Daniel Schreiber Guimarães – 1910462

Laboratório 5 – Troca de Mensagens – Entrega Final

O código completo de cada questão está no final do arquivo.

Questão 1

Para implementação, foi usada a estrutura de troca de mensagens disponibilizada em `<sys/msg.h>`.

O processo produtor coloca na fila de mensagens um inteiro gerado randomicamente, e dorme por um segundo. Ele repete esse processo 128 vezes, e depois fecha. Os processos consumidores acessam a fila de mensagens buscando algum inteiro. Se eles não encontram, não fazem nada de especial. Após isso, eles são colocados para dormir por algum tempo (um ou três segundos), e são finalizados depois de consumirem 128 inteiros, ou percebem que o consumidor não vai mais enviar nenhuma mensagem (para evitar ficarem rodando para sempre).

O código do consumidor:

```
    pid = fork();
    if (pid == 0) {
        // processo filho 1, consumidor
        int quantidade = 0;
        Resposta r;
        while (*variavelCompartilhada < MAX_EXEC) {
            if (msgrcv(msgid, &r, 4, 1, IPC_NOWAIT) == -1) {
                printf("[C1] lido: -- | total: %d\n", quantidade);
            } else {
                quantidade += 1;
                *variavelCompartilhada += 1;
                printf("[C1] lido: %2d | total: %d\n", r.mtext[0], quantidade);
            }
            sleep(TEMPO_CONSUMIDOR1);
        }
        printf("[C1] Total lido: %d. Fechando.\n", quantidade);
        exit(0);
    } else {
        printf("C1 criado! (pid: %d)\n", pid);
    }
```

`variavelCompartilhada` é a variável que mostra quantas execuções o consumidor fez. Quando for menor que `MAX_EXEC` , ou seja, 128, ele irá parar. Ela está em uma memória compartilhada.

`msgrcv` é a função para verificar uma nova mensagem. Se for coletada com sucesso, ela é armazenada na estrutura `Resposta`, cujo conteúdo é exibido, assim como o total de mensagens recebidas por aquele consumidor. A flag avisa para não esperar por alguma mensagem daquele tipo, mas continuar a execução

Um exemplo de exibição do primeiro consumidor:

```
[C1] lido: 14 | total: 81
```

Caso não tenha uma mensagem, ele exibirá “—” no lugar do inteiro lido.

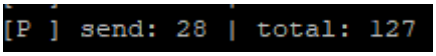
O código do produtor:

```
// processo pai, gerador
    srand(time(NULL));
    int quantidade = 0;
    while (quantidade < MAX_EXEC) {
        Mensagem m;
        struct msqid_ds buf;
        if (msgctl(msgid, IPC_STAT, &buf) == -1) {
            perror("Nao foi possivel obter informacoes da fila");
            exit(-4);
        }
        if (buf.msg_qnum < MAX_FILA) {
            m.mtype = 1;
            m.mtext[0] = rand() % 100;
            if (msgsnd(msgid, &m, 4, IPC_NOWAIT) == -1) {
                perror("Erro no envio de mensagem");
                exit(-5);
            } else {
                quantidade += 1;
                printf("[P ] send: %2d | total: %d\n", m.mtext[0], quantidade);
            }
        }
        sleep(TEMPO_PRODUTOR);
    }
```

A repetição funciona da mesma maneira. Porém, antes de enviar a mensagem, o comando `msgctl` é executado para coletar informações da fila. Essa informação é importante para verificar se a fila já não está cheia (com o limite definido da questão, de 32 mensagens).

Após isso, a mensagem é construída (com um tipo qualquer, desde que seja o mesmo que os consumidores esperam) com um inteiro aleatório de conteúdo, e é enviada.

