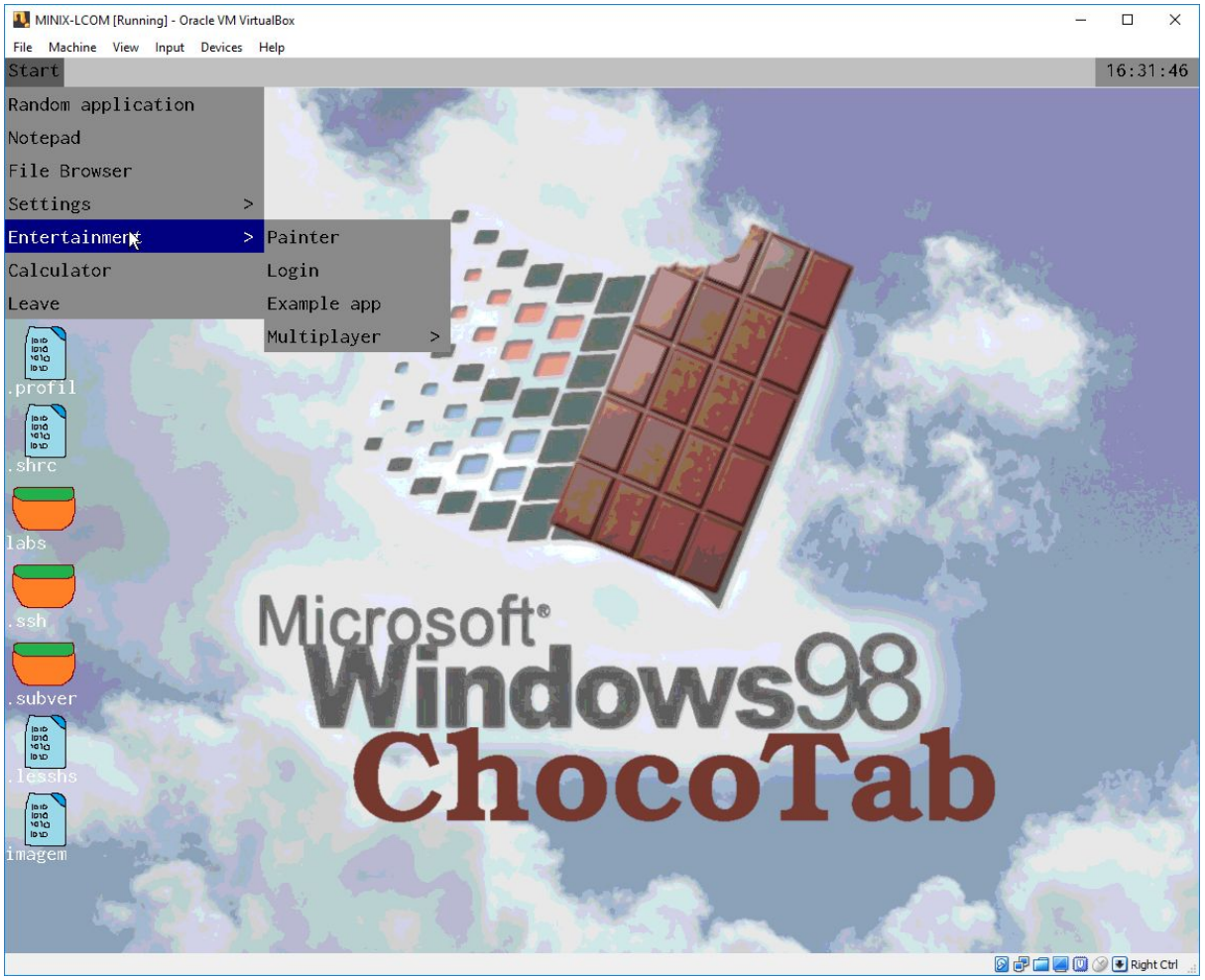
Inicialmente o programa apresenta um ambiente de trabalho vazio onde é possível visualizar a imagem de fundo selecionada, os items do ambiente de trabalho e a barra de tarefas (taskbar).



No canto superior esquerdo existe um menu que apresenta várias opções ao clicar, cada uma com a sua funcionalidade.

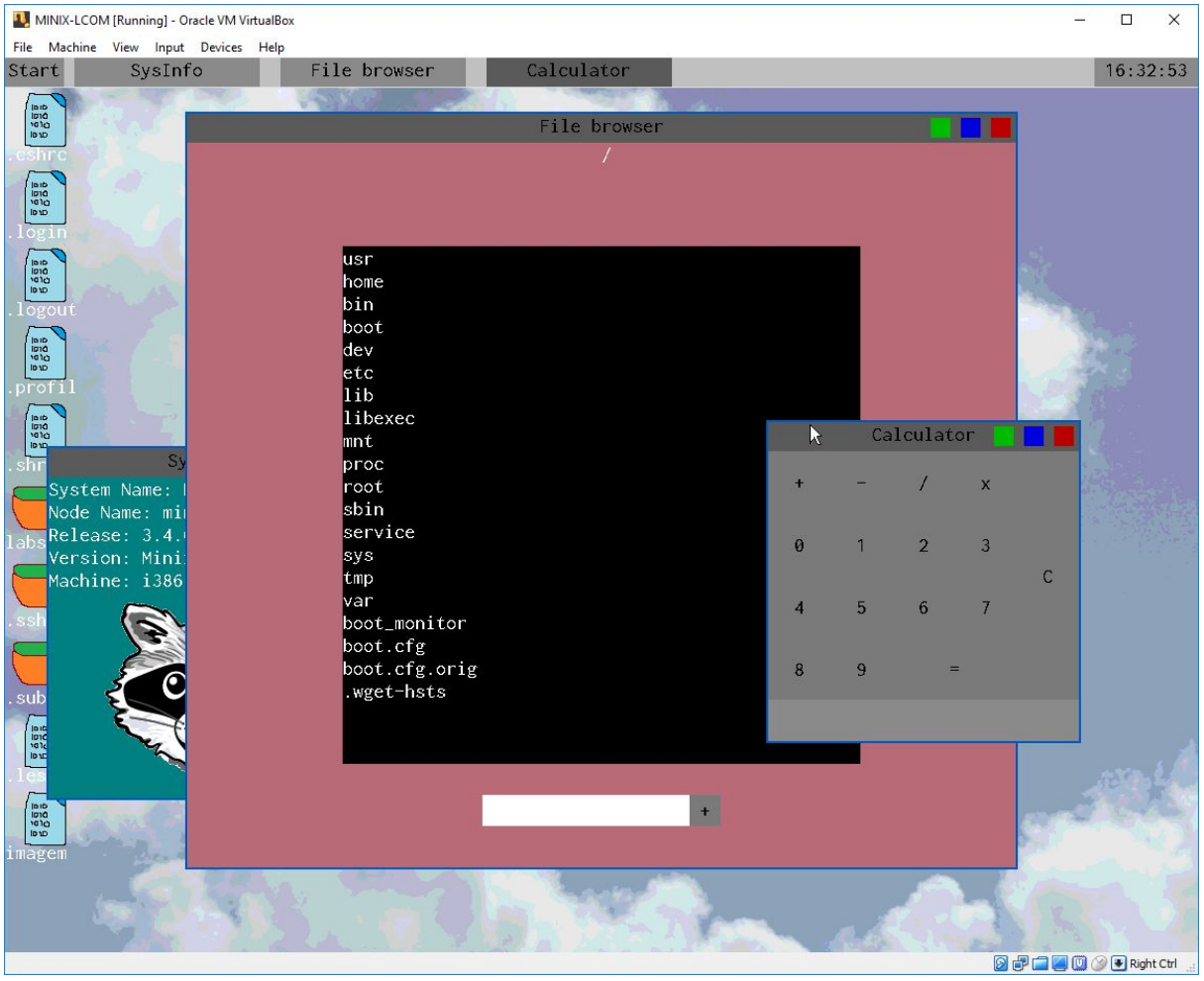


Opção com o rato por cima pode abrir um sub-menu:

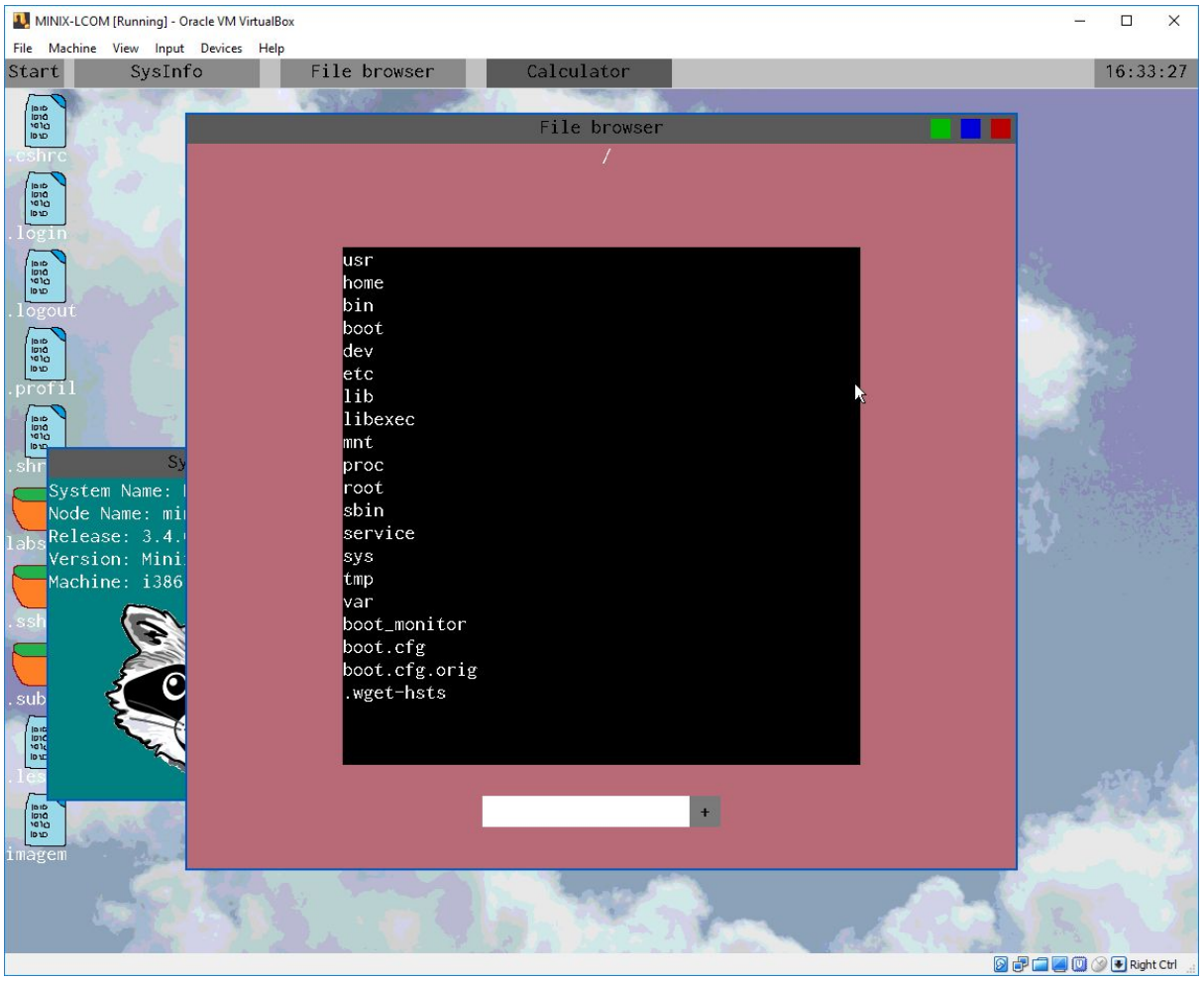


As janelas abertas pelo programa podem ser movidas pelo rato, clicando nelas e arrastando. Permitem ser minimizadas, maximizadas e fechadas. Cada janela possui a respetiva referência na taskbar localizada em cima, podendo ser minimizadas e trazidas para primeiro plano pressionando na respetiva referência. É possível também encadear janelas em cima das outras, respeitando a ordem por que foram sobrepostas. De modo a trazer uma janela para primeiro plano, basta clicar nela ou na sua referência e ela entrará em foco.

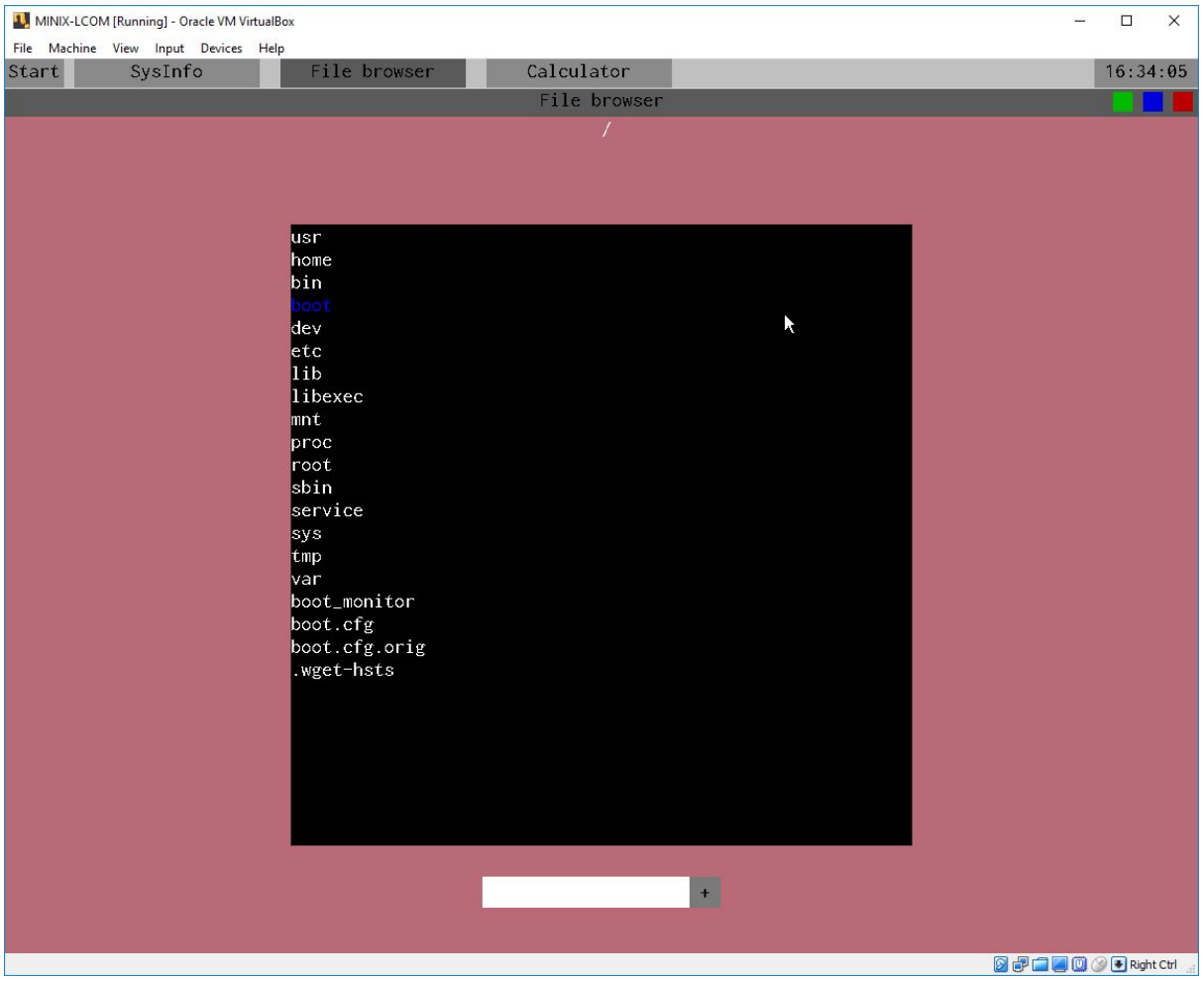
Foram abertas três janelas distintas:



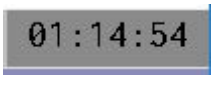
Minimizou-se a calculadora



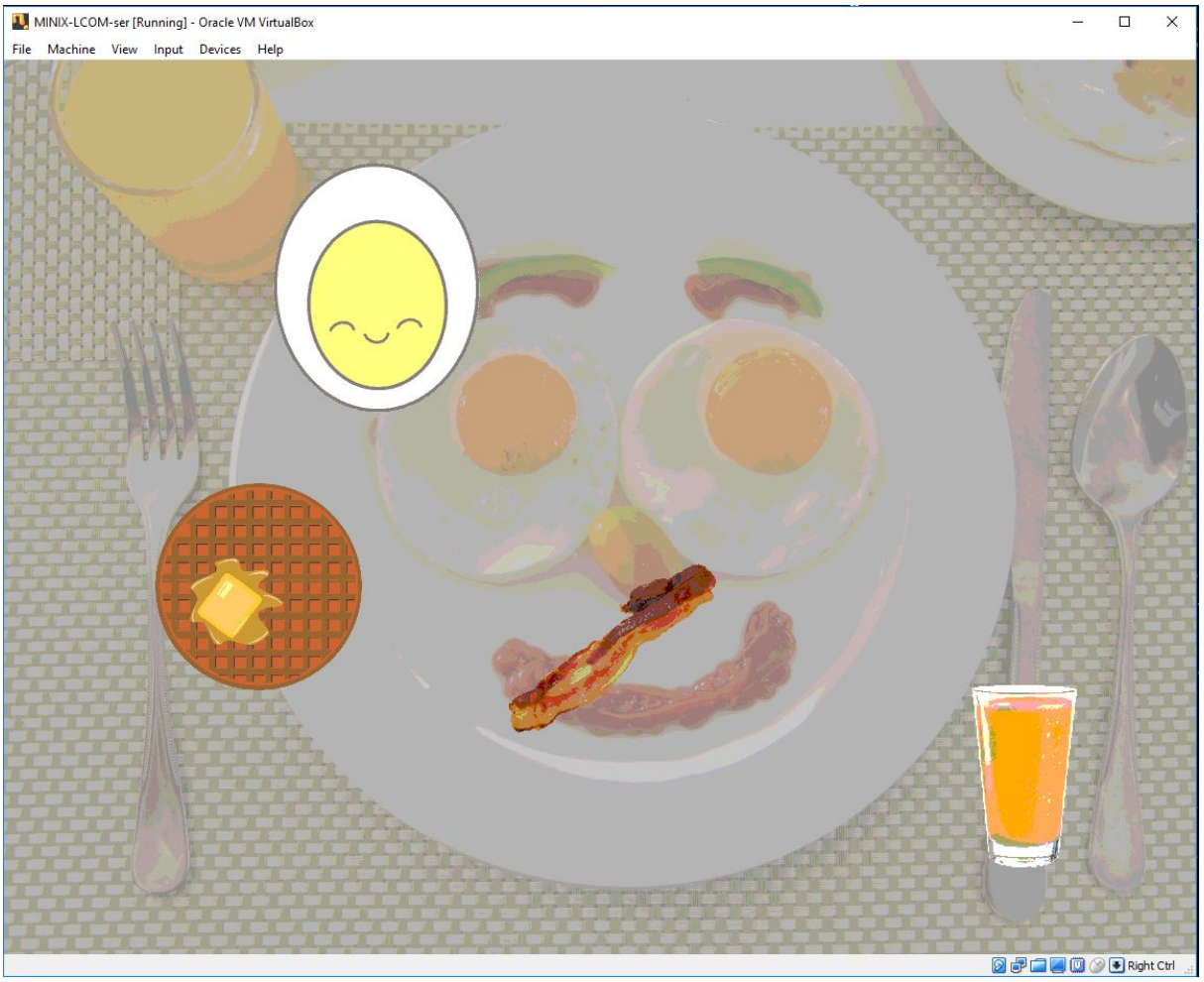
Maximizou-se o file browser



No canto superior direito está presente um relógio configurado para mostrar o tempo atual no formato dd:mm:yyyy.



