

Licenciatura em Engenharia e Gestão de Sistemas de Informação

Universidade do Minho

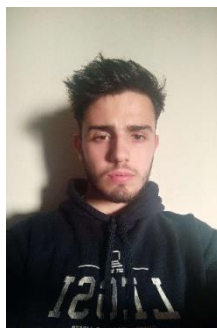
Sistemas Operativos

2023/2024

Relatório Final



Fernando Fernandes
Pires
a103887
a103887@uminho.pt



José Miguel Sousa
Pereira
a103808
a103808@uminho.pt



Tomás Magalhães
Gonçalves
a103810
a103810@uminho.pt

Índice

Objetivos do Trabalho.....	3
Principais Questões a Considerar	4
1. Como o programa interage com o sistema de arquivos proc para obter informações sobre os processos ativos no sistema?	4
2. Como é implementado o redirecionamento de entrada e saída nos comandos que envolvem operadores como '>' e '<'?	4
3. Como o programa executa comandos Linux, tanto simples quanto aqueles envolvendo pipelines (' ')?	4
4. Como o programa lida com situações de erro, como comandos inválidos ou falhas na execução?	4
5. Como o programa gerência a atualização periódica das informações exibidas no comando 'top'?	4
6. Como é tratada a alocação e desalocação de memória para evitar vazamentos?	5
7. Quais medidas de segurança são implementadas para garantir a integridade e confidencialidade das operações do programa?	5
8. O código é projetado considerando a portabilidade para diferentes ambientes Linux?.....	5
9. Como o código-fonte está documentado, facilitando a compreensão e manutenção futura?	5
Fundamentos Teóricos/Tecnológicos Subjacentes	6
1. Manipulação de Processos em Sistemas Linux	6
2. Chamadas de Sistema e API do Linux.....	6
3. Redirecionamento de Entrada e Saída em Sistemas Unix	6
4. Eficiência na Atualização de Dados em Tempo Real	6
5. Gerenciamento Dinâmico de Memória em C	6
6. Segurança em Programação de Sistemas	7
7. Portabilidade em Ambientes Linux.....	7
8. Documentação e Boas Práticas de Programação.....	7
Descrição da Implementação dos Comandos	8
1. Comando top.....	8
2. Comandos de Execução Simples (cmd).....	8
3. Comandos com Redirecionamento de Saída (cmd > file).....	8
4. Comandos com Redirecionamento de Entrada (cmd < file).....	8
5. Comandos com Pipelines (cmd1 cmd2).....	8

Manual dos Comandos	9
1. Comando top.....	9
2. Comando de Execução Simples.....	9
3. Comando com Redireccionamento de Saída	9
4. Comando com Redireccionamento de Entrada	10
5. Comando com Pipelines	10
Conclusões.....	11

Objetivos do Trabalho

Este relatório visa oferecer uma análise abrangente do desenvolvimento do comando/programa mycmd, conforme proposto na disciplina de Sistemas Operativos. Os objetivos delineados são essenciais para garantir a eficácia e qualidade do mycmd, refletindo um compromisso sério com os padrões estabelecidos.

O primeiro objetivo concentra-se na implementação do mycmd, seguindo estritamente as especificações fornecidas. Neste contexto, é fundamental que o mycmd obtenha informações sobre os processos ativos no sistema de forma autónoma, sem depender dos comandos padrão do Linux.

A segunda meta é capacitar o mycmd para a execução de comandos Linux, incluindo redirecionamento de entrada e saída. A terceira prioridade consiste na implementação de um sistema robusto de deteção e tratamento de erros. O mycmd deve ser capaz de apresentar mensagens explicativas e contextualizadas em situações de comandos inválidos, visando aprimorar a usabilidade do programa.

Por fim, a validação da implementação por meio de testes é crucial. A condução de testes abrangentes e criteriosos visa assegurar a robustez, confiabilidade e desempenho do mycmd, evidenciando sua qualidade em cenários práticos. Essa abordagem estratégica reflete empenho em alcançar a excelência na implementação do mycmd, priorizando não apenas a conformidade técnica, mas também a eficiência operacional e a experiência do utilizador.

Principais Questões a Considerar

1. Como o programa interage com o sistema de arquivos `proc` para obter informações sobre os processos ativos no sistema?

O programa utiliza funções da biblioteca C, como `opendir` e `readdir`, para acessar as informações das estruturas de dados do kernel presentes no diretório `/proc`. Dessa forma, consegue identificar e exibir detalhes sobre os processos em execução, como PID, estado, linha de comando e nome de usuário.