O produtor também exibe uma mensagem ao enviar a mensagem, dizendo qual inteiro foi produzido, e quantas mensagens já foram produzidas até agora. Um exemplo de exibição:



Executando:

```
pi@raspberrypi:/mnt/rede/Daniel/rasc/sinais2 $ sudo ./q
Memoria compartilhada alocada.
Memoria compartilhada anexada.
Fila de mensagens criada.
C1 criado! (pid: 20062)
C2 criado! (pid: 20063)
[C1] lido: -- | total: 0
[P ] send: 25 | total: 1
[C2] lido: 25 | total: 1
[C1] lido: -- | total: 0
[P ] send: 20 | total: 2
[C1] lido: 20 | total: 1
[P ] send:  1 | total: 3
[C1] lido:  1 | total: 2
[P ] send: 79 | total: 4
[C2] lido: 79 | total: 2
[C1] lido: -- | total: 2
[P ] send: 56 | total: 5
[P ] send: 22 | total: 6
[C1] lido: 56 | total: 3
[C2] lido: 22 | total: 3
[C1] lido: -- | total: 3
[P ] send: 45 | total: 7
[C1] lido: 45 | total: 4
[P ] send: 33 | total: 8
[P ] send: 96 | total: 9
[C1] lido: 33 | total: 5
[C2] lido: 96 | total: 4
[C1] lido: -- | total: 5
[P ] send: 49 | total: 10
[P ] send: 90 | total: 11
```

Ao executar, é possível ver que o consumidor 1 quase sempre captura as mensagens, mas de vez em quando o consumidor 2 consegue capturar, mesmo executando menos vezes.

```
[C2] lido: 11 | total: 43
[C1] lido: -- | total: 83
[P ] send: 28 | total: 127
[C1] lido: 28 | total: 84
[P ] send: 57 | total: 128
[C1] lido: 57 | total: 85
[C2] Total lido: 43. Fechando.
pi@raspberrypi:/mnt/rede/Daniel/rasc/sinais2 $ [C1] Total lido: 85. Fechando.
```

No final da execução, foi possível observar que quase sempre o consumidor 2 ao tentar pegar uma mensagem, conseguiu pegar antes do consumidor 1, pois a proporção está bem próxima de 1:2 de mensagens. De fato, esse número é o maior valor possível de mensagens que o consumidor 2 poderia pegar (128 / 3). Caso os dois consumidores houvesse tempos mais competitivos de execução, essa proporção exata não seria a mesma.

Questão 2

Neste caso, o código é bem semelhante, porém a principal diferença é que o consumidor não dorme, ele espera até que tenha uma mensagem do tipo pré-definido, e só quando houver uma mensagem ele continua a execução.

Código do consumidor:

```
    if (fork() == 0) {
        // processo filho, que recebera as mensagens
        for (int i = 0; i < QUANTIDADE; i ++) {
            Resposta r;
            // le ou espera ate chegar uma mensagem
            // daquele tipo (tipo 1 para envio, tipo 1 para receber)

            if (msgrcv(msqid, &r, 4, 1, 0) == -1) {
                perror("Erro na leitura da mensagem");
                exit(-2);
            } else {
                printf("Chegou mensagem!: %d\n", r.mtext[0]);
            }
        }
        exit(0);
```

Para que o processo espere a mensagem, não foi utilizada a flag da questão anterior, `IPC_NOWAIT`. Caso houvesse a flag, o loop iria ficar repetindo sem parar, sempre vendo se havia alguma mensagem. Mas agora, isso permitiu uma comunicação síncrona (no caso, ao receber uma mensagem, o processo somente exibe o valor inteiro enviado pela mensagem).

Código do produtor:

```
    // processo pai, que envia as mensagens
        srand(time(NULL));
        for (int i = 0; i < QUANTIDADE; i ++) {
            Mensagem m;
            m.mtype = 1;
            m.mtext[0] = rand() % 500;

            if (msgsnd(msqid, &m, 4, IPC_NOWAIT) == -1) {
                perror("Erro no envio da mensagem");
                exit(-1);
            } else {
                printf("Enviou mensagem!: %d\n", m.mtext[0]);
            }
            sleep(1);
        }
```

O código é semelhante ao produtor da questão anterior.

