Universidade Federal de São João del Rei
Departamento de Ciência da Computação
Curso de Ciência da Computação

Simulador de gerenciador de processos

Adélson de Oliveira Carmo Júnior, Renan Rodrigues Sousa

1 Introdução

Um processo é apenas uma instância de um programa em execução, essencialmente uma tarefa que está sendo realizada por um sistema operacional. Cada processo possui seu próprio espaço de memória, que inclui o contador de programas, registradores, e variáveis, permitindo que opere de forma independente. Além disso, cada processo é identificado por um identificador único que o diferencia dos outros processos em execução no sistema. O gerenciamento de processos é uma das funcionalidades mais importantes de um sistema operacional moderno, sendo responsável por controlar a execução de múltiplos processos em um sistema.

Este projeto visa implementar um simulador para cinco funções principais do gerenciamento de processos, oferecendo uma visão prática sobre como essas operações são executadas. O simulador abordará a criação de processos, a substituição do processo atual por outro, conhecida como troca de processos, fundamental para a multitarefa, onde o sistema precisa interromper a execução de um processo e retomar outro. Além disso, o simulador permitirá a observação da transição de estados de um processo, como "pronto", "executando"ou "bloqueado". O escalonamento de processos será abordado para decidir qual processo será executado em determinado momento, utilizando um algoritmo para otimizar o desempenho do sistema. Por fim, o simulador demonstrará a troca de contexto, que envolve salvar o estado atual de um processo e carregar o estado do próximo processo a ser executado quando um processo é interrompido.

A visão geral do funcionamento do simulador consiste em três tipos de processos:

- O ***Commander*** é o processo responsável por iniciar a simulação. Ele cria um pipe e, em seguida, um processo do tipo *Process Manager*. O *Commander* atua como a interface principal do sistema, lendo comandos de entrada do usuário e transmitindo-os para o Process Manager através de um pipe. Ele permite que o usuário interaja diretamente com a simulação, enviando instruções que controlam a execução dos processos simulados
- O ***Process Manager*** é responsável por simular as cinco funções de gerenciamento de processos mencionadas. Ele inicia criando o primeiro processo simulado e, em seguida, gerencia a execução de instruções e a atualização das estruturas de dados relacionadas ao estado dos processos. Além disso, o *Process Manager* é responsável por criar o processo *Reporter* sempre que é necessário imprimir o estado do sistema.
- O ***Reporter*** é criado pelo *Process Manager* sempre que é necessário imprimir o estado do sistema para monitorização do desempenho e execução correta dos processos simulados.

2 Implementação

Para implementar a simulação das cinco funções do gerenciador de processos - criação de processos, substituição do processo atual, transição de estados, escalonamento e troca de contexto - utilizamos chamadas de sistema do Linux. A função fork() é usada para criar novos processos, enquanto wait() permite esperar pela conclusão dos processos filhos. A comunicação entre processos é feita por meio de um pipe(), e a função

sleep() é empregada para simular atrasos na execução, replicando o comportamento real do gerenciamento de processos em sistemas operacionais.

Como mencionado anteriormente, o simulador é estruturado em três processos principais: Commander, Process Manager e Reporter. A seguir, fornecemos uma explicação mais detalhada de cada um desses processos e as estruturas de dados utilizadas.

2.1 Commander

O Commander é responsável por duas funções principais: a criação de um novo processo e a interação com o usuário. Ele inicia o Process Manager por meio de um fork e estabelece um pipe para se comunicar com ele. O Commander também atua como a interface de entrada do sistema, permitindo que o usuário interaja com a simulação. Ele recebe comandos do usuário e os envia para o Process Manager através do pipe.

Os comandos aceitos pelo sistema são: Q, que executa uma única instrução do processo em execução; U, que desbloqueia o primeiro processo bloqueado; P, que imprime o estado atual do sistema; e T, que termina a simulação. Esses comandos permitem ao usuário controlar e monitorar a execução dos processos simulados de forma eficaz.

2.2 Process Manager

O Process Manager é responsável por gerenciar a execução, bloqueio, e término dos processos simulados. Ele manipula uma lista de processos, executa instruções de cada um, e mantém o estado de cada processo atualizado.

