Ponto de Controle 03

Sistemas Embarcados

Leonardo Brandão Borges de Freitas
RA: 14/0025197
Universidade de Brasília, Campus Gama
Brasília, Brasil
leonardobbfga@gmail.com

Ygor Pereira Borgonove
RA: 14/0166408
Universidade de Brasília, Campus Gama
Brasília Brasil
ygor.borgonove@hotmail.com

Resumo – ***Desenvolvimento de um pequeno equipamento constituído por um display para interfaceamento gráfico, um HD para armazenamento de arquivos e um Raspberry Pi para embarcar procedimentos de upload e downloads de dados para uma nuvem física, offline e com alcance local de roteiamento, em que o usuário so terá acesso com autenticação de login.***

Palavras-chave — Skullpi, Nuvem, Raspberry Pi, Download, Upload, Segurança, Sistema Embarcado.

I. Introdução

Armazenamento e transferência de dados são palavras chaves em nosso cotidiano. Com o avanço da tecnologia digital, geramos cada vez mais material a ser compartilhado e distribuído. Para tanto, expandir o a capacidade de servidores e acelerar o compartilhamento online de arquivos é foco de muitos desenvolvedores como a Google, Youtube, Amazon e vários outros.

Figura 1: Logo Google Drive

Em meio a tanta demanda criou-se o serviço de nuvem , em que são fornecidos servidores para armazenamentos de arquivos em banco de dados, backup, dentre outros serviços. Atualmente os maiores serviços de núvem prestados são: Google Drive, Dropbox, One Drive e iCloud. Existem, basicamente, três tipos de nuvens: IaaS, PaaS, SaaS.

IaaS é a mais básica das três em que o usuário aluga a infraestrutura de TI, servidores e máquinas virtuais, são as mais fornecidas pelas empresas citadas anteriormente.

PaaS, plataforma como serviço, é usada para serviços de computação em nuvem que fornecem um ambiente para desenvolvimento, testes, gerenciamento, dentre outras aplicações para empresas. Foi criada para auxiliar os desenvolvedores a criar aplicativos.

SaaS, software como serviço, basicamente é uma forma de distribuição e comercialização de software. Ao utilizar o SaaS as empresas contratantes não precisam se preocupar com instalação, manutenção e atualização de hardware ou software, só precisaria de uma conexão com a internet para gerenciar o que estivesse na núvem. Muito utilizado para serviço de assinaturas.

Em meio globalizado a segurança de arquivos está cada vez mais delicada e com alto custo. Muitas vezes nos deparamos com situações de vazamentos de arquivos pessoais em que o usuário paga caro por um produto ou serviço para armazenamento de seus materiais e devido a falhas na segurança, alguns mal intencionados conseguem ter acesso aos seus arquivos pessoais podendo causar danos irreparáveis ao usuário. Um grande exemplo dessa quebra de segurança foi o que aconteceu com usuários da Apple, pessoas famosas tiveram fotos particulares expostas na internet recentemente. Uma simples solução para esse

problema seria uma nuvem offline com caráter físico que só poderia ser acessada em curto alcance e por poucos usuários cadastrados, evitando assim problemas de quebra de segurança e vazamento de dados.

II. Desenvolvimento

Para tanto, será construído um sistema de nuvem com um software chamado Skullpi codificado em linguagem C, desenvolvido no sistema operacional Linux Raspbian, embarcado em um Raspberry Pi 3 Model B, interfaceado por um display de 7" touchscreen, com armazenamento em um cartão micro SD de 16 GB, com adaptação para HD externo caso mais memoria seja demandada.

O funcionamento do Skullpi tem 4 etapas associadas (A, B, C e D):

A. Conexão:

O Skullpi é um aparelho eletrônico pessoal, contudo, sua utilidade pode ser compartilhada de forma segura. Devido a isso o proprietário deve definir uma senha para a conexão wireless, a qual ficará visível a todo momento, no entanto só terão acesso a nuvem quem souber a senha do wi-fi.

