//Eric Goulart da Cunha - 2110878
//João Pedro Biscaia Fernandes - 2110361

Relatório Lab 6 Sistemas Operacionais

Implementação:
A estrutura de código é semelhante nos 2 itens. Criamos 2 processos, um chamado sender e outro receiver, um que manda mensagem, e outro que recebe.

A maior diferença é que, na letra a, como é síncrono, enviamos uma mensagem de confirmação do processo receiver para o processo sender. Isso é para impedir a concorrência, logo que um print pode ser mais “lento” que o envio da próxima mensagem.

Porém, na letra b, garantimos somente que existe um limite de bytes(32), enquanto a letra a possui de 4 bytes.
Código:

```c
int main(void) {
    pid_t sender, receiver;
    int msqid, status;
    struct msqid_ds buf;
    struct msgbuf aux;

    msqid = msgget(IPC_PRIVATE, IPC_CREAT | 0666);
    if (msqid == -1) {
        perror("msgget");
        exit(EXIT_FAILURE);
    }
```

criação da fila ^

```c
    if (msgctl(msqid, IPC_STAT, &buf) == -1) {
        perror("msgctl IPC_STAT");
        exit(EXIT_FAILURE);
    }

    buf.msg_qbytes = 4; // Permitir 1 msg de int na fila

    if (msgctl(msqid, IPC_SET, &buf) == -1) {
        perror("msgctl IPC_SET");
        exit(EXIT_FAILURE);
    }
```

Alteração da fila (nesse caso da letra a, para 4 bytes -> 1 inteiro - 1 msg por vez)

```c
    sender = fork();
    if (sender < 0) {
        perror("Error creating sender process");
        exit(EXIT_FAILURE);
    }
    if (sender == 0) {
        for (int i = 1; i <= 128; i++) {
            aux.mtype = 1;
            aux.mtext = i;
            if (msgsnd(msqid, &aux, sizeof(aux.mtext), 0) == -1) {
                perror("msgsnd");
                exit(EXIT_FAILURE);
            }
            printf("Enviou msg %d\n", aux.mtext);
            if(msgrcv(msqid, &aux2, sizeof(aux2.mtext),2,0)==-1){
                perror("msgrcv");
                exit(EXIT_FAILURE);
            }
        }
        exit(0);
```

Processo que envia mensagens(exemplo da letra_a);

```c
 if (sender == 0) {
     for (int i = 1; i <= 128; i++) {
         aux.mtype = 1;
         aux.mtext = i;
         if (msgsnd(msqid, &aux, sizeof(aux.mtext), 0) == -1) {
             perror("msgsnd");
             exit(EXIT_FAILURE);
         }
         printf("Enviou mensagem %d\n", aux.mtext);
     }
     exit(0);
 }
```

Processo que envia mensagens(letra_b)

```c
receiver = fork();
if (receiver < 0) {
    perror("Error creating receiver process");
    exit(EXIT_FAILURE);
}
if (receiver == 0) {
    for (int i = 1; i <= 128; i++) {
        if (msgrcv(msqid, &aux, sizeof(aux.mtext), 1, 0) == -1)
            perror("msgrcv");
            exit(EXIT_FAILURE);
        }
        printf("Recebeu message %d\n", aux.mtext);
        aux2.mtype = 2; // manda mensagem do tipo 2, de confirma
        aux2.mtext = i;
        if (msgsnd(msqid, &aux2, sizeof(aux.mtext), 0) == -1) {
            perror("msgsnd");
            exit(EXIT_FAILURE);
        }
    }
    exit(0);
}
```

Processo que recebe mensagens(letra_a, note que envia confirmação para sender)

```c
receiver = fork();
if (receiver < 0) {
    perror("Erro ao criar processo receptor");
    exit(EXIT_FAILURE);
}
if (receiver == 0) {
    for (int i = 1; i <= 128; i++) {
        if (msgrcv(msqid, &aux, sizeof(aux.mtext), 0, 0) == -1) {
            perror("msgrcv");
            exit(EXIT_FAILURE);
        }
        printf("Recebeu mensagem %d\n", aux.mtext);
    }
    exit(0);
}
```

Processo que recebe na letra b.

Com isso, garantimos a concorrência somente na letra_b, pois não há o controle síncrono do envio de mensagens, gerando o seguinte exemplo de saída:

```
Enviou mensagem 1
Recebeu mensagem 1
Enviou mensagem 2
Enviou mensagem 3
Recebeu mensagem 2
Enviou mensagem 4
Recebeu mensagem 3
Enviou mensagem 5
Recebeu mensagem 4
Enviou mensagem 6
Recebeu mensagem 5
Enviou mensagem 7
Recebeu mensagem 6
Enviou mensagem 8
Recebeu mensagem 7
Enviou mensagem 9
Recebeu mensagem 8
Enviou mensagem 10
Recebeu mensagem 9
Enviou mensagem 11
Recebeu mensagem 10
Enviou mensagem 12
Recebeu mensagem 11
Enviou mensagem 13
Recebeu mensagem 12
```

E a letra_a, sem concorrência:

```
Sending msg 1
Caught message 1
Sending msg 2
Caught message 2
Sending msg 3
Caught message 3
Sending msg 4
Caught message 4
Sending msg 5
Caught message 5
Sending msg 6
Caught message 6
Sending msg 7
Caught message 7
Sending msg 8
Caught message 8
Sending msg 9
Caught message 9
Sending msg 10
Caught message 10
Sending msg 11
Caught message 11
Sending msg 12
Caught message 12
Sending msg 13
```

Conclusões:
Tivemos dificuldades para entender a biblioteca sys/msg.h, e seus usos. Após entender o uso da função msgctl, conseguimos fazer as questões A e B de forma trivial. Aprendemos melhor conceitos de concorrência nesse laboratório, assim como uma útil biblioteca para troca de mensagens.