Ao deixar o programa inativo durante 5 segundos, isto é, sem inputs do teclado e rato pelo utilizador, o ecrã é alterado para apresentar um *screensaver*. Este consiste em imagens de waffles rotativos, ovos, bacon e sumo de laranja em movimento, a colidir entre si e com as margens do ecrã. Com qualquer interação o ecrã volta ao normal.



Estado do projeto

A informação sobre a nossa utilização de periféricos encontra-se resumida na tabela seguinte. As colunas representam o nome do periférico, as funcionalidades para que foi utilizado e se funciona com base em interrupções ou não. Todos os periféricos com excepção do UART e Timer têm os seus handlers implementados em assembly.

Periférico	Utilização	Interrupções
Timer	- Controlo da *frame rate* - Controlo do tempo de inatividade	Sim
KBD	- Atalhos para funcionalidades - Escrita de texto - Verificação da (in)atividade - Interrupt handler em assembly	Sim
Mouse	- Movimento do apontador do utilizador - Interação com botões - Controlo das janelas - Verificação da (in)atividade - Interrupt handler em assembly	Sim
Video Card	- Apresentar no ecrã o ambiente de trabalho e todas as suas componentes - Apresentar o *screensaver* e objetos em movimento - Interrupt handler em assembly	Não
RTC	- Obtenção do tempo atual sempre que este é atualizado	Sim
UART	- Interação de um utilizador com um ambiente de trabalho remoto	Sim

TIMER

Utiliza-se o timer 0 para controlar a *frame rate* do programa, que definimos ser 30 frames por segundo. Opera internamente a uma frequência de 60 hertz.

É também usado para medir o tempo de inatividade do utilizador, de modo a decidir se deve ser desenhado o ambiente de trabalho ou o *screensaver*.

Foram implementadas funções para gerir a subscrição de interrupções, **timer_subscribe_int()** e **timer_unsubscribe_int()**. Além disso, destaca-se a função **timer_int_handler(),** que deve ser chamada a cada interrupção do timer. Esta última incrementa um contador que regista quantas interrupções passaram, baseado na sua frequência interna.

KBD

Com o teclado é possível fazer entrada de texto em caixas de texto ou áreas de texto e também ao pressionar certas teclas ou combinações é possível executar certos atalhos. Um exemplo é quando se pressiona na tecla do Windows é aberto o menu iniciar.

Atalhos conhecidos como *ALT-F4* e *ALT-TAB* também estão presentes e a tecla *DELETE* pode ser utilizada para eliminar entradas no ambiente de trabalho.

Em adição, o teclado, juntamente com o rato, controla a (in)atividade do utilizador, de modo a permitir decidir se deve ser desenhado o ambiente de trabalho ou o *screensaver*. Com qualquer interação, o tempo de inatividade é reduzido a 0.

São utilizadas funções para subscrição de interrupções (**keyboard_subscribe_int()** e **keyboard_unsubscribe_int())**, para gerir interrupções (**keyboard_ih()**) e para processar scancodes (**opcode_available()**).

Mouse

O rato é o meio principal pelo qual o utilizador interage com o programa, através de um cursor que pode ser movimentado no ecrã. Este permite clicar e interagir com os diversos botões e elementos do programa, seja para abrir menus, selecionar opções, arrastar / focar / abrir / fechar / minimizar / maximizar janelas.

Além disso, juntamente com o teclado, controla a (in)atividade do utilizador, de modo a permitir decidir se deve ser desenhado o ambiente de trabalho ou o *screensaver*. Com qualquer interação, o tempo de inatividade é reduzido a 0.

Relativamente ao mouse foram implementadas funções relativas à subscrição de interrupções (**mouse_subscribe_int()** e **mouse_unsubscribe_int()**), para gerir cada interrupção (**mouse_ih()**) e para processar os pacotes do rato (**assemble_mouse_packet()** e **parse_mouse_packet()**).

Video Card

O programa opera no modo **0x14C**, com uma resolução de **1152x864** e **32 bits** por pixel em **RGB (8)-8-8-8** (8 bits para cada componente azul, verde e vermelha, e 8 bits reservados).

Foi utilizada a técnica de double buffering para desenhar no ecrã, sendo que tudo é desenhado primeiro para um backbuffer e quando é dada um interrupção do timer o backbuffer é trocado com o frontbuffer, havendo assim uma troca de buffers (**swap_buffers**).

As imagens e objetos no programa são **xpm**s, fazendo uso da função load_xpm() fornecida pelos professores da cadeira para os converter em pixmaps.

Recorre-se a um xpm com todos os símbolos da tabela ASCII desde '!' até '~' para representar a **fonte**, tendo este sido preparado pelo grupo. Ao longo da execução é utilizado para apresentar informação em diversos locais.

No ambiente de trabalho são verificadas várias **colisões**, seja entre o rato e as janelas / botões ou entre as janelas e as margens do ecrã. No entanto, destaca-se o *screensaver* que entra em ação quando o utilizador permanece inativo durante 5 segundos. Neste surgem objetos de waffles, ovos, bacon e sumo de laranja em movimento e a colidir, tendo sido implementada **colisão pixel a pixel.** Os waffles apresentam também uma **animação**, encontrando-se em rotação.

RTC

O RTC é utilizado para obter o tempo atual e apresentá-lo na taskbar, no canto superior direito, sob a forma de um relógio. A informação encontra-se no formato hh:mm:ss.

São utilizadas as interrupções de *update*, de modo a atualizar o relógio sempre que o tempo muda (a cada segundo).

Foram implementadas funções para gerir a subscrição de interrupções (**rtc_subscribe_int()** e **rtc_unsubscribe_int()**), para ativar as interrupções necessárias (neste caso apenas as de *update*, **rtc_enable_update_int()**) e para gerir cada interrupção (**rtc_int_handler_asm()**). Esta última verifica se a fonte da interrupção foi um *update* dos registos e atualiza o relógio em caso afirmativo.

UART

O UART é utilizado para comunicação entre programas em máquinas distintas, como é caso do **multi_painter**. São utilizadas **FIFOs** para enviar e receber informação.

É configurado para trabalhar com uma **bit rate** de 9600, **paridade** par, 2 **stop bits** e 8 **bits por mensagem**. São subscritas as **interrupções** de **Received Data Available**, **Transmitter Empty** e **Line Status**. O **trigger level** utilizado na comunicação com FIFOs é de 8 bytes por interrupção.

Na comunicação relativa ao multi_painter são enviados 12 bytes, sendo que os primeiro 4 indicam o tipo de mensagem e os últimos 8 são a mensagem em si, a ser interpretada de forma diferente consoante o tipo de mensagem. Existem as mensagens **SERIAL_DRAW**, utilizado quando se quer enviar uma atividade de desenho, e **SERIAL_SLIDER**, quando os sliders das cores ou do pincel é deslocado.

Foram implementadas funções para gerir a subscrição de interrupções (**ser_subscribe_int()** e **ser_unsubscribe_int()**), para configurar o UART com determinados parâmetros (**ser_configure_settings()**), para ler / escrever de registos, para comunicação utilizando FIFOs (**ser_write_msg_fifo()**) e para gerir cada interrupção.

Organização / Estrutura do código

Ao longo dos laboratórios durante o semestre e agora durante a realização do projeto, foi-se organizando o código em módulos para torná-lo mais fácil de ler e interpretar. Uma das decisões que foi tomado foi organizá-lo em pastas, agrupando módulos relacionados.

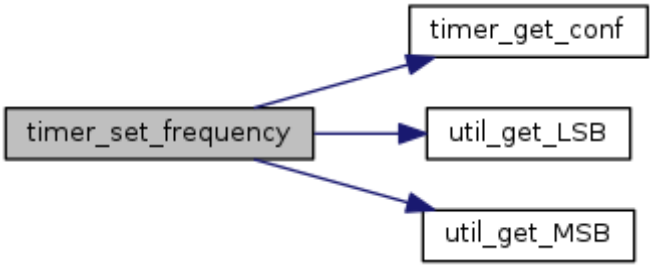
Periféricos

Os módulos descritos sob esta secção dizem respeito a periféricos e, à exceção do *rtc* e do *serial port*, foram desenvolvidos durante o decorrer dos Laboratórios de LCOM. Para o projeto apenas foram adicionadas algumas funcionalidades necessárias e / ou refatorizado o código.

- ### Timer

  #### *Módulo 'timer'*

  Este módulo contém código relativo à manipulação dos timers (que no nosso caso apenas usamos o timer 0), com funções para subscrição de interrupções, handlers e métodos para alterar o funcionamento interno. Foi desenvolvido durante o Laboratório 2 e ambos os membros contribuíram igualmente na sua criação.



Módulo 'i8254'

Este módulo contém as constantes e macros definidas para a programação do timer. Foi nos fornecido durante o Laboratório 2, e, portanto, não foi desenvolvido por nós.

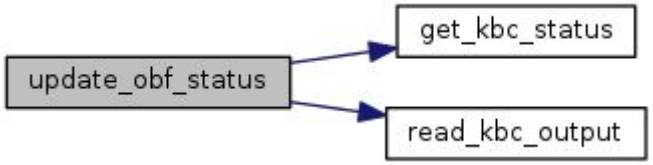
Módulo 'timer_user'

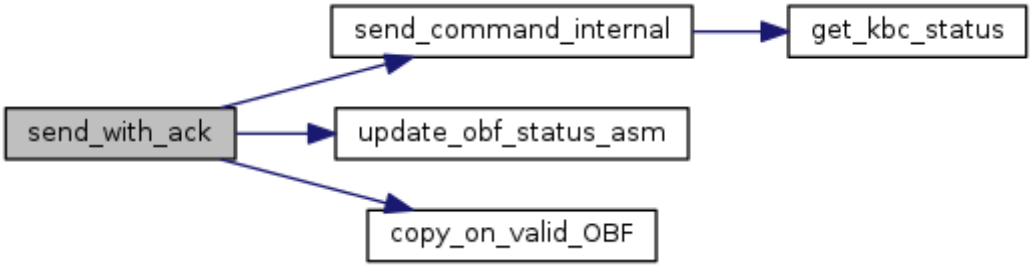
Este módulo contém algumas constantes e macros adicionais que consideramos ser úteis para a programação do timer, tendo sido desenvolvido durante o Laboratório 2. Ambos os membros contribuíram igualmente na sua criação.

- KBC

Módulo 'kbc'

Este módulo contém código relativo à manipulação do kbc, seja para gerir interrupções ou enviar / receber comandos. Foi desenvolvido durante o Laboratório 3 para o teclado e, mais tarde, refatorizado para permitir usá-lo também com o rato. Ambos os membros contribuíram igualmente na sua criação.



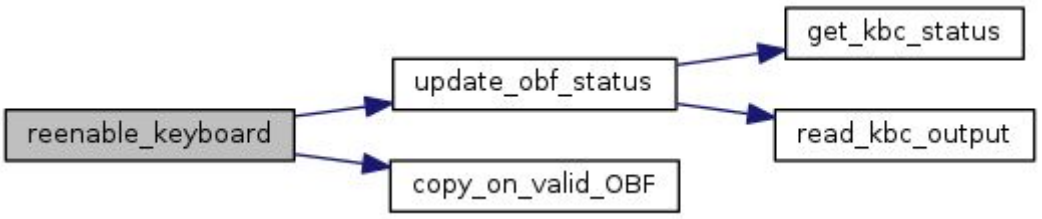


Módulo 'i8042'

Este módulo contém as constantes e macros definidas para a programação do teclado e rato. Foi desenvolvido durante os Laboratórios 3 e 4 e ambos os membros contribuíram igualmente na sua criação.

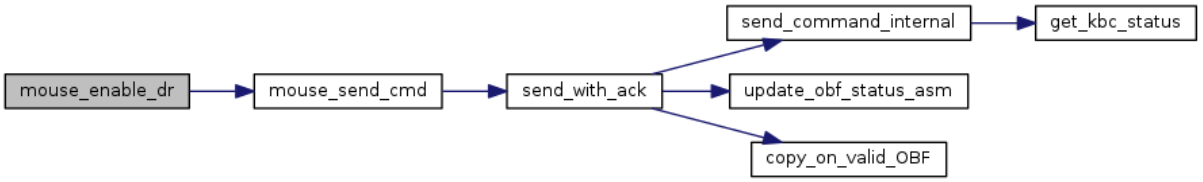
Módulo 'keyboard'

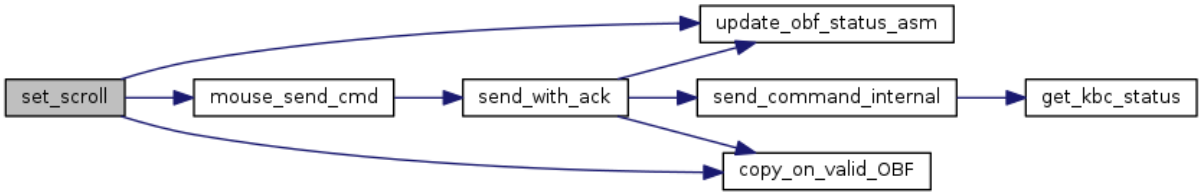
Este módulo contém código relativo à manipulação do teclado, seja para subscrever e gerir interrupções ou processar *scancodes*. Foi desenvolvido durante o Laboratório 3 e ambos os membros contribuíram igualmente na sua criação.

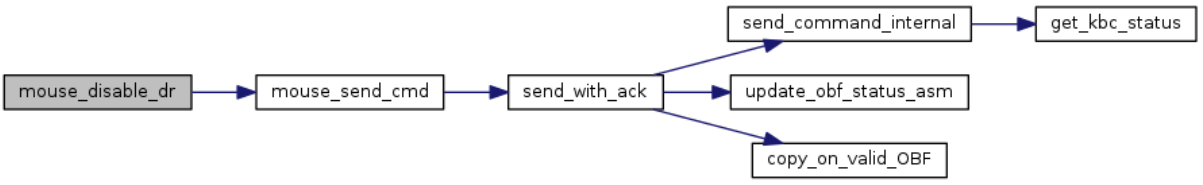


Módulo 'mouse'

Este módulo contém código relativo à manipulação do rato, seja para subscrever e gerir interrupções ou processar *packets*. Foi desenvolvido durante o Laboratório 4 e ambos os membros contribuíram igualmente na sua criação. Por fim a capacidade de utilização da scroll wheel foi adicionada por José Silva.



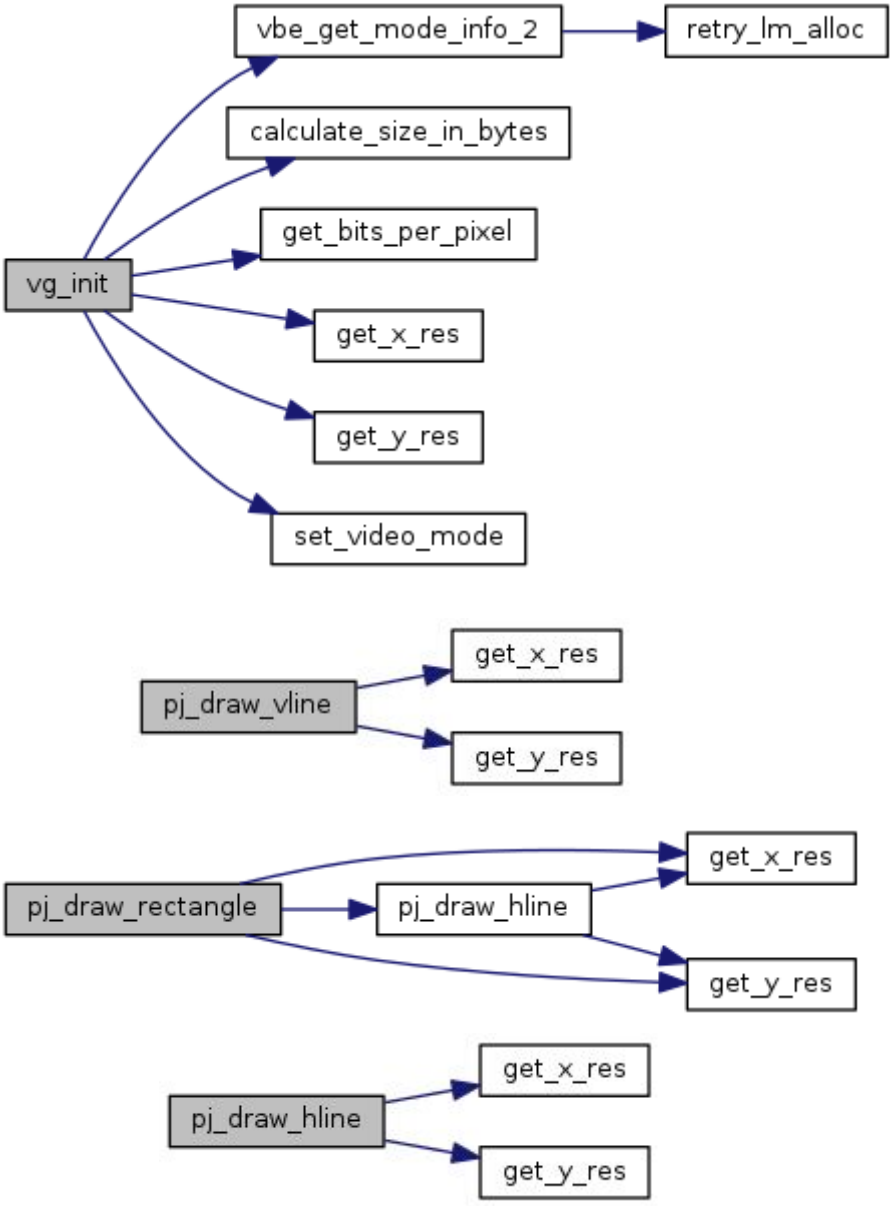




- ## Video Card

Módulo 'vbe'

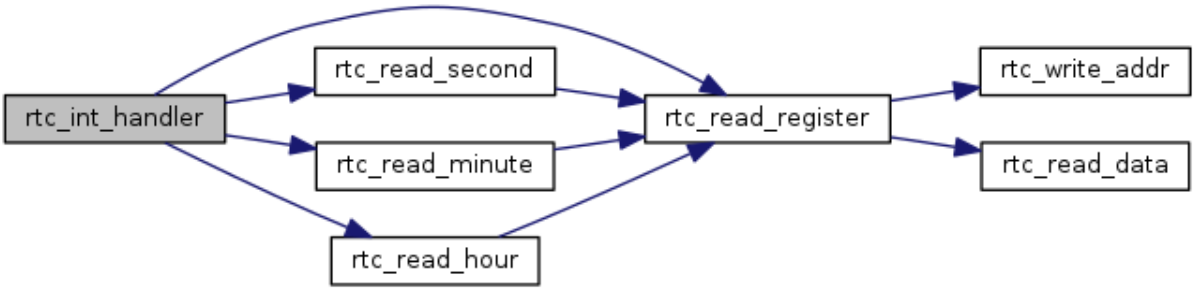
Este módulo contém código relativo à manipulação da VBE, com funções para inicializar o modo de vídeo, alterar o modo de vídeo, desenhar no ecrã e obter / alterar configurações. Foi desenvolvido durante o Laboratório 5 e ambos os membros contribuíram igualmente na sua criação.

