Instituto Superior
de Engenharia
Politécnico de Coimbra

CTeSP de Tecnologias e Programação de Sistemas de Informação

Gestão Automatizada de Utilizadores e Tarefas em Ambiente Unix/Linux com Bash e C

17/06/2025

Autores:	Filipe Jerónimo	Diogo Moreira	Luís Simões
Curso:	TPSI		

Índice

Índice de Imagens

Introdução

Este manual tem como objetivo guiar a instalação e configuração do sistema operativo **Zorin OS 17.2** dentro de uma máquina virtual utilizando o software **VirtualBox**. A virtualização permite testar e utilizar diferentes sistemas sem afetar a máquina física, proporcionando um ambiente seguro e isolado para aprendizagem e experimentação.

No contexto do projeto de **Sistemas Operativos**, este ambiente foi utilizado como base para desenvolver um conjunto de ferramentas que exploram conceitos essenciais como:

- Gestão de utilizadores e permissões
- Automação de tarefas administrativas com **scripts Bash**
- Programação de baixo nível em **linguagem C**, incluindo:
    - Comunicação entre processos com pipe
    - Tratamento de sinais
    - Simulação de escalonamento de processos

O uso do Zorin OS 17.2 (baseado em Ubuntu) ofereceu uma interface acessível e compatível com os comandos Linux padrão, tornando-se ideal para o desenvolvimento e testes realizados ao longo deste trabalho.

Requisitos do Sistema

Requisitos para o Host

- Processador: Intel/AMD com suporte a virtualização (VT-x/AMD-V)
- Memória RAM: Mínimo 4GB (Recomendado 8GB)
- Armazenamento: Mínimo 20GB de espaço livre
- Sistema Operativo: Windows, macOS ou Linux

Requisitos para a Máquina Virtual

- Memória RAM: 2GB (Recomendado 4GB)
- Espaço em disco: 15GB
- Tipo de sistema: Linux (Baseado em Debian)

Download e Instalação da VirtualBox

1. Aceda ao site oficial da VirtualBox: https://www.virtualbox.org
2. Faça o download da versão correspondente ao seu sistema operativo.
3. Execute o instalador e siga as instruções na tela.
4. Após a instalação, abra o VirtualBox.

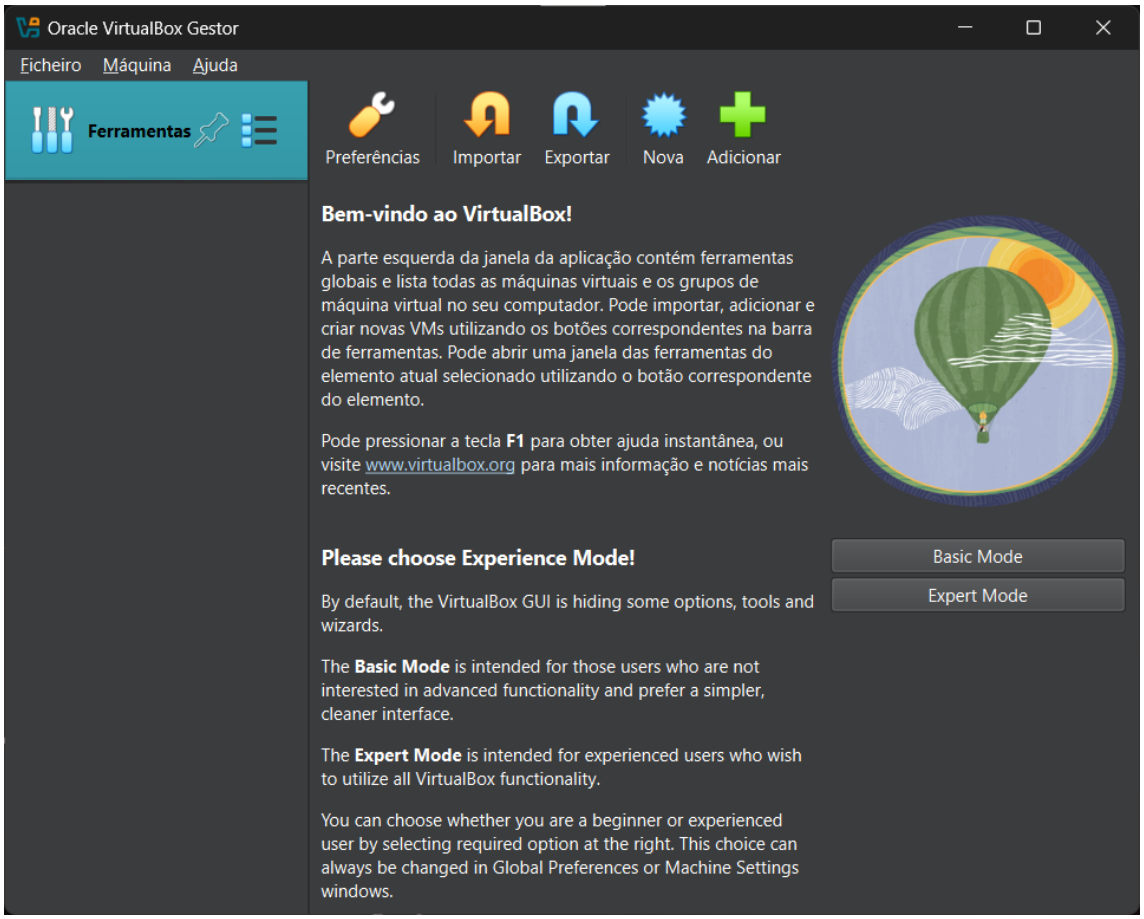

Imagem 1 – VirtualBox

Download do ISO do Zorin OS 17.2

1. Acesse o site oficial do Zorin OS: https://zorin.com
2. Faça a transferência da versão 17.2 em formato ISO.
3. Guarde o ficheiro em um local acessível para a instalação.

