Especificação do Projeto

Objetivo do Projeto

Desenvolver uma Máquina Virtual (VM) que simule a execução de processos em um sistema operacional de propósito específico. O projeto deve incluir a implementação de componentes essenciais como gerenciamento de processos, memória, entrada/saída (E/S), escalonamento de processos, um sistema de arquivos simples, uma linguagem de instruções e uma Interface de Linha de Comando (CLI).

Obs: o projeto deverá ser desenvolvido em duplas

Componentes Principais

1. Linguagem de Instruções
2. Interface de Linha de Comando (CLI)
3. Gerenciador de Processos
4. Gerenciador de Memória
5. Escalonador de Processos
6. Sistema de Arquivos Simples

1. Linguagem de Instruções

Requisitos

- Definir uma linguagem de instruções simples que os processos possam executar, dentro de um escopo escolhido pela equipe. Por exemplo, pode ser uma VM específica para rodar jogos (veja o Anexo).
- A linguagem deve incluir instruções para operações aritméticas, controle de fluxo, operações de E/S e manipulação de memória.

Tarefas

- Especificar a sintaxe e a semântica das instruções.
- Implementar um interpretador para a linguagem de instruções.
- Garantir que o interpretador possa decodificar e executar as instruções corretamente.

Exemplo de Instruções

- Operações Aritméticas: ADD, SUB, MUL, DIV
- Controle de Fluxo: JMP, JZ (Jump if Zero), JNZ (Jump if Not Zero)
- Operações de E/S: READ, WRITE
- Manipulação de Memória: LOAD, STORE

2. Interface de Linha de Comando (CLI)

Requisitos

- Implementar uma CLI que permita aos usuários interagir com a VM.
- A CLI deve suportar comandos para criar, executar e gerenciar processos, bem como para monitorar o estado da VM.

Tarefas

- Criar uma interface de linha de comando que aceite comandos do usuário.
- Implementar comandos para criar e gerenciar processos.
- Implementar comandos para monitorar o uso de recursos (CPU, memória, E/S).

Exemplo de Comandos

- Criar Processo: `create_process <arquivo_instrucoes>`
- Executar Processo: `run_process <PID>`
- Listar Processos: `list_processes`
- Monitorar Recursos: `monitor_resources`

3. Gerenciador de Processos

Requisitos

- Implementar uma estrutura de dados para representar um processo.
- Implementar funcionalidades para criar, gerenciar e terminar processos.
- Cada processo deve conter:
    - PID (Process Identifier)
    - Estado do Processo (pronto, executando, bloqueado, terminado)
    - Contador de Programa (PC)
    - Registros
    - Memória Alocada
    - Instruções

Tarefas

- Criar uma classe para representar um processo.
- Criar uma classe para gerenciar processos.
- Implementar métodos para criar e terminar processos.
- Garantir que o gerenciador de processos possa manter e atualizar o estado dos processos.

4. Gerenciador de Memória

Requisitos

- Implementar um sistema de gerenciamento de memória que permita a alocação e liberação de memória para os processos.
- Implementar um esquema simples de paginação ou segmentação.

Tarefas

- Criar uma classe para gerenciar a memória.
- Implementar métodos para alocar e liberar memória.
- Garantir que o gerenciador de memória possa lidar com solicitações de memória de diferentes tamanhos e liberar memória quando não for mais necessária.

5. Gerenciador de E/S

Requisitos

- Implementar um sistema de gerenciamento de dispositivos de entrada/saída.
- Implementar um sistema de filas para gerenciar requisições de E/S.

Tarefas

- Criar uma classe para gerenciar operações de E/S.
- Implementar métodos para solicitar e processar operações de E/S.
- Garantir que o gerenciador de E/S possa simular operações de dispositivos como discos, interfaces de rede e dispositivos de entrada.

6. Escalonador de Processos

Requisitos

- Implementar um escalonador de processos que decida qual processo deve ser executado em um dado momento.
- Implementar diferentes algoritmos de escalonamento, como FIFO, Round Robin, ou SJF.

Tarefas

- Criar uma classe para gerenciar o escalonamento de processos.
- Implementar métodos para adicionar processos à fila de prontos e selecionar o próximo processo a ser executado.
- Garantir que o escalonador possa alternar entre processos de maneira eficiente e justa.

7. Sistema de Arquivos Simples

Requisitos

- Implementar um sistema de arquivos simples que permita operações básicas de leitura e escrita de arquivos.

- Implementar funcionalidades para criar, ler, escrever e deletar arquivos.

Tarefas

- Criar uma classe para gerenciar o sistema de arquivos.
- Implementar métodos para criar, ler, escrever e deletar arquivos.
- Garantir que o sistema de arquivos possa armazenar e recuperar dados de maneira eficiente.

