Técnicas de Sistemas Operacionais numa arquitetura Von Newmann e MIPS simulada

1st Getúlio Santos Mendes
DECOM-DV
CEFET-MG Campus V
Divinópolis, Brasil
getuliosantosmendes@gmail.com

2nd Frank Leite Lemos Costa
DECOM-DV
CEFET-MG Campus V
Divinópolis, Brasil
frankcefet090@gmail.com

***Abstract*—Este artigo apresenta o processo, escolhas e resultados do desenvolvimento de uma arquitetura simulada baseada na arquitetura MIPS com o objetivo de aplicar, analisar e discutir didaticamente conceitos fundamentais para Sistemas Operacionais modernos e sua relação com o Hardware.**

***Index Terms*—Arquitetura de Von Neumann, Arquitetura MIPS, Sistemas Operacionais, CPU, Multi-core, Preempção, Pipeline**

I. Introdução

Esse trabalho foi desenvolvido como parte da disciplina de Sistemas operacionais e subdivido em módulos corres a conceitos fundamentais para Sistemas Operacionais modernos.

O principal objetivo é a implementação de um simulador de uma arquitetura simplificada com conceitos de pseudo-paralelismo (Pipeline) e paralelismo real e a construção de um sistema operacional simplificado para gerenciar esses recursos de Hardware.

II. Metodologia

A linguagem de implementação escolhida foi o C++, devido ao seu suporte a tanto recursos de mais auto nível de abstração, como a possibilidade de operações de baixo nível e controle direto sobre memória; além de sua boa performance.

A. Arquitetura MIPS customizada

A arquitetura MIPS é um exemplo de processador RISC (Reduced Instruction Set Computing), projetada para maximizar eficiência e desempenho com um conjunto reduzido e simplificado de instruções.

A pipeline é uma característica central da MIPS, permitindo a execução simultânea de diferentes etapas de várias instruções. Dividindo a execução em estágios independentes e aumentando o *throughput* geral do processador. Essa abordagem, torna a arquitetura MIPS altamente eficiente e ideal para aplicações de alto desempenho, como sistemas embarcados e dispositivos de consumo.

O simulador foi diretamente inspirado pela arquitetura buscando fidelidade dentro do possível, sendo dividido em Unidade Lógica e Aritmética (ULA), Unidade de Controle, Registradores e memória principal e memória secundária; fazendo uso dos mesmos estágios de pipeline que a MIPS: busca, decodificação, execução, acesso à memória e escrita de volta (*Fetch, Decode, Execute, Memory Access, Write back*.

No estágio *Fetch*, a próxima instrução é buscada da memória, com base no contador de programa (PC), enquanto o PC é atualizado para a próxima instrução. Em seguida, no estágio *Decode*, a instrução é decodificada, identificando os registradores e valores necessários para a execução.

No estágio Execute, a operação especificada pela instrução é realizada, com suporte da Unidade Lógica e Aritmética (ULA) para cálculos ou ajustes no PC em caso de saltos e *loops*. O estágio *Memory Access* trata de leitura na memória para instruções como *load* e *print*. Por fim, no estágio *Write Back*, os resultados são armazenados dos registradores à memória, onde instuções como *store* são implementadas.

B. Arquitetura Multicore e Preempção básica

Neste modulo foi implementado uma arquitetura multicore com o uso da biblioteca *pthreads*, a arquitetura single core foi abstraída de modo que para a execução de cada core da CPU bastasse criar um *thread* dentro da linguagem de programação utilizada que execute essa abstração de core. É claro que isso inevitavelmente introduz problemas a respeito do acesso de recursos em paralelo e alocação de programas para executar em cada um desses cores.

Para resolver esses problemas foi-se implementado um mecanismo básico de preempção. O Sistema Operacional mantêm uma fila de processos a serem executados, seus estados e recursos em seu *Program Control Block* (PCB). Cada *thread* representando um core busca um processo para a execução dentro dessa fila e executa esse processo por um determinado *quantum* que representa um quantidade de ciclos de *clock* do core. Essa fila segue a orientação FCFS (i.e., *First Come First Service*) por simplificação de implementação.

Além disso, um *thread* roda dedicadamente a função de Escalonador de Processos que verifica se todos os processos foram totalmente executados para finalizar o simulador, futuramente também a organização da fila de processos de acordo com políticas de escalonamento mais robustas.

Um *thread* dedicado também faz o gerenciamento do requisições de *output*, ele utiliza um *delay* artificial para gerar o bloqueio dos processos, enquanto o processo está bloqueado

o núcleo busca outro programa para executar, quando a requisição é atendida o *thread* é desbloqueado.

