UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE INFORMÁTICA

DISCIPLINA: SISTEMAS OPERACIONAIS

Equipe: José Samuel Alves da Silva // André Luiz Rodrigues Castro Da Nobrega

Virtual Machine Created in Python

João Pessoa – 22 de Agosto de 2024

cli.py

```python
import pygame
import time

class CLI:
    def __init__(self, vm):
        self.vm = vm

    def start(self):
        while True:
            command = input("> ").strip().split()  # Recebe comando do usuário e divide em partes
            if not command:
                continue
            cmd, *args = command
            if cmd == "create_process":
                self.create_process(args)  # Cria um novo processo
            elif cmd == "run_vm":
                self.vm.run()  # Executa a VM
            elif cmd == "run_process":
                self.run_process(args)  # Executa um processo específico
            elif cmd == "list_processes":
                self.list_processes()  # Lista todos os processos
            elif cmd == "monitor_resources":
                self.monitor_resources()  # Monitora recursos
            elif cmd == "exit":
                break
                print(f"Unknown command: {cmd}")

    # Lista todos os processos ativos na VM
    def list_processes(self):
        processes = self.vm.process_manager.get_all_processes()  # Obtém todos os processos
        if not processes:
            print("No processes are running.")
        else:
            print(f"Total of {len(processes)} process(es):")
            for pid, process in processes.items():
                print(f"PID: {pid}, State: {process.state}")  # Exibe PID e estado de cada processo

    # Menu de músicas para o usuário escolher
    def show_music_menu(self):
        print("Escolha uma das músicas para tocar:")
        print("1. Música 1")
        print("2. Música 2")
        print("3. Parar música")
        choice = input("Digite o número da sua escolha: ")
        return choice

    # Processa a escolha da música e retorna as instruções correspondentes
    def handle_music_choice(self, choice):
        if choice == "1":
            return [("CHOOSE_MUSIC", "song1"), ("PLAY_MUSIC", "song1")]
        elif choice == "2":
            return [("CHOOSE_MUSIC", "song2"), ("PLAY_MUSIC", "song2")]
        elif choice == "3":
            return [("STOP_MUSIC",)]
        else:
            print("Escolha inválida.")
            return []
```

cli.py

```python
    # Verifica e garante que a música seja parada
    def ensure_music_stopped(self):
        import time
        max_retries = 5  # Tentativas máximas para garantir a parada
        retries = 0
        while retries < max_retries and pygame.mixer.music.get_busy():
            print("Parando música...")
            pygame.mixer.music.stop()
            time.sleep(1)  # Aguarda um tempo antes de verificar novamente
            retries += 1

        if not pygame.mixer.music.get_busy():
            print("Música foi parada com sucesso.")
        else:
            print("Falha ao parar a música após várias tentativas.")

    # Cria um novo processo na VM
    def create_process(self, args):
        if args[0] == "music_player":
            choice = self.show_music_menu()  # Exibe o menu de músicas
            instructions = self.handle_music_choice(choice)

            if instructions:
                process = self.vm.process_manager.create_process(instructions)
                self.vm.scheduler.add_process(process)  # Adiciona o processo ao escalonador
                print(f"Processo 'music_player' criado com PID {process.pid}")

        elif args[0] == "guess_game":
            instructions = [("GUESS_NUMBER",)]  # Instruções para o jogo de adivinhação
            process = self.vm.process_manager.create_process(instructions)
            self.vm.scheduler.add_process(process)
            print(f"Processo 'guess_game' criado com PID {process.pid}")
        else:
            print("Unknown process")

    # Executa um processo específico
    def run_process(self, args):
        print(f"Running process with PID {args[0]}")

    # Monitora os recursos do sistema
    def monitor_resources(self):
        print("Monitoring resources (mock implementation)")
```

instruction_set.py

```python
import pygame
import os
import random

class InstructionSet:
    def __init__(self, vm):
        self.vm = vm
        pygame.mixer.init()

    # Toca uma música em loop
    def PLAY_MUSIC(self, args):
        song_path = os.path.join("assets", "songs", f"{args[0]}.mp3")
        pygame.mixer.music.load(song_path)
        pygame.mixer.music.play(-1)  # O "-1" faz a música tocar em loop.
        print(f"Tocando música: {args[0]}")
        if self.vm.current_process:
            self.vm.current_process.state = 'RUNNING'  # Define o processo como em execução

    # Para a música em execução
    def STOP_MUSIC(self, args):
        if pygame.mixer.music.get_busy():
            pygame.mixer.music.stop()
            print("Música parada")
            if self.vm.current_process:
                self.vm.current_process.state = 'TERMINATED'  # Coloca o processo como terminado

    # Escolhe uma música para tocar
    def CHOOSE_MUSIC(self, args):
        self.vm.current_music = args[0]
        print(f"Música selecionada: {args[0]}")

    # Implementa um jogo de adivinhação de número
    def GUESS_NUMBER(self, args):
        number_to_guess = random.randint(1, 100)
        attempts = 0
        print("Adivinhe o número entre 1 e 100:")
        while True:
            guess = int(input("Seu palpite: "))
            attempts += 1
            if guess < number_to_guess:
                print("Muito baixo!")
            elif guess > number_to_guess:
                print("Muito alto!")
            else:
                print(f"Parabéns! Você acertou em {attempts} tentativas.")
                if self.vm.current_process:
                    self.vm.current_process.state = 'TERMINATED'  # Termina o processo ao final do jogo
                break
```