Com a conexão estabelecida o usuário pode executar o programa SkullMain, o qual o apresentará o menu principal do user interface.

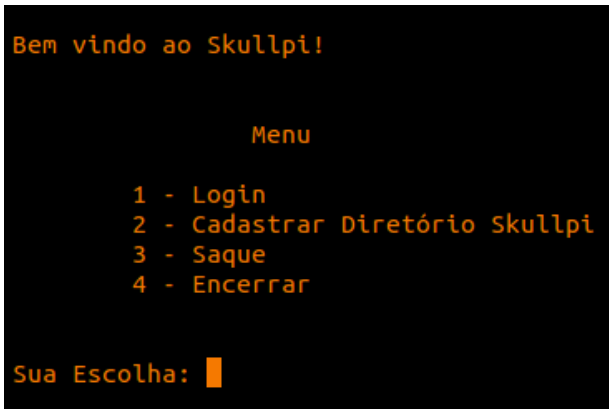

Figura 2– Menu User Interface (SkullMain)

B. Login:

O segundo passo é a etapa de login, que possui duas instâncias: cadastro e autenticação.

O procedimento de cadastro pede ao usuário o endereço da sua pasta pessoal (/home/nome_de_usuário) e uma senha a ser criada, os quais serão seu passaporte de acesso para o Skullpi. Essa etapa também é responsável por criar dois diretórios: um na máquina do usuário (/home/nome_de_usuário/Skullpi), a qual será utilizada para o download e upload de arquivos para a nuvem, e outra dentro do hd do raspberry (/home/pi/Skullpi/nome_de_usuário) a qual armazenará os arquivos enviados. Desta forma, o processo de cadastro linka o login com apenas um diretório para gerenciamento de dados, conferindo maior segurança e organização.

A autenticação é a fase necessária para acionar o menu de Gerenciamento do programa. Ela pedirá o nome de usuário e a senha e se for verificado a existência desses dados na sessão de cadastrados no raspberry o usuário passa para a etapa de gerenciamento de arquivos em seus diretório.

C. Gerenciamento:

Aqui o usuário comum tem permissões de gerenciamento apenas em seu diretório, sendo que o super usuário pode gerenciar todos os diretórios e arquivos presentes em seu Skullpi.

De forma simples e intuitiva o menu possui 3 opções: Upload, Download e Encerrar.

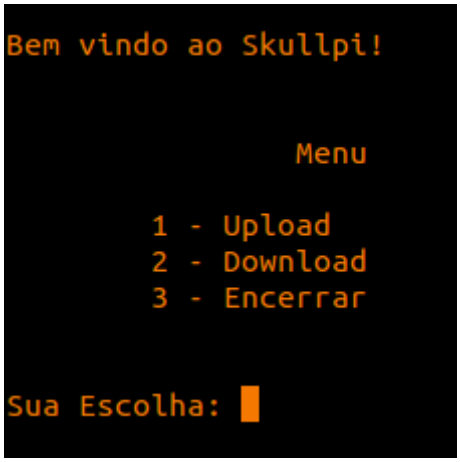

Figura 3 – menu de Gerenciamento (login efetivado)

Para o upload de dados, deve-se colocar os arquivos, que deseja armazenar na nuvem, dentro da pasta Skullpi

(/home/nome_de_usuário/Skullpi) e selecionar a primeira opção do menu. Se o procedimento ocorrer de forma correta, os dados são armazenados no HD do raspberry e a pasta Skullpi passa por um processo de "refresh" o qual apaga os arquivos que já foram enviados, deixando a pasta vazia.

Ao selecionar a segunda opção do menu o usuário faz o download de todos os seus arquivos contidos no seu diretório do Skullpi do raspberry. Esses arquivos aparecem na pasta Skullpi na maquina do usuário.