Imagem 2 - Iso

Configuração da Máquina Virtual

1. No VirtualBox, clique em **Novo**.

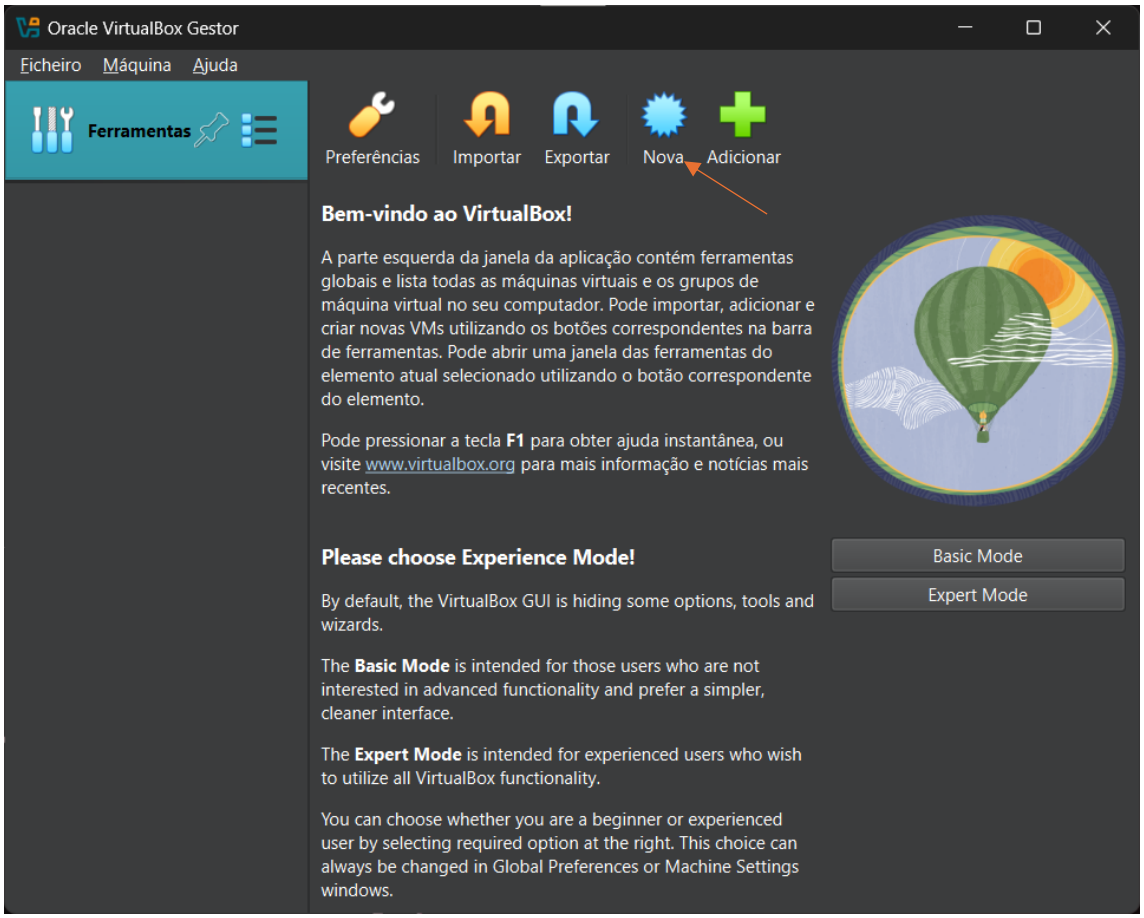

Imagem 3 - Botão novo

2. Defina um nome para a máquina virtual (ex.: "Zorin OS") e selecione o ISSO que fez transferência.

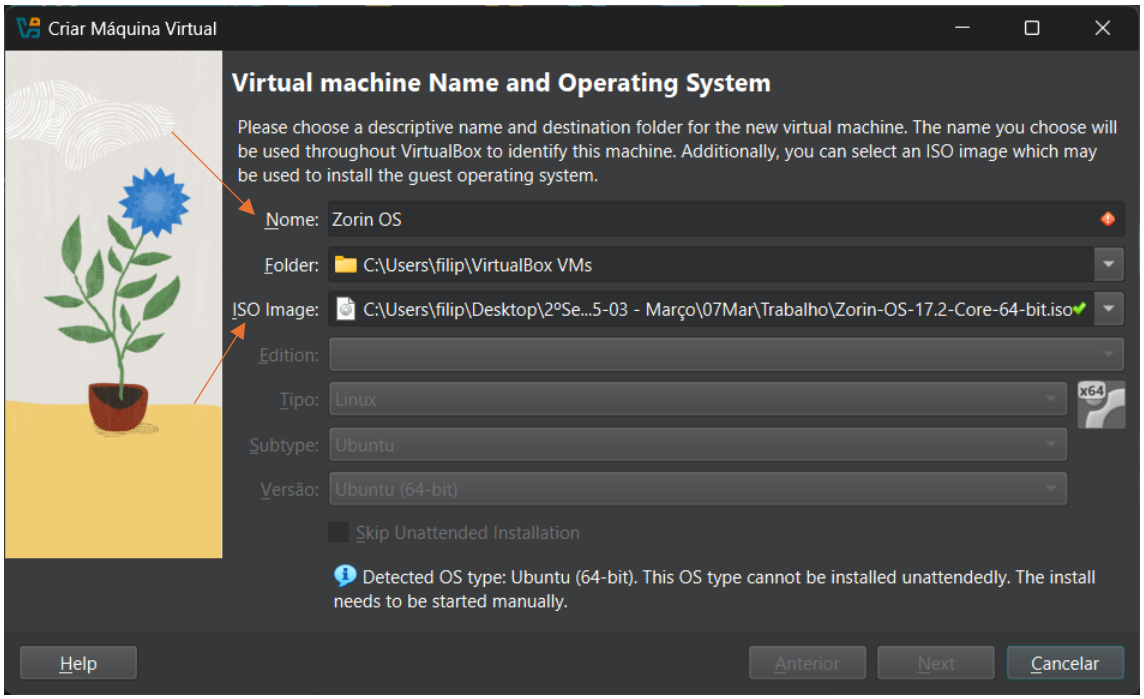

Imagem 4 - Definição do nome e selecionar o ISO

3. Defina a memória RAM (recomendado: 4GB).

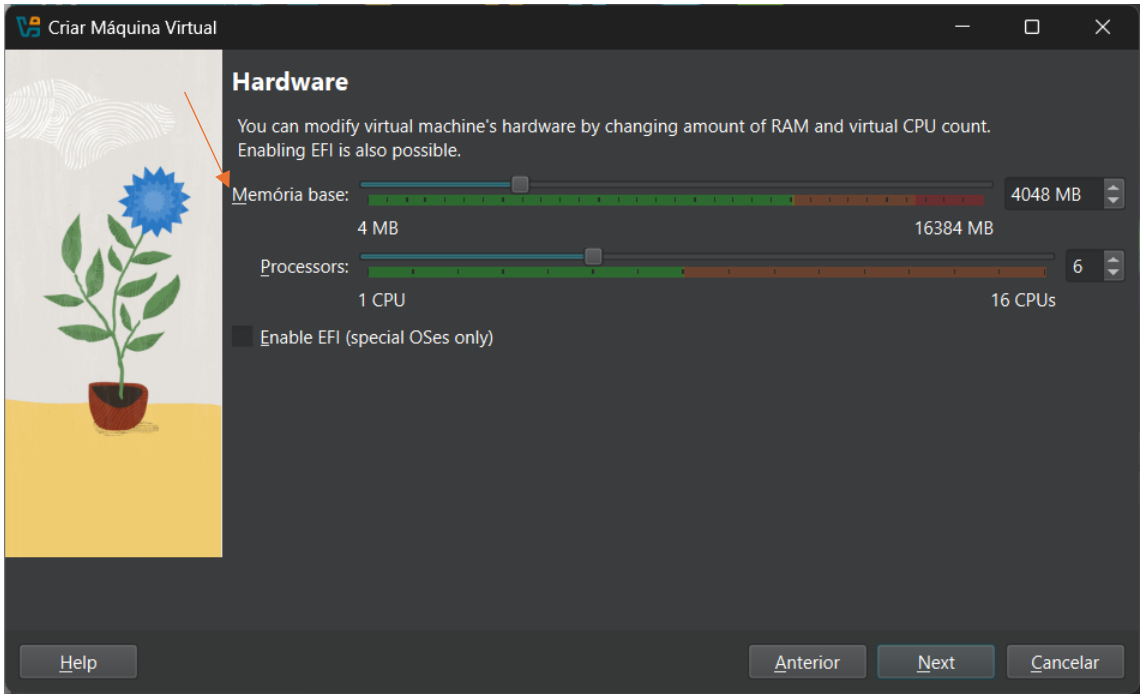

Imagem 5 - Memória Ram

4. Crie um disco rígido virtual do tipo **VDI**(Virtual Hard Disk), aloque dinamicamente e defina 15GB.

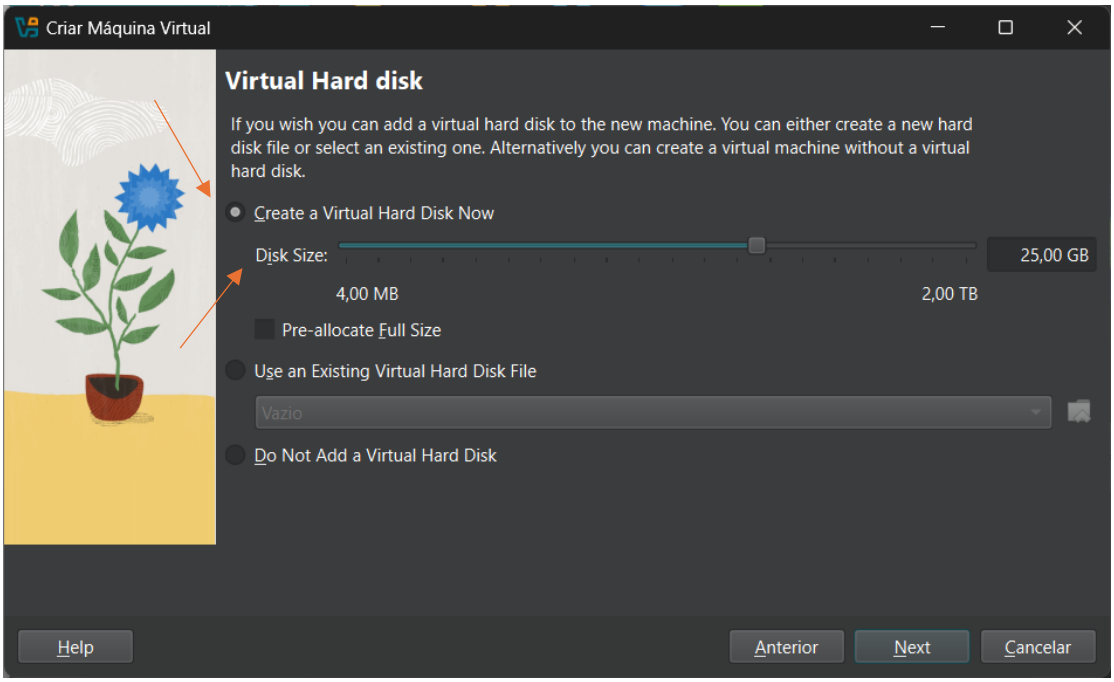

Imagem 6 - VDI (Virtual Hard Disk)