Executando:

```
pi@raspberrypi:/mnt/rede/Daniel/rasc/sinais2 $ sudo ./q2
Fila criada:  2
Enviou mensagem!: 336
Chegou mensagem!: 336
Enviou mensagem!: 101
Chegou mensagem!: 101
Enviou mensagem!: 65
Chegou mensagem!: 65
Enviou mensagem!: 22
Chegou mensagem!: 22
Enviou mensagem!: 156
Chegou mensagem!: 156
Enviou mensagem!: 467
Chegou mensagem!: 467
Enviou mensagem!: 272
Chegou mensagem!: 272
Enviou mensagem!: 44
Chegou mensagem!: 44
Enviou mensagem!: 244
Chegou mensagem!: 244
Enviou mensagem!: 34
```

Não é possível observar nessa imagem, mas enquanto a mensagem de envio só aparece a cada um segundo, sempre que aparece o envio logo em seguida aparece a chegada. Ou seja, é uma comunicação síncrona.

OBS: O terminal é de um *Raspberry PI*, pois não consegui colocar para funcionar meu *WSL* novamente, por isso eu salvava o arquivo através de uma rede interna, e executava via uma conexão *ssh*.

Código completo da questão 1

```c
/*
Resolução:
processo gerador:
    * envia uma mensagem na fila com tipo 1, com o inteiro
    * adiciona 1 em uma variavel compartilhada
    * dorme
    * repete enquanto nao tiver enviado 32 elementos

processos consumidores:
    * tenta ler da fila algo do tipo 1
    * dorme algum tempo
    * repete enquanto uma variavel compartilhada nao for 128
*/

# include <stdio.h>
# include <stdlib.h>
# include <errno.h>

# include <unistd.h>
# include <sys/shm.h>
# include <sys/stat.h>
# include <sys/wait.h>
# include <time.h>

# include <sys/ipc.h>
# include <sys/msg.h>

const int MAX_FILA = 32;
const int MAX_EXEC = 128;
const int TIPO_MENSAGEM = 1;
const int CHAVE_FILA = 1234;
const int CHAVE_MEMORIA = 9998;
const int TEMPO_PRODUTOR = 1;
const int TEMPO_CONSUMIDOR1 = 1;
const int TEMPO_CONSUMIDOR2 = 3;

// estrutura da mensagem enviada
typedef struct msgtext {
    long mtype;
    int mtext[1];
} Mensagem;

// estrutura da mensagem recebida
typedef struct msgbuf {
    long mtype;
```

```
    int mtext[1];
} Resposta;


int main() {
    int segmento, i, status, pid, msgid;
    int *variavelCompartilhada;

    // alocando memoria compartilhada para a variavel
    segmento = shmget(CHAVE_MEMORIA, sizeof(int), IPC_CREAT | IPC_EXCL | S_IRUSR | S_IW
USR);
    if (segmento == -1) {
        perror("Nao foi possivel alocar memoria.");
        exit(-1);
    } else {
        printf("Memoria compartilhada alocada.\n");
    }

    // anexando a memoria compartilhada
    variavelCompartilhada = (int *)shmat(segmento, 0, 0);
    if ((int) variavelCompartilhada == -1) {
        perror("Nao foi possivel anexar a memoria compartilhada.");
        shmctl(segmento, IPC_RMID, 0);
        exit(-2);
    } else {
        printf("Memoria compartilhada anexada.\n");
    }
    *variavelCompartilhada = 0;

    // criando fila de mensagens
    msgid = msgget(CHAVE_FILA, IPC_CREAT | IPC_EXCL);
    if (msgid == -1) {
        perror("Nao foi possivel criar a fila");
        exit(-3);
    } else {
        printf("Fila de mensagens criada.\n");
    }

    // criando os processos filhos (consumidores)
    pid = fork();
    if (pid == 0) {
        // processo filho 1, consumidor
        int quantidade = 0;
        Resposta r;
        while (*variavelCompartilhada < MAX_EXEC) {
            if (msgrcv(msgid, &r, 4, 1, IPC_NOWAIT) == -1) {
```

```
            printf("[C1] lido: -- | total: %d\n", quantidade);
        } else {
            quantidade += 1;
            *variavelCompartilhada += 1;
            printf("[C1] lido: %2d | total: %d\n", r.mtext[0], quantidade);
        }
        sleep(TEMPO_CONSUMIDOR1);
    }
    printf("[C1] Total lido: %d. Fechando.\n", quantidade);
    exit(0);
} else {
    printf("C1 criado! (pid: %d)\n", pid);
}

pid = fork();
if (pid == 0) {
    // processo filho 2, consumidor
    int quantidade = 0;
    Resposta r;
    while (*variavelCompartilhada < MAX_EXEC) {
        if (msgrcv(msgid, &r, 4, 1, IPC_NOWAIT) == -1) {
            printf("[C2] lid: -- | total: %d\n", quantidade);
        } else {
            quantidade += 1;
            *variavelCompartilhada += 1;
            printf("[C2] lido: %2d | total: %d\n", r.mtext[0], quantidade);
        }
        sleep(TEMPO_CONSUMIDOR2);
    }
    printf("[C2] Total lido: %d. Fechando.\n", quantidade);
    exit(0);
} else {
    printf("C2 criado! (pid: %d)\n", pid);
}

// processo pai, gerador
srand(time(NULL));
int quantidade = 0;
while (quantidade < MAX_EXEC) {
    Mensagem m;
    struct msqid_ds buf;
    if (msgctl(msgid, IPC_STAT, &buf) == -1) {
        perror("Nao foi possivel obter informacoes da fila");
        exit(-4);
    }
    if (buf.msg_qnum < MAX_FILA) {
```

```
            m.mtype = 1;
            m.mtext[0] = rand() % 100;
            if (msgsnd(msgid, &m, 4, IPC_NOWAIT) == -1) {
                perror("Erro no envio de mensagem");
                exit(-5);
            } else {
                quantidade += 1;
                printf("[P ] send: %2d | total: %d\n", m.mtext[0], quantidade);
            }
        }
        sleep(TEMPO_PRODUTOR);
    }

    // fechando tudo
    wait(NULL);
    msgctl(msgid, IPC_RMID, NULL);
    shmdt(variavelCompartilhada);
    shmctl(segmento, IPC_RMID, 0);

    return 0;
}
```

