UNIVALI – UNIVERSIDADE DO VALE DO ITAJAI

CIENCIA DA COMPUTAÇÃO

ENRICO BELO RIBEIRO & PAULO KUNRATH

RELATÓRIO

Servidor com pool threads

Itajai

2024

Sobre o Projeto

O objetivo do projeto é desenvolver um **servidor local** que deve estar sempre disponível para processar requisições de clientes que solicitam **números** ou **strings**. O servidor usa uma **pool de threads** para otimizar o processamento de múltiplas requisições simultâneas, evitando a criação contínua de novas threads, o que reduziria a performance. A comunicação entre o servidor e os clientes é feita utilizando **pipes nomeados (FIFO)**.

- **Algumas threads** são responsáveis por processar requisições de **números**.
- **Outras threads** são responsáveis por processar requisições de **strings**.
- Cada cliente sabe previamente se deseja um **número** ou uma **string** e se conecta ao pipe correspondente para fazer a requisição.
- O servidor deve responder de acordo com o pipe de requisição, enviando um número ou uma string, conforme o tipo de cliente.

Explicação e Contexto de Aplicação

A comunicação entre processos (**IPC - Inter-Process Communication**) é essencial em sistemas onde processos distintos precisam trocar informações de maneira eficiente. Neste caso, estamos desenvolvendo um sistema que envolve um **servidor** que responde a múltiplas requisições de **clientes**. O servidor utiliza um **pool de threads** para garantir que as requisições sejam processadas de maneira simultânea, sem necessidade de criar uma nova thread para cada requisição.

A comunicação entre o **servidor** e os **clientes** é feita via **pipes nomeados (FIFO)**, que permitem que processos diferentes, mas no mesmo sistema, se comuniquem de maneira eficiente. Os **named pipes** são arquivos especiais no sistema de arquivos que podem ser abertos para leitura ou escrita por diferentes processos.

Objetivo da Solução

- **Clientes**: O cliente que deseja um **número** se conecta ao **pipe de números**, enquanto o cliente que deseja uma **string** se conecta ao **pipe de strings**.
- **Servidor**: O servidor, por meio de um **pool de threads**, processa essas requisições em threads específicas para **números** ou **strings** e responde ao cliente.

Essa abordagem reduz o overhead de criação contínua de threads, melhorando a eficiência do servidor e escalabilidade em cenários de múltiplas requisições.

Resultados obtidos durante os Testes

Teste 1: Requisição de Número

- Cliente faz a requisição de um número e recebe o mesmo;

```
@EnricoBRibeiro ➜/workspaces/SO-Codes (main) $ python3 cliente.py
[Cliente] Enviando requisição: Quero um número
[Cliente] Resposta recebida: 71
```

- Resposta do servidor

```
@EnricoBRibeiro ➜/workspaces/SO-Codes (main) $ python3 servidor.py

[Servidor] Servidor iniciado e aguardando requisições...

[Servidor] Requisição recebida: Quero um número
[Servidor] Respondendo com número: 71
```

Teste 2: Requisição de String

- Cliente faz a requisição de uma string ao servidor e recebe a mesma;

```
[Cliente] Enviando requisição: Quero uma string
[Cliente] Resposta recebida: Olá
```

- Resposta do servidor

```
[Servidor] Requisição recebida: Quero uma string
[Servidor] Respondendo com string: Olá
```

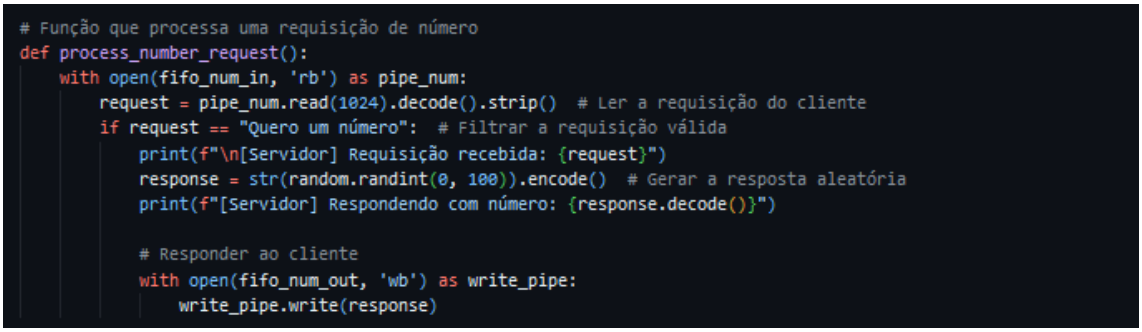
Observações:

- As requisições de números e strings foram processadas corretamente.
- O servidor foi capaz de lidar com as requisições simultâneas sem sobrecarga ou erros, aproveitando a pool de threads para dividir a carga de trabalho.

Sobre o Código

Abaixo estão as principais funções para o funcionamento do código de **Servidor** (servidor.py) e **Cliente** (cliente.py) e uma breve explicação sobre cada;

Servidor



```python
# Função que processa uma requisição de número
def process_number_request():
    with open(fifo_num_in, 'rb') as pipe_num:
        request = pipe_num.read(1024).decode().strip()  # Ler a requisição do cliente
        if request == "Quero um número":  # Filtrar a requisição válida
            print(f"\n[Servidor] Requisição recebida: {request}")
            response = str(random.randint(0, 100)).encode()  # Gerar a resposta aleatória
            print(f"[Servidor] Respondendo com número: {response.decode()}")

            # Responder ao cliente
            with open(fifo_num_out, 'wb') as write_pipe:
                write_pipe.write(response)
```

- Objetivo: Essa função processa a requisição de número enviada pelo cliente e responde com um número aleatório.