Instalação do Zorin OS 17.2

1. Inicie a máquina virtual e selecione a opção **Instalar Zorin OS**.

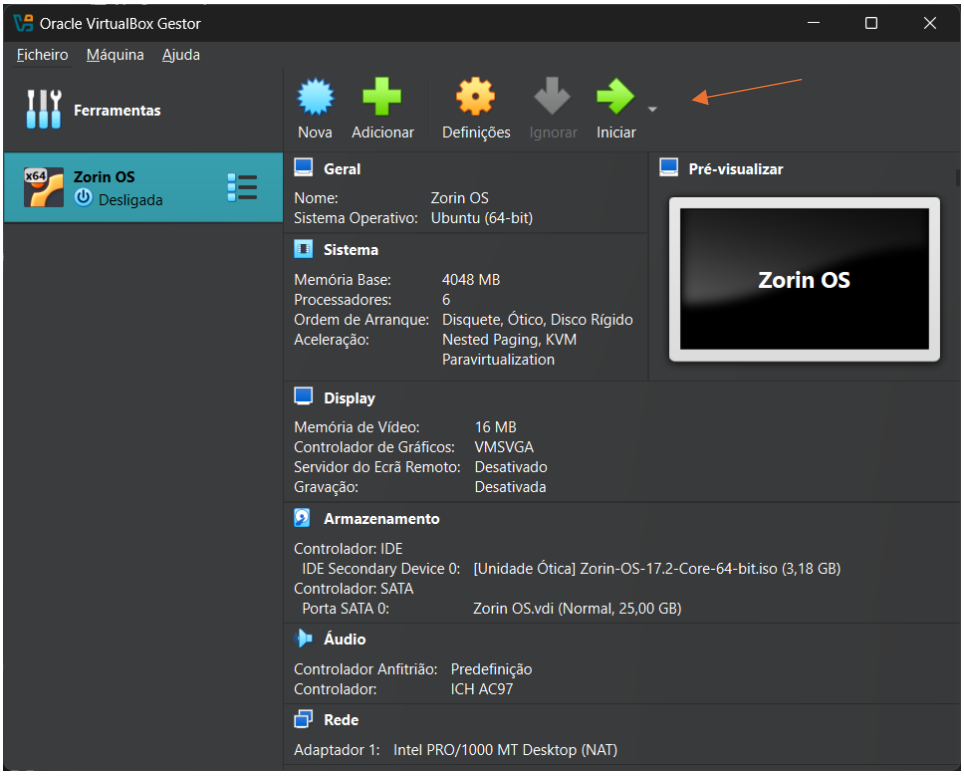

Imagem 7 - Iniciar Zorin OS

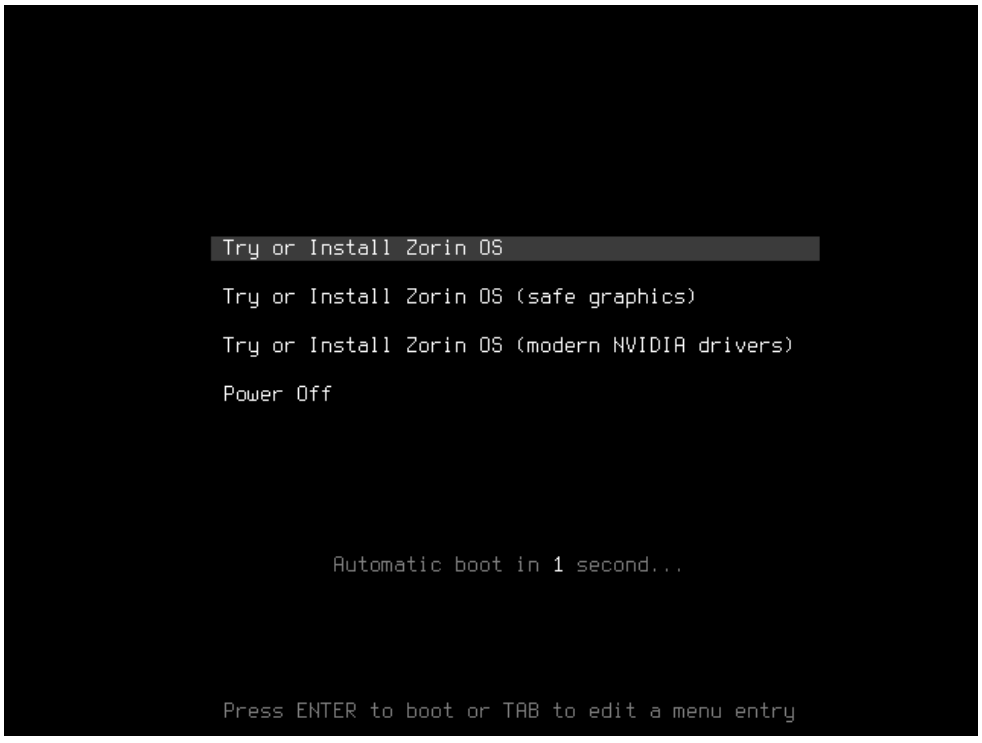

Imagem 8 - Instalar Zorin OS

2. Siga as instruções do assistente de instalação:
    - Escolha do idioma.

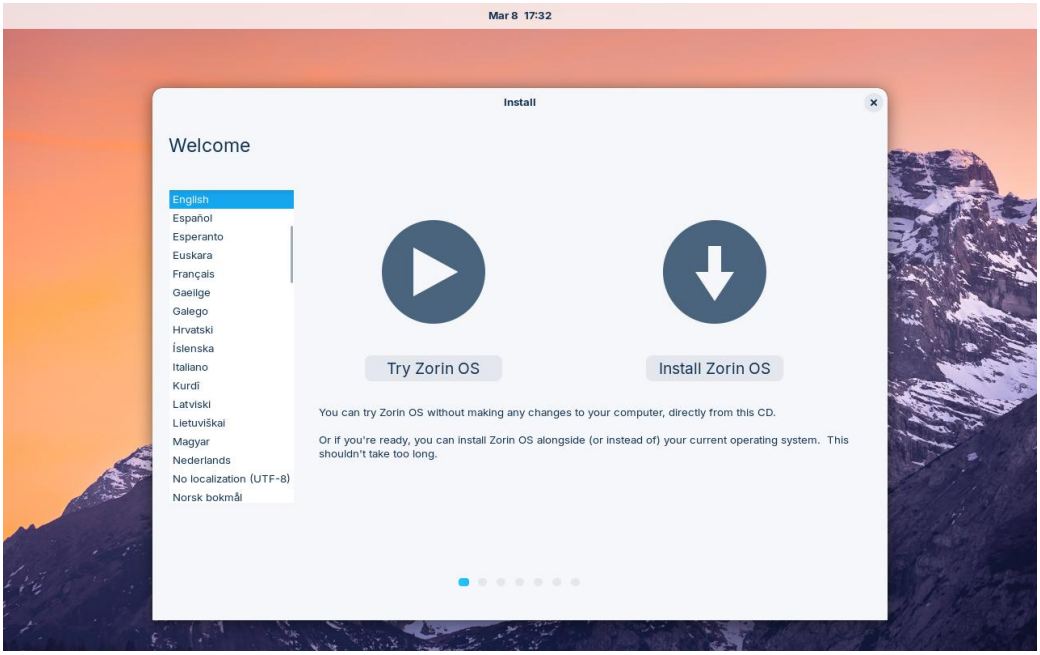

Imagem 9 - Escolha Idioma 1

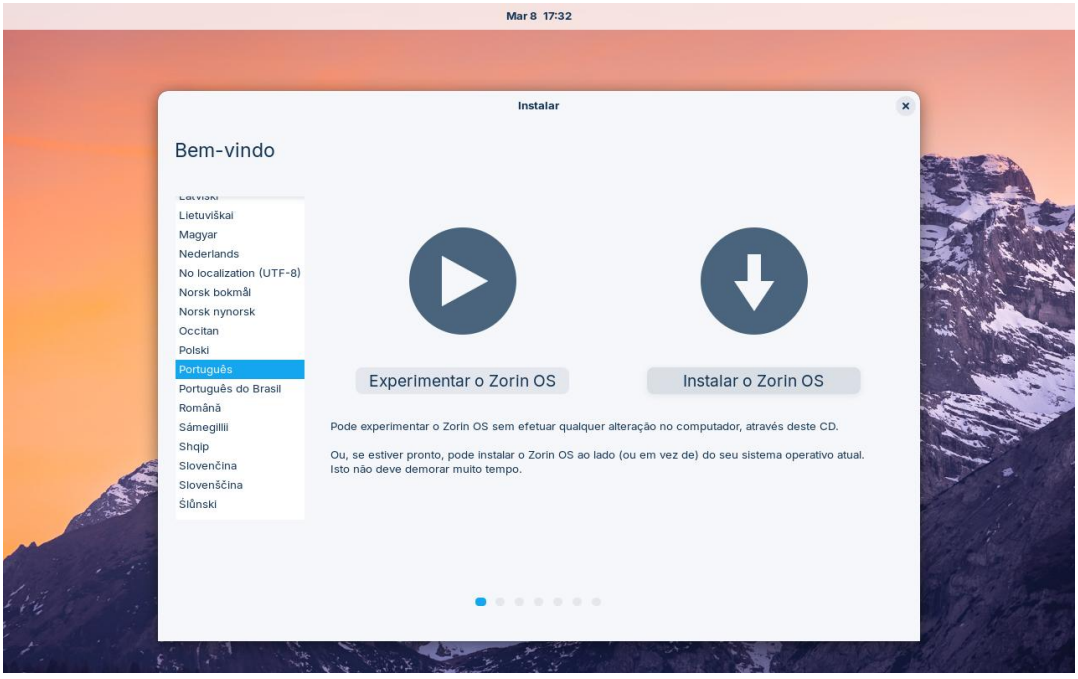

Imagem 10 - Escolha Idioma 2

- Escolha do layout do teclado.