A ultima opção encerra o login e retorna o programa para o menu principal e além disso, aqui o usuário também define se seu diretório ficará em modo de Saque ou não.

D. Saque:

Ao selecionar o Saque no menu principal o programa pede a pasta pessoal do usuário (/home/nome_de_usuário) e também qual o diretório que será "sacado", ou seja, qualquer pessoa que tenha acesso ao wi-fi do raspberry, pode utilizar dessa opção. No entanto, o diretório cadastrado só entra nesse modo se seu proprietário autorizar. Para tanto, o Skullpi só mostrará na interface gráfica as pastas que tem o modo saque ativado.

Um PIN diferente é gerado e mostrado no display toda vez que essa opção é escolhida e o download dos arquivos só é efetivado inserindo o PIN corretamente, se isso ocorrer, o diretório escolhido para download surgirá em sua pasta pessoal.

Portanto, o modo saque tem a mesma função do Download, no entanto, o login não é necessário e apenas diretórios autorizadas podem ser baixados nesse modo.

E. Comunicação e transferência de dados

A comunicação entre a máquina do usuário e o Raspberry embarcado com o software Skullpi é realizada pela a leitura e escrita de arquivos através das funções **open()**, **read()**, **write()** e **close()** presentes nas bibliotecas **fcntl.h** e **unistd.h** que seguem as normas POSIX (IEEE 1003). Também utiliza-se a função **system()** da biblioteca **stdlib.h** para execução de algumas subrotinas em linguagem Unix.

A transferência de dados se dá através do protocolo de rede criptografado **Secure Shell** ou **SSH,** o qual utiliza modelo Cliente-Servidor para comunicação e algumas funções como **SCP** (Secure Copy) para a transferência de dados de uma máquina para a outra.

III. Descrição de Software

O funcionamento do Skullpi é dividido em dois programas, um que fica compilado e rodando no Raspberry e outro que fica como executável na máquina do usuário.

A. Skullpi Raspbian

Esse é o programa que está em constante execução dentro do Raspberry. Na versão que se encontra, esse software é basicamente constituído por dois processos paralelos que executam scripts em linguagem Unix para bash. O processo pai lida com a compactação e descompactação de arquivos .tar que são enviados pelo usuário e o processo filho verifica novos cadastramentos. Através do tunelamento de 3 comandos Unix (**ls**, **grep** e **wc**) o software cria arquivos verificadores, os quais indicam ao programa o que fazer: Se devem descompactar um compilado novo que chegou, ou se devem armazenar um novo pedido de login, por exemplo. Além disso, com a leitura e escrita em arquivos o Skullpi executa as alterações necessárias nos arquivos de cadastros de novos logins.

Obs.: O programa em C, junto com os demais scripts necessários para a execução do programa estarão disponíveis no git e o link estará nas referências.

B. SkullpiMain:

O Programa principal de interface com o usuário apresenta as opções de menu para que esse possa se cadastrar, ou logar no gerenciamento de seu diretório no sistema Skullpi.

```c
int menu(){

    int user_input = 0;

    if(flag_login == 0){

        puts("\n\t\tMenu\t\t\n");
        puts("\t1 - Login\t");
        puts("\t2 - Cadastrar Diretório Skullpi\t");
        puts("\t3 - Saque\t");
        puts("\t4 - Encerrar\t\n\n");

        printf("Sua Escolha: ");
        scanf("%d", &user_input);

        while(user_input < 1 || user_input > 4){
            printf("\nEntrada invalida, escolha outra: ");
            scanf("%d", &user_input);
        }

        if(user_input == 1) return 1;
        if(user_input == 2) return 2;
        if(user_input == 3) return 3;
        if(user_input == 4) return 4;

    }


    if(flag_login == 1){

        printf("\nBem vindo %s\n", user_adress);
        puts("\n\t\tMenu\t\t\n");
        puts("\t1 - Upload\t");
        puts("\t2 - Download\t");
        puts("\t3 - Encerrar\t\n\n");

        printf("Sua Escolha: ");
        scanf("%d", &user_input);

        while(user_input < 1 || user_input > 3){
            printf("\nEntrada invalida, escolha outra: ");
            scanf("%d", &user_input);
        }

        if(user_input == 1) return 5;
        if(user_input == 2) return 6;
        if(user_input == 3) return 4;

    }

}
```

