Universidade do Minho
Licenciatura em Engenharia Informática

Unidade Curricular de Sistemas Operativos

Ano Lectivo de 2023/2024

TPSO68

David Figueiredo (a104360)

Diogo Ferreira (a104266)

Ricardo Ochoa (a103996)

SO

Índice

Resumo

Este relatório explica o nosso programa que tinha como objetivo implementar algum serviço que permitisse a execução e orquestração de tarefas num computador. Há um programa orchestrator que configura o servidor que, em seguida, permite que o programa cliente execute várias tarefas. O programa cliente também permite visualizar quais tarefas estão em execução, aguardando a execução e quais são concluídas. Há também subarquivos como avl.c, parser.c, linkedlist.c e request.c que contêm funções auxiliares para o programa.

Arquivos

client.c

O arquivo client.c contém a implementação de um programa cliente projetado para interagir com um servidor através de comunicação entre processos (IPC) usando FIFOs (First In, First out).

A função main() determina a ação a ser executada pelo cliente seguindo uma estrutura de caso de comutação com base no número de argumentos passados.

Switch(args)

1) Sem argumentos
    - Cria um código de erro e envia-o para o servidor
2) Execute
    - Constrói uma solicitação com base em argumentos
    - Envia a solicitação para o servidor
    - Lê e exibe a resposta do servidor
3) Status
    - Recebe, lê e exibe informações de status do servidor
4) Default
    - Lida com formatos de entrada incorretos e exibe mensagens de erro

Error Handling

- O programa utiliza perror() para imprimir mensagens de erro juntamente com o motivo da falha.
- A memória alocada dinamicamente usando malloc() é corretamente desalocada usando free().

- FIFOs temporários criados para receber respostas do servidor são desvinculados (unlink()) após o uso para garantir a limpeza

Conclusão

Em resumo, client.c fornece funcionalidade para que os clientes se comuniquem com o servidor enviando solicitações e recebendo respostas via FIFOs. Ele suporta mecanismos básicos de tratamento e limpeza de erros para garantir o bom funcionamento dentro do sistema cliente-servidor maior.

orchestrator.c

O arquivo orchestrator.c contém a implementação de um programa de servidor responsável por gerenciar tarefas simultâneas recebidas de clientes e distribuí-las aos processos de trabalho para execução. Esse orquestrador é um componente crítico de um sistema distribuído maior, projetado para lidar com solicitações de clientes com eficiência.

A função main() serve como ponto de entrada do programa e é responsável por orquestrar a execução das tarefas. Ele lida com argumentos de linha de comando para inicializar o orquestrador com os parâmetros necessários, como o caminho da pasta de saída e o número de tarefas paralelas

main(int args, char ** argv)

1) Validação de parâmetros
    - Verifica se o programa foi inicializado com os parâmetros necessários. Caso contrário, ele exibe uma mensagem de erro e sai
2) Inicialização do controlador:
    - Inicializa o processo do controlador responsável por gerenciar a distribuição de tarefas e a comunicação entre clientes e trabalhadores.
    - Cria um pipe (fdServerController) para comunicação entre o orquestrador e o processo do controlador.
3) Processo do Controlador:
    - Forks um processo filho para executar a lógica do controlador. O processo controlador gerencia a distribuição de tarefas entre os processos de trabalho e cuida da comunicação com os clientes.
4) Processos de trabalho:
    - Forks vários processos filhos (numOfSons) para executar tarefas simultaneamente.
    - Cada processo de trabalho escuta tarefas do controlador por meio de um canal (fdControllerTasks). Ao receber uma tarefa, o processo de trabalho a executa, redireciona a saída para o arquivo de saída correspondente e notifica o controlador após a conclusão.
5) Comunicação com o cliente:

- O orquestrador escuta solicitações recebidas de clientes por meio de um FIFO (fifoComum).
- Ao receber uma solicitação, o orquestrador a encaminha ao processo controlador para distribuição entre os processos de trabalho.

6) Limpar:
    - Ao receber um sinal de encerramento (-404), os processos do orquestrador e do controlador limpam os recursos, encerram os processos de trabalho e saem normalmente.