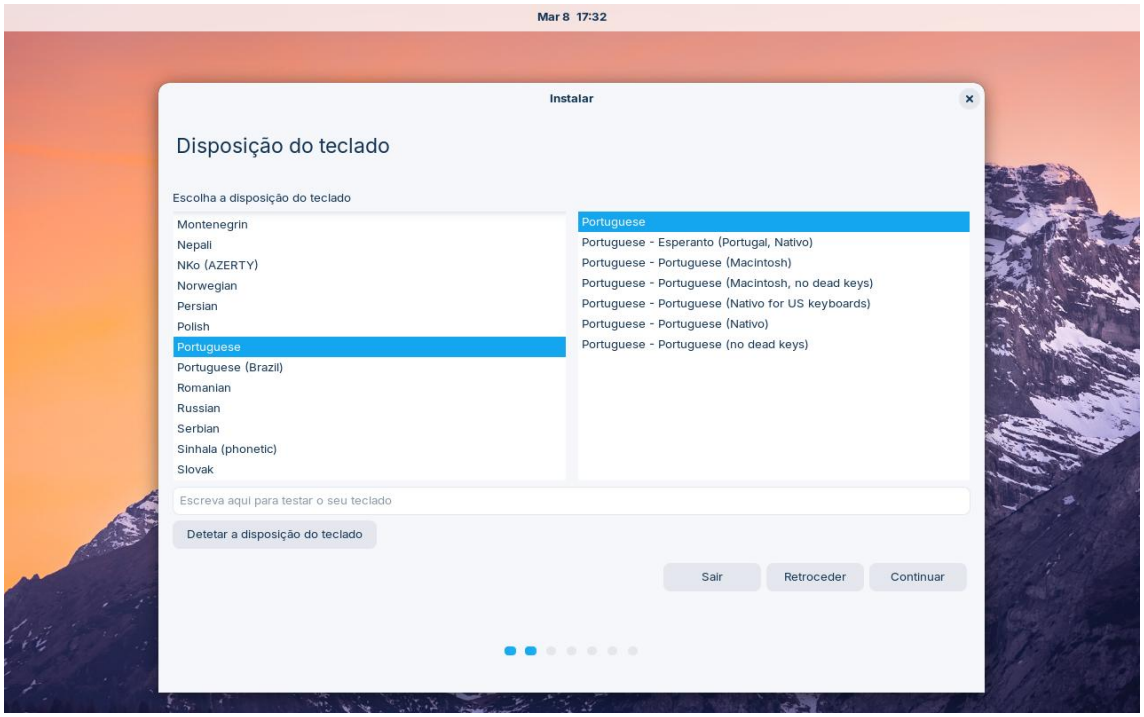

Imagem 11 - Layout do teclado

- Atualizações e outros Softwares

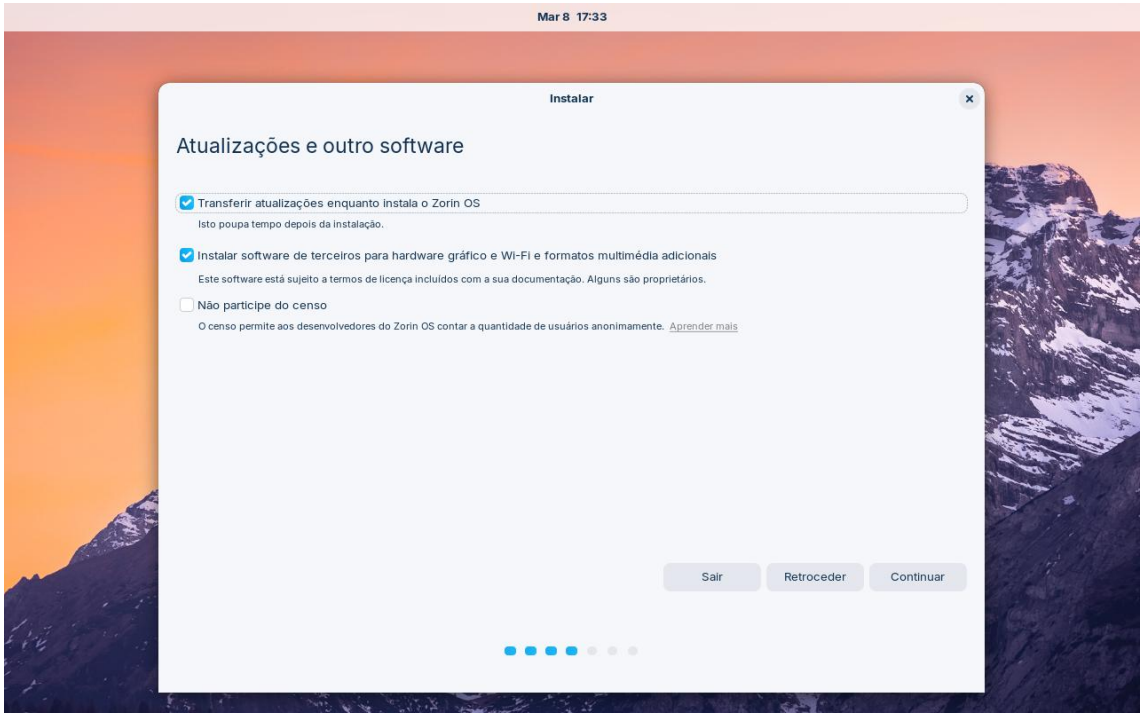

Imagem 12 - Atualizações e Outros Softwares

- Tipo de Instalação

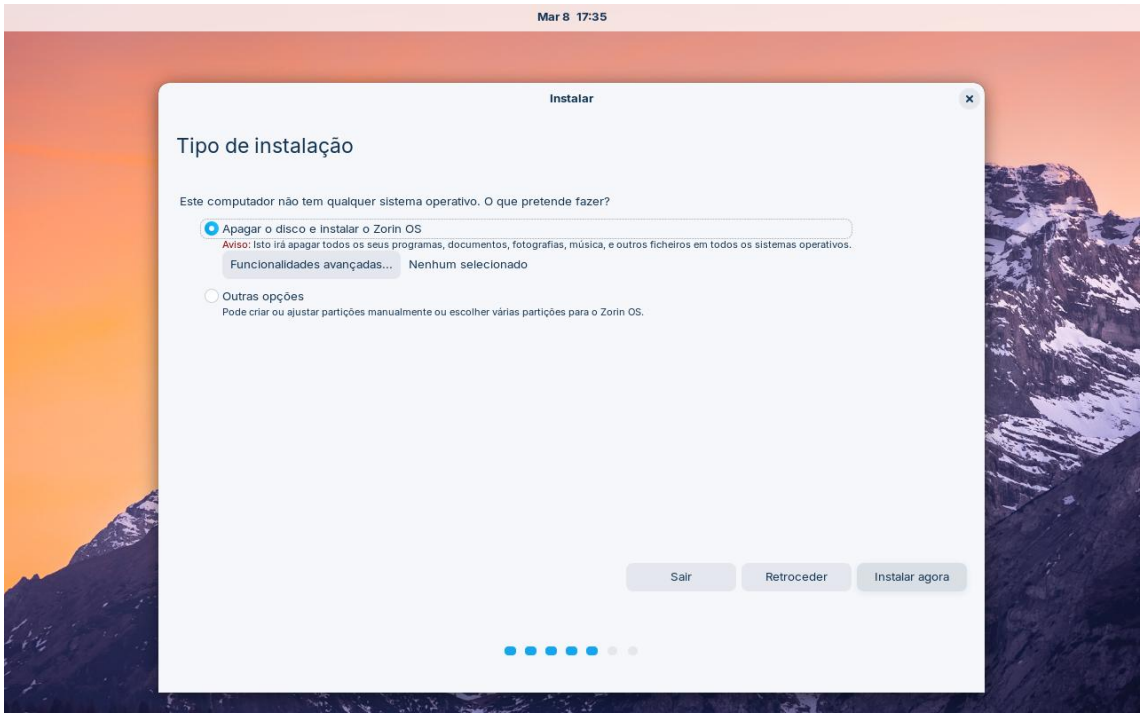

Imagem 13 - Tipo de Instalação parte 1

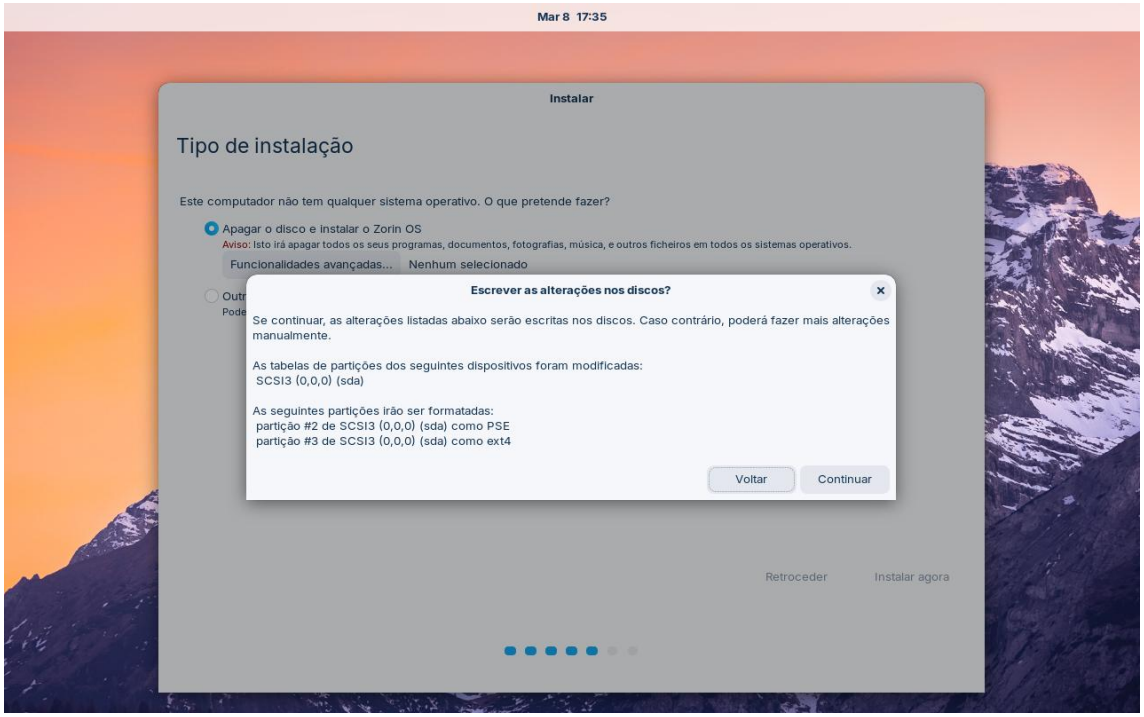

Imagem 14 - Tipo de Instalação parte 2

- Configure o fuso horário.

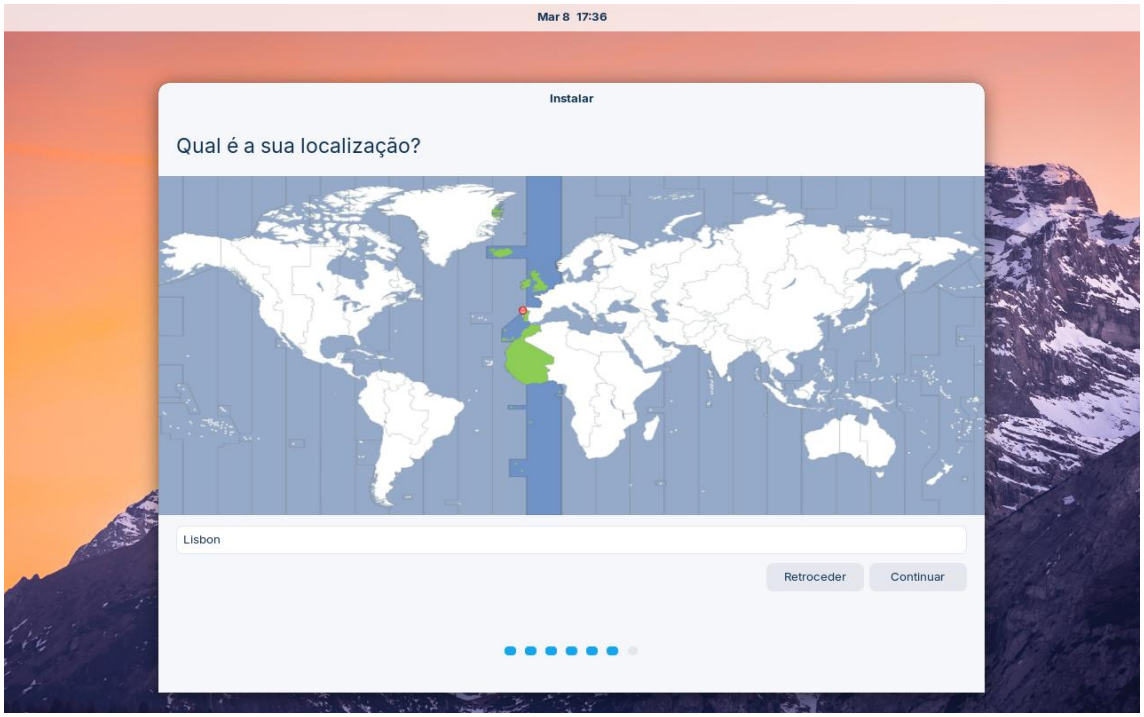

Imagem 15 - Configuração do Fuso Horário

- Defina nome de utilizador e password.

