Minix Calendar

Daniel Pereira da Silva - up201503212

Frederico Portugal Pinho Rocha - up201408030

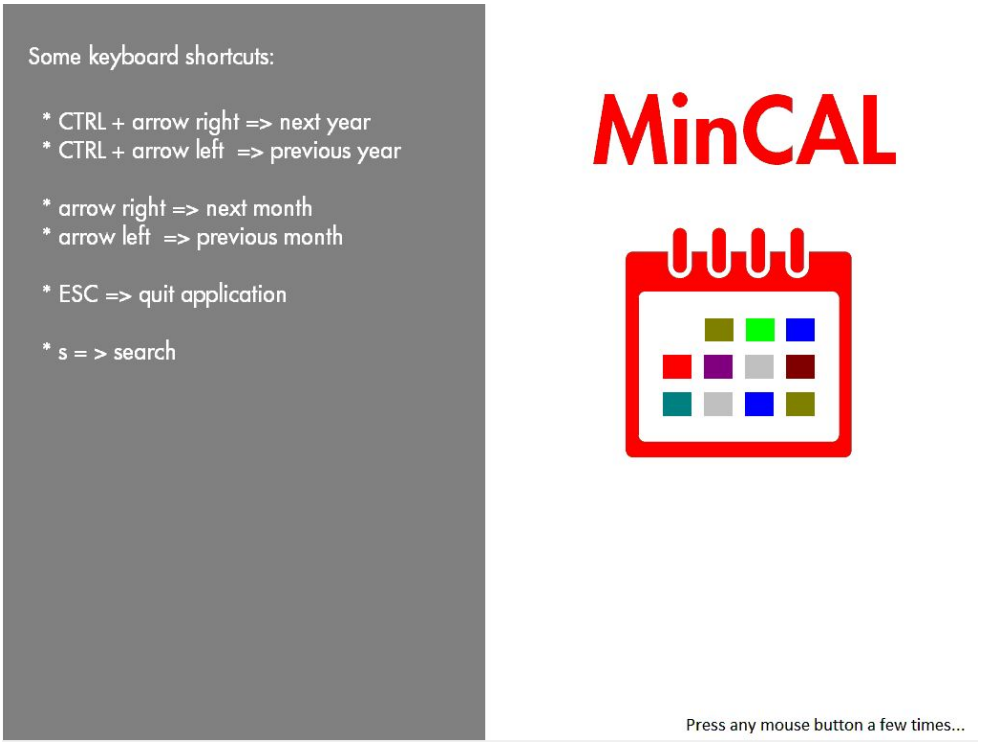
FEUP-LCOM
Turma 3MIEIC10
`MinCal`

Trabalho realizado usando a linguagem C, com o objetivo
de criar um calendário em Minix.