process_manager.py

```python
class Process:
    def __init__(self, pid, instructions):
        self.pid = pid  # Identificador único do processo
        self.instructions = instructions
        self.state = "READY"  # Estado inicial do processo
        self.pc = 0  # Contador de programa (posição atual na lista de instruções)
        self.registers = [0] * 8  # Registros do processo
        self.memory = [0] * 256  # Memória local do processo

    # Executa a próxima instrução do processo
    def execute_instruction(self):
        if self.pc < len(self.instructions):
            instruction = self.instructions[self.pc]  # Pega a próxima instrução
            self.pc += 1  # Avança o contador de programa
            return instruction
        else:
            self.state = 'TERMINATED'  # Termina o processo se não houver mais instruções
            return None

# Gerencia os processos na VM
class ProcessManager:
    def __init__(self):
        self.processes = {}  # Dicionário para armazenar processos
        self.next_pid = 1  # Contador para atribuir PIDs únicos

    # Cria um novo processo e retorna sua instância
    def create_process(self, instructions):
        pid = self.next_pid  # Atribui o próximo PID
        self.next_pid += 1  # Incrementa o contador de PIDs
        process = Process(pid, instructions)  # Cria o processo
        self.processes[pid] = process  # Armazena o processo no dicionário
        return process

    # Remove um processo pelo PID
    def terminate_process(self, pid):
        if pid in self.processes:
            del self.processes[pid]  # Remove o processo do dicionário

    # Retorna todos os processos ativos
    def get_all_processes(self):
        return self.processes
```

scheduler.py

```python
class Scheduler:
    def __init__(self):
        self.ready_queue = []  # Fila de processos prontos para execução
        self.current_process = None  # Processo em execução no momento

    # Adiciona um processo à fila de prontos
    def add_process(self, process):
        if process.state == 'READY':
            self.ready_queue.append(process)

    # Seleciona o próximo processo a ser executado
    def schedule_next(self):
        if self.ready_queue:
            self.current_process = self.ready_queue.pop(0)  # Pega o primeiro processo da fila
            self.current_process.state = 'RUNNING'  # Define o estado como em execução
            return self.current_process
        else:
            self.current_process = None
            return None

    # Marca um processo como aguardando (ex: I/O)
    def mark_as_waiting(self, process):
        process.state = 'WAITING'

    # Marca um processo como terminado
    def mark_as_terminated(self, process):
        process.state = 'TERMINATED'

    # Reprograma o processo atual (volta para a fila de prontos)
    def reschedule(self):
        if self.current_process:
            self.current_process.state = 'READY'
            self.ready_queue.append(self.current_process)
```

virtual_machine.py

```python
from process_manager import ProcessManager
from scheduler import Scheduler
from instruction_set import InstructionSet

class VM:
    def __init__(self):
        # Inicializa a VM com os componentes principais:
        # Gerenciador de processos, escalonador e o conjunto de instruções.
        self.process_manager = ProcessManager()
        self.scheduler = Scheduler()
        self.instruction_set = InstructionSet(self)
        self.current_process = None  # Armazena o processo que está sendo executado no momento.

    def run(self):
        # Loop principal que mantém a VM rodando até que não haja mais processos a serem executados.
        while True:
            # Obtém o próximo processo pronto para execução.
            process = self.scheduler.schedule_next()
            if not process:
                # Se não houver mais processos, termina a execução da VM.
                print("No more processes to run.")
                break

            self.current_process = process  # Define o processo atual.
            print(f'{process.pid}' +': '+ process.state)  # Exibe o PID e o estado do processo.

            # Loop que mantém o processo rodando enquanto ele estiver no estado 'RUNNING'.
            while process.state == 'RUNNING':
                # Executa a próxima instrução do processo.
                instruction = process.execute_instruction()

                if instruction:
                    # Se a instrução for válida, executa-a.
                    self.execute_instruction(process, instruction)


                # Se o processo foi terminado durante a execução da instrução,
                # ele é marcado como 'TERMINATED' e não é reprogramado.
                if process.state == 'TERMINATED':
                    self.scheduler.mark_as_terminated(process)
                else:
                    # Se o processo ainda estiver pronto para rodar, reprograma-o.
                    self.scheduler.reschedule()

    def execute_instruction(self, process, instruction):
        opcode, *args = instruction
        # Busca o método correspondente ao opcode no conjunto de instruções.
        method = getattr(self.instruction_set, opcode, None)
        if method:
            # Se o método existe, executa-o passando os argumentos.
            method(args)
        else:
            # Se o opcode não for reconhecido, exibe uma mensagem de erro.
            print(f"Unknown instruction {opcode} for process {process.pid}")
```

DOCUMENTAÇÃO

Módulo virtual_machine.py

Classe VM

A classe VM representa a máquina virtual que gerencia a execução dos processos e suas instruções.