2.2.1 Estruturas de Dados

Na implementação do Process Manager foram utilizadas duas estruturas de dados:

- **Processos Simulados:** representa um processo simulado dentro do sistema. Contém o identificador único **id** do processo e a variável inteira **var**, que é utilizada pelas instruções do programa. O campo **contP** serve como o contador de programa, indicando a posição atual no vetor de strings programa, que representa o conjunto de instruções que o processo deve executar. O **tamProg** indica o tamanho desse programa, enquanto **bloqueio** é um indicador de bloqueio que mostra se o processo está atualmente impedido de continuar sua execução. O campo **completo** assinala se o processo já foi finalizado. Por último, **tempo_inicio** registra o tempo em que o processo foi iniciado, o que é útil para calcular o tempo total de execução e análise de desempenho. Essa estrutura é crucial para gerenciar o estado de cada processo dentro do simulador.

- **Gerenciador:** administra a execução de processos simulados e contém diversos campos para gerenciar diferentes aspectos do sistema. O campo **tempo_gasto** é um inteiro que rastreia o total de tempo consumido na execução dos processos até o momento. **contador_processo** atua como um índice ou contador para processos, ajudando a identificar ou gerenciar o processo atual. **processos_finalizados** é um ponteiro para um array de inteiros que registra os ids dos processos que foram concluídos. **processos_bloqueado**s é um ponteiro para um array de inteiros que armazena os IDs ou índices dos processos que estão atualmente em estado bloqueado. processos armazena todos os processos do programa, é a tabela pcb. **processos_espera** é um ponteiro para um array de ponteiros para ProcessoSimulado que mantém os processos em espera, aguardando condições para serem executados. **processo_execucao** é um ponteiro para um ProcessoSimulado que indica o processo que está atualmente utilizando a CPU.

2.2.2 Gerenciamento de Processos

Na implementação do nosso simulador, a lógica de gerenciamento foi desenvolvida da seguinte maneira: o Process Manager recebe comandos do usuário, que são obtidos pelo Commander e transmitidos através de um pipe. A função *novoGerenciador* é criar uma nova instância do gerenciador de processos, alocando espaço para armazenar processos ativos, em espera, bloqueados, e finalizados. Ele também inicializa contadores e variáveis de estado.

O *ProcessManager* cria novos processos usando a função *novoProcesso*, que instancia um novo processo simulado a partir de um programa fornecido. Cada processo possui um identificador único, contador de programa, tamanho do programa e variáveis de controle como bloqueio e status de completude. Essa funcionalidade é utilizada para criar o processo inicial com base no arquivo init. O gerenciador pode então executar quatro ações diferentes.

O comando **Q** inicia a execução do primeiro processo da fila. Cada vez que o usuário escolhe o comando Q, o processo em execução avança uma instrução em seu programa simulado. A lógica utilizada para esse comando é implementada no método simular, que coordena a execução dos processos, assegurando que o processo correto esteja em execução. Ele verifica se o processo atual está bloqueado ou completo e chama a função *executarInstrucao* para realizar uma instrução de cada vez, garantindo o avanço correto do estado do processo.

Cada processo simulado é composto por um conjunto de instruções. A instrução S atualiza o valor da variável inteira para o valor especificado na instrução. A instrução A soma um valor ao valor atual da variável inteira, enquanto D subtrai um valor desse total. A instrução B bloqueia o processo simulado, e E termina o processo simulado. A instrução F cria um novo processo simulado que é uma cópia exata do processo pai, começando a execução a partir da instrução imediatamente após a instrução F. Finalmente, a instrução R substitui o programa do processo simulado pelo programa contido em um arquivo.

A medida que o programa avança na execução de cada instrução, ele utiliza funções auxiliares para realizar as ações correspondentes. Entre essas funções principais, a instrução B é responsável por bloquear o processo atual, movendo-o para a lista de processos bloqueados. A instrução F cria uma cópia do processo atual e coloca o processo original na fila de espera, permitindo que ambos prossigam de forma independente. Por fim, a instrução R substitui o programa do processo atual por um novo, carregado a partir de um arquivo especificado.

O comando **U** é utilizado para desbloquear o primeiro processo da lista de processos bloqueados. Quando esse comando é recebido, o Process Manager identifica o primeiro processo na lista de bloqueados e o remove dessa lista. O processo desbloqueado é então movido para a fila de processos em espera, tornando-o disponível para execução. Essa ação permite que o processo, que estava aguardando devido ao bloqueio, retome sua execução quando for o momento apropriado.

