UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
ESCOLA POLITÉCNICA DE ENGENHARIA
ENGENHARIA DE COMPUTAÇÃO E INFORMAÇÃO
SISTEMAS DISTRIBUÍDOS - 2021.1

GABRIEL XARÁ
ANA LUCIA CANTO

Objetivo

Este relatório foi desenvolvido para apresentar os resultados e aprendizados obtidos durante a implementação do primeiro trabalho prático da disciplina.
O objetivo do trabalho era testar, na prática, os principais mecanismos de IPC (Interprocess Communication): Sinais, Pipes e Sockets.

Decisões do projeto

Após deliberar sobre as características das linguagens de programação que dominamos, optamos pela utilização do C + +. Os motivos que nos deram respaldo a essa decisão foram:

- Por se tratar de uma linguagem de baixo nível (maior proximidade com o sistema operacional), a linguagem nos oferece recursos interessantes na criação de estruturas de dados e no acesso direto à memória.
- Ambos os integrantes da dupla já possuíam experiência prévia com a linguagem
- A linguagem habilita o paradigma de programação orientado a objetos (entretanto este paradigma não foi utilizado para este primeiro trabalho).

Implementação

Nesta seção serão apresentados os detalhes de desenvolvimento que consideramos mais importantes. Além disso, serão exibidos algumas demonstrações da execução dos programas criados, de modo que possam ser avaliados como estudos de caso.

Sinais

Para esta tarefa foram implementados dois métodos:

- signal_sender: utilizado para disparar sinais para um outro processo qualquer, com a ajuda do sistema operacional.

Para efetuar o envio de sinais utilizamos o método “kill”, definido na biblioteca **signal**. Apesar do nome, este método permite o envio de qualquer sinal, e não apenas sinais para terminação do programa de destino.

- signal_handler: utilizado a fim de se testar o recebimento de um sinal e implementar no programa diferentes comportamentos para tipos distintos de sinais capturados.
  Optamos por utilizar recursos disponibilizados pela biblioteca **csignal**, que facilita a criação de *handlers* customizados para sinais recebidos pelo processo.
  Implementamos no programa 3 *signal handlers* diferentes, sendo eles:
  sigABRTsignalHandler -> Dispara uma mensagem “Signal ABRT (‘ 1 ‘) received.” quando o processo capturar um sinal SIGABRT (que por padrão indica uma terminação anormal do programa).
  sigFPEsignalHandler -> Dispara uma mensagem “Signal SIGFPE (‘ 2 ‘) received.” quando o processo capturar um sinal SIGFPE.
  sigINTsignalHandler -> Dispara uma mensagem “Signal SIGINT (‘ 4 ‘) received. The execution will stop” quando o processo captura um sinal SIGINT, e termina a execução do programa.

```
if (strcmp(argv[1], "block") == 0)
{
    while (true)
    {
        cout << "Waiting for a signal ... " << endl;
        sleep(4);
    }
}
else if (strcmp(argv[1], "busy") == 0)
{
    while (true)
    {
        cout << "Waiting for a signal ... " << endl;
    }
}
```

Implementação de blocking wait e busy wait

Pipes

A segunda etapa do trabalho consistia na implementação de dois programas que deveriam se comunicar com a utilização de pipes. Um dos programas, o produtor, deveria gerar números inteiros em uma sequência crescente, a serem enviados para o programa consumidor. O consumidor por sua vez deve verificar, para cada número recebido, se o mesmo é um número primo.
A implementação que fizemos recebe como parâmetro de linha de comando a quantidade de mensagens (números inteiros) a serem geradas e enviadas pelo programa produtor.

```
gxara@ubuntu:~/.../TP1/pipes(main)$ make run messages=7
# @make clean
make[1]: Entering directory '/home/gxara/Workspaces/UFRJ/Disciplinas/Sistemas Distribu
Compiling program
g++ -std=c++11 src/pipes.cpp -o bin/tp1_pipes
make[1]: Leaving directory '/home/gxara/Workspaces/UFRJ/Disciplinas/Sistemas Distribuí

11 is prime
16 is NOT prime
24 is NOT prime
25 is NOT prime
31 is prime
39 is NOT prime
40 is NOT prime
```

Funcionamento do programa com o parâmetro messages=7

```
gxara@ubuntu:~/.../TP1/pipes(main)$ make run messages=15
# @make clean
make[1]: Entering directory '/home/gxara/Workspaces/UFRJ/Disciplinas/Sistemas Distribuídos/Trabalhos Práticos/TP1/pipes'
Compiling program
g++ -std=c++11 src/pipes.cpp -o bin/tp1_pipes
make[1]: Leaving directory '/home/gxara/Workspaces/UFRJ/Disciplinas/Sistemas Distribuídos/Trabalhos Práticos/TP1/pipes'

2 is prime
11 is prime
16 is NOT prime
17 is prime
26 is NOT prime
28 is NOT prime
37 is prime
41 is prime
44 is NOT prime
50 is NOT prime
55 is NOT prime
57 is NOT prime
62 is NOT prime
69 is NOT prime
76 is NOT prime
```

Funcionamento do programa com o parâmetro messages=15

Sockets

A proposta desta tarefa era implementar novamente o desafio de produtores-consumidores, mas, desta vez, com a utilização de sockets. Neste modelo de IPC o programa consumidor (server.cpp) é exposto em um endereço IPv4 + Porta, e aguarda por solicitações enviadas por um ou mais programas produtores (client.cpp) com os números a serem testados.
A parte mais complicada da implementação deste último modelo de IPC foi efetuar a criação bem sucedida de um socket. A solução foi abstraída da seguinte forma:

```cpp
int createSocket()
{
    cout << "Will create socket \n";
    int mySocket = socket(
        AF_INET,      // IPV4 Family
        SOCK_STREAM, // SOCK_STREAM for TCP and SOCK_DGRAM for UDP
        0            //Protocol code ...  0 for Internet Protocol(IP)
    );

    return mySocket;
};

int main(int argc, char **argv)
{
    struct sockaddr_in serv_addr;
    int addrlen = sizeof(serv_addr);
    int valread;
    char buffer[1024] = {0};
    int myFirstCppSocket;
    int myFirstCppSocketFd = createSocket();
    if (myFirstCppSocketFd < 0)
    {
        cout << "\n Socket creation error \n";
        return -1;
    };

    cout << "Socket created" << endl;
    ...
```

```
gxara@ubuntu:~/.../TP1/sockets(main)$ make run_client N=7
# @make clean
make[1]: Entering directory '/home/gxara/Workspaces/UFRJ/Disciplinas/Sistemas Distribuídos/Trabalhos Práticos/TP1/sockets'
Compiling program client
g++ -std=c++11 src/client.cpp -o bin/tp1_client
make[1]: Leaving directory '/home/gxara/Workspaces/UFRJ/Disciplinas/Sistemas Distribuídos/Trabalhos Práticos/TP1/sockets'

Sending number: 19... Received that this number is prime
Sending number: 38... Received that this number is prime
Sending number: 57... Received that this number is prime
Sending number: 76... Received that this number is NOT prime
Sending number: 95... Received that this number is NOT prime
Sending number: 114... Received that this number is NOT prime
Sending number: 133... Received that this number is NOT prime
Encerrando execução ...
gxara@ubuntu:~/.../TP1/sockets(main)$
```

Código Fonte

O código pode ser encontrado em https://github.com/gxara/cos470-sd