Relatório: 10
Código: 9,0
Nota: 9,5

UFRJ

Universidade Federal do Rio de Janeiro
Engenharia de Computação e Informação

Trabalho prático 1
Sistemas distribuídos COS 470

Prof. Daniel Ratton Figueiredo

Lucas Máximo Dantas
Amanda Aparecida Cordeiro Lucio

2022.1
Rio de Janeiro

1. Decisões de Implementação

1.1. Linguagem de programação

A linguagem de programação escolhida foi **c**, sendo a mesma uma linguagem de baixo nível, desta forma, facilitando e permitindo a interação com o hardware. A escolha da linguagem baseou-se, principalmente, na nossa proximidade com a linguagem, visto que era necessário otimizar o tempo devido às nossas outras demandas da faculdade.

1.2. Sistema Operacional

Para o desenvolvimento, decidimos utilizar a *SysCall* do linux, sendo a distribuição utilizada o Ubuntu.

1.3. Implementação

A implementação pode ser acompanhada através do seguinte repositório: https://github.com/AmandaACLucio/Sistemas-Distribuidos

2. Implementações

2.1. Signals

Para Signals foram desenvolvidas duas funções, sendo uma responsável por enviar um sinal a qualquer outro processo e a outra responsável por capturar e tratar sinais diferentes, imprimindo uma mensagem específica para cada sinal. Os processos UNIX possuem um número de identificação (PID) e a partir desses id únicos é possível decidir para qual processo o sinal será enviado. Além disso, os sinais UNIX também possuem um número de identificação, dos quais foram escolhidos 3 sinais: SIGUSR1 (10); SIGUSR2 (12); SIGQUIT (3), sendo o SIGQUIT o responsável por finalizar o programa.

A função responsável por enviar sinal realiza essa função através do pid do processo desejado. Primeiramente verifica-se se o processo existe, através da função kill com parâmetro 0. Existindo, um sinal é enviado

utilizando a função kill do c, direcionando o número do sinal UNIX e o número de identificação do processo.

Enquanto isso, a função destinada ao recebimento de processos utiliza signal handler para gerar impressões de acordo com o sinal recebido. A espera pelo sinal dessa função pode ocorrer de duas formas, por busy wait e blocking wait. Busy Wait consiste em um loop infinito aguardando por determinado sinal. O outro modo de espera, blocking wait também é construído com um loop infinito, no entanto o processo permanece em espera até o recebimento do sinal.

2.2. Pipes

Para essa implementação, foram geradas duas funções, uma para o produtor e uma para o consumidor. Essas funções foram chamadas através da função fork e pipe para criar a comunicação entre os processos. O fork gera dois processos, o pai e o filho, identificados pelo PID. Sendo assim, para utilização do pipe, o filho foi definido como o produtor e o pai foi definido como o consumidor.

Os inputs deste programa são feitos na função main, responsável também pela criação do fork. Desta forma, o produtor gera números aleatórios com base no número anterior e escreve em um lado do pipe. Ao passo disso, conforme o Sistema Operacional escalona os processos, o consumidor irá ler os dados deste pipe em uma outra fiação, converter para inteiro e posteriormente verificar se esse dado recebido é um primo ou nao, imprimindo uma mensagem.

2.3. Sockets

A implementação do programa que utiliza sockets, foi dividido em dois programas, um para realizar o papel do produtor(cliente) e outro para o consumidor (servidor). Os dois programas utilizam a porta 8080 para comunicação através do protocolo TCP.

Esse sistema de sockets, foi realizado para funcionar da seguinte forma: primeiro, executamos o programa consumidor(servidor), que fica aguardando receber alguma mensagem, logo em seguida, executamos o segundo programa, o produtor (cliente), que por sua vez inicia calculando o primeiro número aleatório, e envia para o servidor, o servidor responde se o número se trata de um número aleatório ou não, e é impresso no console do cliente o resultado de cada um dos 5 números, como pode ser visto a seguir.

```
(base) max@elementaryosdesktop712ef8a3:~/Documentos/Programas/Sistemas-Distribuidos/3.Socket$ ./cliente
83 o numero nao e primo.
169 o numero e primorimo.
246 o numero e primorimo.
261 o numero e primorimo.
354 o numero e primorimo.
fim programa cliente
(base) max@elementaryosdesktop712ef8a3:~/Documentos/Programas/Sistemas-Distribuidos/3.Socket$
```

No lado do servidor, a mensagens com os números aleatórios são recebidas, e lidas proceduralmente, após o servidor realizar suas respectivas respostas, como pode ser observado:

```
(base) max@elementaryosdesktop712ef8a3:~/Documentos/Programas/Sistemas-Distribuidos/3.Socket$ ./servidor
83 nao e primo
169 e primo
246 e primo
261 e primo
354 e primo
```

Comentários código:
Não separaram a parte de sinais em 2 programas (-0.5pts).
O código de sockets poderia ter sido melhor modularizado (-0,5pts).