8. Simulação da Execução de Processos

Requisitos

- Implementar um ciclo de execução que inclua busca, decodificação, execução e atualização do estado dos processos.
- Implementar um conjunto de instruções que os processos possam executar, como operações aritméticas, controle de fluxo e operações de E/S.

Tarefas

- Criar uma classe principal para a VM que coordene a execução dos processos.
- Implementar o ciclo de execução dos processos.
- Implementar um conjunto de instruções que os processos possam executar.
- Garantir que a VM possa alternar entre processos e gerenciar o estado de cada processo de maneira eficiente.

9. Funcionalidades Adicionais

Isolamento e Segurança

- Garantir que as VMs sejam isoladas umas das outras e do sistema host.
- Implementar técnicas de isolamento de processos para evitar que uma VM interfira em outra.
- Gerenciar permissões e acessos para proteger os recursos do host e das VMs.

Monitoramento e Diagnóstico

- Implementar ferramentas para monitorar o desempenho e diagnosticar problemas nas VMs.
- Coletar e exibir métricas de desempenho, como uso de CPU, memória e E/S.
- Implementar logs e ferramentas de diagnóstico para identificar e resolver problemas.

10. Entrega e Avaliação

Documentação

- Fornecer uma documentação detalhada do código, explicando a arquitetura da VM, os componentes implementados e como usá-la.

Demonstração

- Realizar uma apresentação ou vídeo demonstrando a criação, execução e gerenciamento da VM.

Código-Fonte

- Entregar o código-fonte completo do projeto, hospedado em um repositório como GitHub ou GitLab.

ANEXO

Exemplo de instruções para VM voltada a executar jogos

CREATE_CHARACTER

- Sintaxe: `CREATE_CHARACTER name, sprite_file, x, y`
- Descrição: Cria um personagem com o nome `name`, usando o sprite `sprite_file` e posiciona-o nas coordenadas `(x, y)`.

MOVE_CHARACTER

- Sintaxe: `MOVE_CHARACTER name, direction, distance`
- Descrição: Move o personagem `name` na direção `direction` (UP, DOWN, LEFT, RIGHT) por uma distância `distance`.

SET_CHARACTER_POSITION

- Sintaxe: `SET_CHARACTER_POSITION name, x, y`
- Descrição: Define a posição do personagem `name` nas coordenadas `(x, y)`.

GET_CHARACTER_POSITION

- Sintaxe: `GET_CHARACTER_POSITION name, var_x, var_y`
- Descrição: Obtém a posição do personagem `name` e armazena em `var_x` e `var_y`.

ON_KEY_PRESS

- Sintaxe: `ON_KEY_PRESS key, action`

- Descrição: Define uma ação `action` a ser executada quando a tecla `key` for pressionada.

ON_COLLISION

- Sintaxe: `ON_COLLISION char1, char2, action`
- Descrição: Define uma ação `action` a ser executada quando `char1` colidir com `char2`.

WAIT

- Sintaxe: `WAIT time`
- Descrição: Pausa a execução por um tempo `time` (em milissegundos).

LOAD_SCENE

- Sintaxe: `LOAD_SCENE scene_file`
- Descrição: Carrega um cenário a partir do arquivo `scene_file`.

DRAW_SCENE

- Sintaxe: `DRAW_SCENE`
- Descrição: Desenha o cenário atual na tela.

CLEAR_SCENE

- Sintaxe: `CLEAR_SCENE`
- Descrição: Limpa o cenário atual.

PRINT

- Sintaxe: `PRINT message`
- Descrição: Exibe uma mensagem `message` na tela.

GET_INPUT

- Sintaxe: `GET_INPUT var`
- Descrição: Obtém a entrada do usuário e armazena em `var`.

Exemplo de processo utilizando a linguagem criada:

```
# Carrega o cenário inicial
LOAD_SCENE "initial_scene.txt"
DRAW_SCENE

# Cria personagens
CREATE_CHARACTER "hero", "hero_sprite.png", 100, 100
```

```
CREATE_CHARACTER "obstacle", "obstacle_sprite.png", 200, 200

# Define ações para teclas de movimento
ON_KEY_PRESS "UP", MOVE_CHARACTER "hero", UP, 10
ON_KEY_PRESS "DOWN", MOVE_CHARACTER "hero", DOWN, 10
ON_KEY_PRESS "LEFT", MOVE_CHARACTER "hero", LEFT, 10
ON_KEY_PRESS "RIGHT", MOVE_CHARACTER "hero", RIGHT, 10

# Define ação para colisão
ON_COLLISION "hero", "obstacle", PRINT "Collision detected!"

# Loop principal do jogo
MAIN_LOOP:
    CLEAR_SCENE
    DRAW_SCENE
    DRAW_CHARACTER "hero"
    DRAW_CHARACTER "obstacle"
    WAIT 100
    GOTO MAIN_LOOP
```