

# Relatório 1 - Sistemas Distribuídos

Guilherme Bergman de Souza  
Thiago Guimarães Rebello Mendonça de Alcântara

Julho 2021

## 1 Decisões de projeto

Utilizou-se a linguagem C++ para escrever os programas deste trabalho. Essa decisão leva em consideração o amplo acesso a bibliotecas que disponibilizam funcionalidades do sistema operacional no C++. Os programas foram escritos para serem executados em ambientes UNIX. Todos os códigos apresentados foram compilados com sucesso em uma máquina Linux distribuição Manjaro. Os programas apresentaram resultados condizentes com os requisitos definidos, esses resultados serão apresentados em figuras ao longo do relatório.

Todo o processo de desenvolvimento foi feito utilizando as ferramentas git, a extensão VSCode Liveshare (ferramenta para pair programming) e o github para armazenamento. Todo o código pode ser encontrado [aqui](#). Cada uma das soluções possui seu próprio Makefile, para facilitar a compilação dos códigos.

## 2 Sinais

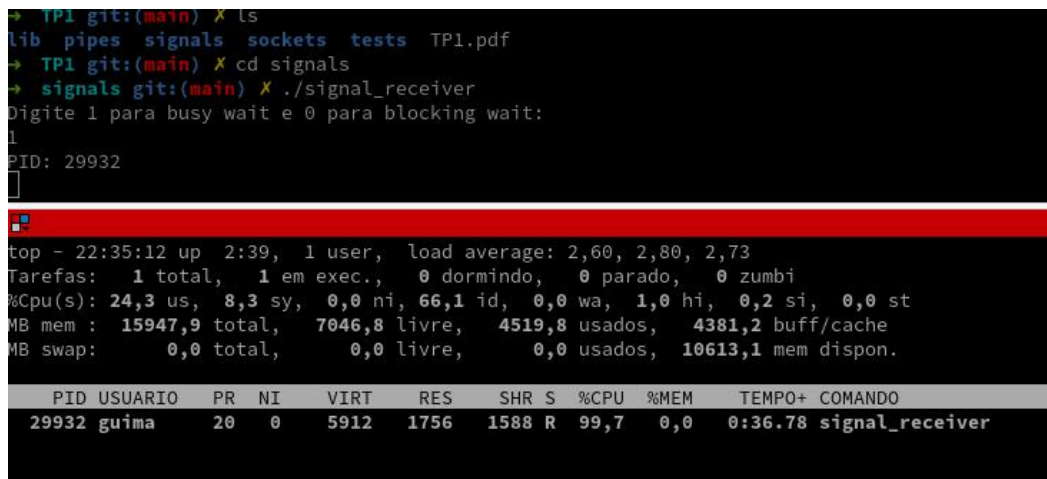
Para a tarefa de sinais foram escritos dois programas. O primeiro realiza a função de enviar um sinal a outro processo, o segundo recebe o sinal e imprime uma mensagem diferente para cada processo recebido.

Ao iniciar o programa receptor, é pedido para o usuário escolher entre o modo busy wait ou blocking wait. No busy wait, o programa espera o sinal de forma a continuar rodando na CPU, rodando um loop infinito. Assim, o processo continua rodando e utilizando tempo de CPU. No blocking wait, para evitar que o programa fique utilizando a CPU quando não necessário, o processo é colocado em espera, de forma a não utilizar a CPU. Isto pode ser observado nas Figuras 1 e 2.

Ao iniciar o programa emissor, é pedido para o usuário fornecer o sinal e PID do processo alvo.

A biblioteca signal.h foi utilizada para acessar os mecanismos de sinais. Os três sinais escolhidos para serem tratados são: SIGUSR1, SIGUSR2, SIGABRT - o SIGABRT foi o sinal escolhido para finalizar o programa.

Para testar, rodamos os programas para os casos possíveis, como pode ser conferido na Figura 3.



```
+ TP1 git:(main) X ls
lib pipes signals sockets tests TP1.pdf
+ TP1 git:(main) X cd signals
+ signals git:(main) X ./signal_receiver
Digite 1 para busy wait e 0 para blocking wait:
1
PID: 29932
[ ]

top - 22:35:12 up 2:39, 1 user, load average: 2,60, 2,80, 2,73
Tarefas: 1 total, 1 em exec., 0 dormindo, 0 parado, 0 zumbi
%Cpu(s): 24,3 us, 8,3 sy, 0,0 ni, 66,1 id, 0,0 wa, 1,0 hi, 0,2 si, 0,0 st
MB mem : 15947,9 total, 7046,8 livre, 4519,8 usados, 4381,2 buff/cache
MB swap: 0,0 total, 0,0 livre, 0,0 usados, 10613,1 mem dispon.

  PID USUARIO PR NI  VIRT  RES   SHR S  %CPU %MEM  TEMPO+ COMANDO
 29932 guima  20  0   5912  1756  1588 R   99,7  0,0  0:36.78 signal_receiver
```