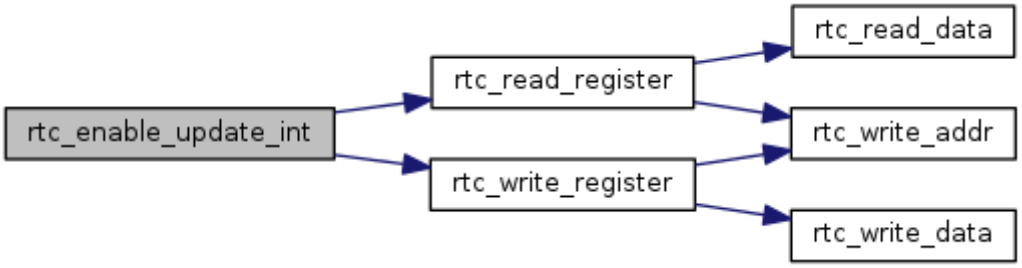


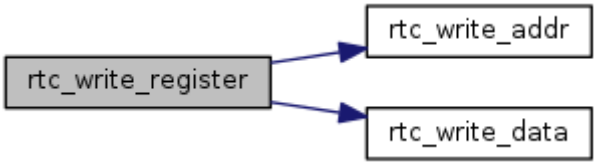
- RTC

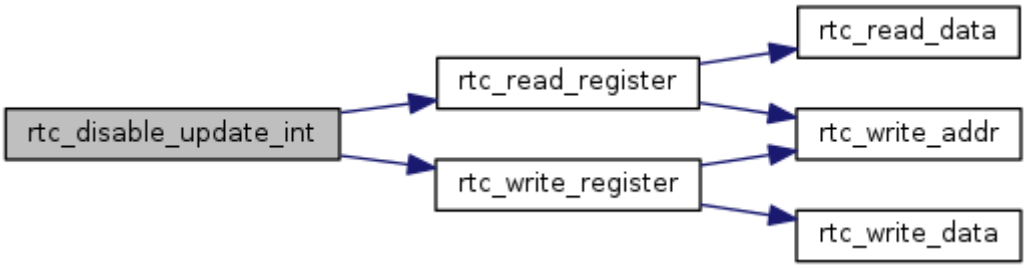
Módulo 'rtc'

Este módulo contém código relativo à manipulação do RTC, com funções para gerir e processar interrupções, alterar o funcionamento do periférico e ler / escrever nos seus registos. Encontram-se também neste módulo todas as constantes e macros definidas para a sua programação. Foi desenvolvido maioritariamente pelo Mário Mesquita com contribuições de José Silva.





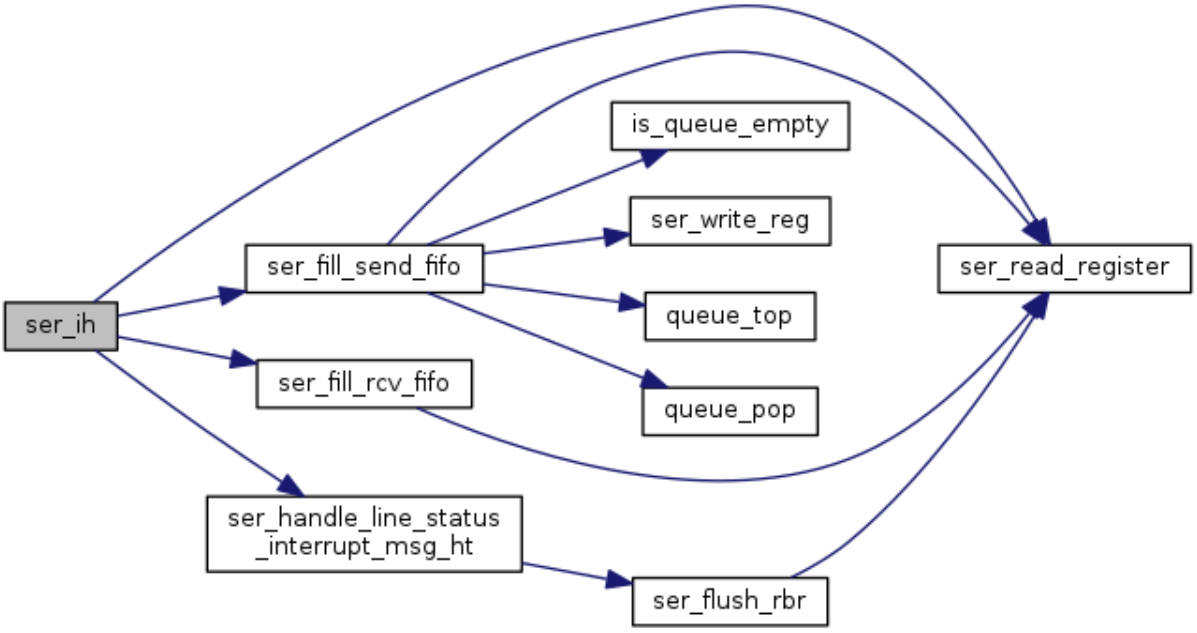


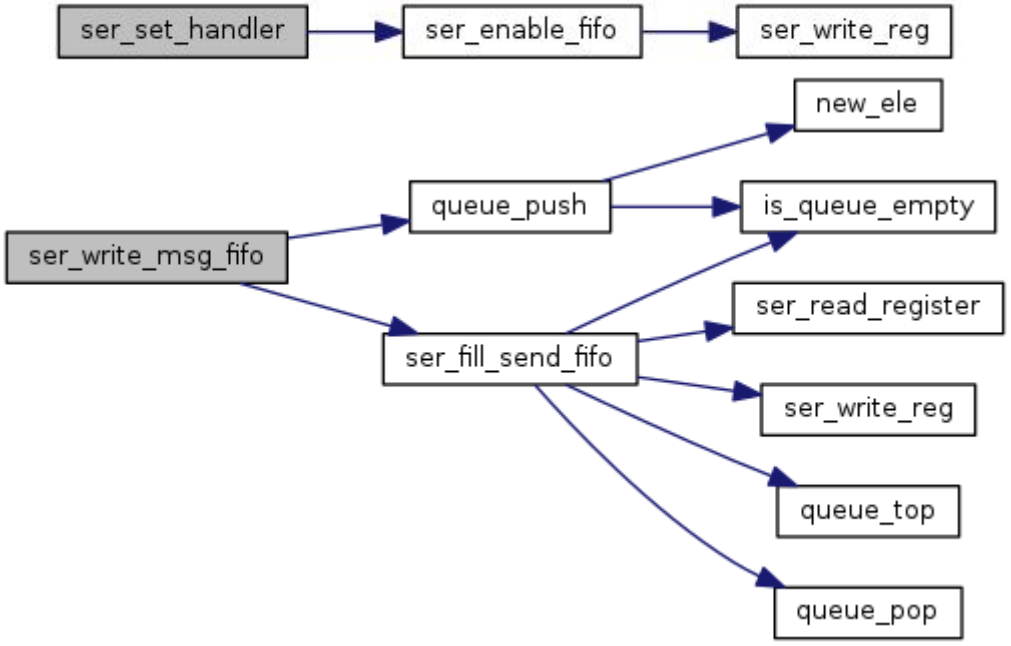


- ## Serial Port

Módulo 'serial_port'

Este módulo contém código relativo à manipulação da UART, com funções para gerir e processar interrupções, alterar o funcionamento do periférico, ler / escrever nos seus registos e enviar / receber informação. Encontram-se também neste módulo todas as constantes e macros definidas para a sua programação. Foi desenvolvido maioritariamente pelo Mário Mesquita, com uma contribuição de debugging pelo José Silva.





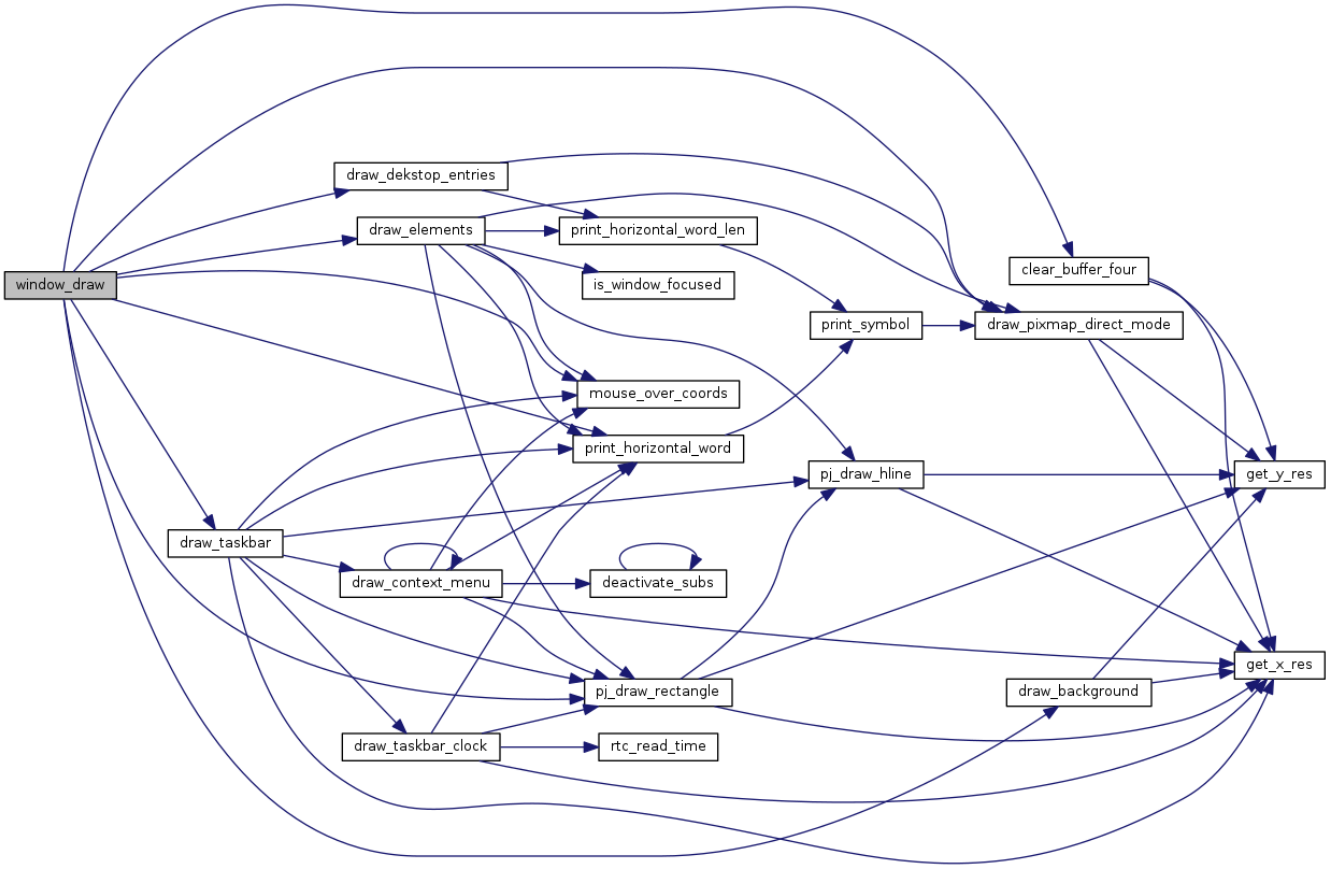
Waffle Lib

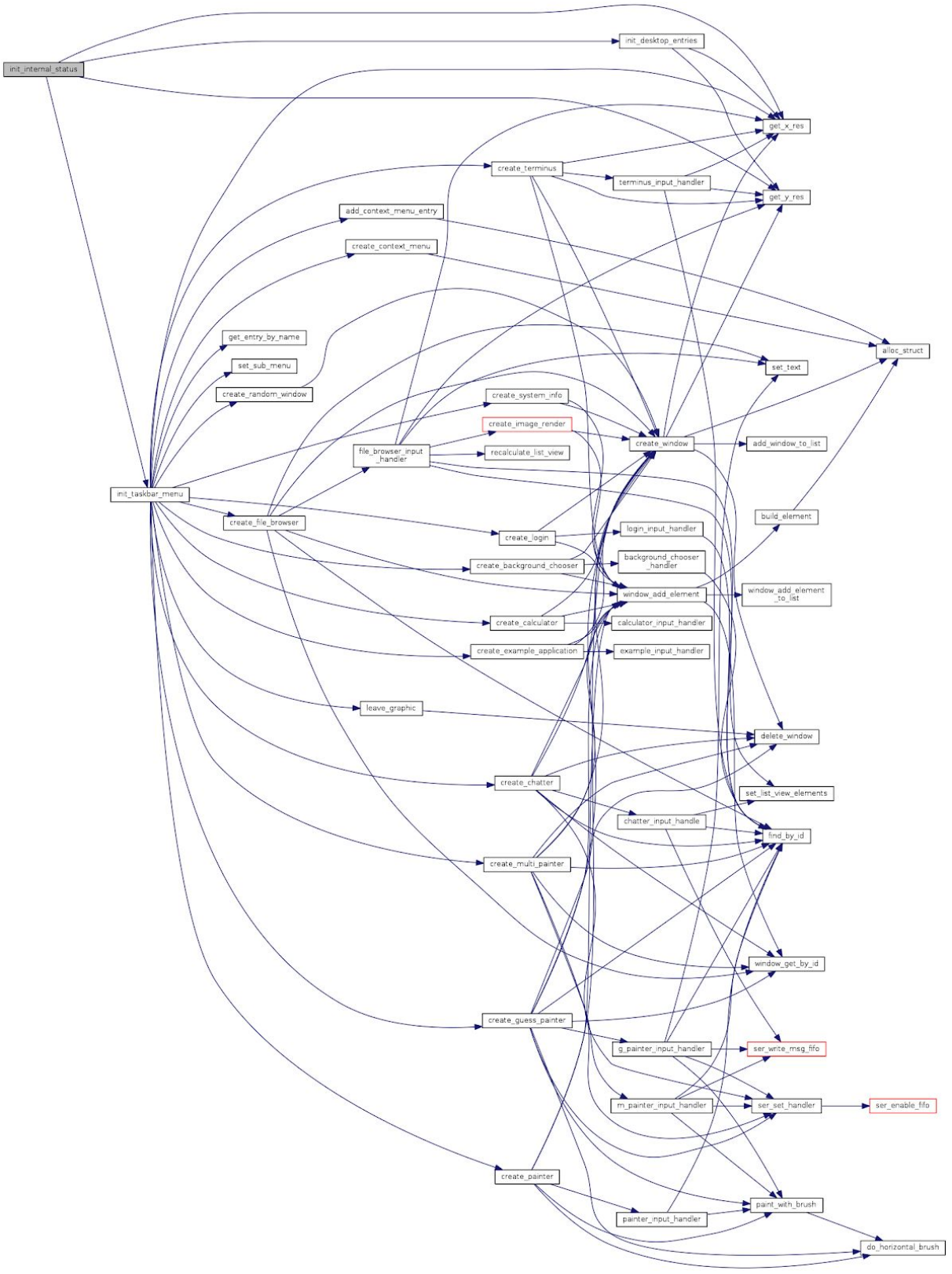
Os módulos descritos sob esta secção foram desenvolvidos para o projeto e, portanto, são específicos para este contexto.

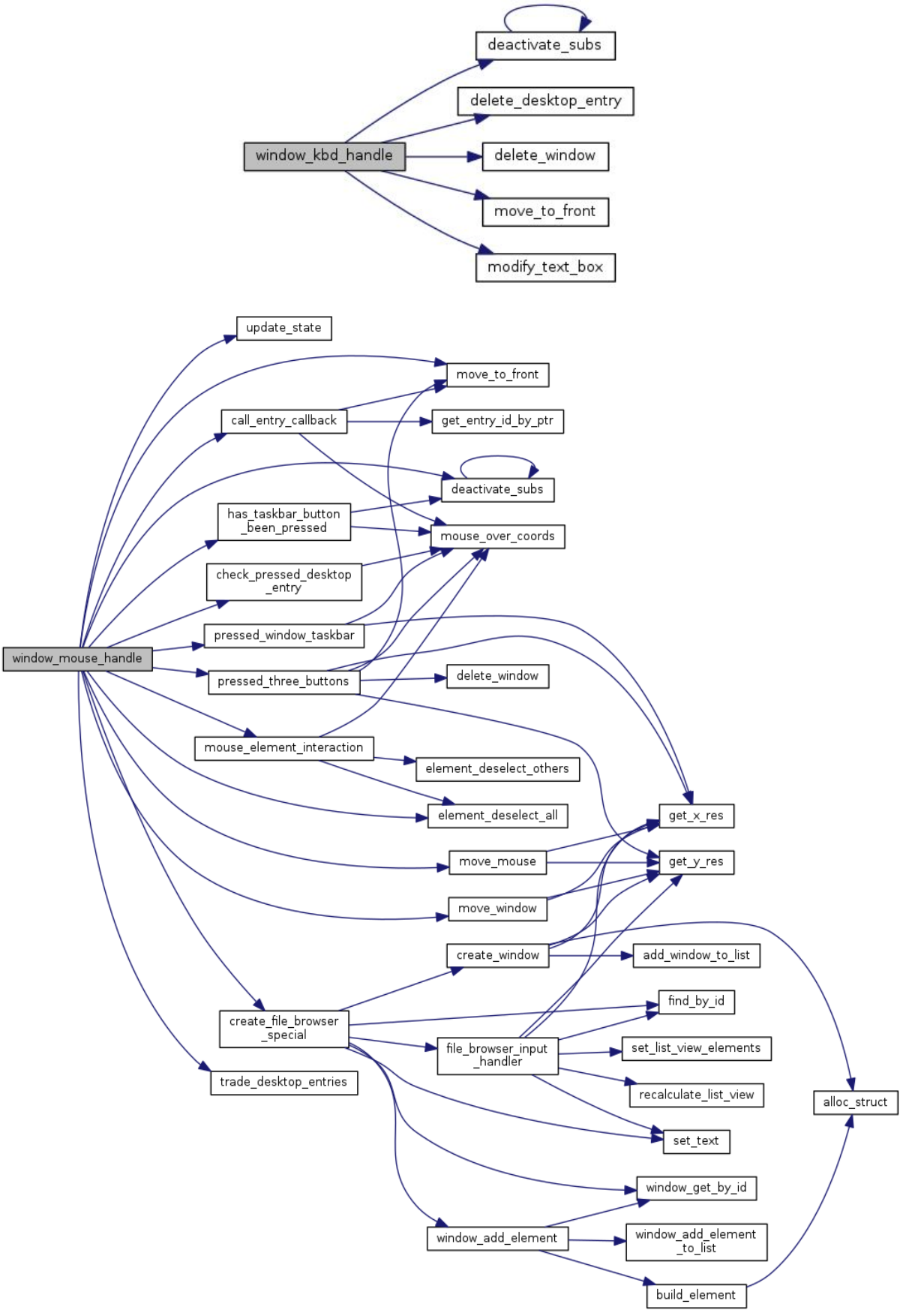
- ### Window

Módulo 'window'

É o ponto de entrada do contacto das interrupções dos diversos periféricos com as janelas e o ambiente gráfico em si. Contêm todo o código para gerir e interagir com janelas e os diversos elementos destas. As porções mais importantes de cada componente do ambiente gráfico foram separados deste módulo de forma a garantir a modularização e gestão do código. Desenvolvido por José Silva.