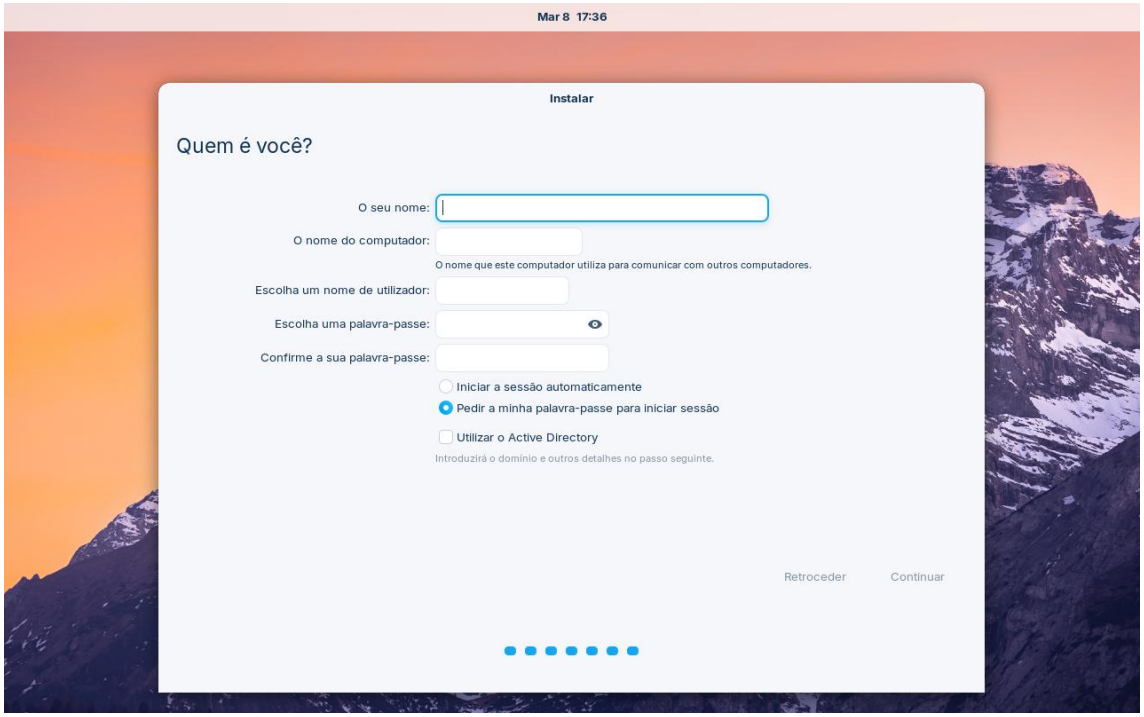

Imagem 16 - Definição do nome de utilizador e password parte 1

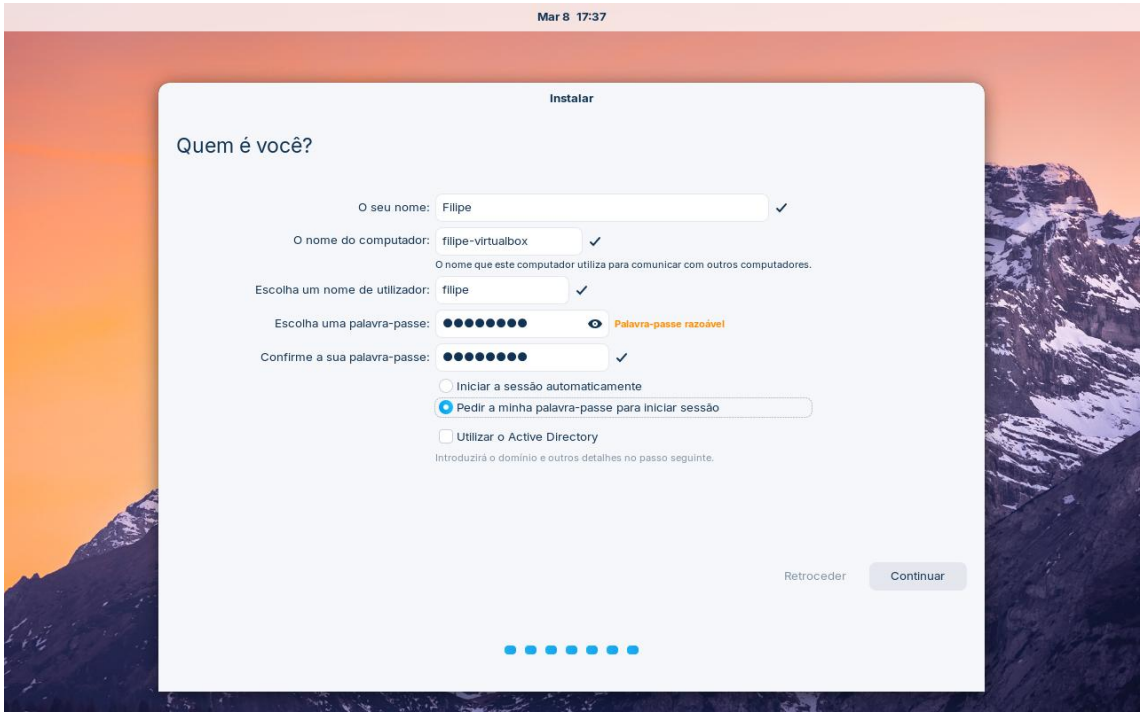

Imagem 17 - Definição do nome de utilizador e password parte 2

- Conclusão da Instalação, demorou 30mins, mas depende da internet e do computador.