Código completo da questão 2

```
# include <stdio.h>
# include <stdlib.h>
# include <unistd.h>
# include <time.h>
# include <sys/wait.h>
# include <string.h>

# include <sys/ipc.h>
# include <sys/msg.h>
# include <sys/stat.h>

# include <errno.h>

const int CHAVE_FILA = 1234;
const int QUANTIDADE = 64;

typedef struct msgtext {
    long mtype;
    int mtext[1];
} Mensagem;

typedef struct msgbuf {
```

```c
    long mtype;
    int mtext[1];
} Resposta;

int main() {
    int msqid;

    // criando a fila de mensagens
    msqid = msgget(CHAVE_FILA, IPC_CREAT | IPC_EXCL);
    if (msqid == -1) {
        perror("Erro criando fila de mensagens");
        exit(-1);
    } else {
        printf("Fila criada:  %d\n", msqid);
    }

    if (fork() == 0) {
        // processo filho, que recebera as mensagens
        for (int i = 0; i < QUANTIDADE; i ++) {
            Resposta r;
            // le ou espera ate chegar uma mensagem
            // daquele tipo (tipo 1 para envio, tipo 1 para receber)

            if (msgrcv(msqid, &r, 4, 1, 0) == -1) {
                perror("Erro na leitura da mensagem");
                exit(-2);
            } else {
                printf("Chegou mensagem!: %d\n", r.mtext[0]);
            }
        }
        exit(0);
    } else {
        // processo pai, que envia as mensagens
        srand(time(NULL));
        for (int i = 0; i < QUANTIDADE; i ++) {
            Mensagem m;
            m.mtype = 1;
            m.mtext[0] = rand() % 500;

            if (msgsnd(msqid, &m, 4, IPC_NOWAIT) == -1) {
                perror("Erro no envio da mensagem");
                exit(-1);
            } else {
                printf("Enviou mensagem!: %d\n", m.mtext[0]);
            }
            sleep(1);
```

```
        }
    }

    // fechando a fila e saindo
    wait(NULL);
    msgctl(msqid, IPC_RMID, NULL);
    return 0;
}
```