Error Handling

- O tratamento adequado de erros é implementado usando perror() para exibir mensagens de erro em caso de falhas.
- Recursos como pipes e FIFOs são devidamente fechados (close()) e desvinculados (unlink()) para garantir a limpeza após o uso

Conclusão

Em resumo, orchestrator.c serve como componente central responsável por gerenciar a execução simultânea de tarefas em um sistema distribuído. Ele coordena com eficiência a comunicação entre clientes e trabalhadores, garantindo que as tarefas sejam executadas de maneira confiável e que os resultados sejam armazenados de forma adequada.

linkedList.c

O arquivo linkedList.c fornece a implementação de uma estrutura de dados de listas ligadas com várias operações para manipular e gerenciar a lista. Esta lista vinculada foi projetada para armazenar objetos genéricos, proporcionando flexibilidade para diferentes tipos e estruturas de dados.

Estrutura node: Cada node da lista ligada contém um ponteiro para um objeto genérico (void *) e um ponteiro para o próximo node.

Funções

1) createLinkedList():
    - Inicializa e retorna uma lista vinculada vazia.
2) acrescentar(LinkedList * lista, void * obj):
    - Acrescenta um novo nó contendo o objeto fornecido ao final da lista vinculada.
3) inserir(LinkedList ** lista, void * obj):
    - Insere um novo nó contendo o objeto fornecido no início da lista vinculada.
4) orderInsert(LinkedList ** lista, void * obj, int (*cmp)(void *, void *)):
    - Insere um novo nó contendo o objeto fornecido na lista vinculada em ordem de classificação, com base na função de comparação fornecida.
5) pop(LinkedList * lista, void (*destruir)(void *)):

- Remove e retorna o objeto do final da lista vinculada.

6) popFront(LinkedList ** lista, void (*destruir)(void *)):
   - Remove e retorna o objeto do início da lista vinculada.
7) destruirLinkedList(LinkedList * lista, void (*destruir)(void *)):
   - Destrói a lista vinculada, liberando toda a memória alocada. Opcionalmente, aplica uma função de destruição a cada objeto da lista.
8) printLinkedList(LinkedList * lista):
   - Imprime os objetos armazenados na lista vinculada.

Error Handling

O tratamento de erros é implementado para falhas de alocação de memória durante a criação da lista vinculada. A desalocação adequada de memória é garantida em funções que removem nós da lista.

Conclusão

Em resumo, linkedList.c fornece uma implementação robusta de uma estrutura de dados de lista vinculada junto com operações para inserção, exclusão e impressão de elementos. Essa estrutura de dados flexível pode ser usada para armazenar e gerenciar vários tipos de objetos de forma eficiente.

parser.c

O arquivo Parser.c contém a implementação de funções responsáveis por analisar cadeias de comandos e gerenciar relatórios de tarefas. Estas funções são essenciais para processar comandos de entrada e gerar relatórios de tarefas em um ambiente de computação distribuído.

Command Parsing Funções:

1) countCommands(const char * cmd):
   - Conta o número de comandos separados por caracteres de barra vertical (|) em uma determinada sequência de comandos.
2) countTokens(const char *cmd):
   - Conta o número de tokens (palavras) em uma sequência de comandos.
3) cmdTok(const char *cmd):
   - Tokenizes uma sequência de comandos em uma matriz de sequências que representam comandos individuais.
4) parsePipe(const char *cmdPipe):
   - Analisa uma sequência de comandos contendo vários comandos separados por caracteres verticais (|) em uma matriz multidimensional de tokens de comando.

Query and Pipeline Management Funções

5) printQuery(Consulta cmd):

- Imprime uma consulta representada como uma matriz de strings.

6) freeQuery(consulta tok):
    - Deslocar memória alocada para uma consulta representada como uma matriz de strings.
7) freeCmdPipeline (consulta ** pipeline):
    - Deslocar memória alocada para um pipeline de comando, que é uma matriz de consultas

Task Report Funções

8) writeTaskReport(int id, muito tempo):
    - Grava um relatório de tarefa contendo o ID da tarefa e o tempo de execução em um arquivo de log.
9) nomeFifo(int pid):
    - Gera um nome de arquivo de pipe nomeado (FIFO) com base no ID do processo (pid).
10) writeReply(int fd, int id):
    - Grava uma mensagem de resposta em um descritor de arquivo (fd) indicando o recebimento de uma solicitação com um determinado ID

Error Handling

O tratamento adequado de erros é implementado usando perror() para exibir mensagens de erro em caso de falhas durante as operações do arquivo. A memória alocada dinamicamente é desalocada adequadamente usando free() para evitar vazamentos de memória.