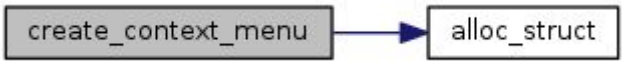


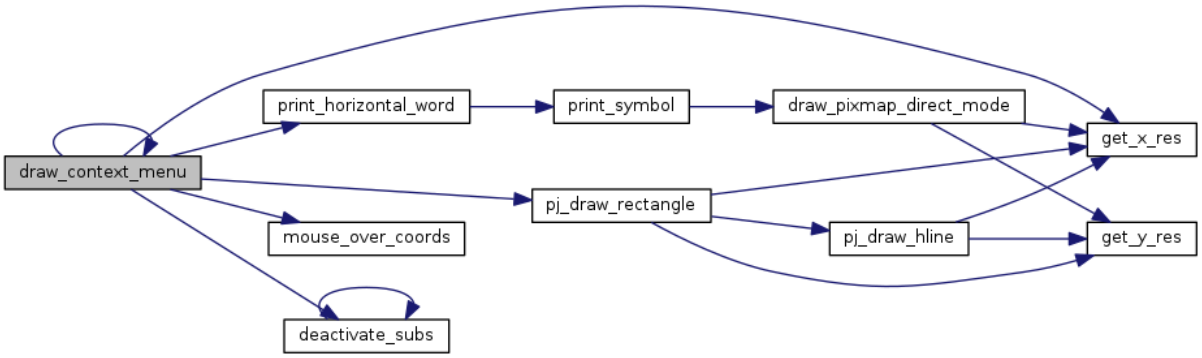

init_internal_status
init_desktop_entries
get_x_res
get_y_res
create_terminus
terminus_input_handler
add_context_menu_entry
create_context_menu
get_entry_by_name
set_sub_menu
create_random_window
alloc_struct
set_text
create_system_info
create_image_render
recalculate_list_view
file_browser_input_handler
create_window
add_window_to_list
init_taskbar_menu
create_file_browser
create_login
login_input_handler
build_element
create_background_chooser
background_chooser_handler
window_add_element
window_add_element_to_list
create_calculator
calculator_input_handler
create_example_application
example_input_handler
leave_graphic
delete_window
create_chatter
set_list_view_elements
chatter_input_handle
find_by_id
create_multi_painter
window_get_by_id
create_guess_painter
g_painter_input_handler
ser_write_msg_fifo
m_painter_input_handler
ser_set_handler
ser_enable_fifo
create_painter
paint_with_brush
painter_input_handler
do_horizontal_brush


deactivate_subs
delete_desktop_entry
window_kbd_handle
delete_window
move_to_front
modify_text_box
update_state
move_to_front
call_entry_callback
get_entry_id_by_ptr
deactivate_subs
has_taskbar_button
_been_pressed
mouse_over_coords
check_pressed_desktop
_entry
pressed_window_taskbar
window_mouse_handle
pressed_three_buttons
delete_window
mouse_element_interaction
element_deselect_others
element_deselect_all
get_x_res
move_mouse
get_y_res
move_window
create_window
add_window_to_list
find_by_id
create_file_browser
_special
file_browser_input
_handler
set_list_view_elements
recalculate_list_view
alloc_struct
trade_desktop_entries
set_text
window_get_by_id
window_add_element
window_add_element
_to_list
build_element

Módulo 'context_menu'

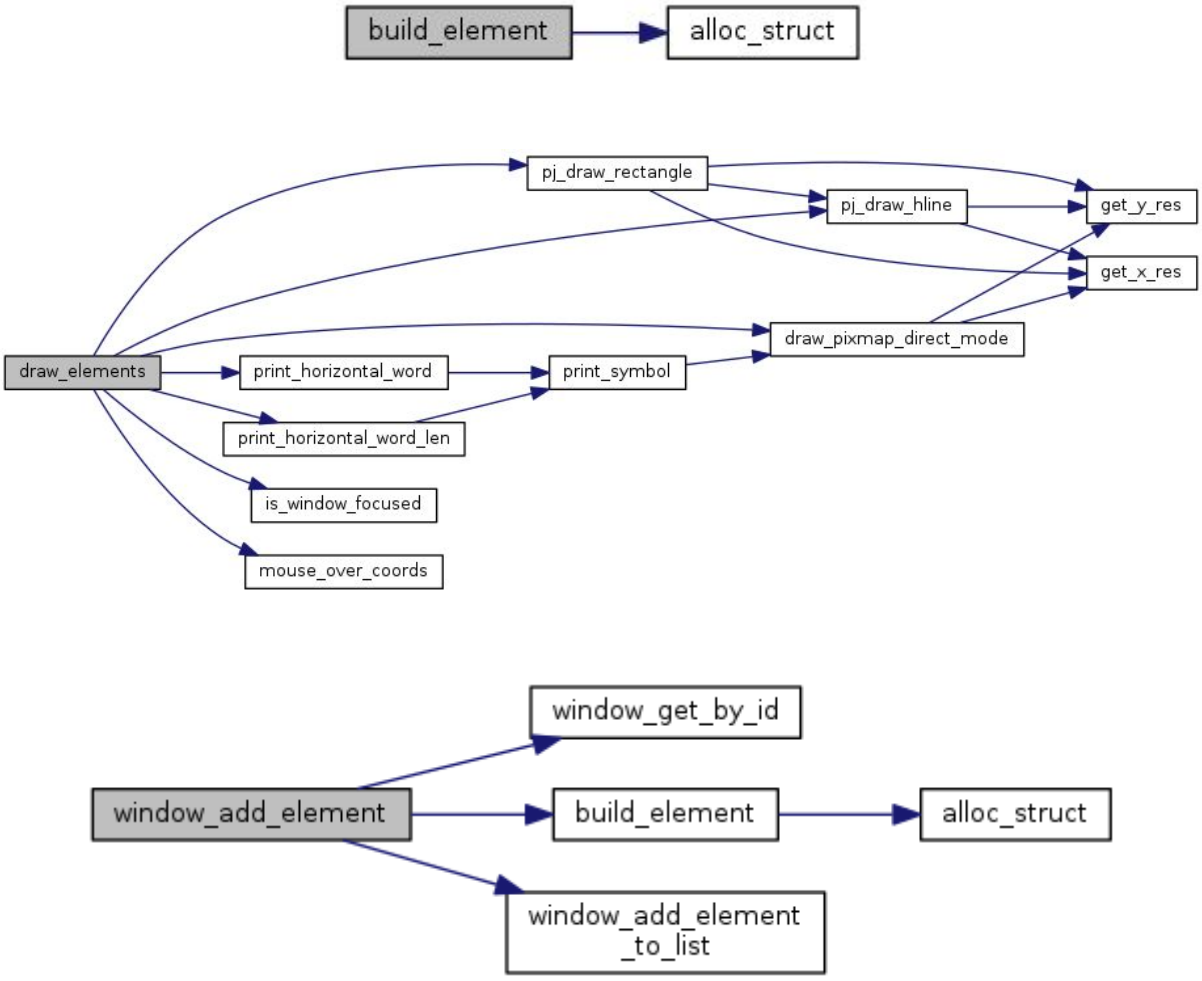
Responsável pelos menus de contexto, isto é, pela sua criação, desenho e gestão. Desenvolvido por José Silva.





Módulo 'elements'

Responsável por todos os componentes das janelas, isto é, sua criação e desenho, porque a gestão está ao critério do utilizador. No entanto, é sempre fornecido um comportamento padrão caso o utilizador não deseje nenhum funcionamento especial. Desenvolvido por José Silva.

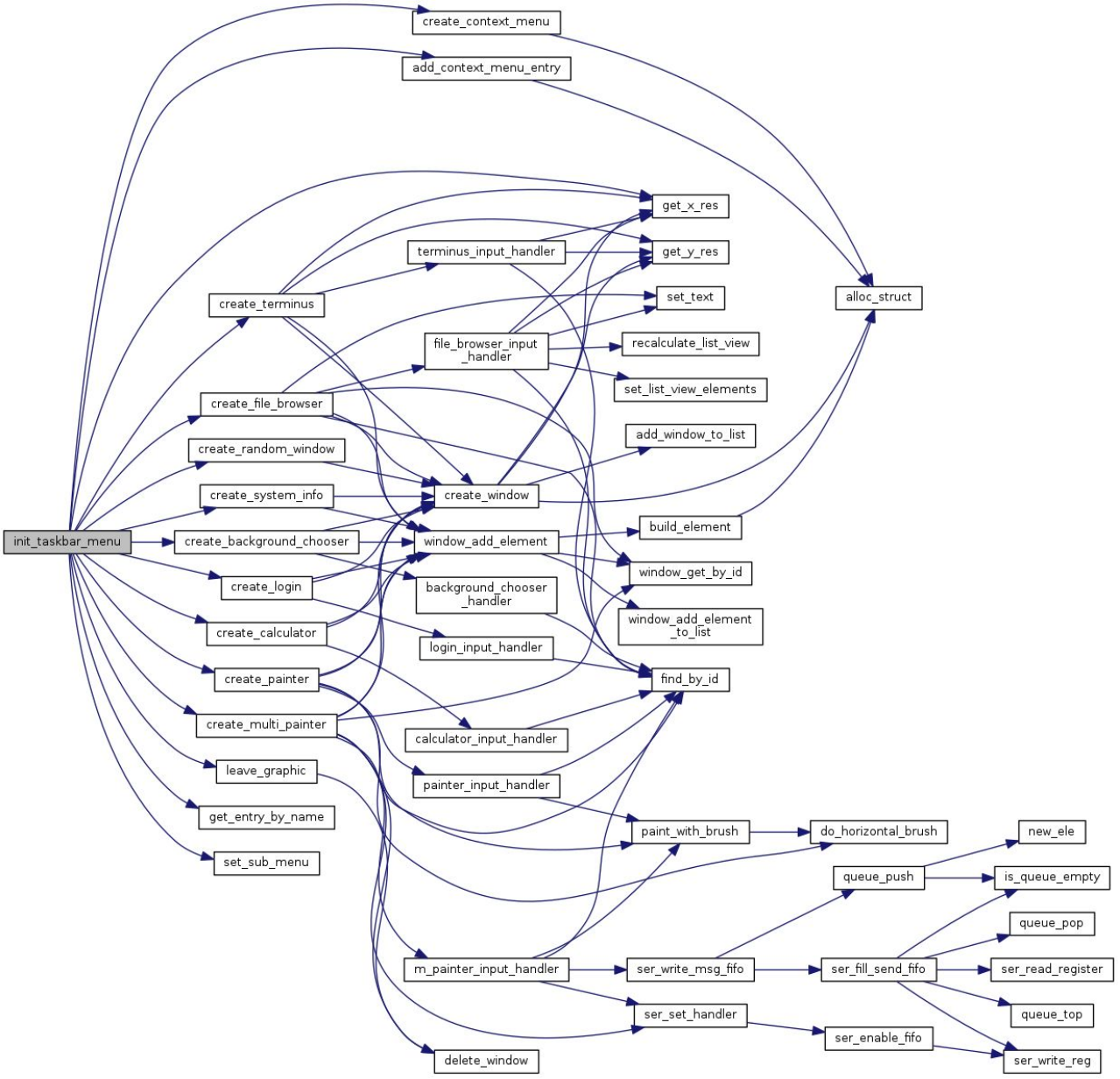


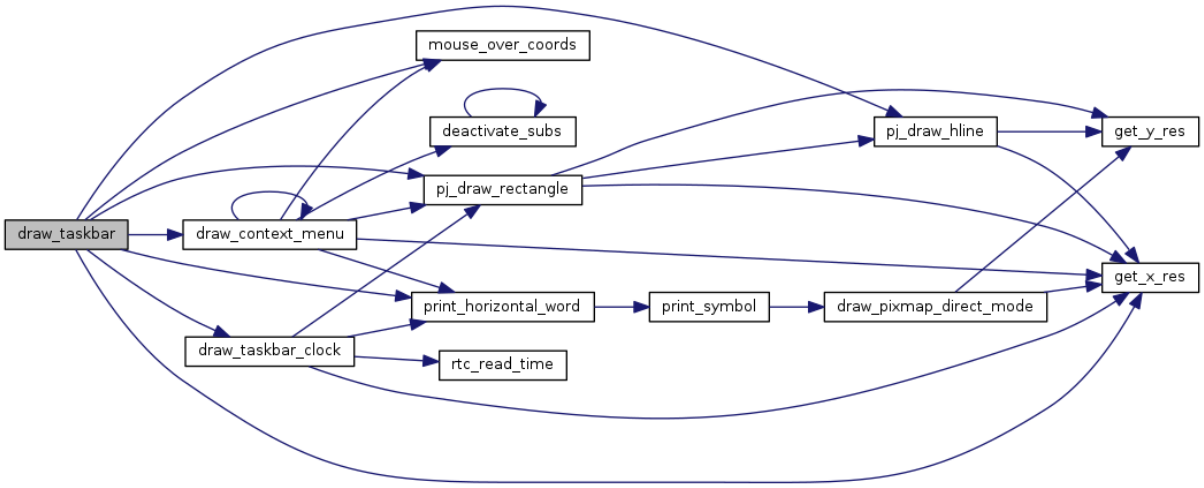
Módulo 'state_machine'

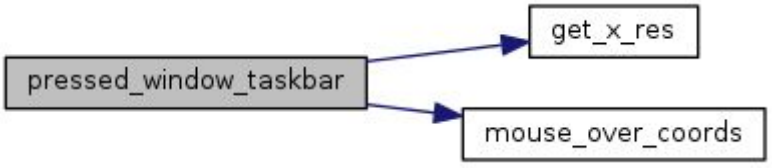
Este módulo contém a máquina de estados que gere o estado do rato. É uma implementação sucinta e simples que utiliza bitmasks para passar a informação. Desenvolvido por José Silva.

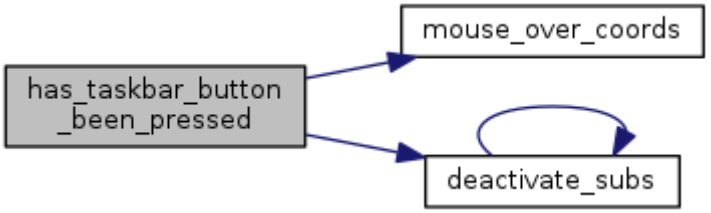
Módulo 'taskbar'

Responsável pela taskbar. Contém todo o código de gestão, desenho e interação com a barra. Desenvolvido por José Silva.




mouse_over_coords
deactivate_subs
pj_draw_hline
get_y_res
pj_draw_rectangle
draw_taskbar
draw_context_menu
get_x_res
print_horizontal_word
print_symbol
draw_pixmap_direct_mode
draw_taskbar_clock
rtc_read_time


get_x_res
pressed_window_taskbar
mouse_over_coords

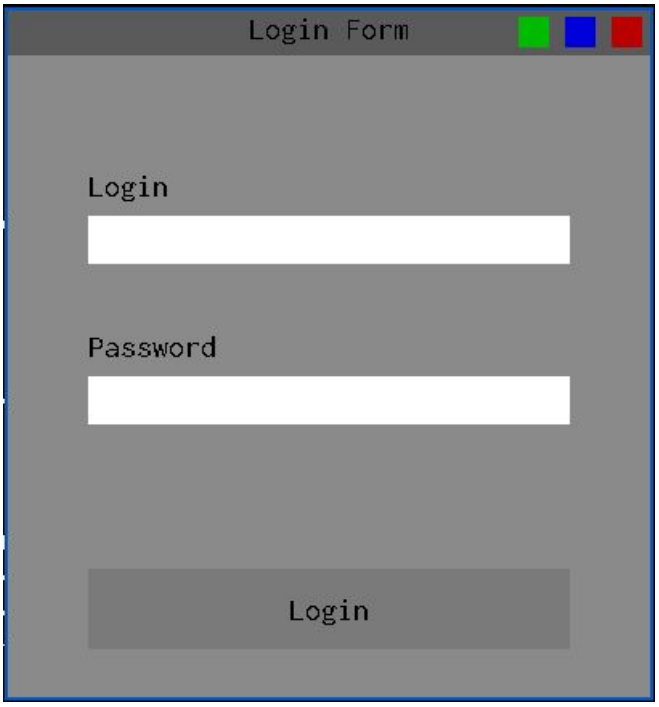

mouse_over_coords
has_taskbar_button
_been_pressed
deactivate_subs

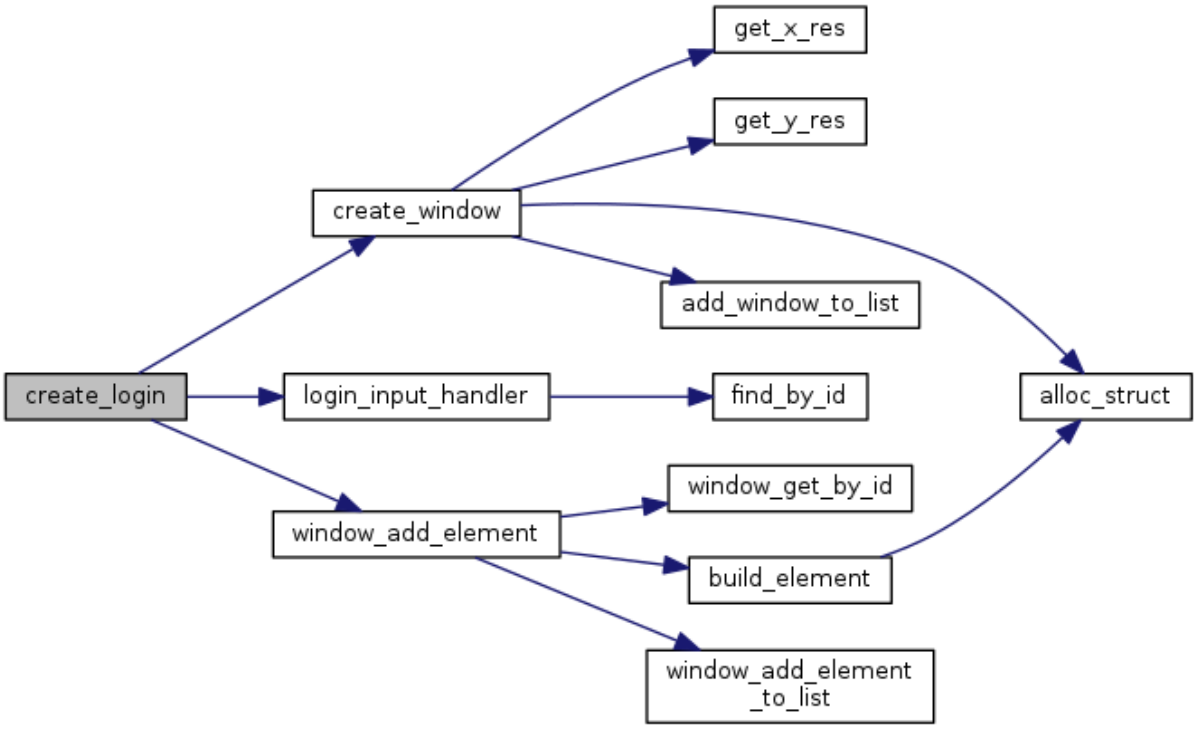
- Aplicações

Módulo 'login'

Implementação simples de um ecrã de login utilizando a biblioteca de gestão de janelas que foi desenvolvida. Recebe input pelo teclado e pelo rato, no formato de text boxes(caixas de texto) e botões, para efetuar o login.