2. Como é implementado o redirecionamento de entrada e saída nos comandos que envolvem operadores como `'>'` e `'<'`?

O redirecionamento de entrada e saída é principalmente realizado por meio da chamada do sistema `dup2`. Os operadores `">"`, `"<"` e `"|"` são manipulados para redirecionar a saída ou a entrada dos comandos conforme especificado. A implementação cuida eficientemente do redirecionamento para arquivos e criação de pipes anônimos para suportar pipelines.

3. Como o programa executa comandos Linux, tanto simples quanto aqueles envolvendo pipelines (`'|'`)?

O programa executa comandos Linux utilizando a família de funções `exec`. Para comandos simples, como `mycmd ls -l`, o programa cria um novo processo filho e substitui sua imagem usando `execlp`. No caso de comandos envolvendo pipelines (`|`), o programa cria pipes anônimos e utiliza `forks` para executar comandos consecutivos, redirecionando a saída do primeiro para a entrada do segundo.

4. Como o programa lida com situações de erro, como comandos inválidos ou falhas na execução?

O programa é robusto em relação ao tratamento de erros. Mensagens de erro explicativas são apresentadas em situações como comandos inválidos ou falhas na execução. O usuário é informado claramente sobre problemas que possam ocorrer durante a execução.

5. Como o programa gerencia a atualização periódica das informações exibidas no comando `'top'`?

A eficiência é priorizada na gestão da atualização periódica das informações apresentadas pelo comando `'top'`. O código foi ajustado para adotar um método baseado em sinais, especificamente utilizando o sinal `SIGALRM`, para acionar a atualização em intervalos de 10 segundos. Essa abordagem emprega um loop infinito e sinais para garantir uma exibição sempre atualizada. Além disso, o programa continua a gerenciar conjuntos volumosos de processos de maneira eficiente, mantendo a visualização ordenada e limitada a 20 processos.

6. Como é tratada a alocação e desalocação de memória para evitar vazamentos?

A alocação e desalocação de memória são tratadas de maneira eficiente. O programa utiliza `realloc` para ajustar dinamicamente o tamanho dos arrays de argumentos, garantindo o uso eficiente da memória. A liberação de memória é realizada adequadamente ao final da execução do programa.

7. Quais medidas de segurança são implementadas para garantir a integridade e confidencialidade das operações do programa?

Medidas de segurança são implementadas para garantir a integridade e confidencialidade das operações. O programa valida entradas do usuário para evitar vulnerabilidades, especialmente ao lidar com arquivos e execução de comandos externos.

8. O código é projetado considerando a portabilidade para diferentes ambientes Linux?

O código é desenvolvido considerando a portabilidade para diferentes ambientes Linux. Depende de chamadas de sistema padrão e utiliza bibliotecas comuns, minimizando dependências específicas de distribuição.

9. Como o código-fonte está documentado, facilitando a compreensão e manutenção futura?

O código-fonte está bem documentado, facilitando a compreensão e manutenção futura. Comentários explicativos são fornecidos para destacar decisões de design e implementação, tornando o código mais acessível a desenvolvedores.

Fundamentos Teóricos/Tecnológicos Subjacentes

1. Manipulação de Processos em Sistemas Linux

O programa mycmd baseia-se nos fundamentos teóricos da manipulação de processos em sistemas operacionais Linux. O acesso às informações dos processos é realizado por meio do sistema de arquivos /proc, que fornece uma interface para ler dados das estruturas do kernel relacionadas aos processos. Essa abordagem é fundamentada na teoria de que o diretório /proc é uma representação virtual do estado do sistema em tempo de execução.

2. Chamadas de Sistema e API do Linux

O desenvolvimento do programa faz uso extensivo de chamadas de sistema e da API do Linux. As chamadas de sistema, como open, read, close, fork, exec, pipe, e dup2, são essenciais para interagir com o sistema operacional, realizar operações de entrada/saída, gerenciar processos e implementar o redirecionamento de entrada/saída. Essa abordagem reflete a compreensão profunda dos recursos oferecidos pelo kernel Linux.

3. Redirecionamento de Entrada e Saída em Sistemas Unix

A implementação eficaz do redirecionamento de entrada e saída segue os princípios fundamentais dos sistemas Unix. O uso da chamada de sistema dup2 para redirecionamento é uma técnica amplamente reconhecida em programação de sistemas Unix/Linux. A manipulação cuidadosa dos operadores >, <, e | demonstra o conhecimento sólido sobre a sintaxe e semântica desses redirecionamentos.