1. Instruções de Utilização

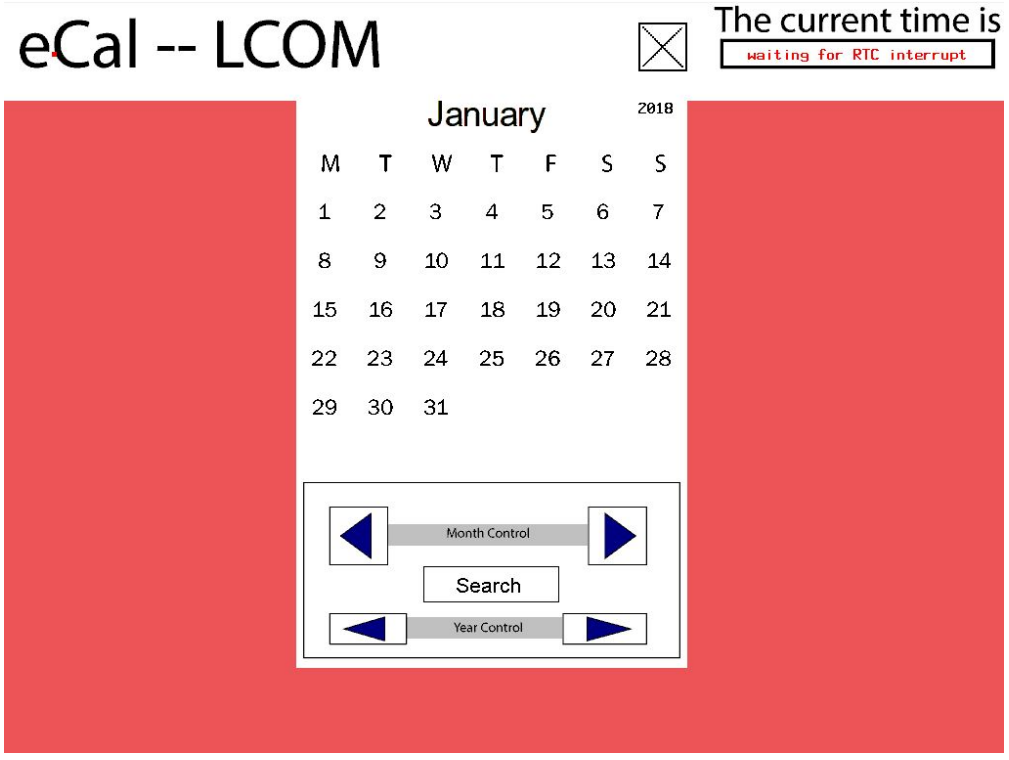
1.1 Ecrã Inicial

Ao iniciar o programa, é apresentado um ecrã inicial, mostrando o logotipo do projeto e algumas mensagens úteis para o utilizador. De seguida, passa-se para o ecrã principal. No topo do ecrã são demonstradas informações adicionais, tal como o titulo do programa, a data atual e horas, e uma 'cruz' para encerrar o programa. No centro está o calendário, e, abaixo deste encontra-se botões de navegação. É de salientar também um ponto vermelho, no canto superior esquerdo do ecrã, representativo do rato.



1.2 Navegar o Calendário

De modo a navegar o calendário, podem ser utilizadas as setas do teclado, ou os botões presentes no ecrã. A tecla da seta esquerda muda o calendário para o mês anterior, enquanto que a seta direita muda o calendário para o mês seguinte. A tecla CTRL ativa a funcionalidade de mudar o ano, em vez do mês. Com o rato, pode se aceder ao botões do ecrã e navegar no calendário, premindo o botão esquerdo do rato, ou usando a *mousewheel*.



1.3 Terminar o Programa

De modo a terminar o programa, o utilizador pode premir a tecla ESC, ou, com o rato, clicar na cruz acima do calendário.

2. Project Status

2.1 Devices

Device	Use	Interrupt
Timer	Controlling framerate	Y
Keyboard	Menu navigation + exiting	Y
Mouse	Menu navigation + exiting	Y
Video Card	Display	N
RTC	Date + current time	Y

Timer

O timer é utilizado para redesenhar o ecrã a cada interrupt do Timer0.

```
void timer0_int_handler() {
    static int tick_elapsed=0;
    if((++tick_elapsed)%1==0){
        if(landing){
            draw_landing_page();
        }else{
            rfill_screen();
            if(search){
                draw_search_box();
            }else{
                draw_main_page();
            }
            draw_cursor(cursorX,cursorY);
        }
        vg_flush();
    }
    if(tick_elapsed%60==0){
        clear_regC();
    }

    return;
}
```

Keyboard

O teclado é utilizado para navegar o calendário. Sempre que é premida uma tecla, entra-se na condição de interrupt apresentada abaixo, onde ocorrem as verificações relacionadas as teclas premidas. O teclado é também utilizado para o input de texto, servindo para procurar meses e anos especificos no calendario, uzando o formato decimal.

```
if (msg.NOTIFY_ARG & irq_kbdset) {
    //printf("handling keyboard\n");
    landing=0;
    KEY_PRESS* kp;
    kp=kbd_int_handler();
    if(kp==NULL)
        continue;
    if(search){
        if(kp->code==0x81){//W
            search=0; continue;
        }else if(kp->code==0x1c){//return/enter
            search=interpretSrchStr();
        }else{
            textInput(kp);
        }
    }else{
        if(kp->code==0x81){
            stop=1; continue;
        }
        if(kp->code==0x4d){ //right
            if(CTRL_status)
                nextYear(&cal);
            else
                nextMonth(&cal);
        }
        if(kp->code==0x4b){ //left
            if(CTRL_status)
                prevYear(&cal);
            else
                prevMonth(&cal);
        }
        if(kp->code==0x1f){ //W
            search=1;
        }
        if(kp->code==0x1d || kp->code==0x9d){ //CTRL
            if(kp->mk){
                //makecode
                CTRL_status=1;
            }else{
                CTRL_status=0;
            }
        }
    }

    free(kp);
}
```