Código 3.0 – Função do Menu;

O menu é uma função do tipo inteiro que retorna para a função principal (main) o resultado da escolha do usuário e sua visualização depende diretamente de uma variável interia global chamada **flag_login**, a qual indica se o usuário está logado ou não com seu diretório Skullpi.

```c
void create(){

    char buffer[50];
    char buffer2[50];

        printf("\n\tColoque o endereço da sua pasta de usuário /home/");
        scanf("%s",buffer);
        sprintf(user_adress,"/home/%s", buffer);

        printf("\tcrie uma Senha: ");
        scanf("%s",user_pass);

        system("clear");

        printf("\nUsuário: %s\n", buffer);
        printf("Senha: %s\n\n", user_pass);

        //criando e dando permissão para o arquivo de novo login
        sprintf(buffer2, "echo %s %s > /tmp/login.txt", buffer, user_pass);
        system(buffer2);
        system("chmod 777 /tmp/login.txt");

        //enviando para o raspberry e removendo da maquina do usuário
        system("sshpass -p \"leonard7\" scp /tmp/login.txt pi@192.168.42.1:/tmp");
        system("rm /tmp/login.txt");
```

Código 3.1 – Função de cadastro

A subrotina **create()** é uma função do tipo void que quando executada realiza o cadastramento de um novo usuário. Este deve inserir qual o nome de sua pasta principal (/home/nome_de_usuário) e a criação de uma senha, possibilitando o funcionamento das demais subrotinas do programa **SkullMain**. Além disso, através do protocolo **ssh** essa função envia um arquivo txt (login.txt) contendo o usuário e a senha do novo cadastro realizado para o aparato **Skullpi** que o armazena no arquivo **cadastrados.txt** no software que está rodando no Raspberry, para futuramente ser requisitado em procedimento de login.

```c
void login(){

    char buffer[50];
    char buffer2[50];
    char autenticador[100];

        printf("\n\tUsuário: ");
        scanf("%s",buffer);
        sprintf(nome_pasta,"%s", buffer);
        sprintf(user_adress,"/home/%s", buffer);

        printf("\tSua Senha: ");
        scanf("%s",user_pass);
        puts("\n");

        system("clear");

        //super usuário
        if(!(strcmp(buffer, "leonardo")) && !(strcmp(user_pass, "leobbf"))) flag_login = 1;

        else{

            //comparar arquivo cadastrados.txt (raspberry) com user_adress e user_pass

            sprintf(buffer2,"%s %s\n", buffer, user_pass);

            system("sshpass -p \"leonard7\" scp pi@192.168.42.1:/home/pi/cadastrados.txt /tmp");

            fd2 = fopen("/tmp/cadastrados.txt", "r");
            fseek(fd2, 0, SEEK_SET);
            do{

                fgets(autenticador, 100, fd2);

                if(strcmp(buffer2, autenticador) == 0){
                    flag_login = 1;
                }


            }while(!feof(fd2));

            if(flag_login == 0){
                puts("usuario e senha incorretos ou não cadastrado!");

            }

             fclose(fd2);

        }
}
```

