Relatório M1: Sistema de Servidor com Pool de Threads para Comunicação via Pipes

Autor: Erik Luiz Gervasi
Disciplina: Sistemas Operacionais
Professor: Felipe Viel
Instituição: Universidade do Vale do Itajaí (Univali)
Data de entrega: 17/09/2024

O objetivo deste trabalho é desenvolver um sistema de servidor local que utiliza pool de threads para atender a múltiplas requisições de clientes simultaneamente. O servidor responde a dois tipos de requisições: números aleatórios e strings aleatórias. A comunicação entre o servidor e os clientes é feita via pipes nomeados.

Explicação e Contexto da Aplicação

O projeto explora conceitos de **Interprocess Communication (IPC)**, **threads**, **concorrência** e **paralelismo**. O uso de pool de threads foi escolhido para melhorar o desempenho do servidor, evitando a sobrecarga de criação e destruição de threads para cada requisição recebida.

A comunicação por **pipes nomeados** foi utilizada para permitir que clientes se conectem ao servidor e enviem ou recebam informações, simulando um ambiente de comunicação entre processos. A aplicação foi desenvolvida em Python, utilizando a biblioteca win32pipe para a criação e manipulação dos pipes, e ThreadPoolExecutor para a criação do pool de threads.

Resultados Obtidos com as Simulações

Durante a execução do sistema, o servidor foi capaz de responder a requisições de múltiplos clientes simultaneamente, cada um conectado a um pipe específico. Clientes que requisitavam números aleatórios se conectavam ao pipe numeros_pipe, enquanto clientes que requisitavam strings aleatórias se conectavam ao pipe strings_pipe.

Testamos o sistema com dois clientes ativos, um para cada tipo de pipe, e o servidor respondeu corretamente às requisições enviando números e strings aleatórios. O sistema foi capaz de manter uma comunicação contínua, com as threads do servidor gerenciando as requisições de forma eficiente e sem bloqueios.

Tabela de Resultados:

Cliente	Tipo de Requisição	Resposta Recebida
Cliente 1	Números	Numero: 42
Cliente 2	Strings	String: "O código funcionou"
Cliente 1	Números	Numero: 17
Cliente 2	Strings	String: "Teste mensagem aleatória"

Códigos Importantes da Implementação

Servidor.py (código do servidor):

```
import win32pipe, win32file, pywintypes
import threading
import random
import time
from concurrent.futures import ThreadPoolExecutor

frases = [
    "Aula de Sistemas Operacionais",
    "O código funcionou",
    "Teste mensagem aleatória"
]

def handle_numeros(pipe_name):
    try:
        pipe = win32pipe.CreateNamedPipe(
            pipe_name,
            win32pipe.PIPE_ACCESS_OUTBOUND,
            win32pipe.PIPE_TYPE_MESSAGE | win32pipe.PIPE_WAIT,
            10, 65536, 65536, 0, None)
        win32pipe.ConnectNamedPipe(pipe, None)
        while True:
            time.sleep(random.randint(1, 3))
            num = random.randint(1, 100)
            message = f"Numero: {num}\n".encode('utf-8')
            win32file.WriteFile(pipe, message)
    except pywintypes.error as e:
        print(f"Erro: {e}")
```

```
    finally:
        win32pipe.DisconnectNamedPipe(pipe)
        win32file.CloseHandle(pipe)

def handle_strings(pipe_name):
    try:
        pipe = win32pipe.CreateNamedPipe(
            pipe_name,
            win32pipe.PIPE_ACCESS_OUTBOUND,
            win32pipe.PIPE_TYPE_MESSAGE | win32pipe.PIPE_WAIT,
            10, 65536, 65536, 0, None)
        win32pipe.ConnectNamedPipe(pipe, None)
        while True:
            time.sleep(random.randint(1, 3))
            frase = random.choice(frases)
            message = f"String: {frase}\n".encode('utf-8')
            win32file.WriteFile(pipe, message)
    except pywintypes.error as e:
        print(f"Erro: {e}")
    finally:
        win32pipe.DisconnectNamedPipe(pipe)
        win32file.CloseHandle(pipe)

def servidor():
    print("Servidor rodando e esperando por conexões...")
    with ThreadPoolExecutor(max_workers=10) as executor:
        while True:
            executor.submit(handle_numeros, r'\\.\pipe\numeros_pipe')
            executor.submit(handle_strings, r'\\.\pipe\strings_pipe')

if __name__ == "__main__":
    servidor()
```

Cliente.py (código do cliente):

```
import win32file, pywintypes
import time

def receber_dados(pipe_name):
    pipe = None
    try:
        while True:
            try:
                pipe = win32file.CreateFile(
                    pipe_name,
                    win32file.GENERIC_READ,
```

```
                0,
                None,
                win32file.OPEN_EXISTING,
                0,
                None
            )
            break
        except pywintypes.error as e:
            if e.winerror == 231:
                time.sleep(2)
            else:
                return

    while True:
        result, data = win32file.ReadFile(pipe, 1024)
        if data:
            message = data.decode('utf-8')
            print(f"Recebido: {message.strip()}")
            time.sleep(1)
except pywintypes.error as e:
    print(f"Erro: {e}")
finally:
    if pipe:
        win32file.CloseHandle(pipe)

if __name__ == "__main__":
    escolha = input("1 - Números\n2 - Strings\nEscolha: ")
    pipe_name = r'\\.\pipe\numeros_pipe' if escolha == '1' else r'\\.\pipe\strings_pipe'
    while True:
        receber_dados(pipe_name)
```

Análise e Discussão dos Resultados Finais

O sistema de servidor utilizando pool de threads para atender múltiplos clientes via pipes nomeados demonstrou ser eficiente, permitindo comunicação simultânea e contínua com clientes conectados. O uso de ThreadPoolExecutor ajudou a otimizar o desempenho, evitando a sobrecarga de criação de novas threads a cada requisição.

O sistema foi testado com sucesso em um ambiente de simulação, com threads separadas respondendo a diferentes tipos de requisições. Embora o sistema tenha funcionado conforme esperado, algumas melhorias futuras incluem a otimização do tratamento de erros e a criação de mais clientes para testar a escalabilidade do servidor.