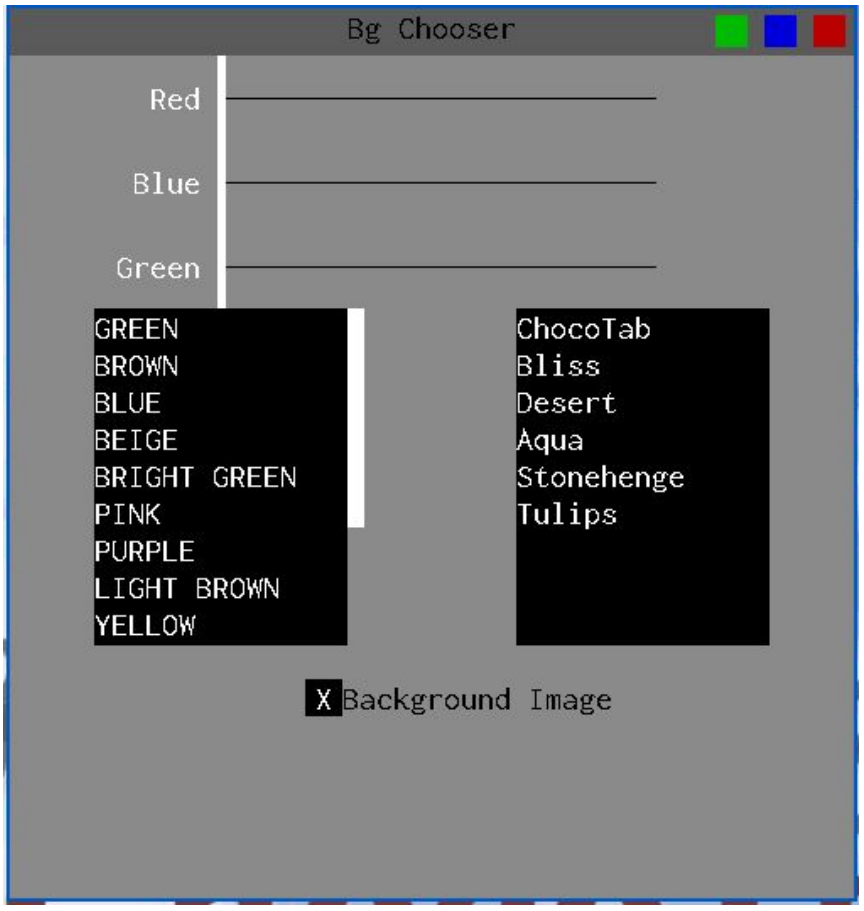
Imagem que demonstra a criação da janela, com a definição do input handler.

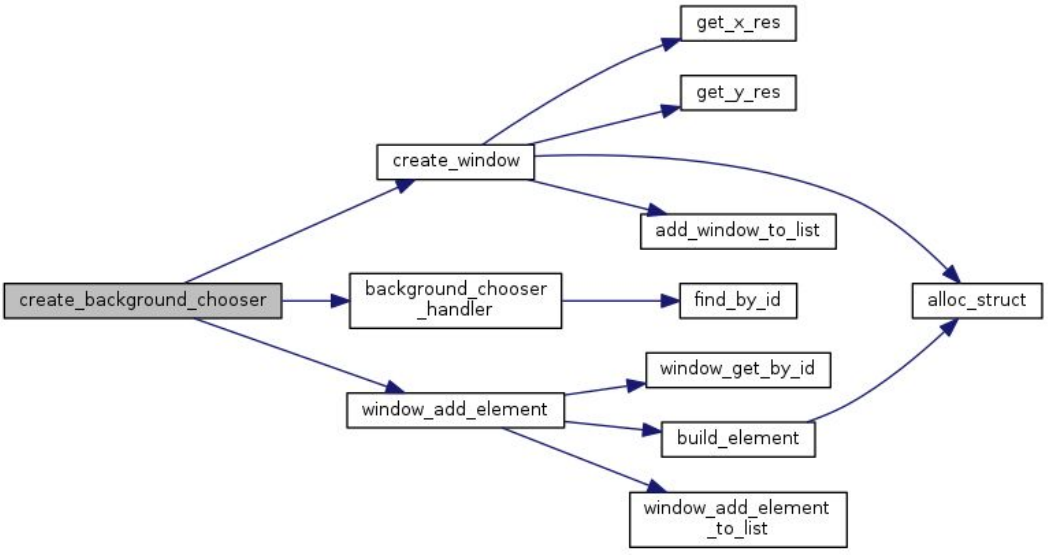




Módulo 'background_chooser'

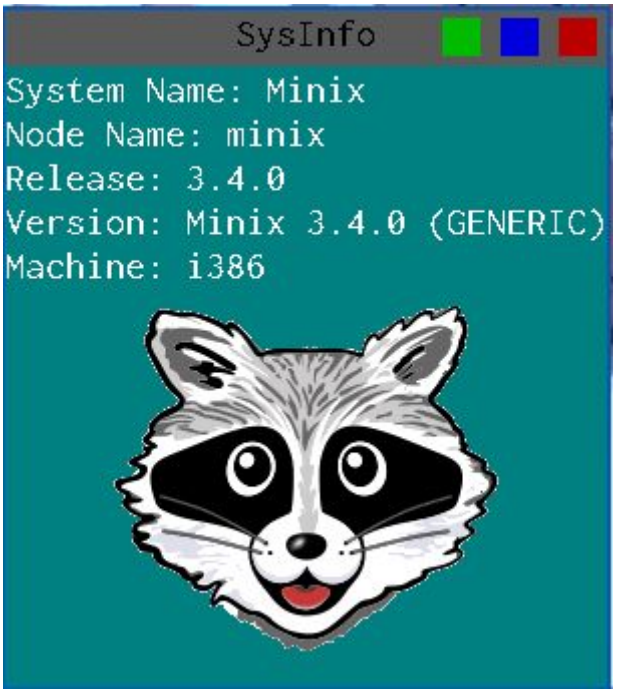
Implementação de um gestor do fundo de ambiente de trabalho. Permite alternar entre cores simples, controladas pelo utilizador e imagens que são disponibilizadas. Utiliza **SLIDER** para gerir a cor do fundo e **checkboxes.**

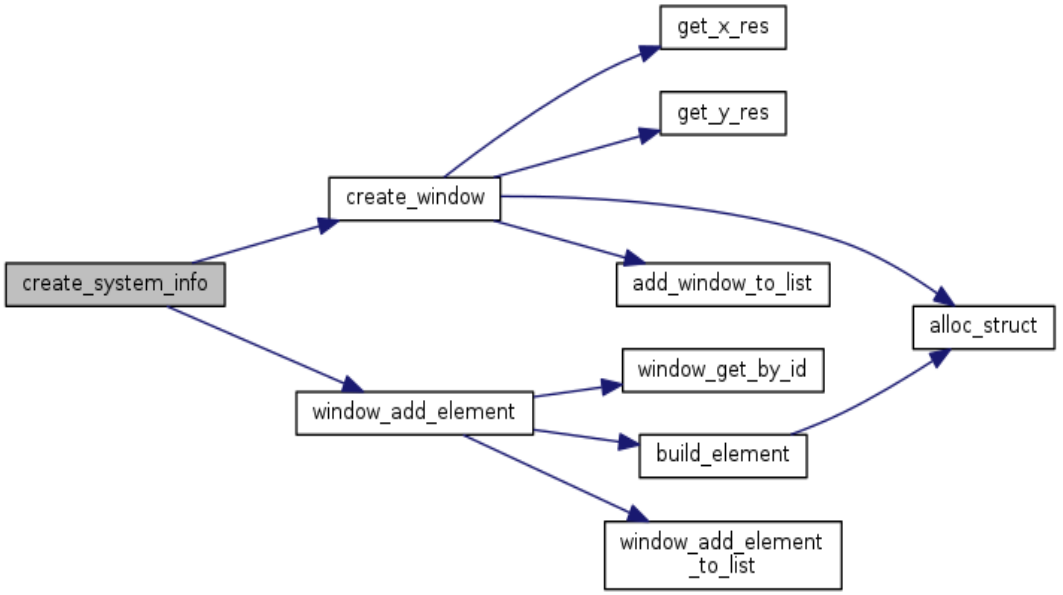




Módulo 'system_info'

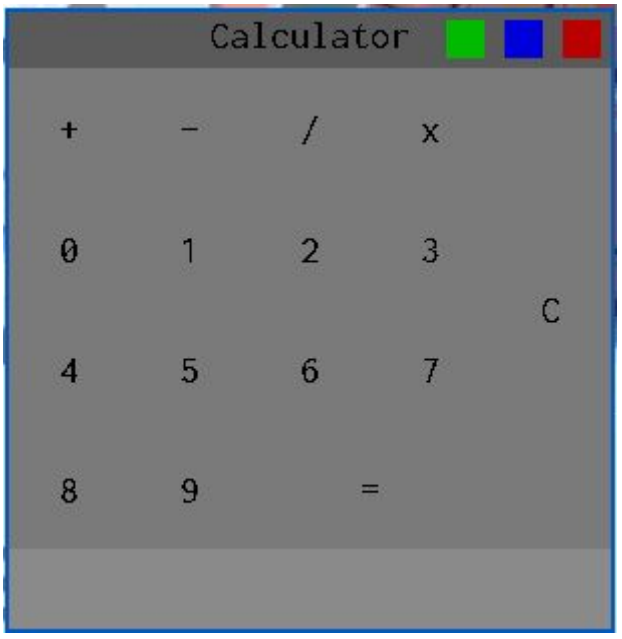
Implementação de uma janela informativa que mostra informações do sistema operativo recolhidas dinamicamente, recorrendo à biblioteca **utsname** e para mostrar o **guaxinim do Minix** utiliza o elemento **IMAGE** de **elements.**

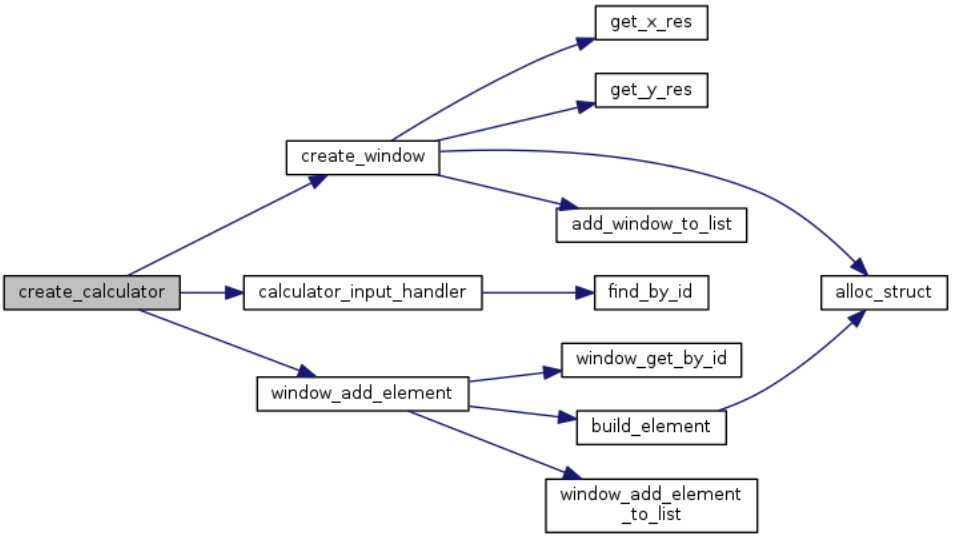




Módulo 'calculator'

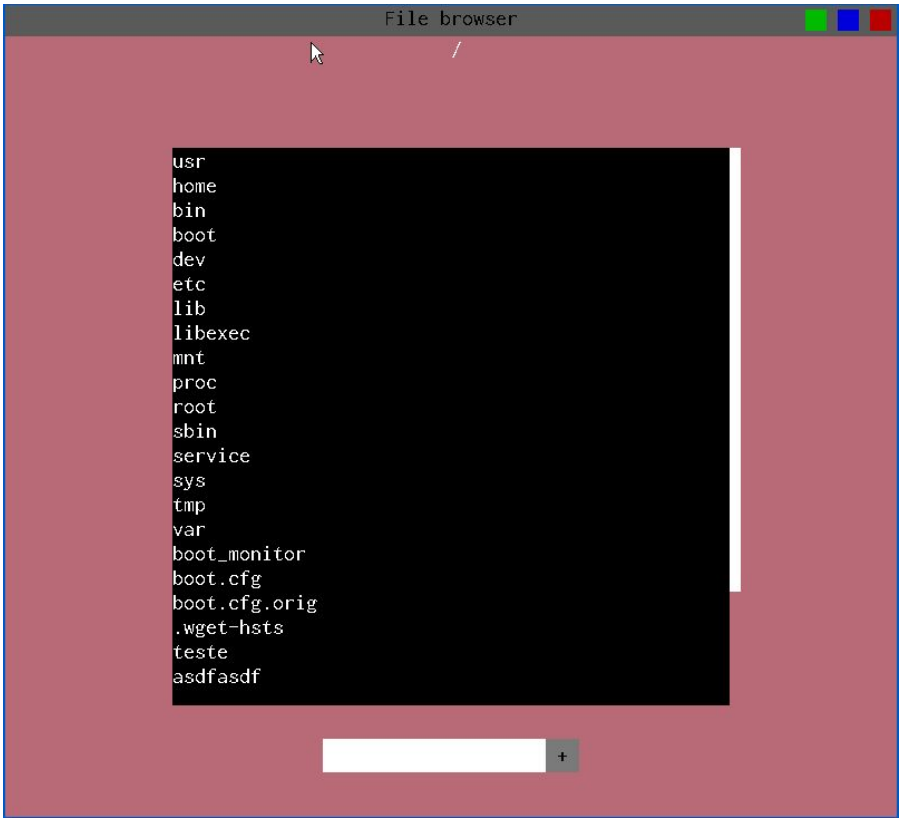
Implementação de uma calculadora, para mostrar o quão customizáveis são os elementos das janelas.

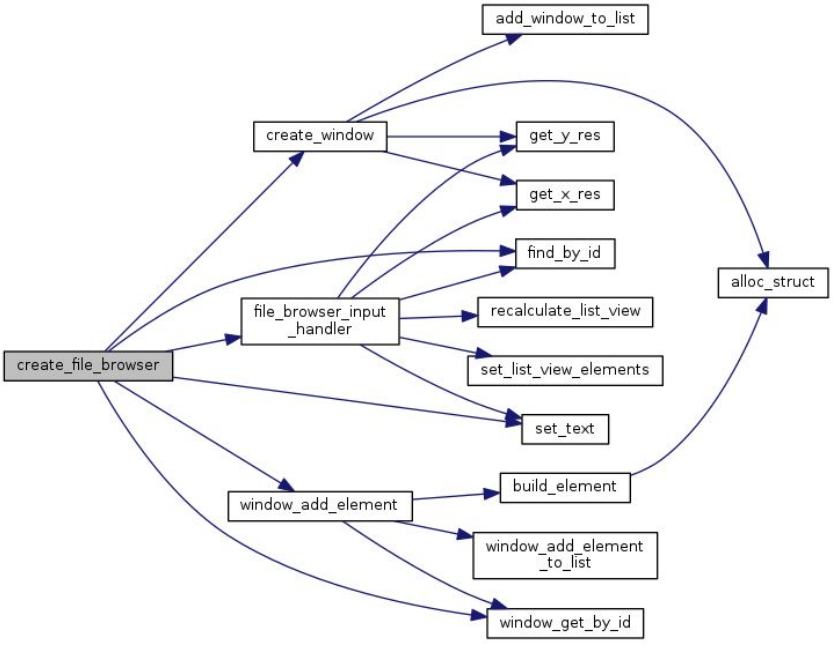




Módulo 'file_browser'

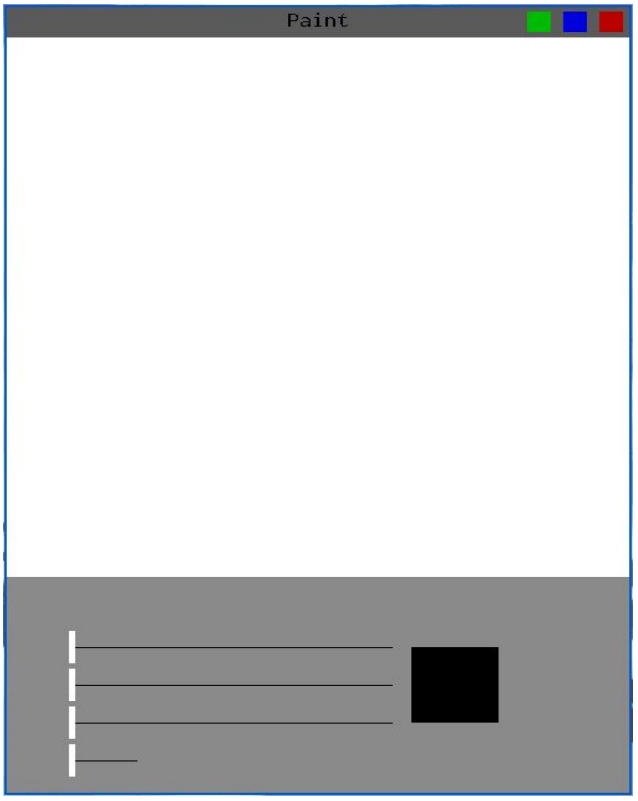
Contém o código da aplicação file browser, sendo que a listagem dos diretórios é feita com recurso à biblioteca **dirent**. A informação é disposta então numa **vista de lista**. Desenvolvido por José Silva.