O comando **P** invoca a função *reporter*, que imprime o estado atual do sistema. Reporter imprime informações organizadas sobre cada estado dos processos.

O comando **T** encerra a simulação e imprime o tempo de retorno médio dos processos. Quando recebido, o Process Manager chama a função reporter para exibir o estado final do sistema. e, em seguida, encerra todos os processos e limpando os recursos utilizados. Esse comando permite uma análise final do desempenho da simulação.

2.3 Reporter

O processo reporter é responsável por imprimir o estado atual do sistema de processos simulados, oferecendo uma visão detalhada dos processos em execução, bloqueados e finalizados. A função reporter utiliza um *fork* para criar um processo filho. Isso permite que o relatório seja gerado de maneira isolada e paralela à execução principal, sem interromper ou atrasar o gerenciamento dos processos.

A função reporter exibe detalhes do processo que está atualmente em execução, incluindo seu ID, tempo de início e o tempo de CPU utilizado até o momento. Além disso, lista os processos bloqueados, apresentando informações semelhantes às do processo em execução. A função também exibe os processos finalizados, fornecendo dados que ajudam a entender o desempenho geral da simulação. Ao terminar, o processo filho se encerra, retornando o controle ao processo principal do simulador.

3 Resultados e Discussões

3.1 Resultados

Para testar e verificar a correta implementação da aplicação, foi elaborada uma entrada para a função init que percorre todas as instruções disponíveis. A figura 1 mostra a entrada escolhida, enquanto as figuras 2 e 3 complementam essa entrada.

```
S 100
A 19
A 20
D 23
A 55
F 1
R file_a.txt
D 1
F 1
R file_b.txt
E
```

Figura 1: init

```
S 10
A 5
A 5
B
A 5
E
```

Figura 2: file_a.txt

```
S 50
D 1
E
```

Figura 3: file_b.txt

Ao executar o código e passar *init* como entrada, o painel de escolhas geral aparece. O primeiro comando, Q, inicia o processo simulado. O sistema resolve a primeira expressão e, em seguida, o comando P é usado para imprimir o processo atual na tela. Esta etapa pode ser visualizada na figura 4 abaixo, juntamente com o valor 100, que foi atribuído corretamente à variável.

```
Digite um comando (Q, U, P, T): Q
Digite um comando (Q, U, P, T): P

****************************************************************
Estado do sistema:
****************************************************************
TEMPO ATUAL: 1

PROCESSO EXECUTANDO:
pid: 0, ppid: 29890, valor: 100, tempo inicio: 0, CPU usada ate agora: 1

PROCESSOS BLOQUEADO:

PROCESSOS FINALIZADOS:

PROCESSOS ESPERANDO:
****************************************************************
```

Figura 4: Teste dos comandos Q, P e S

Por conseguinte, Q será apertado novamente mais 4 vezes, a fim de testar a funcionalidade das instruções A e D, então ao imprimir a tela, obtem-se a figura 5.

```
Digite um comando (Q, U, P, T): Q
Digite um comando (Q, U, P, T): Q
Digite um comando (Q, U, P, T): Q
Digite um comando (Q, U, P, T): Q
Digite um comando (Q, U, P, T): P

****************************************************************
Estado do sistema:
****************************************************************
TEMPO ATUAL: 5

PROCESSO EXECUTANDO:
pid: 0, ppid: 29890, valor: 171, tempo inicio: 0, CPU usada ate agora: 5

PROCESSOS BLOQUEADO:

PROCESSOS FINALIZADOS:

PROCESSOS ESPERANDO:
****************************************************************
```

Figura 5: Teste dos comandos A e D

Nessa figura, nota-se que inicialmente o valor atribuído é 100. Em seguida, 19 é somado, depois 20 é adicionado, seguido por uma subtração de 23, e finalmente 55 é somado. O resultado dessa expressão é descrito abaixo:

$$100 + 19 + 20 - 23 + 55 = 171 \tag{1}$$

Na sequência, as próximas duas instruções a serem testadas são F e R. A primeira, F, cria um novo processo idêntico ao do pai, faz o pai pular n instruções e torna o novo processo o processo em execução. Isso é mostrado na figura 6. A segunda instrução, R, lê um arquivo e substitui as instruções do processo atual pelo conteúdo do arquivo de texto fornecido, resetando os parâmetros. Isso é exemplificado na figura 7.