Mouse

A funcionalidade do rato é navegar o calendário, através dos butões presentes no ecrã. Adicionalmento, pode ser usado para terminar o programa com o botão 'X'. Na imagem que se segue está as funções que tratam da colisão. O check_clicks é chamado no final do *interrupt handler* do rato.

```
int mouseInBox(unsigned tlX, unsigned tlY, unsigned brX, unsigned brY){

    return cursorX>tlX && cursorX<brX && cursorY>tlY && cursorY<brY;
}

void check_clicks(struct mouse_action_t* ma){
    if(search){
        if(mouseInBox(812, 228, 862, 278) && ma->lmb){ // exit box
            search=0;
        }
        if(mouseInBox(698, 427, 748, 463) && ma->lmb){ // ok box
            search=interpretSrchStr();
        }
        if(mouseInBox(755, 427, 853, 463) && ma->lmb){ // reset field box
            srchStr[0]='\0';
        }
    }else{
        if(mouseInBox(650, 20, 700, 70) && ma->lmb){ // exit box
            stop=1;
        }

        if(mouseInBox(335, 516, 393, 574) && ma->lmb){ // arrow left month
            prevMonth(&cal);
        }

        if(mouseInBox(600, 516, 658, 574) && ma->lmb){ // arrow right month
            nextMonth(&cal);
        }

        if(mouseInBox(335, 625, 412, 655) && ma->lmb){ // arrow left year
            prevYear(&cal);
        }

        if(mouseInBox(574, 625, 654, 655) && ma->lmb){ // arrow right year
            nextYear(&cal);
        }
        if(mouseInBox(425, 570, 574, 620) && ma->lmb){ // search box
            search=1;
        }
    }


}
```

Video Card

O modo de video é utilizado para desenhar o *background* do ecrã, através da atribuição de cores aleatórias. É também utilizado para desenhar todas as *sprites* no ecrã, através de funções próprias de leitura de ficheiros .xpm.

O VBE é inicializado em modo 0x118, com uma resolução de 1024x768 e suportando cores de 24 bits. É utilizado *doublebuffering* de modo a se poder manter uma *framerate* estável, visto que o *background* é redesenhado com valores diferentes a cada interrupt do timer.

Usa-se deteção de colisão nos botões. Não se usa animação de sprites.

```
void draw_character(char asciiCode, unsigned short x, unsigned short y, pixel_t color){
    /**
     * x and y mark the bottom left corner.
     */

    if(asciiCode<32){
        fprintf(stderr, "could not write unprintable character ASCII: %d!\n", asciiCode);
        return ;
    }

    video_info_t vi = get_vi();
    if(((short)y)-8<0 || x+13>vi.x || y>vi.y){
        fprintf(stderr, "could not write character out of screen (on %dx%d)!\n", x, y);
        return;
    }
    y-=8;
    // we've checked our args, they look good to go.

    /**
     * A small crash course on our letters mini pixmaps is in order:
     *
     * - a bit set to 1 represents a pixel to be filled with the color passed in the args.
     * - a bit set to 0 should be left untouched. Another function must set the background before.
     * - every char is 13 pixels in height, and 8 pixels in width (monospace), thus occupying 104 pixels.
     * - that's all folks!
     */

    int i, j;
    const unsigned char* a=letters[(int) asciiCode-32];
    //printf("selecting index %d\n", (int) asciiCode-32);
    for(i=y;i<y+13;i++){
        for(j=x;j<x+8;j++){
            const char bit = a[12-(i-y)] & BIT(7-(j-x));
            //printf("selecting %d -> %d got 0x%x\n", i-y, j-x, bit);
            if(bit!=0){
                setP(j, i, color);
            }
        }
    }
}
```

função do VBE para desenhar carateres

Real Time Clock

O RTC é utilizado para ler a data e as horas, e atualizando-as a cada interrupt. Estas são desenhadas no ecrã através da função de VBE draw_character() apresentada acima.