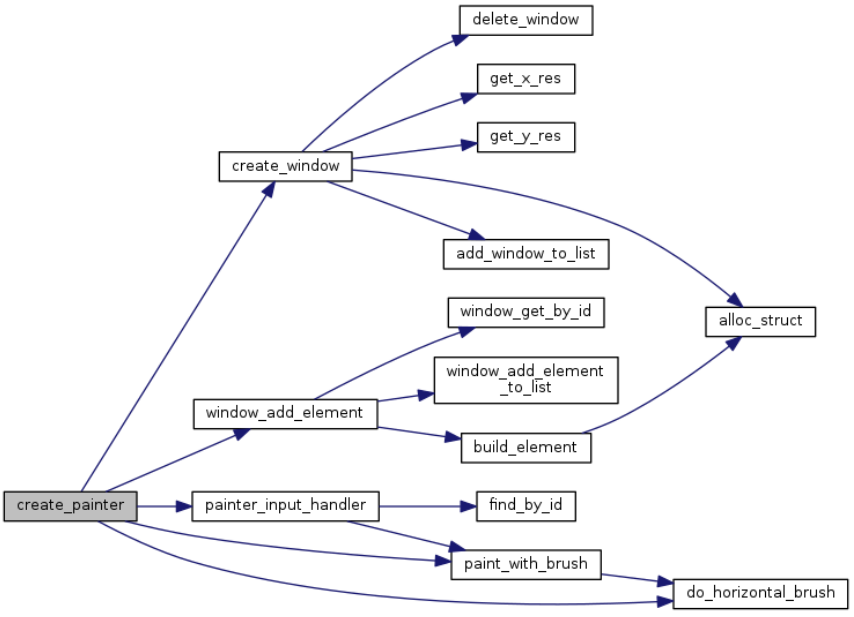




Módulo 'painter'

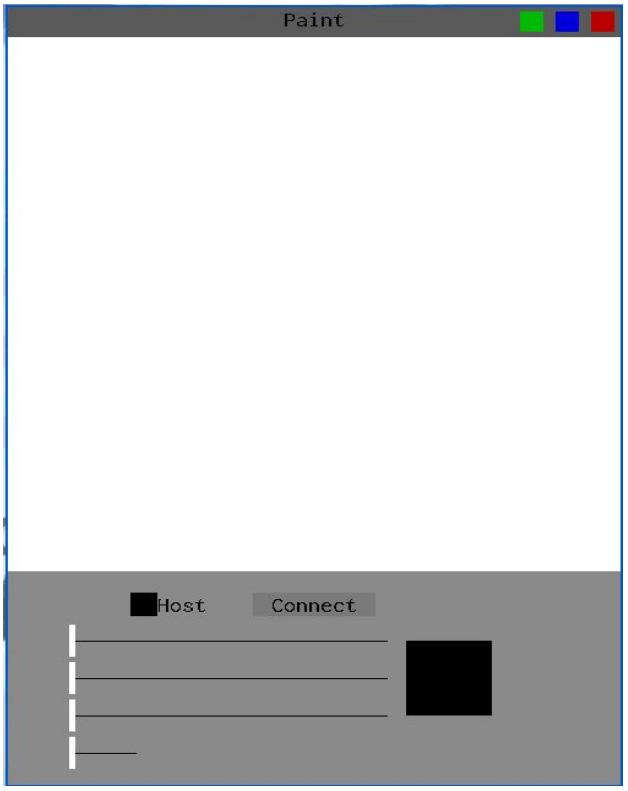
Implementa uma cópia do famoso programa Paint, permite regulação de cor e do tamanho do pincel, recorrendo a **SLIDERS**. Ser também como demonstração do elemento **CANVAS.**

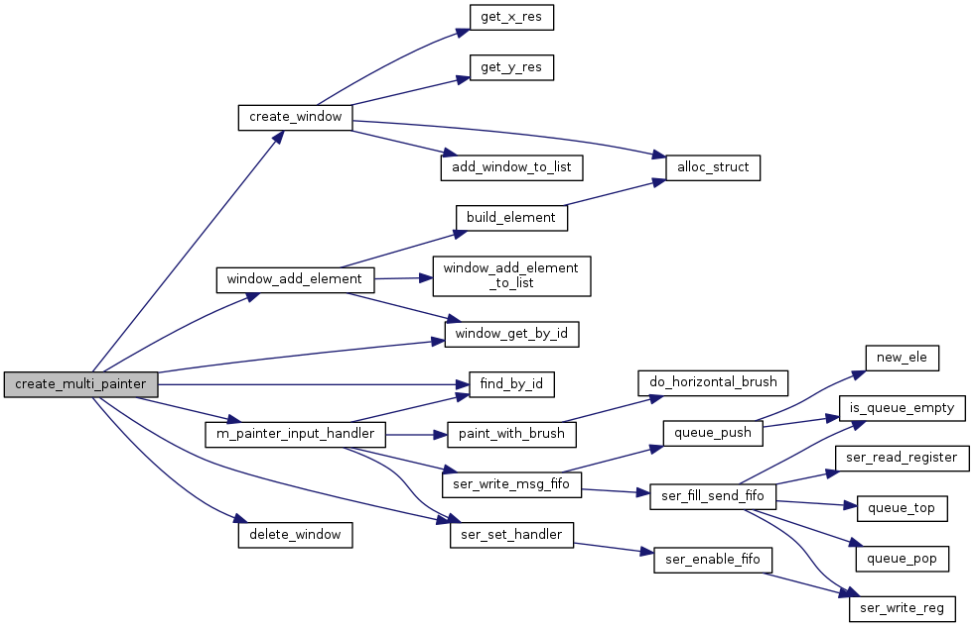




Módulo 'multi_painter'

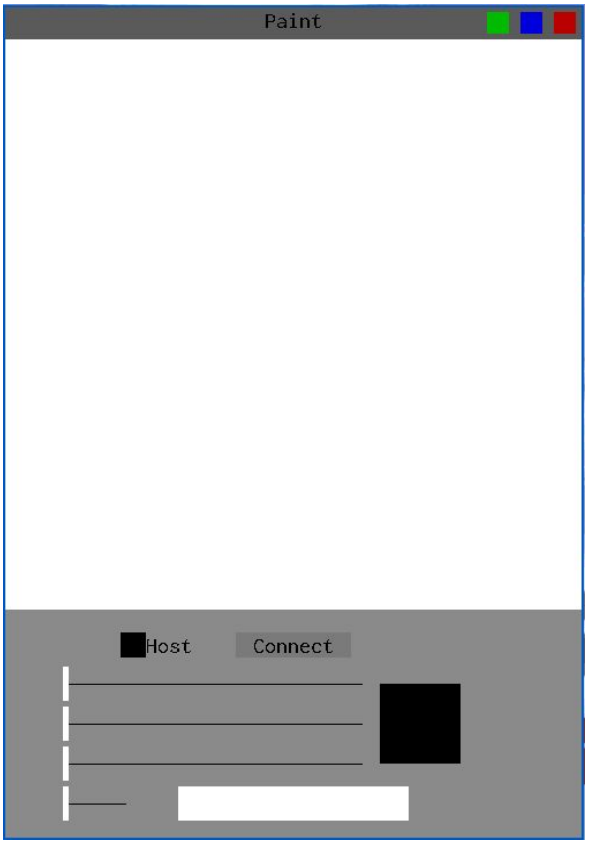
Implementa uma versão **multi-pessoa** do **Paint**. O botão de connect inicia o protocolo específico desta aplicação. Desenvolido por José Silva.

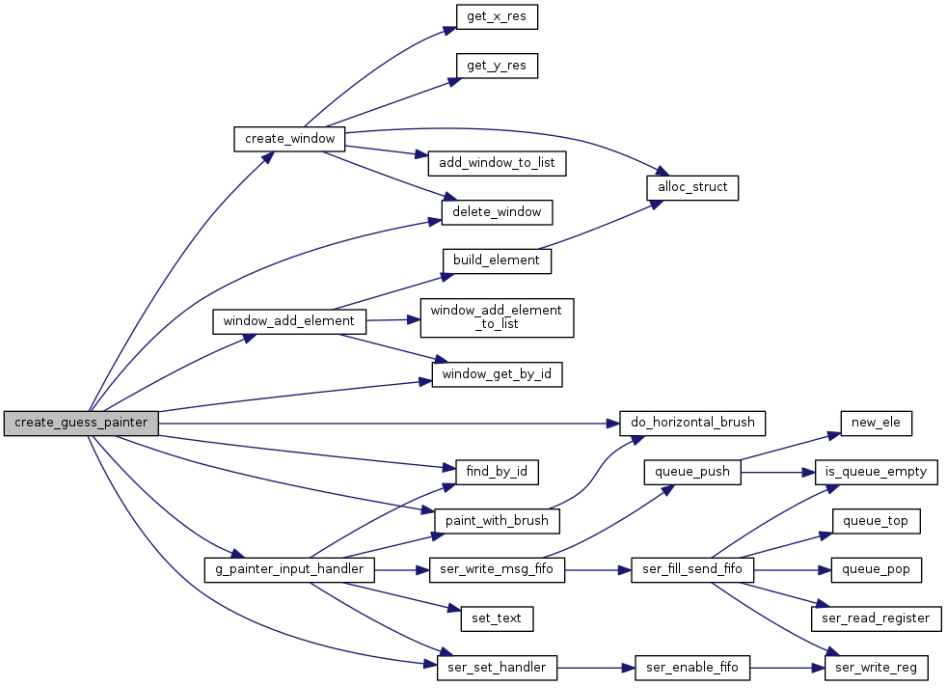




Módulo 'guess_painter'

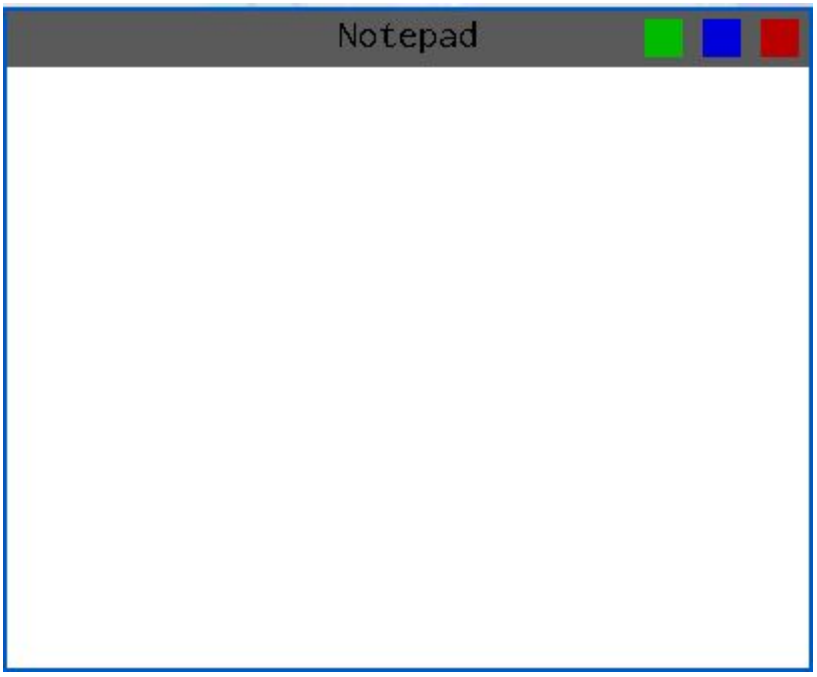
Implementa uma versão do jogo ***Draw With Friends***. em que um jogador tem de desenhar a palavra atribuída e outro tem de adivinhar a palavra através do desenho. A palavra é inserida numa **TEXT_BOX** e enviada quando a tecla Enter é pressionada. O botão de connect inicia o protocolo específico desta aplicação. Desenvolvido por José Silva.

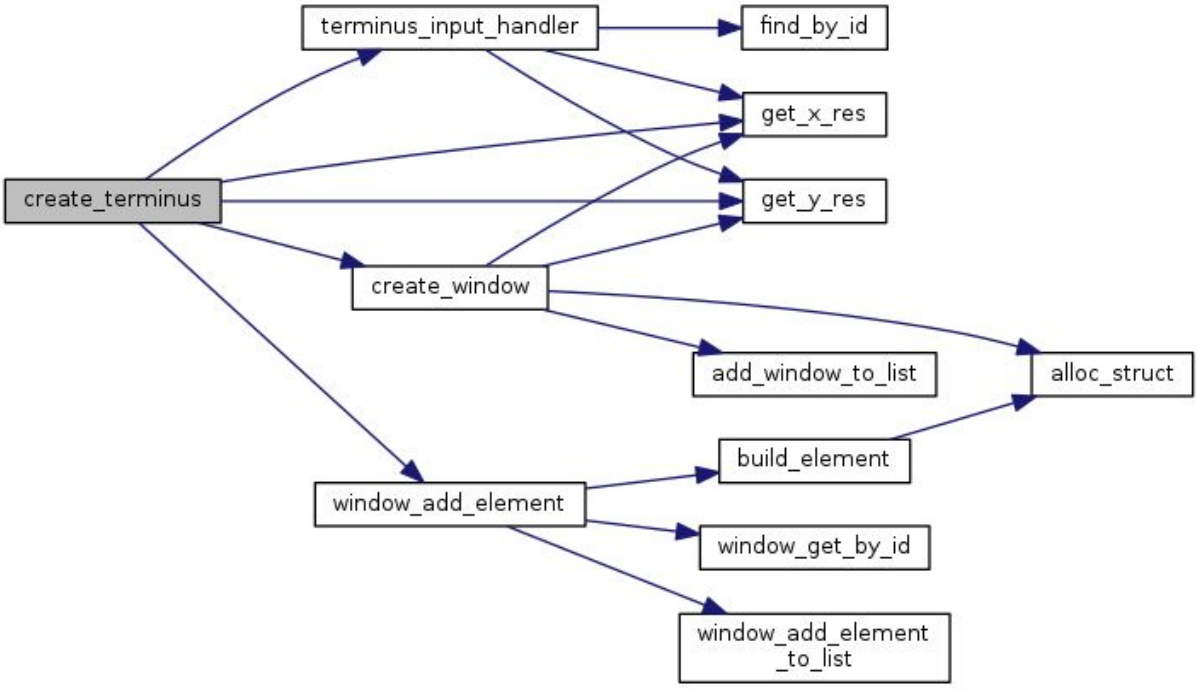




Módulo 'notepad'

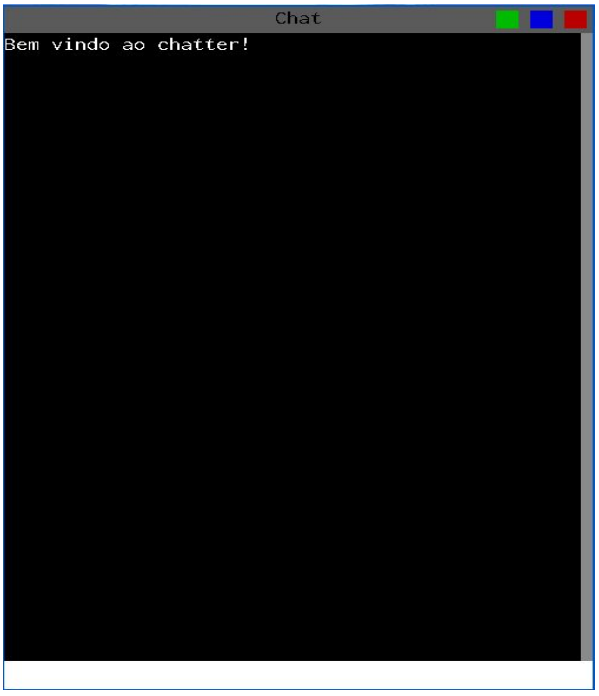
Implementa uma cópia do Notepad (bloco de notas) recorrendo a caixas de texto, **TEXT_BOX**.

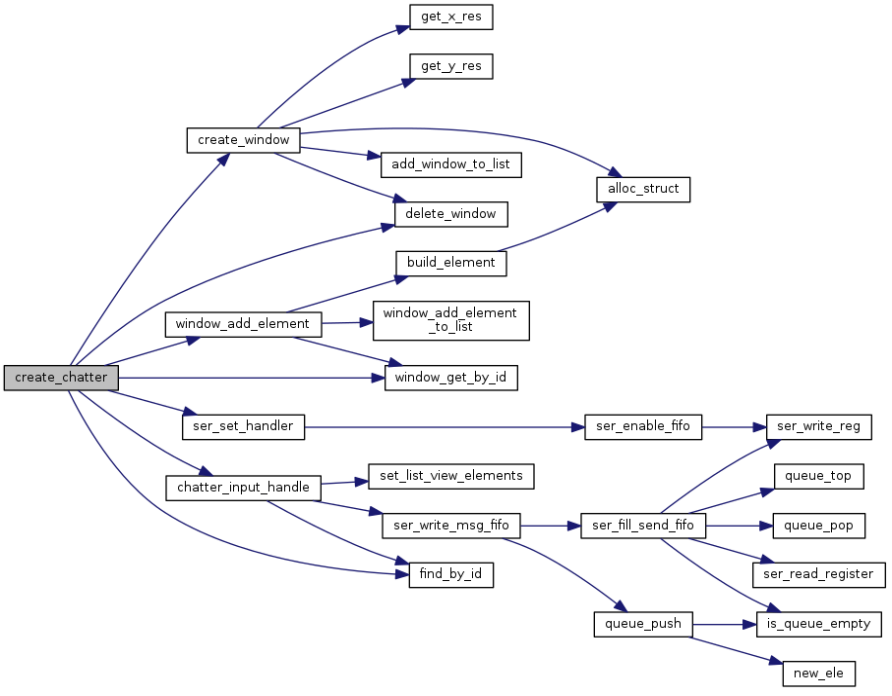




Módulo 'chatter'

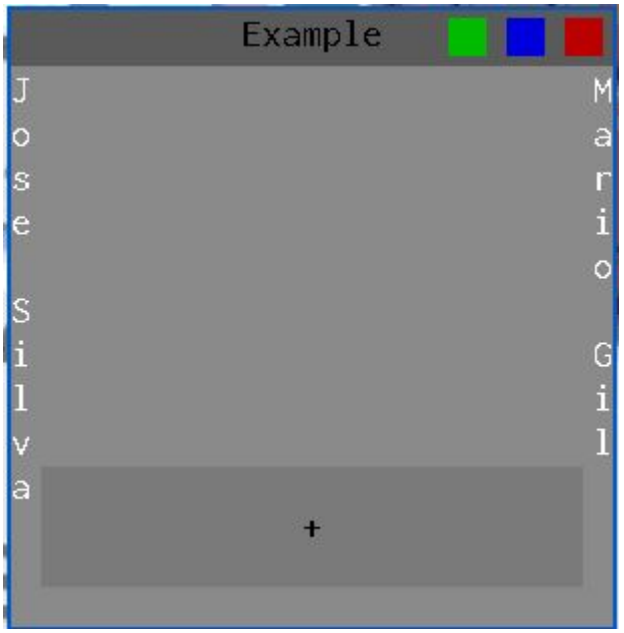
Implementa uma aplicação de chat que permite conversar com outra máquina. Utiliza **LIST_VIEW** para mostrar mensagens e uma **TEXT_BOX** para inserir as mensagens. Utiliza um protocolo especial para enviar mensagens com mais de 8 caracteres. Desenvolvido por José Silva.

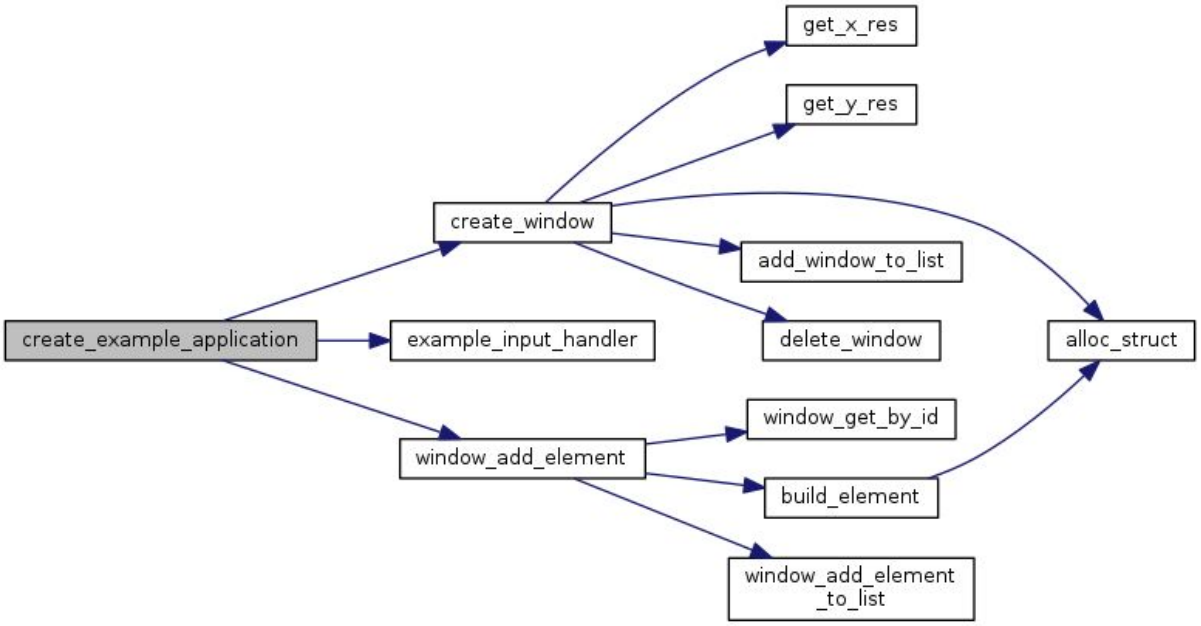




Módulo 'example'

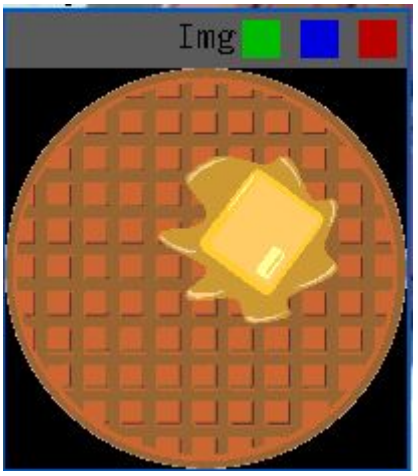
Este módulo contém uma implementação de uma janela de exemplo, com renderização vertical de texto e interação do utilizador via um botão. Foi desenvolvido por José Silva.