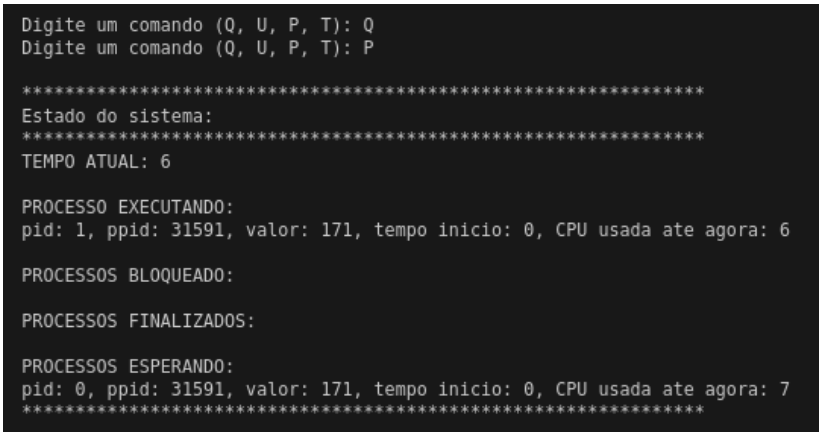

Figura 6: Teste do comando F

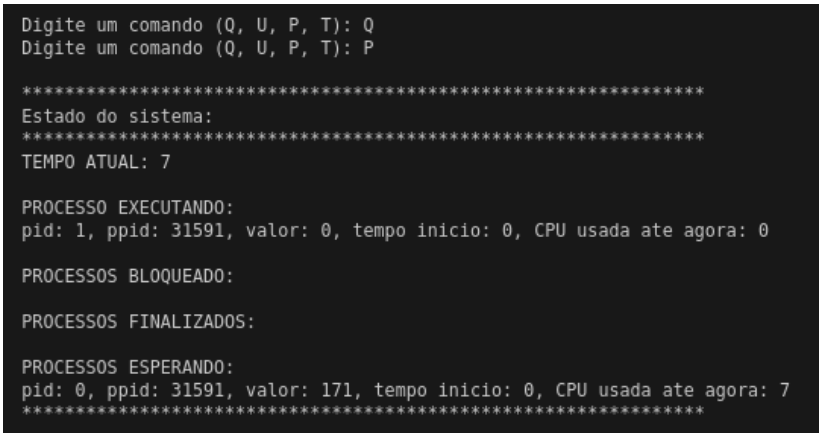

Figura 7: Teste do comando R

A figura x mostra as instruções do novo arquivo lido, file_a.txt. A única nova instrução é B. Como a funcionalidade das instruções anteriores já foi comprovada, elas não serão o foco nesta parte. Q será inserido mais quatro vezes no terminal. Ao fazer isso, o processo é bloqueado, saindo da execução e passando para a fila de bloqueados. O processo que estava na fila de espera então assume a execução, como mostrado na figura 8.

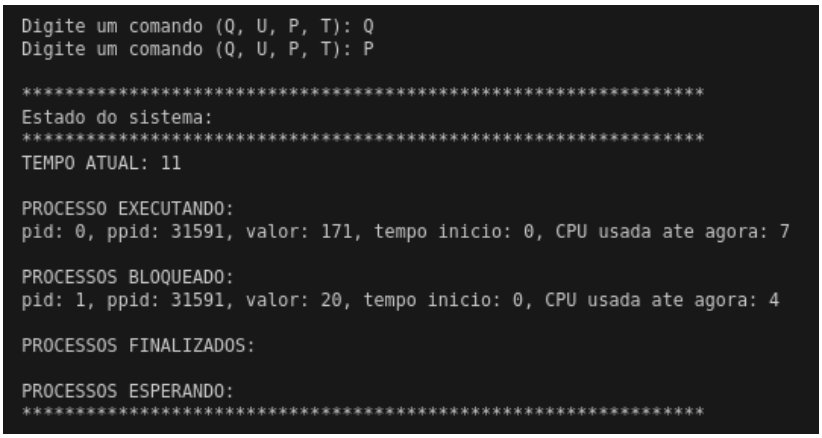

Figura 8: Teste do comando B

Em seguida, o primeiro processo volta a ser executado e mais quatro instruções são processadas, testando algumas instruções vistas anteriormente, conforme ilustrado na figura x. Depois, um novo comando, U, é inserido no terminal, desbloqueando o processo que estava bloqueado e colocando-o na fila de espera. Esse processo se posiciona após o processo 0, respeitando a ordem de chegada na fila, como mostrado na figura 9.