III. Resultados

A. Arquitetura MIPS customizada

A implementação do simulador inspirado na arquitetura MIPS mostrou resultados alinhados aos princípios do design RISC, reproduzindo com fidelidade o *assembly* da arquitetura e até possivelmente executando programas feitos para MIPS com pequenas modificações.

Os cinco estágios do pipeline: *Fetch, Decode, Execute, Memory Access* e *Write Back*. Componentes como a Unidade Lógica e Aritmética (ULA), a Unidade de Controle, os registradores e as memórias foram integrados para criar um modelo funcional e realista, sem muitas abstrações computacionais que simplificam o trabalho feito em um processador real.

O pipeline demonstrou eficiência ao implementar o pseudoparalelismo e aumentar a efiência de execução. A busca e decodificação funcionaram de forma precisa, preparando os dados necessários para operações subsequentes. A ULA foi bem-sucedida na execução de cálculos e controle de fluxo, enquanto o acesso e a escrita na memória garantiram a consistência dos dados e a funcionalidade de operações como load, store e print.

No geral, o simulador capturou a essência da arquitetura MIPS, conseguindo executar um *subset* de seu *assembly* satisfatório, mostrando desempenho sólido em cenários variados e oferecendo um ambiente eficaz para explorar conceitos fundamentais de sistemas baseados em pipeline.

B. Arquitetura Multicore e Preempção básica

A implentação do processamento paralelo e preempção trouxe melhorias significativas em sua capacidade de simular comportamentos realistas de sistemas operacionais modernos. Uma vez que praticamente todo sistema operacional moderno, com exceção de alguns embarcados, suportam arquiteturas *multicore* e praticamente toda arquitetura moderna é *multicore*.

A utilização de *pthreads* para modelar a execução paralela permitiu a criação de um ambiente onde múltiplos processos podem ser gerenciados simultaneamente, respeitando conceitos de exclusão mútua. Esse avanço trouxe maior flexibilidade ao simulador, que agora é capaz de lidar com cargas de trabalho mais complexas.

A lógica de preempção foi integrada com sucesso, permitindo que processos em execução sejam interrompidos e substituídos conforme definido pelo término do quantum ou pela necessidade do sistema operacional. A troca de contexto entre processos foi realizada por meio de uma lista de PCBs (Process Control Blocks), garantindo a persistência das informações críticas de cada processo e a retomada correta de sua execução.

Com o gerenciamento eficiente de recursos de *I/O* através de requisições ao sistema e a troca de processos bloqueados enquanto esperam por recursos, o simulador ganha performance em cargas de trabalho com muita entrada e saída.

A adoção de estratégias como exclusão mútua se provou ser eficaz para gerenciar regiões críticas, resultando em um desempenho robusto.

Esses resultados demonstram que o simulador agora oferece uma base sólida para explorar e experimentar com arquiteturas *multicore* e sistemas preemptivos, servindo como uma ferramenta poderosa para o estudo e análise de sistemas operacionais modernos.

IV. Conclusão

A implementação do simulador baseado na arquitetura MIPS e sua posterior expansão para uma configuração *multicore* com suporte à preempção demonstrou resultados sólidos e alinhados aos objetivos do projeto. O uso de mutilas *threads* permite que cada parte do sistema operacional execute de modo independe e simultâneo, fazendo uso do paralelismo disponível em sistemas modernos.

A ampliação para a arquitetura *multicore* trouxe avanços significativos, permitindo simulações mais próximas da realidade de sistemas modernos. O suporte à preempção e ao gerenciamento de múltiplos núcleos destacou a flexibilidade e a escalabilidade do simulador, embora tenha introduzido desafios adicionais relacionados à sincronização e ao controle de concorrência. O controle de recursos de entrada e saída foram desafiadores de se implementar, mas fornecem um significativo ganho de performance em cenários com grande influxo de entrada e saída.

Os resultados indicam que o simulador oferece uma plataforma eficaz para estudar e compreender os princípios de sistemas computacionais, desde arquiteturas baseadas em pipeline até ambientes *multicore* mais complexos e seus efeitos e consequências em Sistemas Operacionais modernos. No entanto, há espaço para aprimoramentos, como a inclusão de políticas avançadas de escalonamento, otimizações no gerenciamento de regiões críticas e cache. Esses aprimoramentos podem tornar o simulador ainda mais versátil, ampliando suas aplicações para cenários mais diversos e complexos.

References

[1] link do repositório: https://github.com/frankleitelemoscosta/Multcore-SO-work.git

[2] Tanenbaum, A. S., Bos, H. (2016). *Sistemas operacionais: projeto e implementação* (4ª ed.). Pearson.

[3] Silberschatz, A., Galvin, P. B., Gagne, G. (2013). *Fundamentos de sistemas operacionais* (9ª ed.). LTC.