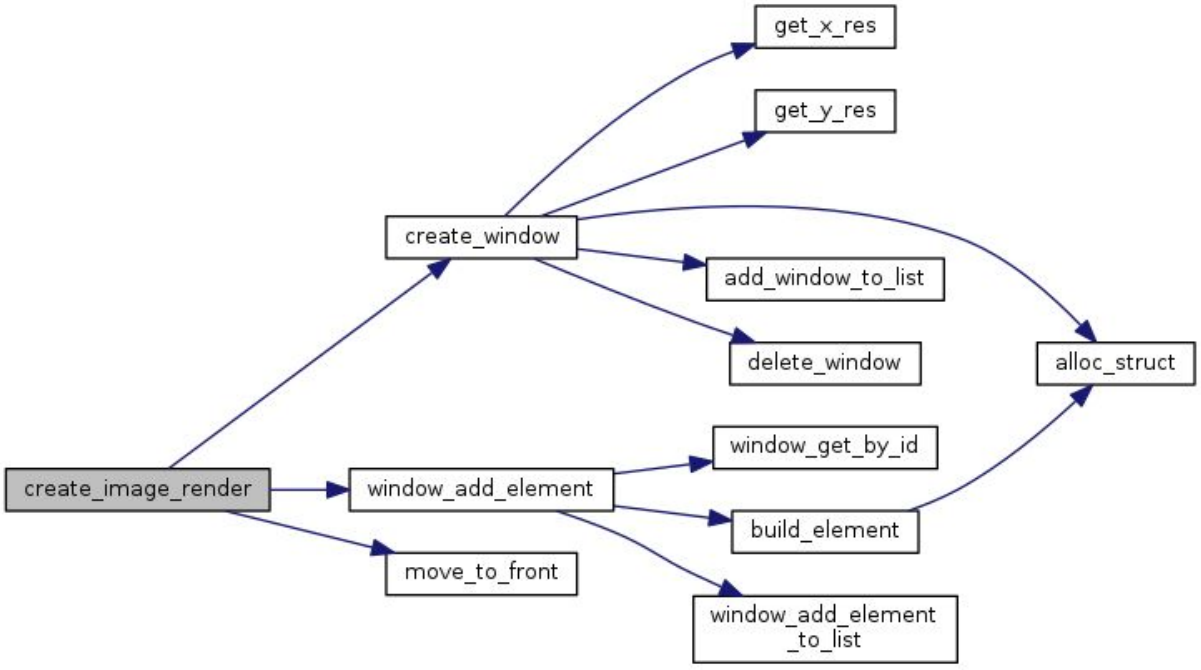




Módulo 'image_render'

Este módulo contém uma implementação de um renderizador de imagens, no formato **XPM RAW**, isto é XPM resultantes da função **xpm_load()** e cuja estrutura é a seguinte. 2 bytes, width, 2 bytes, height, 4*width*height bytes, com as informação da imagem. Foi desenvolvido por José Silva.



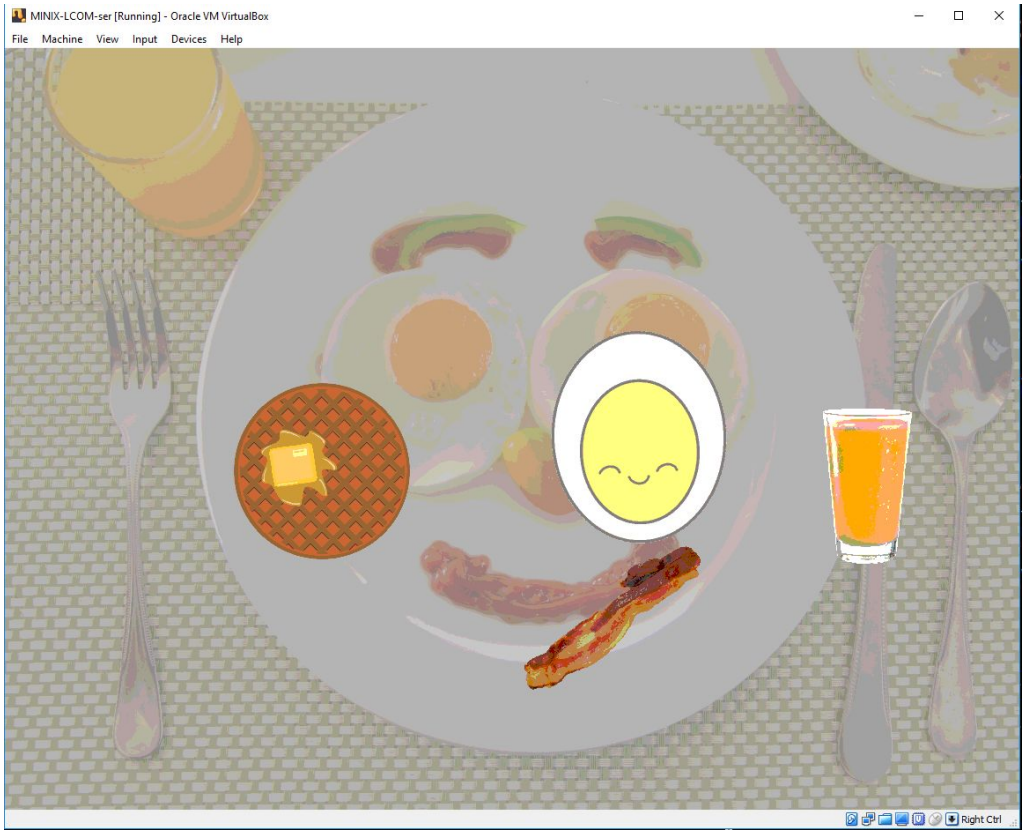


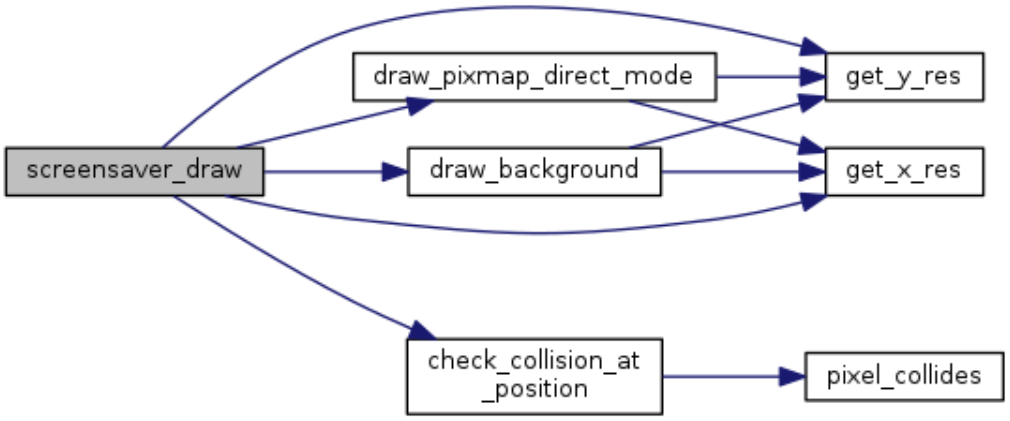
- ## Screensaver

Módulo 'screensaver'

Este módulo contém código relativo ao *screensaver*, seja para desenhar o fundo e elementos no ecrã, para detetar e resolver colisões entre elementos ou para determinar as novas posições dos elementos. Foi desenvolvido pelo Mário Mesquita e debugging contribuído pelo José Silva.

Relativamente ao screensaver, estão também definidos os ficheiros *waffle_xpm.h*, *egg_xpm.h*, *orange_juice_xpm.h, bacon_xpm.h* e *screensaver_background.h*, que contêm os xpms utilizados para desenhar os elementos e fundo do screensaver. Todos estes foram preparados pelo Mário Mesquita.



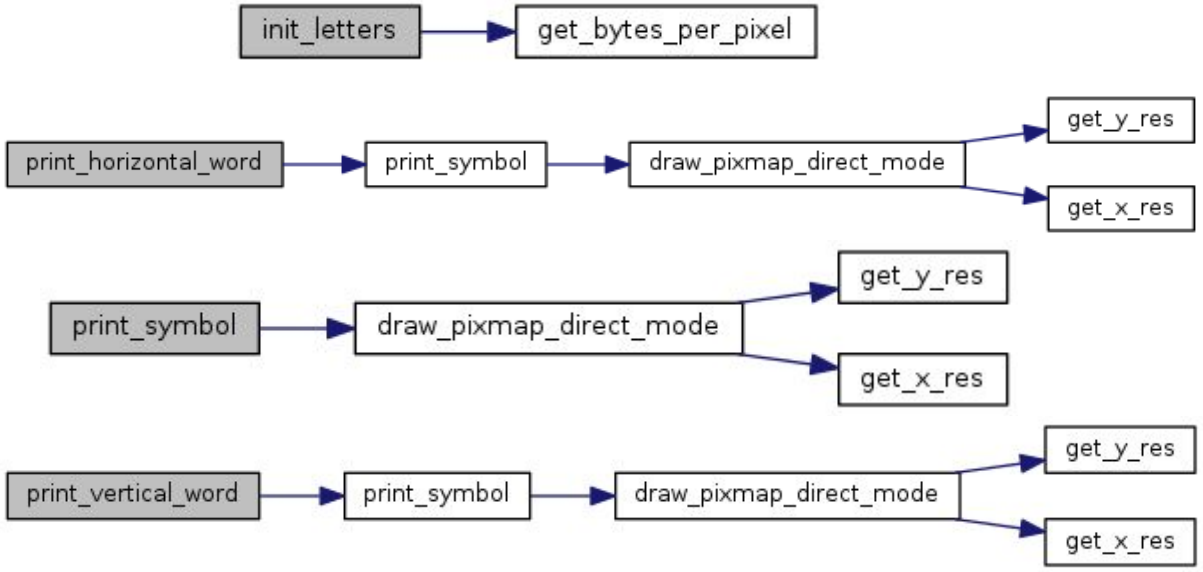


- Font

Módulo 'letters'

Este módulo contém código relativo à manipulação de texto pelo nosso programa. Dispõe de funções para inicializar a fonte utilizada, para imprimir no ecrã um símbolo ou para imprimir no ecrã palavras (horizontal ou vertical). Desenvolvido por Mário Mesquita e correções por José Silva.

Relativamente à fonte, está também definido um ficheiro font.h que contem o xpm de toda a fonte utilizada. Preparada por Mário Mesquita.



- Comunicação

  Módulo 'com_protocol'

  Contém toda a informação relativa ao protocolo de comunicação utilizado. Regista dados sobre as configurações do serial port requeridas.

Outros

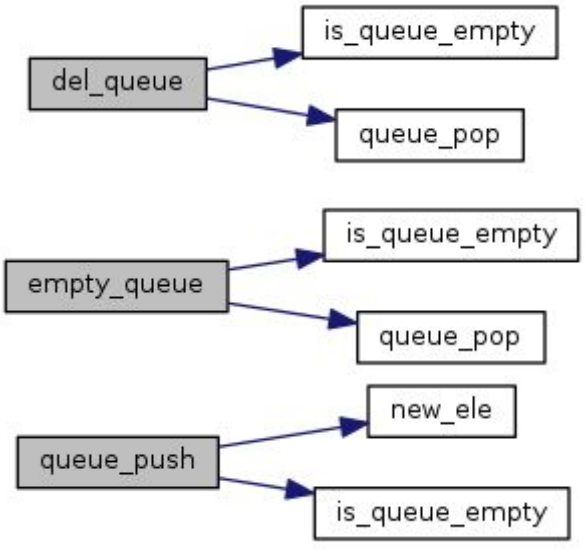
Os módulos descritos sob esta secção são tais que não se enquadram nas secções anteriores. Dizem respeito a funções utilitárias, estruturas de dados ou ao código necessário para correr o projeto.

Módulo 'util'

Este módulo contém código relativo a funções utilitárias para facilitar ou simplificar a programação de outros módulos. Foi desenvolvido igualmente pelos dois membros.

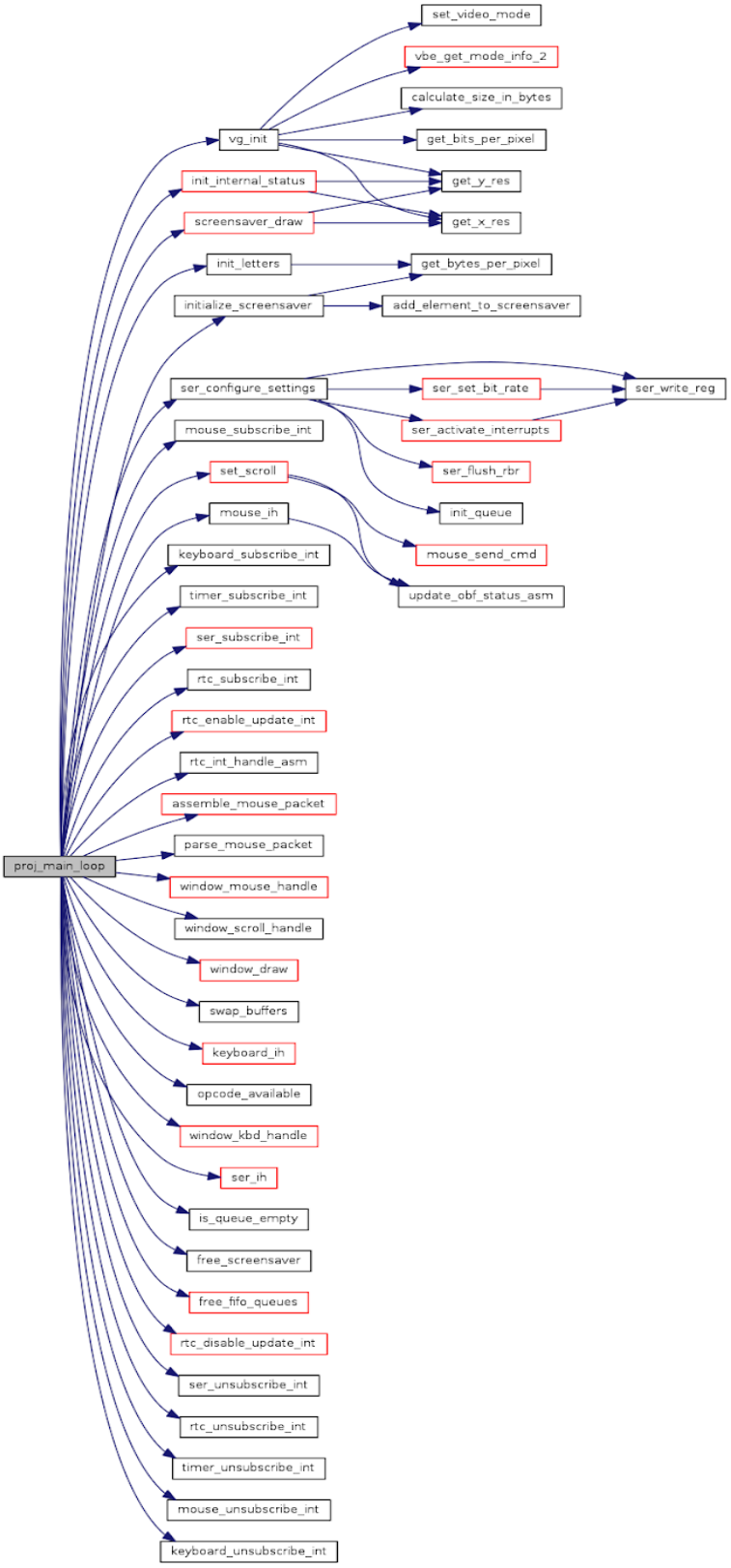
Módulo 'queue'

Este módulo contém a implementação de uma estrutura de dados idêntica a uma queue, utilizada durante a comunicação com FIFOs para guardar a informação a enviar / receber. Foi desenvolvido pelo Mário Mesquita.



Módulo 'proj'

Este módulo contém o código necessário para correr o programa, sendo aqui onde se encontra a função **main()** e o **driver_receive()** loop. Trata da configuração dos periféricos inicial, gestão das interrupções e configuração dos periféricos final. Foi desenvolvido igualmente pelos dois membros.



Detalhes de implementação

Ao longo de todo o semestre, durante os Laboratórios e o projeto, esforçamo-nos por manter o **código organizado e comentado**, de modo a permitir que seja **facilmente interpretado e reutilizado** em outros contextos. Os ficheiros do projeto encontram-se organizados em pastas, agrupando módulos relacionados. Esta abordagem permitiu que facilmente se conseguisse integrar o código relativo aos periféricos, desenvolvido nos laboratórios, no nosso projeto, com poucas ou nenhumas alterações.

No mesmo tópico, definiu-se **macros e constantes simbólicas** para todos os valores que utilizamos no nosso código, de modo a que, em qualquer altura, consigamos identificá-los pelo o que representam. Deste modo, por exemplo, mesmo alguém não familiarizado com o KBC saberá que 0x64 permite aceder ao registo de controlo do dispositivo.

Destaca-se também a distribuição por **camadas de abstração**, separando as funcionalidades de baixo e alto nível que interagem entre si. Um exemplo desta separação é o módulo do KBC, um dispositivo que é usado tanto pelo teclado como pelo rato. Com efeito, os dois periféricos mencionados partilham o mesmo código, chamando as funções de baixo nível definidas para o KBC.

O programa foi desenvolvido com uma forte componente de **programação orientada a objectos**, tendo sido criados várias *structs* para representar elementos. É o caso das Janelas, objeto *Window* e suas componentes, objeto *Element*, os objetos presentes no Screensaver, *ScreensaveEle*, mensagens para e de janelas, *MESSAGE*.

São muito utilizadas as **listas ligadas**, devido à sua facilidade de implementação e gestão. Começando pelas janelas em si, estas encontram-se numa lista duplamente ligada, sendo que o elemento que se encontra no topo é que se encontra mais a frente, mais perto do utilizador, ou seja, a lista de janelas funciona como uma **z-order**. Esta implementação tem duas grandes vantagens, ao desenhar pode-se percorrer a lista de trás para a frente para desenhar, **algoritmo do pintor**, e durante o processamento de input vindos do utilizador percorre-se de frente para trás para se saber para que janela é que esse input deve ser enviado.