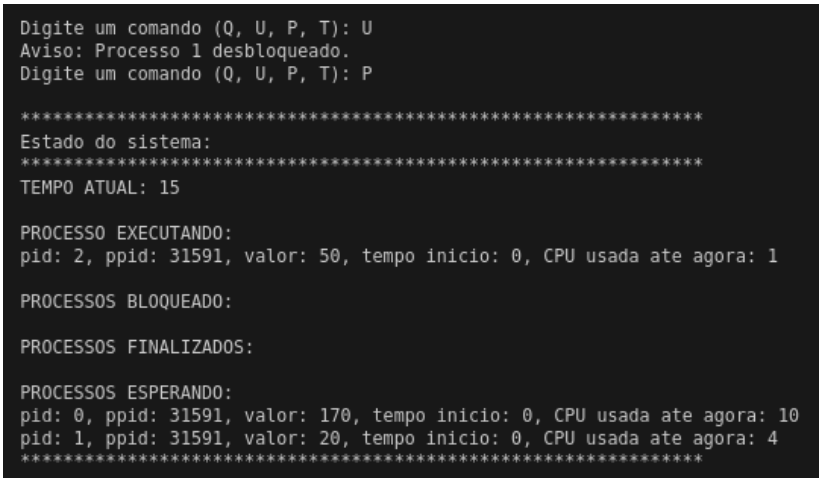

Figura 9: Teste do comando U

Prosseguindo, mais duas instruções são executadas, finalizando o terceiro processo, de ID 2, pois a instrução de término E foi lida. Esse processo é movido para a lista de finalizados, liberando espaço para o primeiro processo na fila de espera. Seguindo o escalonamento FCFS, o processo na primeira posição da fila é o próximo a ser executado, como ilustrado na figura 10.

Ao processar a nova instrução, o processo atual é terminado e adicionado à lista de concluídos. O último processo da fila de espera então passa a ser executado e processa mais duas instruções antes de ser terminado e integrado à lista de finalizados. Com a ausência de processos, uma mensagem aparece na área reservada para o processo em execução, indicando que não há mais processos para serem executados. Isso é mostrado na figura 11.

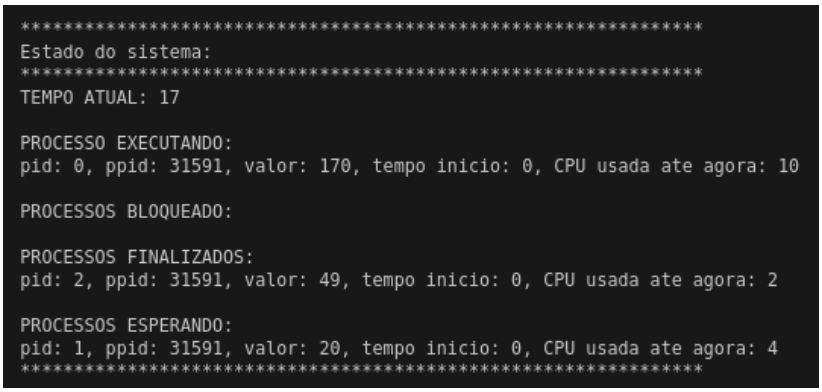

Figura 10: Demonstração do escalonamento

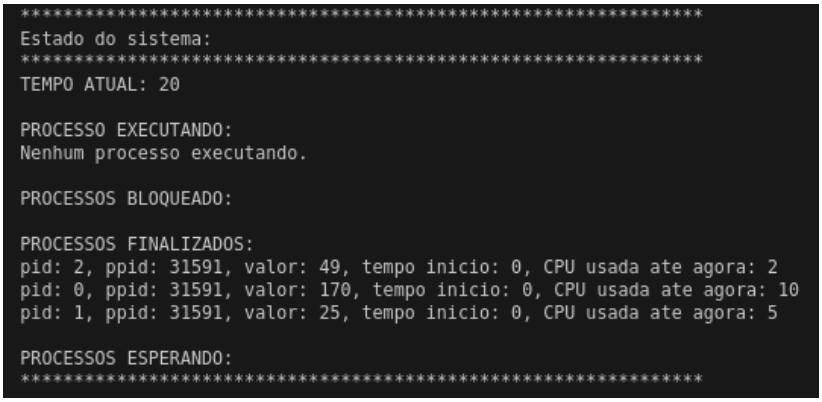

Figura 11: Finalização da simulação

Com isso, a simulação foi concluída, permitindo o uso do último comando que faltava testar: o 'T'. Esse comando encerra o processo geral e gera um relatório final sobre a aplicação, conforme mostrado na Figura 12.

