



INF1316 - Laboratório 07 - Troca de mensagens em sistemas Unix

Nome:

- Guilherme Riechert Senko
- Pedro de Almeida Barizon

Matrícula:

- 2011478
- 2211350

Professor: Luiz Fernando Seibel

Turma: 3WA

Data: 18/11/2024

Objetivo

Implementar e compreender a troca de mensagens entre processos em sistemas Unix com auxílio da biblioteca padrão POSIX `sys/msg.h`.

Observações preliminares

- Para mais detalhes, todos os códigos fonte comentados encontram-se em anexo. Por isso, para evitar redundâncias, não serão aqui exibidos.
- Todos os arquivos fonte foram compilados com auxílio do GNU C Compiler (GCC) fazendo `gcc -Wall -o programa fonte1.c fonte2.c ... fonten.c` e executados com `./programa arg1 arg2 ... argn`. Caso haja particularidades, estas serão abordadas ao longo do relatório.

Exercício 1.a)

Enunciado

Faça um programa que utilize a troca de mensagens para trocar 64 mensagens entre dois processos/programas independentes. O processo 1 escreve as mensagens para serem tratadas pelo processo 2. A primeira forma de comunicação deve ser síncrona. O processo 1 envia a mensagem e aguarda resposta.

1.a.1 Estrutura

São criados dois processos: **sender** e **receiver**, utilizando `fork()`. Ambos os processos compartilham uma fila de mensagens criada com `msgget(IPC_PRIVATE, S_IRWXU)`. O tipo das mensagens é definido como `TYPE_SND` para mensagens enviadas pelo `sender` e `TYPE_RCV` para as respostas do `receiver`.

1.a.2 Solução

O `sender` envia mensagens sequenciais formatadas como `msg1`, `msg2`, ... até `msg64`. A cada envio, aguarda uma resposta do `receiver` para continuar. Já o `receiver` recebe mensagens do `sender`, imprime seu conteúdo, e responde com mensagens formatadas como `rcv1`, `rcv2`, ...

1.a.3 Observações e conclusões

A comunicação síncrona foi garantida, porque o `sender` aguardava explicitamente a resposta antes de enviar a próxima mensagem. Isso se deve à implementação de `msgrcv`, que faz o processo leitor aguardar até que o tipo de mensagem buscado apareça na fila. Dessa forma, definindo-se os tipos distintos `TYPE_SND` e `TYPE_RCV`, asseguraram-se a concorrência e a exclusão mútua de acesso às mensagens, uma vez que `sender` e `receiver` buscavam por tipos diferentes, não tendo sido explicitamente colocados em pausa por qualquer outro mecanismo. Assim, dispensou-se o uso de semáforos.

Apesar disso, por certo desencargo de consciência típico da programação concorrente, criou-se uma versão com semáforos, chamada `ex_a_sem.c`, disponível em anexo.

Por fim, o comportamento esperado foi atingido: todas as 64 mensagens foram trocadas com sucesso, e os resultados foram impressos corretamente em ambas as extremidades, conforme demonstra a saída do programa. Por simplicidade, será exibida apenas a metade final da saída:

```
Snder recebeu: rcv34
Rcver recebeu: msg35
Snder recebeu: rcv35
Rcver recebeu: msg36
Snder recebeu: rcv36
Rcver recebeu: msg37
Snder recebeu: rcv37
Rcver recebeu: msg38
Snder recebeu: rcv38
Rcver recebeu: msg39
Snder recebeu: rcv39
Rcver recebeu: msg40
Snder recebeu: rcv40
Rcver recebeu: msg41
Snder recebeu: rcv41
Rcver recebeu: msg42
Snder recebeu: rcv42
Rcver recebeu: msg43
Snder recebeu: rcv43
Rcver recebeu: msg44
Snder recebeu: rcv44
Rcver recebeu: msg45
Snder recebeu: rcv45
Rcver recebeu: msg46
Snder recebeu: rcv46
Rcver recebeu: msg47
Snder recebeu: rcv47
Rcver recebeu: msg48
Snder recebeu: rcv48
Rcver recebeu: msg49
Snder recebeu: rcv49
Rcver recebeu: msg50
Snder recebeu: rcv50
Rcver recebeu: msg51
Snder recebeu: rcv51
Rcver recebeu: msg52
Snder recebeu: rcv52
Rcver recebeu: msg53
Snder recebeu: rcv53
Rcver recebeu: msg54
Snder recebeu: rcv54
Rcver recebeu: msg55
Snder recebeu: rcv55
Rcver recebeu: msg56
Snder recebeu: rcv56
Rcver recebeu: msg57
Snder recebeu: rcv57
Rcver recebeu: msg58
Snder recebeu: rcv58
Rcver recebeu: msg59
Snder recebeu: rcv59
Rcver recebeu: msg60
Snder recebeu: rcv60
Rcver recebeu: msg61
Snder recebeu: rcv61
Rcver recebeu: msg62
Snder recebeu: rcv62
Rcver recebeu: msg63
Snder recebeu: rcv63
Rcver recebeu: msg64
Snder recebeu: rcv64
○ barizon@barizon-Lenovo-IdeaPad-S145-15IWL:~/Documents/INF1316 LABS/lab07 seibels
```

Figura 1: Parte final da saída de `ex_a`

Exercício 1.b)

Enunciado

Faça um programa que utilize a troca de mensagens para trocar 64 mensagens entre dois processos/programas independentes. Na segunda forma de comunicação, o buffer de comunicação entre os processos é capaz de armazenar apenas 8 mensagens.