Os **elementos das janelas** (botões, vistas de lista, caixas de texto,...) também se encontram numa lista, sendo que a principal vantagem disto é o facto de ser muito fácil a expansão em caso de adição de um novo elemento. Se fosse feito com vetores, quando se chegasse a um determinado limite seria necessário re-alocar memória o que poderia ser mais moroso e intensivo. A criação e implementação de novos elementos é feita de maneira muito simples, sendo os passos os seguintes: adicionar nova entrada à enumeração, adicionar caso à função **build_element** e por fim, colocar sua função de desenho no array **dispatch_draw**, este array contêm todas as funções de desenho dos respetivos elementos sendo que o índice de cada função é dada pela entrada do elemento na enumeração. Sendo assim, sendo que novos elementos, são adicionados no final então é só colocar uma função no final do vetor.

Como todas as janelas se encontram a correr dentro do mesmo programa, não se pode utilizar variáveis globais ou variáveis estáticas, então a especialização do comportamento de janelas iguais é feita com recurso aos identificadores de elementos. Todos os elementos podem possuir um identificador, uma string, e o utilizador pode ir buscar dado elemento recorrendo à função **find_by_id**. Para os casos em que é necessário guardar informação que não pode ser representada sobre qualquer elemento, existe o tipo de elemento **DATA** que simplesmente contém informação e está reservado ao utilizador o que quer fazer com ela. A libertação do espaço de memória também está ao cargo do utilizador.

Para garantir que todos os elementos estão sempre na posição correta, então a sua posição é dada de **forma relativa em relaçào janela**, assim sempre que se diz que um elemento E tem posição em x é em relação à janela em que se encontra e não ao ecrã. Esta decisão fez com que ao deslocar janelas, os seus elementos a acompanhem sempre, dando assim a impressão que estão nela contidos.

De forma a garantir que todos os elementos se encontrariam no mesmo espaço de memória e estavam bem encapsulados foram utilizados **UNIONS**, dando assim uma característica **polimórfica** ao campo **attr** de **Element**. Dependendo do campo type, este campo era interpretado de forma diferente. Esta medida garantiu código mais sano e mais fácil de gerir.

Os **menus de contexto** também são implementados utilizando uma lista ligada, pelas mesmas razões apresentadas nas janelas e nos elementos. Para além disso, a **ideia de cadeia de menus** é representada muito mais naturalmente na forma de listas ligadas. Para se desligar sub-menus ativos utiliza-se um pequeno **truque recursivo** em que cada elemento é capaz de desligar o sub-menu que possui ligado sem "destruir" o caminho que tem para destruir outros sub-menus.

Para tratar as alterações de estado do rato foi implementada uma máquina de estados muito concisa que converte todos os pacotes de rato para uma bitmask que contém todas as alterações dos botões de rato.

Quando há uma interrupção de rato ou de teclado para além do contador de inatividade ser reiniciado estas interrupções são passadas para as janelas atráves das funções **window_mouse_handle()** e **window_kbd_handle()**.

Em termos da função do rato é esta que proporciona toda a interação com rato o ambiente gráfico. Isto é, sempre que há um pacote de rato esta função atualiza a sua máquina de estados e toma uma ação consoante o resultado. Por exemplo, se o botão esquerdo do rato se encontra pressionado e se tal for verdade se o utilizador está a segurar ou simplesmente a clicar. No caso de haver um clique, a função primeiro verifica se a taskbar foi pressionada, caso não tenha sido, percorre a lista de janelas, de cima para baixo, à procura da janela pressionada. Se uma janela tiver sido pressionada, então esta fica focada, passa para o topo da lista, caso nenhuma janela seja pressionada então é verificado se algum item do ambiente de trabalho foi pressionado. Caso o elemento seja pressionado e não esteja selecionado então passa a estar selecionado, se já estiver selecionado então é executada a função que o tenta abrir. Se for uma pasta, abre o **file_browser**, se for imagem, abre o **imagem_render**. Se o utilizador estiver a segurar no botão esquerdo do rato então duas coisas podem acontecer. Se uma janela estiver selecionada pela sua barra então é deslocada, se for um objeto do ambiente de trabalho então este é deslocado. Na situação que o utilizador desloca a scroll wheel, então o input é diretamente passado à janela mais à frente que caso tenha uma vista de lista pode deslocá-la.

Em termos da função do teclado é também esta que proporciona toda a interação do teclado com o ambiente gráfico. Primeiramente verifica combinações especiais como ALT-TAB ou ALT-F4 ou até a tecla DELETE. Caso nenhuma destas tenha sido pressionada então envia a tecla recebida para a janela atualmente em foco. A janela é que decide o que fazer com o input. Se tiver uma **TEXT_BOX** pode até escolher para o gestor de **TEXT_BOX** agir de forma normal, isto é, escrever texto nela.

Os diversos inputs são passados para as respectivas janelas a partir do **input handler** de cada janela que pode não existir se o utilizador quiser o funcionamento padrão da janela. Esta função recebe como argumento o elemento com que foi feita a interação, o tipo de interação, a informação a ser passada e a janela a que o elemento pertence. Como a informação pode ser de diversos tipos este parâmetro é um **void*** que depois pode ser especializada de acordo com o tipo de interação. Por exemplo, quando um elemento de uma vista de lista é pressionado, **LISTVIEW_MSG** é passado uma estrutura que contém o índice do elemento pressionado e quando o **CANVAS** é pressionado é passada a **estrutura packet**, sendo que com esta estrutura é feito depois o desenho do utilizador. O handler **retorna um booleano** que indica ao gestor de janelas se este quer que este execute o tratamento padrão do input. É aqui que interação entre diferentes elementos de uma janela pode ser feita.

No screensaver, onde surgem objetos (waffle, ovo, pedaço de bacon e copo de sumo de laranja) em movimento e a colidir (com as margens e entre si), salienta-se a **colisão pixel a pixel**. Os objetos foram propositadamente escolhidos de diferentes formas e alguns de superfícies irregulares, de modo a conseguir visualizar que, de facto, a colisão não ocorre por *bounding boxes*. Os elementos são deslocados pixel a pixel e em caso de colisão é enviado um sinal para desenhar esse frame tal como se encontra. Ou seja, os elementos nunca se sobrepõem uns aos outros (método muito complicado de implementar e pouco fiável, porque a correção da sobreposição pode gerar outras sobreposições e por aí em diante) e é possível ver todos os frames de colisão.

Após calcular a nova posição de todos os elementos, estes são desenhados sequencialmente no backbuffer, sendo incrementado um valor interno a cada elemento que representa o frame atual em que se encontra. Esta variável interna é utilizada em conjunto com um valor indicativo do número de sprites de cada objeto para criar **animações**, permitindo selecionar em cada frame qual o sprite correto a apresentar. Foi implementada uma animação para o waffle, rodando-o à medida que se move, esta rotação **depende da direção do movimento**.

A UART é utilizada para o módulo **multi_painter**, no programa existe uma checkbox *Host* que define o host da ligação, contudo isto não tem nenhum significado em termos da execução do programa. Ser host apenas avisa o programa que deve estar a espera de um **SERIAL_HELLO**, caso Host não esteja ativado, esta mensagem é ignorada. Recebido o **SERIAL_HELLO**, é enviado **SERIAL_HELLO_RESPONSE** e assim estabelecida a ligação. A partir desse momento todos os movimentos nos **SLIDERS** ou pinturas na **CANVAS** são transmitidas para o outro lado. Sendo as mensagens do tipo **SERIAL_DRAW** e **SERIAL_SLIDER** enviadas. O corpo de todas as mensagens é constituído por 8 bytes: no caso do **SERIAL_DRAW**, os primeiros 4 são a posição x e y onde desenhar (cada um ocupa 2 bytes) e os restantes 4 bytes são o delta_x e o delta_y (cada um ocupa 2 bytes); no caso do **SERIAL_SLIDER** os primeiros 4 bytes indicam a posição do **SLIDER** e os restantes 4 bytes o id do slider a ser deslocado.

A UART é utilizada para o módulo **guess_painter**, tal como no multi_painter a checkbox Host define qual será o primeiro jogador. Ao ser host apenas avisa o programa que deve estar a espera de um **SERIAL_GUESS_HELLO**, caso Host não esteja ativado, esta mensagem é ignorada. Recebido o **SERIAL_GUESS_HELLO**, é enviado **SERIAL_HELLO_GUESS_RESPONSE** e assim

estabelecida a ligação. A partir desse momento os utilizadores vão desenhado à vez, enquanto o outro jogador tenta adivinhar, se o jogador adivinhar a palavra é enviado um **SERIAL_CORRECT_GUESS** passando ser a sua vez de jogar. O jogador que está a desenhar ao mexer nos sliders envia um **SERIAL_GUESS_SLIDER** que possui uma estrutura semelhante à do outro módulo e quando fecha a aplicação envia um **SERIAL_GUESS_GOODBYE.**

Para o módulo **chatter o UART** utiliza um protocolo completamente diferente. Para começar não é feita qualquer distinção Host ou Cliente. Quando um utilizador quer mandar uma mensagem é enviado um **SERIAL_CHAT_HEADER** que contém 8 caráteres da mensagem. Caso a mensagem tenha 8 caracteres ou menos é depois enviado um **SERIAL_CHAT_END**. Avisando o programa que pode mostrar a mensagem. Caso a mensagem tenha mais de 8 caracteres, são enviado tantos pacotes **SERIAL_CHAT_CHARS**, quanto necessário até todos os caráteres serem enviados e termina na mesma com **SERIAL_CHAT_END**. O envio de um número arbitrário de caracteres é disponibilizado graças ao poder das FIFOs.

As janelas que queiram disfrutar do UART terão, aquando da sua criação, de requisitar o UART. Caso este já esteja ocupado, então a janela não é criada. Quando uma janela é fechada, liberta o UART para que outras janelas o possam utilizar. A requisição é feita a partir da função **ser_set_handler()**, assim como a libertação.

Para a comunicação com FIFOs foi implementada a estrutura de dados **queue**. Esta justifica-se pela facilidade e adequação ao funcionamento da UART. Com efeito, a transmissão de informação é feita lendo / escrevendo caráter a caráter para as FIFOs, pelo que as operações de adicionar e remover informações são facilmente realizadas pelas operações **push e pop** caraterísticas desta estrutura de dados.

Quando surge uma interrupção de **Transmitter Empty** e existem ainda carateres a enviar, é removida informação do topo da queue, se a queue estiver vazia então não é colocado nada. Ao aparecer uma interrupção de **Data Available**, é lida informação da FIFO de receção até que esteja esteja vazia, carregando-a sucessivamente num array interno, quando este array fica cheio é enviado uma mensagem para a janela que esteja a escutar, caso exista. Similarmente, numa interrupção de **Char Timeout** é lida informação da FIFO da mesma maneira.

De entre todas as janelas implementadas, há que destacar duas **file_browser** e **image_render**. O file browser para poder a listagem das pastagens recorre à biblioteca *dirent.h* e para criar recorre da função **mkdir()** do *unistd.h*. A listagem das pastas é feita com recurso a listas de vista, **LIST_VIEW**, que são dinamicamente atualizadas recorrendo à função **set_list_view_elements()**. Esta janela apresenta ainda uma **TEXT_BOX** que quando é pressionado o botão ou a tecla ENTER cria um diretório com esse nome, se o diretório for criado no ambiente de trabalho então os elementos do ambiente de trabalho são atualizados. Para fazer um renderização de imagens simples e para demonstrar o poder do elemento **IMAGE**, foi definido o formato **XPM RAW**, definido atrás. Quando um utilizador tenta abrir um ficheiro se este for ao encontro da estrutura descrita então o image render é aberto com a imagem.

No caso da implementação da renderização dos items do ambiente de trabalho foi também utilizada a biblioteca *dirent.h*. Estes são colocados num array de **desktop_ee**, que contém o seu nome

e o facto de serem ou não uma pasta, sendo este fator importante na renderização do ícone. Quando o rato desloca um dos elementos para outra posição então eles são trocados no array.

Para se obter as informações acerca do sistema operativo minix, como versão, revisão e máquina, utilizado no módulo **system_info**, como já mencionado, utiliza-se a biblioteca *utsname.h*.

No módulo de **login**, para simular a inserção de uma palavra-passe, com asteriscos, o elemento **DATA** foi fundamental e o facto do input handler das janelas permitirem dar override a funções de inserção de texto também. Isto, pois quando um utilizador pressiona num caráter um asterisco é adicionado no campo da palavra-passe enquanto a verdadeira palavra-passe a ser inserida é colocada no elemento **DATA**, como identificador *real_password*.

No módulo **calculator**, a utilização do elemento **DATA** revelou-se também muito útil, pois guarda a última operação escolhida. Tudo o que se na calculadora até à parte de realmente fazer contas é feita com strings, pois o display é feito com o elemento **TEXT**, como tal o recurso a **sscanf()** foi crucial. É a aplicação que tem o ar mais distinto de todas, graças ao facto de o elemento **BUTTON** ser altamente customizável, sendo assim foi possível fazer uma interface de utilizador que parece estilo de "vidro".

O módulo **example**, tal como o nome indica é para demonstrar a utilização de elementos, bem como a implementação de um **input handler**. Ao se pressionar o botão o fundo da janela é alterado, de forma **aleatória**, bem como a **cor das strings** que se encontram renderizadas de **forma vertical**, utilizando o elemento **TEXT.**

O algoritmo de desenho presente em todos os módulos ***painter** é relativamente simples, servindo o propósito de mostrar a utilidade do elemento **CANVAS**. Para desenhar linhas é feita uma interpolação linear entre o ponto pressionado e o **delta do rato**, resultado em linhas mais cheias. Para desenhar a pincelada são pintadas linhas verticais de comprimento **ímpar**, depois sucessivamente para cima e para baixo reduz-se o comprimento em duas unidades até que o tamanho fique 1, aí desenha-se só um pixel. Esta abordagem permite fazer uma pincelada simples e com realista, sendo também eficiente na memória. Estes módulos também colocam em evidência como diferentes elementos podem interagir, neste caso os **SLIDERS** influenciam a cor na **IMAGE**.

No módulo **chatter**, ao contrário do que acontece nas outras janelas a vista de lista, **LIST_VIEW**, anda para baixo quando as mensagens já não cabem no ecrã, sendo mais uma demonstração da adaptabilidade dos diferentes elementos às diferentes situações. Isto pois de forma geral, a vista de lista anda para o topo quando é alterada.

Nas caixas de texto, **TEXT_BOX**, em que a tecla ENTER tem um comportamento especialmente, por exemplo, no **notepad** faz new line e no **file_browser** cria uma nova pasta, é um comportamento definido também pelo input handler, pois de forma geral não agia quando recebia um ENTER.

A implementação dos diferentes interrupt handlers em assembly foi relativamente sucinta e pequena, pois não tem de se verificar valores de retorno de funções como **sys_inb** e **sys_outb**. A utilização dos comandos **IN** e **OUT** também é relativamente igual à do Minix, logo a transição não foi difícil.

De modo a imprimir texto no ecrã, foi construído um xpm com todos os símbolos da tabela ASCII desde '!' até '~'. Utiliza-se uma **fonte** monoespaçada para que todos os caracteres ocupem exatamente a mesma largura e altura. Esta abordagem permite que, para aceder a um caráter, baste fazer um *slice*

do xpm da fonte utilizando o offset respetivo (calculado retirando ao caráter a imprimir o valor do primeiro caráter definido na fonte, neste caso '!').

Para desenhar no ecrã foi utilizada a técnica de **double buffering**. Primeiramente é tudo desenhado num **backbuffer**, diferente do principal. Aquando de uma interrupção do timer, este é trocado com o **frontbuffer** e é então apresentado no ecrã.

O código encontra-se devidamente documentado utilizando *Doxygen*. Foi criado um *Doxyfile* em que foram selecionadas as opções RECURSIVE, CALL_GRAPH e INPUT.

Por fim, embora não esteja presente na versão final, ao consultar as primeiras versões desta biblioteca, poderá encontrar umas implementações de IPC. Foi um tópico muito interessante de explorar, ainda por cima porque no sistema operativo Minix têm uma abstração própria. Com isto ficou-se a conhecer os diferentes "servidores" que correm no Minix, como o **Data Server** ou **Process Manager** e a interação com eles. Por exemplo, para se obter o endpoint do próprio processo para passar a processo de utilizador é necessário obter a partir do Process Manager que não é acessível por processos de utilizador (ideia original, que não foi prosseguida por sugestão dos professores), como tal, implementar uma passagem do endpoint do driver para o processo de utilizador foi uma experiência interessante. Ainda por cima os processos de utilizador apenas podem chamar **sendrec()**, o que torna a implementação de protocolos de *ACKNOWLEDGE* algo fulcral.

Conclusões

De uma forma geral, ambos consideramos que a cadeira de LCOM, apesar de trabalhosa, foi muito recompensante. Não é uma disciplina fácil, mas por isso mesmo se torna mais desafiante e nos permite aprender novas habilidades, como a capacidade de trabalho e pesquisa autónoma, que, de outra maneira, não conseguiríamos desenvolver. Foram abordados temas interessantes e novos que, mesmo não sendo do agrado de muitos colegas, entendemos serem importantes estarem presentes no nosso percurso académico.

É também de destacar a atitude do nosso professor das aulas práticas, Pedro Silva, perante as dúvidas colocadas. Esteve sempre disposto a esclarecer-nos e discutir connosco, mesmo quando os temas saiam fora do âmbito da disciplina, e por isso estamos muito gratos.

Destaca-se sobre a cadeira os seguintes pontos negativos:

- Tendo em consideração que, previamente a LCOM, no curso não se teve nenhuma introdução a muitos dos tópicos dados na cadeira, pensamos que deveria ter existido um cuidado extra na forma como muitos dos temas são abordados. Por um lado considera-se positivo ter que pesquisar e aprender sozinhos, mas por outro gastamos muito tempo com isso. Como exemplo refere-se a questão de ser necessário, logo desde o início, ter o Linux instalado e saber trabalhar com ele para conseguir acompanhar as aulas. Na verdade, apenas uma minoria dos estudantes já tinha trabalhado com Linux e estava confortável a usá-lo. Verifica-se ainda que uma grande parte dos alunos teve problemas logo na instalação, tendo necessidade de pedir ajuda externa.
- A demora na entrega das notas respetivas aos Laboratórios e teste de Programação, que faz com que não se consiga saber, durante o semestre, como o nosso desempenho está a ser. Não é adequado apenas saber uma nota final no fim do semestre, sem qualquer feedback a meio.

Salientam-se, no entanto, os seguintes pontos positivos:

- A introdução da LCF ajudou consideravelmente o desenvolvimento dos Laboratórios, permitindo mais facilmente descobrir quais os erros do nosso código e testá-lo com os testes disponibilizados. Consideramos que este fator tenha reduzido significativamente o tempo "perdido" à procura de pequenos erros e que contribuiu para uma melhor e mais eficaz aprendizagem.
- O teste individual de programação surge como uma boa opção para distinguir os alunos que têm ou não trabalhado. Não consideramos que deva ser desafiante, visto os tópicos da cadeira serem exigentes e uma pequena distração ser o suficiente para não conseguir completar com sucesso. Mas, mantendo-se o nível de dificuldade apresentando, cobrindo apenas os fundamentais da cadeira que devem ser retidos, consideramos que seja algo a repetir no futuro.

Instruções de instalação

1. Fazer checkout do repositório
2. Aceder à pasta proj/src
3. Copiar os conteúdos da pasta *files* para /home/lcom/ , comando *cp files/* /home/lcom/*.
4. Correr o comando *lcom_conf add conf/proj*
5. Correr o comando *make*
6. Correr o comando *lcom_run proj*