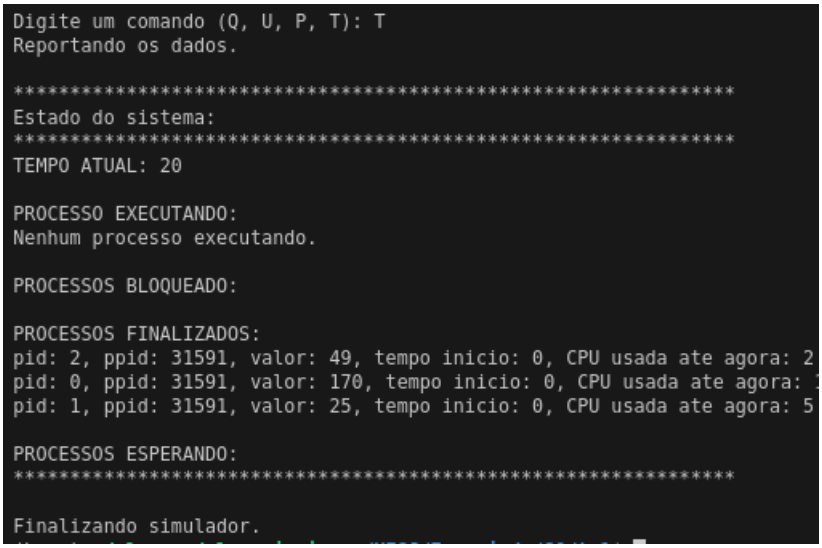

Figura 12: Finalização do processo

3.2 Discussões

Antes mesmo de decidir qual escalonador utilizar, foi necessário estabelecer um método para organizar a ordem dos processos em espera que seguiriam para a execução. Durante a implementação, percebeu-se que a estrutura desenvolvida já funcionava como um escalonador FCFS (First-Come, First-Served), onde os processos são atendidos na ordem de chegada. Portanto, não foi necessário escolher um escalonador diferente.

O mesmo ocorreu com a troca de contexto. Quando foi necessário trocar o processo em execução, a troca de CPU e a atualização da Tabela PCB se mostraram indispensáveis para o funcionamento correto do código. Embora não exista uma função específica chamada "troca de contexto", essa tarefa é realizada pelas funções adicionarNovoExecutando e copiarProcesso, além de outros trechos específicos de código.

As demais funções, como bloquear e adicionarFinalizados, ajudam a manter o código organizado e legível. Juntamente com os locais onde são chamadas, essas funções fazem parte da implementação descrita no tópico 2.

4 Conclusão

Como esperado, o início da execução foi desafiador, tanto para entender o que havia sido solicitado quanto para colocar em prática. No entanto, à medida que o código foi sendo implementado, as ideias foram se tornando mais claras. Esse processo gradual de entendimento foi crucial, pois permitiu enfrentar e superar os obstáculos iniciais. À medida que o projeto avançava, foi possível aplicar conceitos teóricos de sistemas operacionais na prática, o que reforçou o conhecimento.

A implementação do código proporcionou uma experiência prática valiosa, destacando a importância de conceitos como escalonamento de processos e troca de contexto. Além disso, a divisão de tarefas e a colaboração entre os integrantes do grupo foram fundamentais para o sucesso do projeto. Essa abordagem colaborativa permitiu a troca de ideias e a resolução conjunta de problemas, enriquecendo ainda mais o aprendizado.

Em resumo, o desafio inicial foi superado através de um esforço contínuo e colaborativo. A experiência resultou em um entendimento mais profundo dos sistemas operacionais, aumentando a confiança na implementação de soluções práticas e consolidando o conhecimento teórico.

Referências

[1] TANENBAUM, A.; BOS, H. *Sistemas Operacionais Modernos*. 4ª Edição. São Paulo: Pearson Education do Brasil, 2016.