Métodos

- __init__(self):
    - Inicializa a máquina virtual, criando instâncias de ProcessManager, Scheduler, e InstructionSet.
    - self.current_process é usado para rastrear o processo em execução no momento.
- run(self):
    - Controla o ciclo de execução da máquina virtual.
    - A cada iteração, a função verifica se há processos na fila de prontos (ready_queue). Se houver, um processo é selecionado e seu estado é definido como RUNNING.
    - Enquanto o processo está em execução, a instrução atual é executada.
    - Se o processo terminar (TERMINATED), ele é marcado como terminado. Caso contrário, ele é reprogramado na fila.
- execute_instruction(self, process, instruction):
    - Executa a instrução do processo, buscando a função correspondente no conjunto de instruções (InstructionSet).
    - Caso a instrução não seja reconhecida, uma mensagem de erro é impressa.

Módulo scheduler.py

Classe Scheduler

O Scheduler é responsável por gerenciar a fila de processos prontos e determinar a ordem de execução dos processos.

Métodos

- **__init__(self)**:
    - Inicializa a fila de processos prontos (ready_queue) e a referência ao processo em execução no momento.
- **add_process(self, process)**:
    - Adiciona um processo à fila de prontos se seu estado for READY.

- **schedule_next(self)**:
    - Seleciona o próximo processo da fila de prontos, define seu estado como RUNNING e retorna o processo selecionado.
- **mark_as_waiting(self, process)**:
    - Altera o estado do processo para WAITING.
- **mark_as_terminated(self, process)**:
    - Altera o estado do processo para TERMINATED.
- **reschedule(self)**:
    - Reprograma o processo atual (que está em execução) voltando-o para a fila de prontos com o estado READY.

Módulo process_manager.py

Classe ProcessManager

O ProcessManager é responsável por criar e gerenciar os processos.

Métodos

- **__init__(self)**:
    - Inicializa um dicionário para armazenar processos (processes) e um contador para o próximo PID (next_pid).
- **create_process(self, instructions)**:
    - Cria um novo processo com as instruções fornecidas, atribui um PID único e o adiciona ao dicionário de processos.
- **terminate_process(self, pid)**:
    - Remove o processo do dicionário de processos com base no PID.
- **get_all_processes(self)**:
    - Retorna todos os processos ativos.

Módulo process.py

Classe Process

A classe Process representa um processo em execução na VM, contendo suas instruções, estado, e registros.

Métodos

- **__init__(self, pid, instructions)**:
    - Inicializa o processo com um identificador único (pid), uma lista de instruções (instructions), e configura o estado inicial como READY.

- **execute_instruction(self)**:
    - Executa a próxima instrução na lista e avança o contador de programa (pc).
    - Se todas as instruções forem executadas, o estado do processo é definido como TERMINATED.

Módulo instruction_set.py

Classe InstructionSet

O InstructionSet contém todas as operações que podem ser executadas pela VM, incluindo operações aritméticas, controle de fluxo, manipulação de memória, e execução de áudio.

Métodos

- **__init__(self, vm)**:
    - Inicializa o conjunto de instruções, recebendo uma instância da VM.
    - Inicializa o mixer do Pygame para tocar músicas.
- **PLAY_MUSIC(self, args)**:
    - Carrega e toca uma música em loop. O processo que executa esta instrução é mantido no estado RUNNING.
- **STOP_MUSIC(self, args)**:
    - Para a música e termina o processo que a executa.
- **CHOOSE_MUSIC(self, args)**:
    - Escolhe uma música específica para tocar.
- **GUESS_NUMBER(self, args)**:
    - Implementa um jogo de adivinhação de número, que termina o processo ao final do jogo.

Módulo cli.py

Classe CLI

A CLI (Command Line Interface) fornece uma interface para interagir com a máquina virtual.

Métodos

- **__init__(self, vm)**:
    - Inicializa a CLI com uma referência à VM.
- **start(self)**:
    - Inicia o loop principal da CLI, esperando por comandos do usuário.

- **list_processes(self)**:
  - Lista todos os processos ativos e seus estados.
- **show_music_menu(self)**:
  - Exibe um menu para o usuário escolher músicas.
- **handle_music_choice(self, choice)**:
  - Processa a escolha do usuário e retorna as instruções correspondentes.
- **ensure_music_stopped(self)**:
  - Verifica e garante que a música foi interrompida.
- **create_process(self, args)**:
  - Cria um processo com base no tipo de processo especificado (music_player ou guess_game).
- **run_process(self, args)**:
  - Executa um processo específico com base no PID.
- **monitor_resources(self)**:
  - Função fictícia para monitorar recursos (não implementada).

Módulo main.py

Função main()

Ponto de entrada da aplicação. Inicializa a máquina virtual e a CLI, e inicia a interface de linha de comando.