4. Eficiência na Atualização de Dados em Tempo Real

O comando 'top' adota abordagens eficientes para garantir a atualização contínua de dados em tempo real. Agora, por meio do uso de sinais, em especial o SIGALRM, as informações são atualizadas em intervalos regulares de 10 segundos. Esta implementação prescinde da função scanf para esperar a entrada do usuário, sendo substituída por um loop contínuo e sinais, assegurando uma exibição sempre atualizada. Essa otimização reflete o compromisso com o desempenho, proporcionando uma visualização iterativa e responsiva das informações do sistema.

5. Gerenciamento Dinâmico de Memória em C

A alocação e desalocação dinâmica de memória, utilizando as funções malloc, realloc, e free, são fundamentadas nos princípios de gerenciamento eficiente de memória em linguagem C. A adaptação dinâmica do tamanho dos arrays de argumentos assegura uma utilização otimizada da memória durante a execução do programa.

6. Segurança em Programação de Sistemas

Medidas de segurança, como validação de entradas do usuário e tratamento adequado de erros, são incorporadas ao código para garantir a integridade e segurança das operações. Esses conceitos refletem uma compreensão sólida das práticas de programação segura em ambientes de sistema operacional.

7. Portabilidade em Ambientes Linux

O código do mycmd é projetado para ser portátil em ambientes Linux, minimizando dependências específicas de distribuição. A escolha de chamadas de sistema e bibliotecas padrão contribui para a compatibilidade e execução consistente em diferentes sistemas operacionais baseados em Linux.

8. Documentação e Boas Práticas de Programação

A documentação do código, composta por comentários explicativos, segue boas práticas de programação. Essa abordagem facilita a compreensão do código-fonte, auxilia na manutenção futura e promove a transparência no desenvolvimento do programa.

Descrição da Implementação dos Comandos

1. Comando top

O comando top é implementado para fornecer informações detalhadas sobre o desempenho do sistema e os processos em execução. A obtenção das métricas, como a carga média do CPU, o número total de processos e o número de processos no estado de execução, é realizada lendo o arquivo `/proc/loadavg`. O programa apresenta essas informações formatadas e, em seguida, lista os processos em execução, exibindo detalhes como PID, estado, nome de usuário e linha de comando. A manipulação de processos e informações do sistema é realizada lendo os diretórios e arquivos no sistema de arquivos `/proc`. As informações sobre cada processo, como PID, estado e nome de usuário, são obtidas a partir dos arquivos `/proc/{PID}/status` e `/proc/{PID}/cmdline`. A implementação garante eficiência na obtenção dessas informações, permitindo a exibição ordenada e limitada a 20 processos em execução. A atualização ocorre a cada 10 segundos e é interrompida pelo usuário ao pressionar 'q'.

2. Comandos de Execução Simples (cmd)

Os comandos de execução simples, como `mycmd ls -l`, são tratados pela criação de um novo processo filho usando `fork`. O processo filho substitui sua imagem pelo comando fornecido usando `execlp`. Dependendo da presença de operadores de redirecionamento (`>`, `<`), o programa redireciona a saída ou a entrada conforme necessário.

3. Comandos com Redirecionamento de Saída (cmd > file)

Quando o comando `mycmd cmd > file` é fornecido, o programa redireciona a saída do comando para o arquivo especificado. Isso é alcançado abrindo o arquivo com `open` e utilizando a chamada de sistema `dup2` para redirecionar o descritor de arquivo correspondente à saída padrão (`stdout`). O comando é então executado, e a saída é direcionada para o arquivo fornecido.

4. Comandos com Redirecionamento de Entrada (cmd < file)

No caso de comandos com redirecionamento de entrada, como `mycmd cmd < file`, o programa redireciona a entrada do comando lendo o conteúdo do arquivo especificado e usando `dup2` para redirecionar o descritor de arquivo correspondente à entrada padrão (`stdin`). O comando é executado posteriormente.

5. Comandos com Pipelines (cmd1 | cmd2)

Para comandos com pipelines, como `mycmd cmd1 | cmd2`, o programa cria um pipe anônimo usando a chamada de sistema `pipe`. Um processo filho é criado para executar `cmd1`, cuja saída é redirecionada para o extremo de escrita do pipe. O processo pai executa `cmd2`, redirecionando sua entrada para o extremo de leitura do pipe. Isso permite que a saída de `cmd1` seja passada como entrada para `cmd2`.