Código 3.2 – Função para login

A função **login()** é do tipo void e realiza um procedimento de confirmação, o qual permite acesso ao menu de gerenciamento do diretório dentro do **Skullpi**. Através da função **strcmp()** o programa compara a entrada inserida pelo usuário (usuário e senha) com o arquivo **cadastrados.txt,** que é "puxado" através do protocolo **ssh** do **Skullpi**,

verificando linha por linha ate o final do arquivo. A **flag_login** se torna 1 se a comparação for verdadeira, ou seja, se o nome de usuário e senha estiverem cadastrados e corretos, fazendo com que a função **menu()** apresente na interface o menu de gerenciamento (figura 2.2).

```c
void upload(){

    char *buffer;


    puts("iniciando upload...");
    sprintf(buffer, "cd %s && mv Skullpi %s", user_adress, nome_pasta);
    system(buffer);
    sprintf(buffer, "cd %s && tar -zcf pac.tar %s",user_adress, nome_pasta);
    system(buffer);
    sprintf(buffer, "sshpass -p \"leonard7\" scp %s/pac.tar pi@192.168.42.1:/home/pi/Skullpi", user_adress);
    system(buffer);
    sprintf(buffer, "rm %s/pac.tar", user_adress);
    system(buffer);
    sprintf(buffer, "rm -R %s/%s",user_adress, nome_pasta);
    system(buffer);
    sprintf(buffer, "cd %s && mkdir Skullpi", user_adress);
    system(buffer);
    sprintf(buffer, "chmod 777 %s/Skullpi", user_adress);
    system(buffer);
    puts("upload terminado...");

}
```

Código 3.3 – Função de upload de arquivos

A função **upload()** do tipo void executa um sequência de chamadas **system()** para executar comandos diretamente no terminal pela linguam **Unix**. Primeiramente ela renomeia a pasta **Skullpi** com o nome de usuário inserido no login, para que quando enviada para o diretório **Skullpi** do raspberry essa seja designada com o próprio nome de usuário e diferenciável das demais pastas de outros usuários. Logo em seguida a pasta é compactada e enviada via **ssh** para a nuvem. Depois a **upload()** exclui o compactado e faz o procedimento de "refresh" na pasta **Skullpi** da maquina do usuário, limpando-a.

```c
int main(){

    int flag_menu;

    signal(SIGINT, encerrar);
    system("clear");
    puts("\nBem vindo ao Skullpi!\n");

    while(1){

        flag_menu = menu();

        switch(flag_menu){

            case 1:

                system("clear");
                puts("Login...");
                login();

            break;

            case 2:

                system("clear");
                puts("Create...");
                create();

            break;

            case 3:
                system("clear");
                puts("Saque...");
                //saque();

            break;

            case 4:
                encerrar();

            break;

            case 5:
                upload();

            break;

            case 6:
                download();

            break;


        }

    }


return 0;
}
```

Código 3.4 – Função main

A função **main()** do software executa apenas uma loop infinito (while) com estrutura de decisão (Switch/case) para realizar a chamada das funções em relação à escolha do usuário na função **menu()**. Além disso, através da diretiva **signal()** ela atribui o sinal de cancelamento (Ctrl + C) à subrotina **encerrar()** que é uma função do tipo void que apenas trata o sinal fechando os descritores de arquivos que poderiam estar abertos.

```c
void encerrar(){

    puts("\nEncerrando programa...\n");
    close(fd);
    //fclose(fd2);
    exit(0);

}
```

Código 3.5 – Função de encerramento

Para o funcionamento de todas as funções utilizadas pelo software algumas bibliotecas e variáveis globais para manipulação de arquivos estão presentes no header do código. De acordo com o código

3.6.