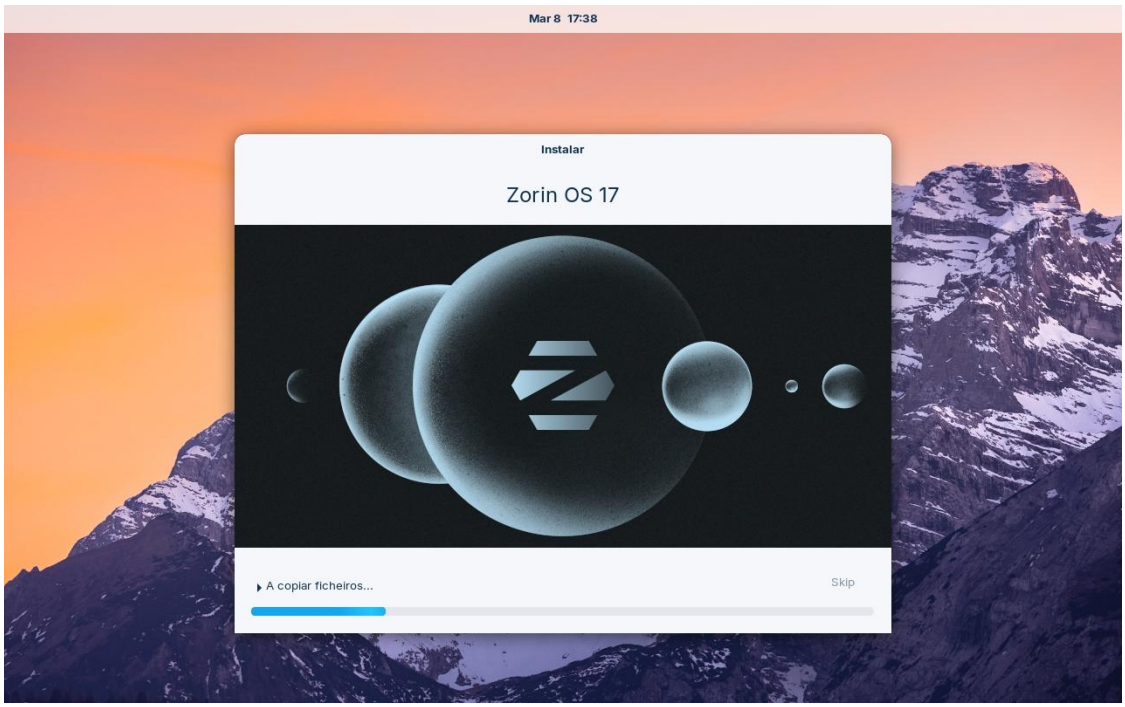

Imagem 18 - Concluindo a Instalação

3. Após a instalação, reinicie a máquina.

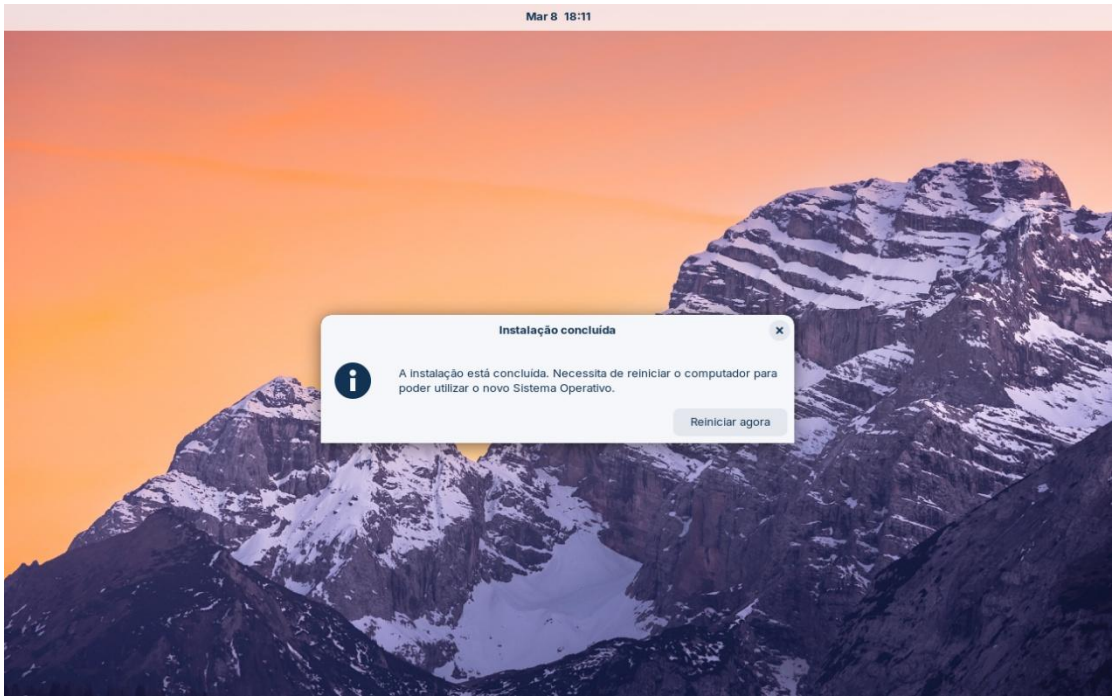

Imagem 19 - Instalação Concluida e Reinício do "Computador

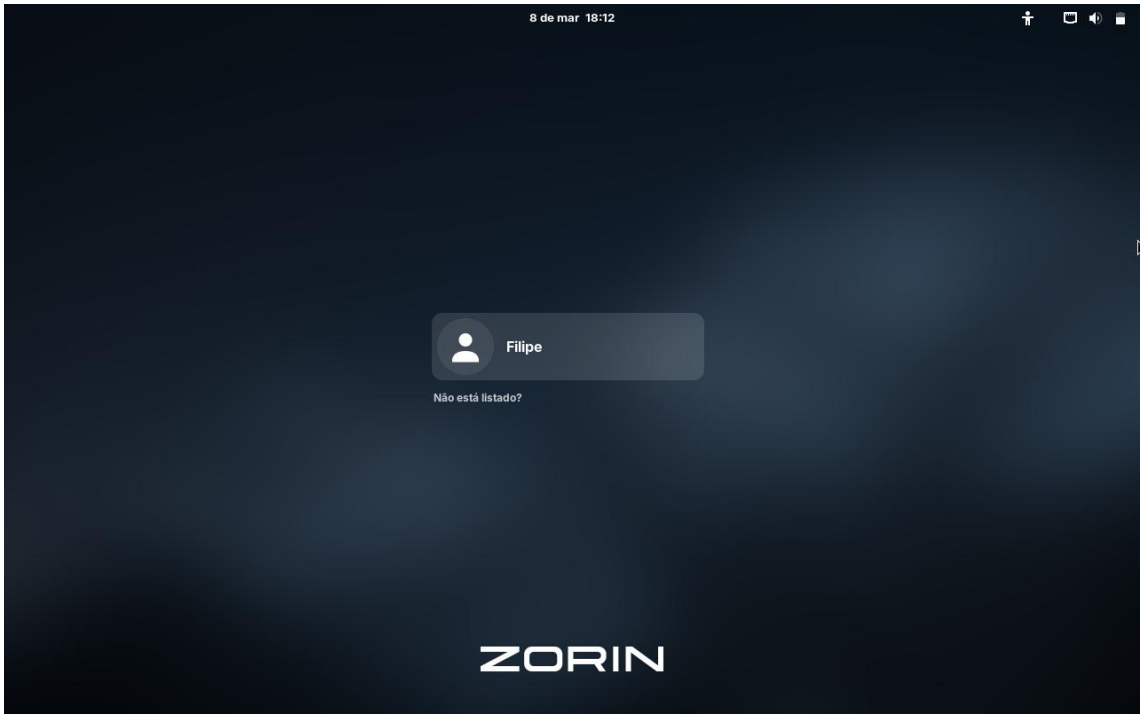

Imagem 20 - Zorion Reiniciado

Configuração/Atualização Pós-Instalação

1. Atualize o sistema:

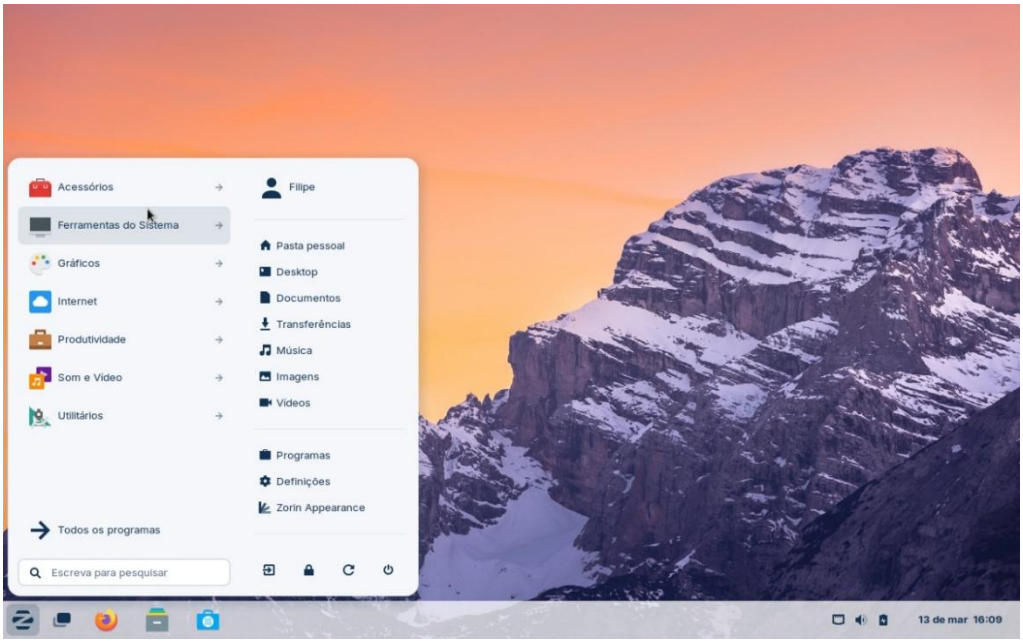

Imagem 21 - Ferramentas do Sistema

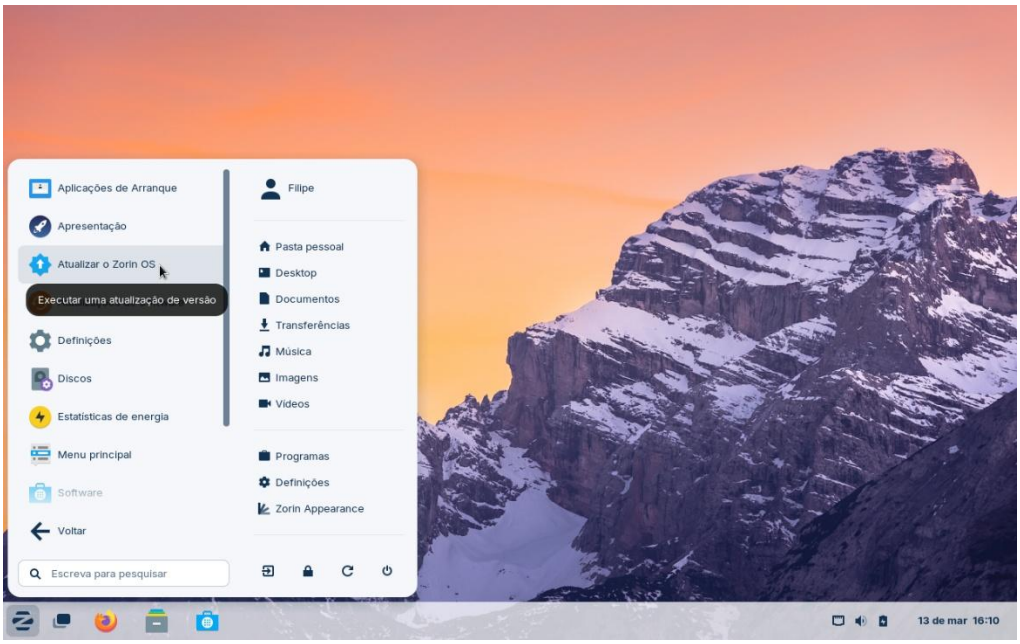

Imagem 22 - Atualizar o Zorion OS

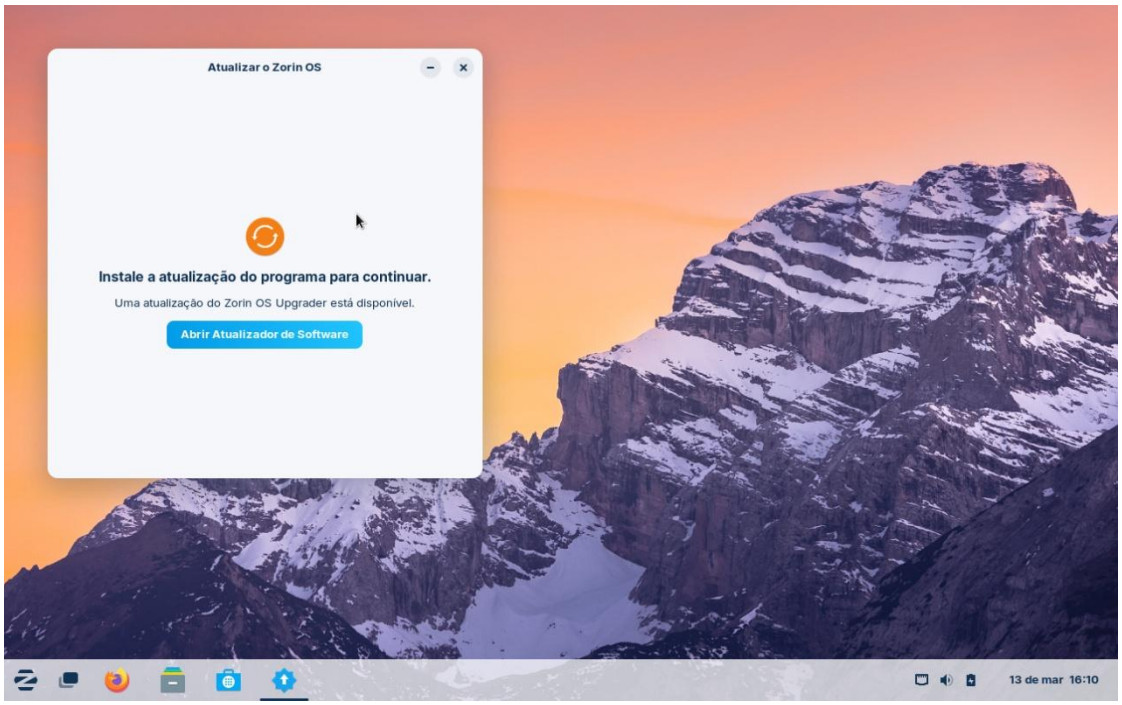

Imagem 23 - Instalar a atualização

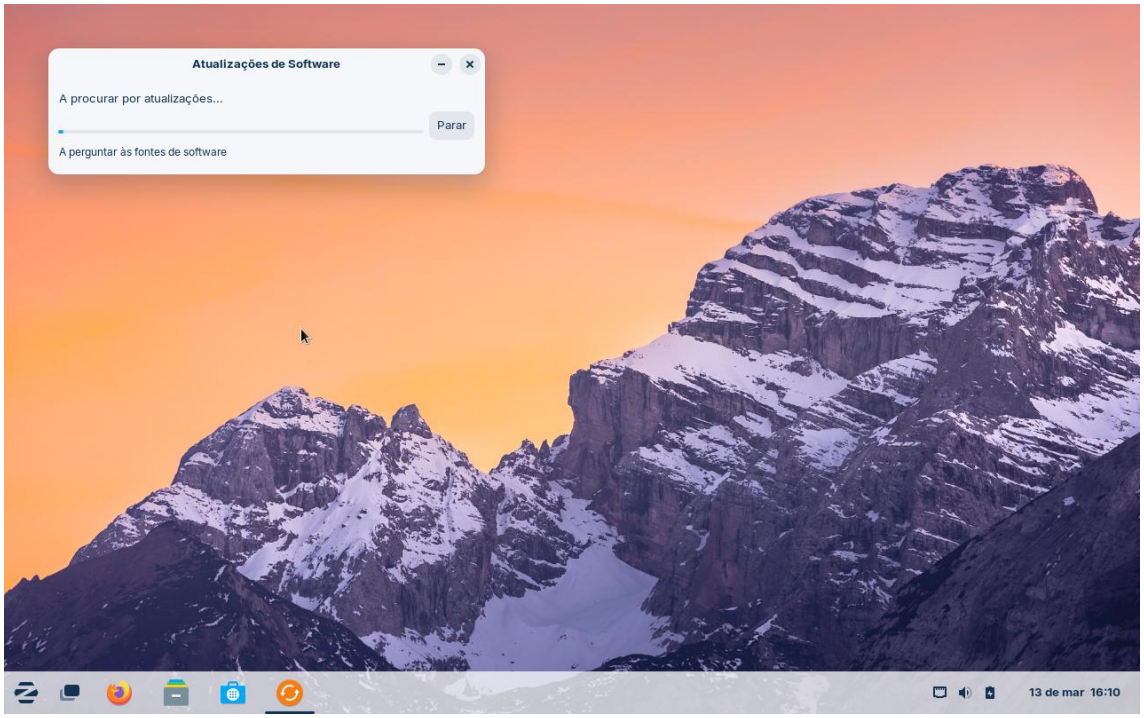

Imagem 24 - Executar Atualização

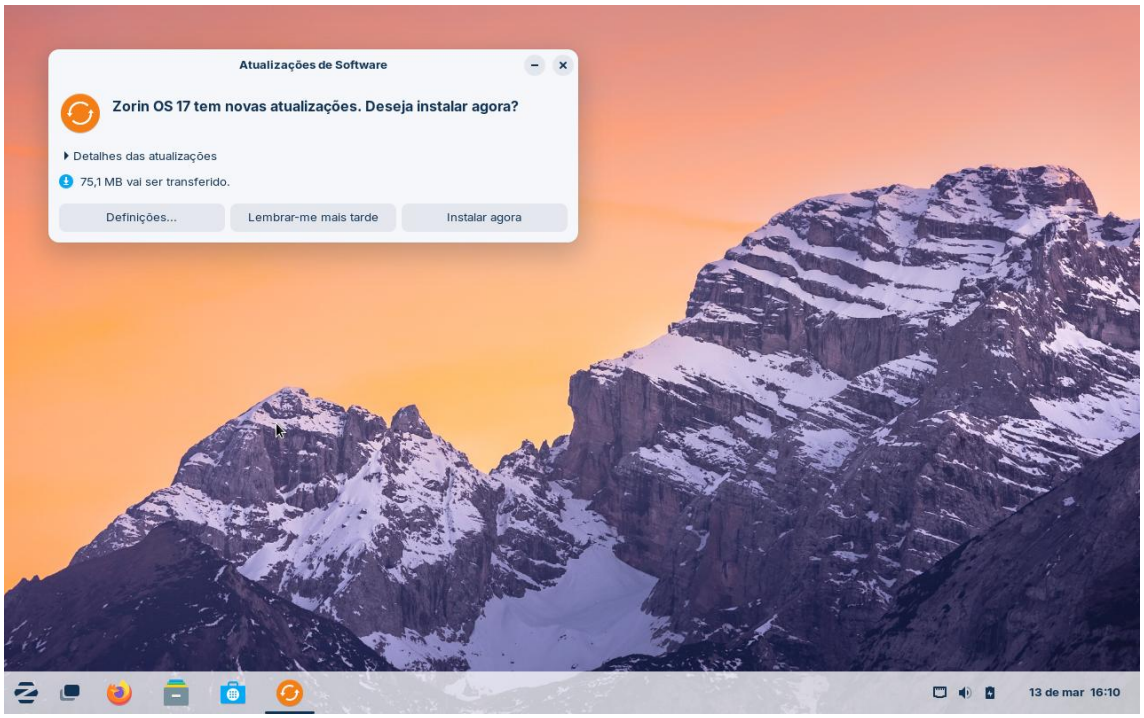

Imagem 25 – Atualização

2. Reinicie a máquina virtual para aplicar as configurações.

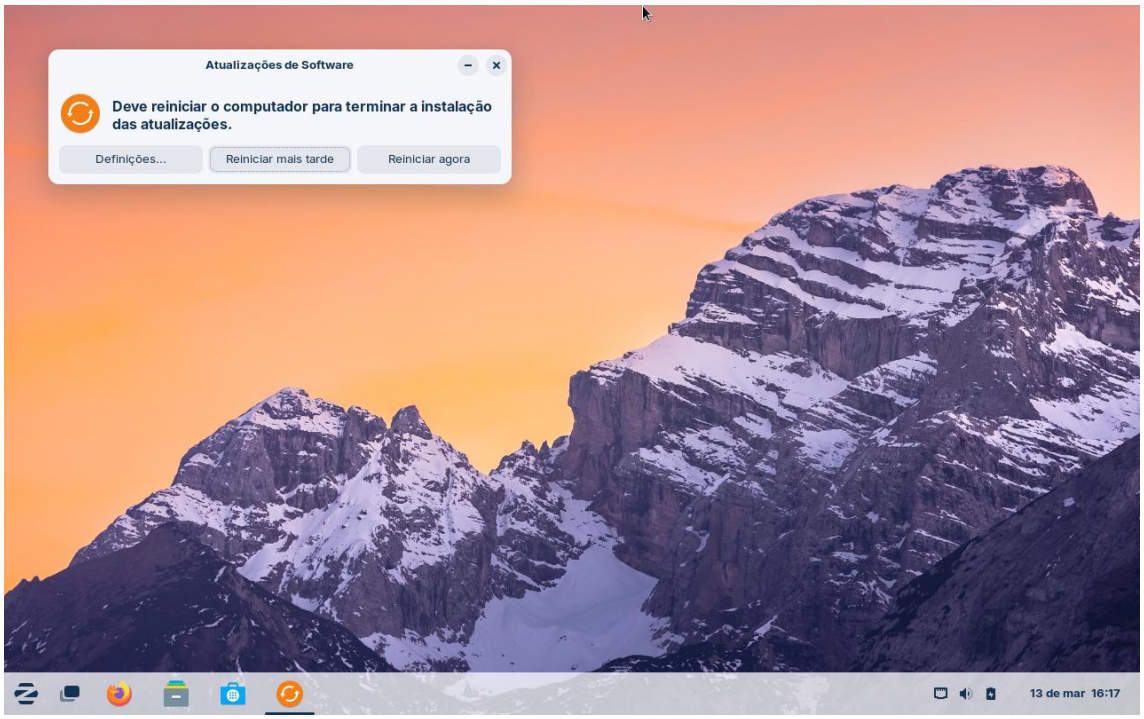

Imagem 26 – Atualização Completa | Reinício do Sistema

Scripts em Bash

Esta secção descreve os scripts desenvolvidos para automatizar tarefas administrativas em sistemas Unix/Linux. Todos os scripts foram testados num ambiente Ubuntu Server.

Script de Criação de Utilizadores

Objetivo:

Criar utilizadores automaticamente e atribuí-los ao grupo **sudo**.

Código:

```
read -p "Nome do novo utilizador: " user
sudo useradd -m -s /bin/bash "$user"
sudo passwd "$user"
sudo usermod -aG sudo "$user"
echo "Utilizador $user criado e adicionado ao grupo sudo."
```

Imagem 27 – Script Criar Utilizador

Exemplo de uso:

```
filipe@filipe-virtualbox:~$ cd Desktop/Trabalho2/scripts/
filipe@filipe-virtualbox:~/Desktop/Trabalho2/scripts$ ./criar_utilizador.sh
Nome do novo utilizador: Luis
[sudo] senha para filipe:
useradd: o utilizador 'Luis' já existe
Nova palavra-passe:
Digite novamente a nova palavra-passe:
passwd: a palavra-passe foi actualizada com sucesso
Utilizador Luis criado e adicionado ao grupo sudo.
```

Imagem 28 - Criação do utilizador Luís

Resultado esperado:

- Um novo utilizador é criado.
- O utilizador tem diretório próprio e permissões administrativas.