Conclusão

Em resumo, parser.c fornece funções essenciais para analisar comandos, gerenciar relatórios de tarefas e lidar com a comunicação entre processos em um ambiente de computação distribuído. Essas funções facilitam o processamento e a execução eficientes de tarefas em vários processos.

request.c

O arquivo request.c fornece a implementação de funções relacionadas ao tratamento de solicitações em um ambiente de computação distribuído. Essas funções facilitam a criação, manipulação e execução de solicitações recebidas de clientes.

Estrutura request: Define a estrutura de uma solicitação contendo um ID, tempo de execução e uma string representando comandos

Funções

1) Funções de criação:

- createRequest (int id, int time, comandos de comando): Cria e inicializa uma solicitação com o ID, tempo de execução e string de comando fornecidos.

2) Funções do acessador:
   - setRid(Request * request, int id): Define o ID de uma solicitação.
   - setRtime(Request * request, int time): Define o tempo de execução de uma solicitação.
   - getRid(Request * request): Recupera o ID de uma solicitação.
   - getRtime(Request * request): Recupera o tempo de execução de uma solicitação.
   - getRCommand(Request * request): Recupera a string de comando de uma solicitação.
3) Funções de manipulação:
   - destroyRequest(void * request): Destrói um objeto de solicitação e libera memória alocada.
   - printRequest(Request * r): Imprime os detalhes de uma solicitação.
4) Funções de I/O:
   - writeRequest(const char * filename, Request * r): Grava uma solicitação em um arquivo.
   - fdWriteRequest(int fd, Request * request): Grava uma solicitação em um descritor de arquivo.
   - readRequest(const char * filename): Lê uma solicitação de um arquivo.
   - fdReadRequest(int fd): Lê uma solicitação de um descritor de arquivo.
5) Funções do utilitário:
   - copyRequest(Request * request): Cria uma cópia profunda de um objeto de solicitação.
   - compareRequest(void *a, void *b): compara duas solicitações com base em seus tempos de execução.
   - executeRequest(Request * request): Executa os comandos especificados em uma solicitação.

Error Handling

O tratamento adequado de erros é implementado usando perror() para exibir mensagens de erro em caso de falhas durante as operações do arquivo. A memória alocada dinamicamente é desalocada adequadamente usando free() para evitar vazamentos de memória.

Conclusão

Em resumo, request.c fornece um conjunto abrangente de funções para criar, manipular e executar solicitações em um ambiente de computação distribuído. Estas funções permitem o processamento e execução eficientes das solicitações dos clientes, contribuindo para a funcionalidade geral do sistema.

Conclusão

Com base na análise dos diversos componentes do programa e nos desafios enfrentados durante o desenvolvimento, é possível concluir que houve um esforço considerável para implementar um sistema robusto de execução e orquestração de tarefas em um ambiente distribuído.

Os arquivos client.c, orchestrator.c, linkedList.c, parser.c e request.c desempenham funções essenciais, desde a interação com os clientes até a gestão eficiente das tarefas e comunicação entre os componentes do sistema. Cada um desses arquivos foi projetado com atenção aos detalhes e inclui tratamento adequado de erros para garantir a confiabilidade do sistema.

Apesar dos desafios encontrados, como os erros identificados no status e no stats-command, o trabalho conseguiu atender às especificações do enunciado, implementando funcionalidades como preservação do estado do servidor, leitura e escrita de informações relevantes dos pedidos e controle de concorrência sobre as estatísticas.

É importante ressaltar que, mesmo com os contratempos enfrentados, o resultado final demonstra um bom entendimento e aplicação de conceitos abordados.

No futuro, com mais experiência e domínio das system calls, é esperado que a equipe possa alcançar um desempenho ainda melhor em projetos semelhantes, aproveitando os aprendizados adquiridos nesta experiência.

Em suma, o trabalho realizado representa um esforço significativo na implementação de um sistema de execução e orquestração de tarefas em um ambiente distribuído, com potencial para melhorias contínuas e aprendizado para projetos futuros.