Obs.: todo o código será disponibilizado no git junto ao relatório e o link estará nas referências.

```
1  //Main User Skullpi
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <fcntl.h>
6  #include <unistd.h>
7  #include <sys/types.h>
8  #include <pthread.h>
9  #include <signal.h>
10 #include <time.h>
11
12 char user_adress[50];
13 char user_pass[50];
14 char nome_pasta[50];
15
16 int flag_login = 0;
17
18 int fd;
19 FILE *fd2;
```

Código 3.6 – bibliotecas e variáveis globais

IV. Descrição de Hardware

O hardware que será utilizado será o Raspberry Pi 3 e a uma tela touchscreen de 7'' desenvolvida pela empresa que fabrica o Raspberry. Essa tela tem uma resolução de 800x600 e acoplada a ela existe um shield que facilita a comunicação entre o Raspberry e a tela. Para ligá-la bastam dois jumpers, VCC e GND comum entre o Rasperry e a tela e um cabo DSI que faz a comunicação entre a tela e o Raspberry. A tela e o Raspberry puxam juntos, em média, de 1 a 1,5 Ampères. Será utilizado também um computador para fazer as transferências de arquivos.

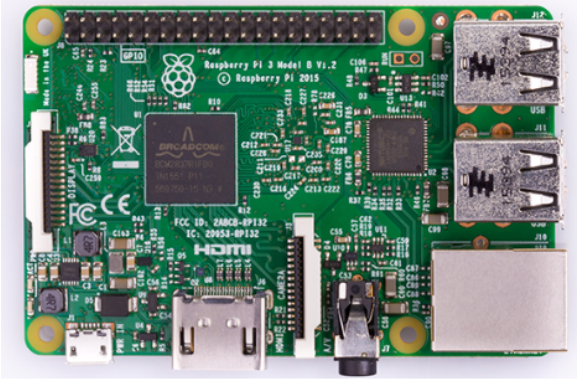

Figura 4: Raspberry Pi 3 Model B

Figura 5: Tela touchscreen 7"

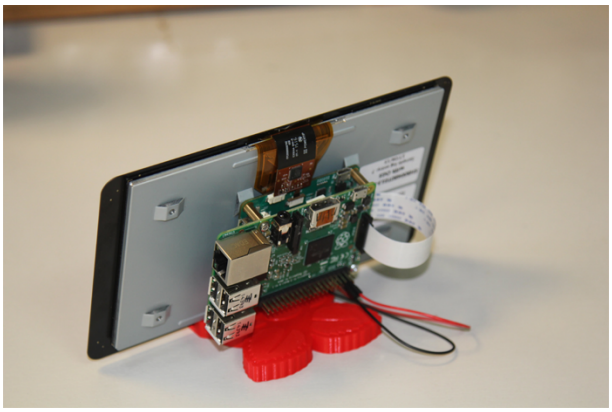

Figura 6: Tela e Raspberry Pi 3 acopladas

V. Resultados

No ponto que está o projeto, ainda temos 2/5 da ideia geral de desenvolvimento para adaptar ao software. As funções de **download** e **saque** precisam ser arrumadas, pois não estão desempenhando 100% de sua capacidade. Além disso, a versão atual do Skullpi não consegue receber envios simultâneos de arquivos, ou seja, não há como dois usuários gerenciarem seus diretórios no Raspberry ao mesmo tempo. Para tanto, criaremos um algoritmo baseado em **thread** para que cada usuário utilize uma thread para gerenciamento de seu diretório no Skullpi. Outro ponto a ser otimizado é a ideia de substituir as funções com arquivos verificadores por funções que recebam sinais, simplificando e acelerando o processamento do software Skullpi.

VI. Conclusão

Conclui-se que mesmo que esteja faltando 3/5 do projeto como um todo, este já está bem encaminhado faltando apenas algumas funções serem atualizadas/ reformuladas e uma otimização do código em geral. Será estudada também uma maneira de fazer uma interface gráfica mais trivial para o usuário.

Referências

[1]https://azure.microsoft.com/pt-br/overview/what-is-cloud-computing/

[2] https://www.salesforce.com/br/saas/

[3]https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

- Skullpi Raspbian: https://github.com/leobbf/Sistemas_Embarcados/tree/master/Skullpi%20Raspbian

- Skullpi Main: https://github.com/leobbf/Sistemas_Embarcados/blob/master/Skullpi%20Main.c