- Passos:
    - Ler a Requisição: A função abre o pipe de requisição de números (fifo_num_in) no modo de leitura ('rb') e lê a mensagem enviada pelo cliente.

    - Gerar Resposta: Se a requisição for "Quero um número", a função gera um número aleatório entre 0 e 100.

    - Enviar Resposta: Após gerar a resposta, a função abre o pipe de resposta de números (fifo_num_out) no modo de escrita ('wb') e envia o número gerado de volta ao cliente.

```python
# Função que processa uma requisição de string
def process_string_request():
    with open(fifo_str_in, 'rb') as pipe_str:
        request = pipe_str.read(1024).decode().strip()  # Ler a requisição do cliente
        if request == "Quero uma string":  # Filtrar a requisição válida
            print(f"\n[Servidor] Requisição recebida: {request}")
            response = random.choice(["Olá", "Mundo", "Teste", "Servidor"]).encode()  # Resposta aleatória
            print(f"[Servidor] Respondendo com string: {response.decode()}")

            # Responder ao cliente
            with open(fifo_str_out, 'wb') as write_pipe:
                write_pipe.write(response)
```

- Objetivo: Esta função processa a requisição de string enviada pelo cliente e responde com uma string aleatória.

- Passos:
    - Ler a Requisição: A função abre o pipe de requisição de strings (fifo_str_in) no modo de leitura ('rb') e lê a mensagem enviada pelo cliente.

    - Gerar Resposta: Se a requisição for "Quero uma string", a função seleciona uma string aleatória da lista ["Olá", "Mundo", "Teste", "Servidor"].

    - Enviar Resposta: Após gerar a string, a função abre o pipe de resposta de strings (fifo_str_out) no modo de escrita ('wb') e envia a string de volta ao cliente.

```python
# Função para iniciar o servidor com pool de threads
def start_server():
    # Criando os named pipes para números e strings
    if not os.path.exists(fifo_num_in):
        os.mkfifo(fifo_num_in)
    if not os.path.exists(fifo_num_out):
        os.mkfifo(fifo_num_out)
    if not os.path.exists(fifo_str_in):
        os.mkfifo(fifo_str_in)
    if not os.path.exists(fifo_str_out):
        os.mkfifo(fifo_str_out)

    print("\n[Servidor] Servidor iniciado e aguardando requisições...")

    # Criando a pool de threads com até 4 threads simultâneas
    with ThreadPoolExecutor(max_workers=4) as executor:
        while True:
            # Escuta as requisições e delega para a pool de threads
            executor.submit(process_number_request)
            executor.submit(process_string_request)
            time.sleep(1)
```

- Objetivo: Esta função configura o servidor para começar a processar requisições de números e strings usando uma pool de threads.

- Passos:
    - Criar Pipes: Se os pipes nomeados para números e strings ainda não existirem, a função os cria usando os.mkfifo().

    - Iniciar a Pool de Threads: A função usa ThreadPoolExecutor para criar uma pool de threads com até 4 threads simultâneas.

    - Processamento de Requisições: Dentro de um loop infinito, a função delega as tarefas de processar requisições de números e strings para a pool de threads usando executor.submit().

Cliente

```python
# Cliente que solicita um número
def client_number_request():
    with open(fifo_num_in, 'wb') as pipe_num:
        request = "Quero um número"
        print(f"[Cliente] Enviando requisição: {request}")
        pipe_num.write(request.encode())  # Enviar a requisição

    with open(fifo_num_out, 'rb') as pipe_num:
        response = pipe_num.read(1024).decode()  # Leitura da resposta
        print(f"[Cliente] Resposta recebida: {response}")
```

- Objetivo: Essa função envia uma requisição de número ao servidor e aguarda a resposta.

- Passos:
    - Abertura do Pipe de Entrada: A função abre o pipe de requisição de números (fifo_num_in) no modo de escrita ('wb') e envia a mensagem "Quero um número" para o servidor.

    - Aguarda a Resposta: Depois de enviar a requisição, a função abre o pipe de resposta de números (fifo_num_out) no modo de leitura ('rb') e lê a resposta do servidor, que é um número gerado aleatoriamente.

    - Saída: A função imprime a resposta recebida.

```python
# Cliente que solicita uma string
def client_string_request():
    with open(fifo_str_in, 'wb') as pipe_str:
        request = "Quero uma string"
        print(f"[Cliente] Enviando requisição: {request}")
        pipe_str.write(request.encode())  # Enviar a requisição

    with open(fifo_str_out, 'rb') as pipe_str:
        response = pipe_str.read(1024).decode()  # Leitura da resposta
        print(f"[Cliente] Resposta recebida: {response}")
```

- Objetivo: Esta função envia uma requisição de string ao servidor e aguarda a resposta.

- Passos:
    - Abertura do Pipe de Entrada: A função abre o pipe de requisição de strings (fifo_str_in) no modo de escrita ('wb') e envia a mensagem "Quero uma string" para o servidor.

    - Aguarda a Resposta: Após enviar a requisição, a função abre o pipe de resposta de strings (fifo_str_out) no modo de leitura ('rb') e lê a resposta do servidor, que é uma string aleatória.

    - Saída: A função exibe a resposta recebida.

Análise e discussão sobre os resultados

A implementação foi **bem-sucedida** em atender os requisitos do projeto:

- **Pool de Threads:** A utilização de um ThreadPoolExecutor é uma solução eficiente, já que permite que múltiplas threads sejam reutilizadas para atender as requisições, evitando a sobrecarga de criar e destruir threads constantemente. Isso torna o servidor mais escalável e eficiente.
- **Named Pipes (FIFO):** A escolha de usar pipes nomeados foi eficaz para comunicação entre processos. Eles permitem que o servidor e os clientes troquem dados de maneira simples e direta, sem necessidade de um sistema de rede ou processos complicados.