3. Estrutura do Codigo

proj

Módulo que contém a função main. Chama a função init() em e_cal se os argumentos do main forem “init”.

e_cal

Módulo responsável por inicializar o modo de video, fazer o *load* dos ficheiros xpm contidos na pasta ‘xpms’, chamar a função de desenho da *main page* e chamar o loop responsavel pelo *listening* dos devices.

utils

Módulo que contém o *device listening loop.* Contém também a função que descarrega todos os XPMs para as suas devidas structs e a função que desenha a *main page*.

read_xpm

Módulo responsável por ler ficheiros xpm, dado o filename do ficheiro ou um char** representativo do conteudo encontrado num ficheiro xpm. Contém a struct *xpm_t* onde se guarda informação relativa aos XPMs lidos de ficheiros.

month_pixmap

Módulo que contém algumas sprites xpm, que se distinguem do resto por serem constantemente chamadas desde o inicio ao fim do programa.

mouse

Módulo que trata de todas as ocorrências relacionadas com o rato. Contém funções feitas previamente no lab4, modificadas para acomodar este projeto. Contém também funções de deteção de colisões, tal como uma struct *MOUSE_ACTION* para guardar dados relativos ao rato, como posição e se certos botões estão premidos.

keyboard

Módulo que contém as funções de subscrição e tratamento das interrupções dos teclado de labs anteriores, modificadas para este projeto. Contém também uma struct *KEY_PRESS* responsável por guardar dados sobre os codigos da ultima tecla premida.

rtc

Módulo que contém informação sobre o tempo atual, guardado numa struct *rtc_time_t.* Contém funções de subscrição e tratamento de interrupções do RTC e funções de *interrupt enabling*. A função bcd_to_decimal() converte binário para decimal e foi retirada de https://stackoverflow.com/a/42340213

timer

Este módulo contém funções de tratamento e subscrição de interrupções do timer0, adaptadas de labs anteriores.

vbe

Este módulo contém duas structs com informação sobre o VBE Mode e VBE Controller, e uma função que retorna a struct relativa ao modo do VBE.

video

Módulo responsável por desenhar no ecrã. Contém funções que preenchem o *background* e que desenham XPMs, strings e characteres especificos. Contém a struct *pixel_t* usada para definir cores rgb.