1.b.1 Estrutura

Similar ao 1.a), são criados dois processos: `sender` e `receiver`. Neste caso, porém, a fila de mensagens é configurada com um limite de *buffer* ajustado para 8 mensagens por meio da alteração do campo `msg_qbytes` da estrutura `msqid_ds`, que define o limite de *bytes* na fila de mensagem. Como cada mensagem ocupa 15 *bytes*, basta definir `msg_qbytes = 8 * 15 = 120 bytes`.

1.b.2 Solução

A fila é criada, e seu limite de armazenamento ajustado com `msgctl()` e a macro `IPC_SET`. Em seguida, `sender` passa a enviar mensagens sequenciais, ao passo que monitora o número de mensagens na fila. Se o limite for ultrapassado, um alerta é exibido no console. Enquanto isso, `receiver` consome as mensagens da fila e imprime os conteúdos recebidos.

1.b.3 Observações e conclusões

A configuração do *buffer* limitado foi eficaz, garantindo que a fila não armazenasse mais do que 8 mensagens de cada vez. Para nos assegurarmos disso, definimos momentaneamente o limiar de aviso na console para 7, e alertas começaram a aparecer. Voltando a 8, cessaram. Dessa forma, constatamos que a fila, de fato, respeitava o limite definido em `msg_qbytes`. Diante disso, diferentemente da situação anterior, o `sender` precisou ajustar sua lógica para aguardar espaço na fila, o que assegurou uma troca eficiente.

Além disso, cabe mencionar que a concorrência harmoniosa entre os processos foi assegurada, porque, se bem que estes acessassem um mesmo recurso (a fila de mensagens), o faziam com papéis diferentes: produtor (*sender*) e consumidor (*receiver*). Assim, enquanto *receiver* consumia mensagens em uma extremidade da fila, *sender* produzia outras na extremidade oposta, de modo que não fossem ocasionados problemas. Admitimos, porém, não ter certeza se, caso a capacidade do *buffer* fosse ainda menor, problemas ocorreriam. Seja como fosse, poderíamos implementar semáforos análogos aos do exercício *ex_a_sem.c*.

Enfim, todos os 64 envios foram realizados com sucesso, e a limitação do *buffer* foi respeitada, conforme evidencia a saída do programa abaixo:

```
• barizon@barizon-Lenovo-IdeaPad-S145-15IWL:~/Documents/INF1316_LABS/lab07_seibels$ ./ex_b
Rcver recebeu: msg1
Rcver recebeu: msg2
Rcver recebeu: msg3
Rcver recebeu: msg4
Rcver recebeu: msg5
Rcver recebeu: msg6
Rcver recebeu: msg7
Rcver recebeu: msg8
Rcver recebeu: msg9
Rcver recebeu: msg10
Rcver recebeu: msg11
Rcver recebeu: msg12
Rcver recebeu: msg13
Rcver recebeu: msg14
Rcver recebeu: msg15
Rcver recebeu: msg16
Rcver recebeu: msg17
Rcver recebeu: msg18
Rcver recebeu: msg19
Rcver recebeu: msg20
Rcver recebeu: msg21
Rcver recebeu: msg22
Rcver recebeu: msg23
Rcver recebeu: msg24
Rcver recebeu: msg25
Rcver recebeu: msg26
Rcver recebeu: msg27
Rcver recebeu: msg28
Rcver recebeu: msg29
Rcver recebeu: msg30
Rcver recebeu: msg31
Rcver recebeu: msg32
Rcver recebeu: msg33
Rcver recebeu: msg34
Rcver recebeu: msg35
Rcver recebeu: msg36
Rcver recebeu: msg37
Rcver recebeu: msg38
Rcver recebeu: msg39
Rcver recebeu: msg40
Rcver recebeu: msg41
Rcver recebeu: msg42
Rcver recebeu: msg43
Rcver recebeu: msg44
Rcver recebeu: msg45
Rcver recebeu: msg46
Rcver recebeu: msg47
Rcver recebeu: msg48
Rcver recebeu: msg49
Rcver recebeu: msg50
Rcver recebeu: msg51
Rcver recebeu: msg52
Rcver recebeu: msg53
Rcver recebeu: msg54
Rcver recebeu: msg55
Rcver recebeu: msg56
Rcver recebeu: msg57
Rcver recebeu: msg58
Rcver recebeu: msg59
Rcver recebeu: msg60
Rcver recebeu: msg61
Rcver recebeu: msg62
Rcver recebeu: msg63
Rcver recebeu: msg64
• barizon@barizon-Lenovo-IdeaPad-S145-15IWL:~/Documents/INF1316_LABS/lab07_seibels$
```

Figura 2: Saída de ex_b

Conclusões finais

Os dois exercícios demonstraram a flexibilidade e o poder das filas de mensagens POSIX para implementar comunicação interprocessual. No caso síncrono (1.a), a espera ativa entre envios e recebimentos foi determinante para garantir a troca completa de mensagens. No caso do *buffer* limitado (1.b), a configuração de atributos da fila foi crucial para forçar uma dinâmica de sincronização implícita entre os processos.

Este laboratório destacou aspectos fundamentais da troca de mensagens, como controle de concorrência, limitações de *buffer*, e o impacto de diferentes abordagens de sincronização na implementação de sistemas Unix.