Manual dos Comandos

1. Comando top

O comando top fornece informações sobre o desempenho do sistema e os processos ativos. A sintaxe do comando é:

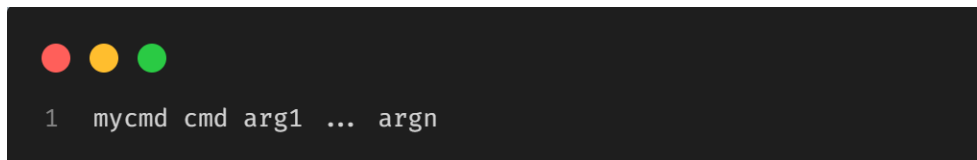
A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The prompt '1 mycmd ' is followed by the command 'top'.

Descrição

Apresenta a carga média do CPU nos últimos 1, 5 e 15 minutos, o número total de processos e o número de processos no estado de execução. Exibe também informações sobre os processos ativos, incluindo PID, estado, linha de comando e nome de usuário. A atualização ocorre em intervalos de 10 segundos até receber o input 'q'.

2. Comando de Execução Simples

O comando de execução simples permite executar comandos Linux diretamente. A sintaxe é:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The prompt '1 mycmd ' is followed by the command syntax 'cmd arg1 ... argn'.

Descrição

Executa o comando Linux especificado (cmd) com os argumentos fornecidos (arg1 a argn).

3. Comando com Redirecionamento de Saída

O comando com redirecionamento de saída permite executar um comando e redirecionar a saída para um arquivo. A sintaxe é:


A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The prompt '1 mycmd ' is followed by the command syntax 'cmd arg1 ... argn > file'.

Descrição

Executa o comando Linux (cmd) com os argumentos fornecidos e redireciona a saída para o arquivo especificado (file).

4. Comando com Redireccionamento de Entrada

O comando com redireccionamento de entrada permite executar um comando lendo a entrada de um arquivo. A sintaxe é:



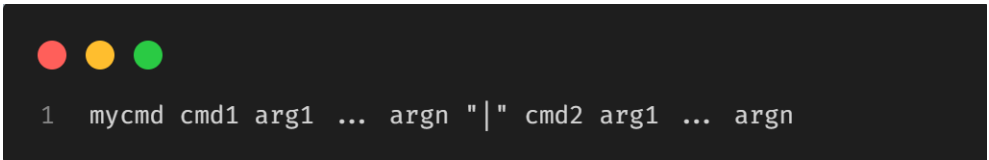
```
1 mycmd cmd arg1 ... argn "<" file
```

Descrição

Executa o comando Linux (cmd) com os argumentos fornecidos, lendo a entrada do arquivo especificado (file).

5. Comando com Pipelines

O comando com pipelines permite executar dois comandos e redirecionar a saída do primeiro como entrada para o segundo. A sintaxe é:



```
1 mycmd cmd1 arg1 ... argn "|" cmd2 arg1 ... argn
```

Descrição

Executa os comandos Linux cmd1 e cmd2, redirecionando a saída de cmd1 como entrada para cmd2.

Conclusões

O desenvolvimento do programa "mycmd" proporcionou uma experiência enriquecedora na esfera da programação de sistemas, evidenciando competências essenciais como a manipulação eficiente de processos, o acesso ao sistema de arquivos /proc e a implementação de comandos fundamentais. No decorrer do processo, a gestão eficaz de desafios relacionados à coordenação de processos, bem como a atenção dedicada à eficiência e usabilidade, emergiram como elementos-chave.

A estratégia de ordenação e a limitação da exibição a 20 processos no comando "top" foi concebida para facilitar a interpretação, enquanto a interatividade integrada com o utilizador contribuiu significativamente para a usabilidade do programa. A manutenção criteriosa e a documentação apropriada do código foram implementadas para assegurar a sua compreensão e facilitar futuras atualizações.

Este projeto não apenas atendeu aos requisitos propostos, mas também estabeleceu uma base sólida para melhorias futuras. O "mycmd" não representa meramente uma conclusão, mas sim um ponto de partida para a exploração contínua e o aprimoramento da programação em sistemas operacionais Linux. A sua implementação reflete um compromisso com os mais elevados padrões de desenvolvimento, consolidando-se como uma contribuição substancial no contexto da programação para sistemas operacionais.