Figure 1: Busy Waiting

```
→ signals git:(main) X ./signal_receiver
Digite 1 para busy wait e 0 para blocking wait:
0
PID: 30344

top - 22:35:42 up 2:39, 1 user, load average: 2,78, 2,83, 2,75
Tarefas: 1 total, 0 em exec., 1 dormindo, 0 parado, 0 zumbi
%Cpu(s): 11,7 us, 5,3 sy, 0,0 ni, 82,2 id, 0,0 wa, 0,6 hi, 0,2 si, 0,0 st
MB mem : 15947,9 total, 7082,4 livre, 4486,5 usados, 4378,9 buff/cache
MB swap: 0,0 total, 0,0 livre, 0,0 usados, 10648,7 mem dispon.

  PID  USUARIO  PR  NI  VIRT  RES  SHR  S  %CPU  %MEM  TEMPO+  COMANDO
  30344 guima   20   0   5912  1876 1708  S   0,0   0,0   0:00.00 signal_receiver
```

Figure 2: Blocking Waiting

```
distributed-systems/TP1/signals
guima@agumon:~/Projetos/SurfNight/distributed-systems/TP1/signals 117x20
→ signals git:(main) X ./signal_receiver
Digite 1 para busy wait e 0 para blocking wait:
1
PID: 19634
Received signal 10
Received signal 12
Received signal 6
→ signals git:(main) X ./signal_receiver
Digite 1 para busy wait e 0 para blocking wait:
0
PID: 20710
Received signal 10
Received signal 12
Received signal 6
→ signals git:(main) X

guima@agumon:~/Projetos/SurfNight/distributed-systems/TP1/signals 117x19
→ signals git:(main) X ./signal_sender
Insira o PID e o sinal
19634 10
→ signals git:(main) X ./signal_sender
Insira o PID e o sinal
19634 12
→ signals git:(main) X ./signal_sender
Insira o PID e o sinal
19634 6
→ signals git:(main) X ./signal_sender
Insira o PID e o sinal
20710 10
→ signals git:(main) X ./signal_sender
Insira o PID e o sinal
20710 12
→ signals git:(main) X ./signal_sender
Insira o PID e o sinal
20710 6
→ signals git:(main) X
```

Figure 3: Casos de Teste - Signals

```
+ pipes git:(main) X ./producer_consumer
Insira quantos números serão enviados no pipe:
30
85 não é primo.
172 não é primo.
250 não é primo.
266 não é primo.
360 não é primo.
396 não é primo.
483 não é primo.
576 não é primo.
626 não é primo.
648 não é primo.
711 não é primo.
739 é primo.
830 não é primo.
890 não é primo.
954 não é primo.
981 não é primo.
1022 não é primo.
1049 é primo.
1122 não é primo.
1159 não é primo.
1171 é primo.
1240 não é primo.
1308 não é primo.
1338 não é primo.
1421 não é primo.
1452 não é primo.
1515 não é primo.
1539 não é primo.
1607 é primo.
1643 não é primo.
+ pipes git:(main) X
```

Figure 4: Teste - Pipes

### 3 Pipes

O programa Produtor-Consumidor foi implementado através de um único binário que cria um pipe e utiliza a função fork. Após o fork, o processo pai executa a função producer, e o processo filho executa a função consumer. Ambos os processos têm acesso a um vetor, cuja a primeira posição indica o file descriptor de leitura do pipe, e a segunda posição o file descriptor de escrita.

A função producer pede para o usuário fornecer quantos números serão escritos no pipe. A função consumer utiliza a função isPrime para determinar se os valores lidos no pipes são primos ou não.

Os valores são escritos no pipe como strings de 30 bytes. Para converter os números de inteiro para string, e de string para inteiro, são utilizadas as funções sprintf e atoi respectivamente.

Para testar, rodamos o programa para diversos valores, como exemplificado na Figura 4.

### 4 Sockets

Para a tarefa de sockets, foram escritos dois programas. O primeiro realiza a função de consumidor, e ao mesmo tempo servidor da conexão TCP. O segundo programa age como produtor e cliente da conexão.

O programa produtor envia os número através da função send, e espera a resposta logo em seguida de forma bloqueante através da função read. Para gerar a resposta o programa consumidor utiliza a mesma função isPrime da tarefa da Seção 3. Também foi utilizado a mesma codificação dos valores em string da Seção 3.

Para testar, rodamos os dois programas, consumidor (servidor) e produtor (cliente), como exemplificado na Figura 5.

```
guima@agumon:~/Projetos/SurfNight/distribut
→ sockets git:(main) X ./consumer
Server listening na porta 8080
Received 85...
Received 172...
Received 250...
Received 266...
Received 360...
Received 396...
Received 483...
Received 576...
Received 626...
Received 648...
→ sockets git:(main) X

guima@agumon:~/Projetos/SurfNight/distribut
→ sockets git:(main) X ./producer
Insira o IPv4 do servidor:
127.0.0.1
Insira quantos números serão enviados ao servidor:
10
Sending 85...
[CONSUMER]: 85 não é primo.
Sending 172...
[CONSUMER]: 172 não é primo.
Sending 250...
[CONSUMER]: 250 não é primo.
Sending 266...
[CONSUMER]: 266 não é primo.
Sending 360...
[CONSUMER]: 360 não é primo.
Sending 396...
[CONSUMER]: 396 não é primo.
Sending 483...
[CONSUMER]: 483 não é primo.
Sending 576...
[CONSUMER]: 576 não é primo.
Sending 626...
[CONSUMER]: 626 não é primo.
Sending 648...
[CONSUMER]: 648 não é primo.
→ sockets git:(main) X
```

Figure 5: Teste - Sockets