Script de Backup Automático

Objetivo:
Criar backups comprimidos da pasta /**home** e armazená-los numa pasta definida pelo utilizador.

Código:

```
src="/home"
dest="$HOME/Desktop/Trabalho2/scripts/backup"
mkdir -p "$dest"
date=$(date +%F)
tar -czf "$dest/backup-$date.tar.gz" "$src"
echo "Backup concluído: $dest/backup-$date.tar.gz"
```

Imagem 29 - Script Backup

Explicação:

- Usa **tar** para comprimir **/home**.
- Salva o ficheiro em **~/Desktop/Trabalho2/scripts/backup**.
- Cria a pasta se não existir.

Exemplo de ficheiro criado:

Imagem 30 - Ficheiro de Backup criado

Script de Monitorização do Sistema

Objetivo:

Monitorizar o uso de CPU, RAM e disco, guardando os dados num ficheiro de log.

Código:

```
if [ "$SUDO_USER" ]; then
    USER_HOME=$(eval echo "~$SUDO_USER")
else
    USER_HOME="$HOME"
fi

log="$USER_HOME/Desktop/Trabalho2/scripts/monitor.log"

mkdir -p "$(dirname "$log")"

echo "Monitorização em: $(date)" >> "$log"
echo "CPU:" >> "$log"
top -bn1 | grep "Cpu(s)" >> "$log"
echo "RAM:" >> "$log"
free -h >> "$log"
echo "Disco:" >> "$log"
df -h >> "$log"
echo "------------------------------" >> "$log"
```

Imagem 31 - Script Monitor

Exemplo de log gerado:

```
Monitorização em: ter 17 jun 2025 17:24:40 WEST
CPU:
RAM:
               total       usada       livre    compart.  buff/cache  disponível
Mem.:          3,8Gi       1,7Gi       149Mi        47Mi       2,0Gi       1,8Gi
Swap:          2,6Gi          0B       2,6Gi
Disco:
Sistema de ficheiros Tamanho   Uso Livre Uso% Montado em
tmpfs                  387M  1,4M  386M   1% /run
/dev/sda3               24G   18G  5,7G  76% /
tmpfs                  1,9G     0  1,9G   0% /dev/shm
tmpfs                  5,0M     0  5,0M   0% /run/lock
/dev/sda2              512M  6,1M  506M   2% /boot/efi
tmpfs                  387M  140K  387M   1% /run/user/1000
------------------------------
```

Imagem 32 - Log gerado do script

Observação:

- Este script funciona corretamente com ou sem sudo.
- Garante que os logs vão para o Desktop do utilizador real.

Programas em C

Foram desenvolvidos três programas em linguagem C para explorar conceitos fundamentais de sistemas operativos: **comunicação entre processos**, **tratamento de sinais**, e **simulação de escalonamento**.

Comunicação entre Processos (Pipe)

Objetivo:

Criar um processo filho que recebe uma mensagem do processo pai através de um pipe.

Código:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>

int main() {
    int pipefd[2];
    pid_t pid;
    char buffer[100];

    if (pipe(pipefd) == -1) {
        perror("Erro ao criar pipe");
        exit(1);
    }

    pid = fork();

    if (pid < 0) {
        perror("Erro no fork");
        exit(1);
    }

    if (pid == 0) {
        close(pipefd[1]);
        read(pipefd[0], buffer, sizeof(buffer));
        printf("Filho leu: %s\n", buffer);
        close(pipefd[0]);
    } else {
        close(pipefd[0]);
        char mensagem[] = "Olá do processo pai!";
        write(pipefd[1], mensagem, strlen(mensagem) + 1);
        close(pipefd[1]);
        wait(NULL);
    }

    return 0;
}
```

Imagem 33 - Código da comunicação (pipes)

Explicação:

- O **pipe()** cria um canal de comunicação entre pai e filho.
- O filho lê do pipe e imprime a mensagem enviada pelo pai.

Exemplo de saída:

```
filipe@filipe-virtualbox:~/Desktop/Trabalho2/progamas_c$ sudo ./processo_pipe_exec
[sudo] senha para filipe:
123Enganou-se, tente de novo.
[sudo] senha para filipe:
Filho leu: Olá do processo pai!
```

Imagem 34 - Exemplo de amostragem no cmd

Tratamento de Sinais

Objetivo:

Detetar sinais como **SIGINT** e **SIGTERM**, registando-os num ficheiro de log.

Código-resumo:

```c
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>

void handler(int sig) {
    FILE *f = fopen("signals.log", "a");
    fprintf(f, "Recebido sinal: %d\n", sig);
    fclose(f);
    if (sig == SIGTERM || sig == SIGINT) {
        exit(0);
    }
}

int main() {
    signal(SIGINT, handler);
    signal(SIGTERM, handler);
    while (1) {
        printf("À espera de sinais...\n");
        sleep(5);
    }
}
```

Imagem 35 - Código de sinais

Testes sugeridos:

- Executar o programa, e usar **Ctrl+C** ou **kill** para enviar sinais.
- Verificar conteúdo de signals.log, por exemplo:

```
Recebido sinal: 2
Recebido sinal: 2
Recebido sinal: 2
Recebido sinal: 2
```

Imagem 36 - Sinais recebidos

Simulador de Escalonamento de Processos

Objetivo:

Implementar dois algoritmos de escalonamento: **FCFS** e **Round-Robin**, com entrada manual de processos.

Campos simulados:

- Tempo de chegada
- Duração do processo
- Tempo de espera
- Tempo de conclusão
- Turnaround

Código-resumo FCFS:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX 10

typedef struct {
    int id;
    int arrival;
    int burst;
    int remaining;
    int finish;
    int waiting;
    int turnaround;
} Process;

void fcfs(Process p[], int n) {
    printf("\n[FCFS]\n");
    int time = 0;
    for (int i = 0; i < n; i++) {
        if (time < p[i].arrival) time = p[i].arrival;
        p[i].waiting = time - p[i].arrival;
        time += p[i].burst;
        p[i].finish = time;
        p[i].turnaround = p[i].finish - p[i].arrival;
    }

    printf("ID | Chegada | Duração | Espera | Conclusão | Turnaround\n");
    for (int i = 0; i < n; i++) {
        printf("%2d |   %4d   |   %4d   | %4d |   %4d    |    %4d\n",
               p[i].id, p[i].arrival, p[i].burst, p[i].waiting, p[i].finish, p[i].turnaround);
    }
}
```

Imagem 37 - Código FCFS

```c
void round_robin(Process p[], int n, int quantum) {
    printf("\n[Round-Robin | Quantum = %d]\n", quantum);
    int time = 0, done = 0;
    int queue[MAX];
    int front = 0, rear = 0;
    int visited[MAX] = {0};

    for (int i = 0; i < n; i++) p[i].remaining = p[i].burst;

    while (done < n) {
        for (int i = 0; i < n; i++) {
            if (p[i].arrival <= time && !visited[i]) {
                queue[rear++] = i;
                visited[i] = 1;
            }
        }

        if (front == rear) {
            time++;
            continue;
        }

        int idx = queue[front++];
        if (p[idx].remaining > quantum) {
            p[idx].remaining -= quantum;
            time += quantum;
        } else {
            time += p[idx].remaining;
            p[idx].remaining = 0;
            p[idx].finish = time;
            p[idx].turnaround = p[idx].finish - p[idx].arrival;
            p[idx].waiting = p[idx].turnaround - p[idx].burst;
            done++;
        }
```

Imagem 38 - Código Round-Robin (1)

```c
        for (int i = 0; i < n; i++) {
            if (p[i].arrival <= time && !visited[i]) {
                queue[rear++] = i;
                visited[i] = 1;
            }
        }

        if (p[idx].remaining > 0) {
            queue[rear++] = idx;
        }
    }

    printf("ID | Chegada | Duração | Espera | Conclusão | Turnaround\n");
    for (int i = 0; i < n; i++) {
        printf("%2d |   %4d   |   %4d   |  %4d |   %4d    |    %4d\n",
               p[i].id, p[i].arrival, p[i].burst, p[i].waiting, p[i].finish, p[i].turnaround);
    }
}
```

Imagem 39 - Código Round-Robin (2)

```c
int main() {
    Process processos[MAX];
    int n, quantum;

    printf("Número de processos: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        processos[i].id = i + 1;
        printf("Processo %d - Tempo de chegada: ", i + 1);
        scanf("%d", &processos[i].arrival);
        printf("Processo %d - Tempo de duração: ", i + 1);
        scanf("%d", &processos[i].burst);
    }

    printf("\n=== Simulação FCFS ===\n");
    fcfs(processos, n);

    // Reset valores alterados
    for (int i = 0; i < n; i++) {
        processos[i].waiting = 0;
        processos[i].finish = 0;
        processos[i].turnaround = 0;
    }

    printf("\nQuantum para Round-Robin: ");
    scanf("%d", &quantum);

    printf("\n=== Simulação Round-Robin ===\n");
    round_robin(processos, n, quantum);

    return 0;
}
```

Imagem 40 - Código Main

Conclusão

Este manual abordou o processo de instalação e configuração do Zorin OS 17.2 dentro do VirtualBox, permitindo aos utilizadores experimentar o sistema operativo de forma segura e eficiente.

Com base nesse ambiente, foram implementadas várias soluções técnicas para automatizar e controlar tarefas comuns em sistemas Linux. Os **scripts Bash** desenvolvidos automatizam a criação de utilizadores, backups e monitorização do sistema. Por outro lado, os **programas em C** permitiram explorar o funcionamento de processos, a comunicação entre eles e o tratamento de sinais.

O ponto alto do projeto foi a construção de um **simulador de escalonamento de processos**, aplicando os algoritmos **FCFS** e **Round-Robin**, permitindo observar na prática os conceitos estudados teoricamente.

Este trabalho consolidou conhecimentos fundamentais de administração de sistemas Unix/Linux e programação em C aplicada a sistemas operativos, desenvolvendo competências técnicas valiosas para ambientes profissionais.

Referências

- Site oficial do Zorin OS: https://zorin.com
- Site oficial da VirtualBox: https://www.virtualbox.org
- Documentação Debian: https://www.debian.org/doc/