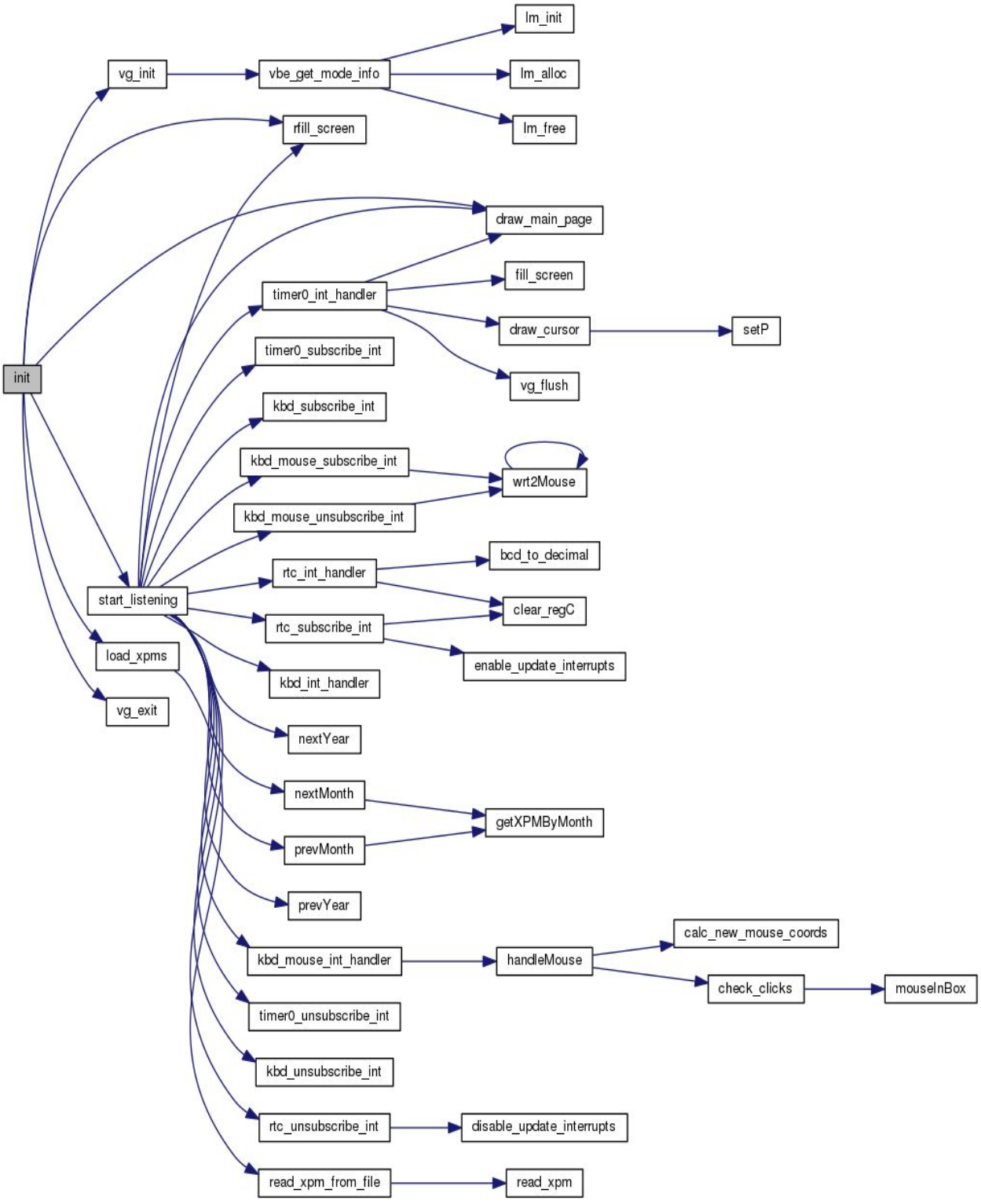
video_gr

Este módulo é responsável por iniciar o modo de video. Contém funções criadas no lab5, re-estruturadas para melhor servir este projeto, com a adição de *double buffering*.

view

Este módulo contem informação sobre o calendário, guardada numa struct *View*, e o tratamento desta mesma. Contém também funções de desenho do calendário. A função bissextile() indica se o ano em questão é ano bissexto e foi retirada de https://github.com/FEUP-MIEEC/Prog1/blob/master/Aula_Pratica_03/Problema_8.c

Function Call Graph



4. Implementation Details

Para a implementação das *sprites*, foi necessário alguma criatividade da nossa parte, visto que precisavamos de usar muitos xpms, ao ponto que o programa não compilava devido a falta de memória. De tal modo, tivemos de criar funções que lêssem diretamente os ficheiros xpm guardados em pastas separadas.

A deteção de colisões foi tratada utilizando coordenadas estáticas para os objetos, neste caso apenas a caixa com o X de exit. As coordenadas do rato são atualizadas a cada interrupção. Na struct *MOUSE_ACTION* são guardados os valores relativos ao movimentos do rato nas coordenadas x e y e estes são incrementados as variáveis cursorX e cursorY, que existem independentemente do rato.

5. Conclusões

Inicialmente, planeavamos implementar mais uma vista para o calendário, onde se apresentava apenas os 7 dias da semana horizontalmente, com as horas no eixo vertical. Porém, com o passar do tempo apercebemo-nos que seria mais sensato não seguir essa abordagem como uma prioridade.

Planeavamos inclusivamente, como melhoria futura, a adição de um sistema de alertas, que utilizaria o alarme do RTC e o rato, para selecionar o dia ao qual se quer adicionar um lembrete. Poderia também usar-se o input do teclado para dar um titulo e descrição ao dito lembrete.