Pontifícia Universidade Católica do Rio de Janeiro

Lab 5 - Troca de Mensagens

Sistemas Operacionais

Alunos	Tharik Lourenço
	Suemy Inagaki
Professor	Luiz Fernando Seibel

Rio de Janeiro, 13 de maio de 2021

Conteúdo

1 Problema 1 **1**
1.1 Código Fonte 1
1.2 Resultado da Execução 4
1.3 Comentário 4

2 Problema 2 **5**
2.1 Código Fonte 5
2.2 Resultado da Execução 8
2.3 Comentário 8

1 Problema 1

Usando processos, escreva um programa C que implemente a troca de mensagens entre 3 processos, um gerando mensagens (produtor) e outros dois processos recebendo as mensagens (consumidores). Os processos compartilham os recursos (filas e demais dados). As mensagens consistem de números inteiros que são gerados pelo produtor e armazenados em um buffer a cada 1 segundo. Os processos consumidores devem retirar os dados da fila, um a cada 1 segundo e o outro a cada 3 segundos. O tamanho máximo da fila deve ser de 32 elementos e tanto o produtor como os dois consumidores (juntos) devem produzir/consumir 128 elementos. Gere um relatório que possibilite entender os processos, suas ações e o resultado de suas ações

1.1 Código Fonte

```
/*
Tharik Lourenço
Suemy Inagaki
Lab 05 Troca de Mensagens
Entrega Final
Sistemas Operacionais
*/

#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

#define KEY 123
#define MSG_SIZE_TEXT 256
#define NUMPROC 2
#define numMsg 32

/* estrutura msg associada as mensagens */
struct msgtext {
    long mtype ;
    int  mtext ;
```

```
};

void receive (int msqid, int t){

    struct msgtext msg;
    int count = 0;
    while(count < numMsg){
        sleep(t);
        while (msgrcv(msqid,&msg,sizeof(msg.mtext),1,IPC_NOWAIT)==-1);
        printf("pid %d leu mensagem: %d\n",getpid(), msg.mtext) ;
        count++;

    }
    return;
}

void sender (int msqid, int t){

    struct msgtext msg;
    int i =1;
    msg.mtype = 1 ;           /* tipo das mensagens */
    while(i<=numMsg*2){

        /* envia a mensagem a fila */
        msg.mtext = i;
        if(msgsnd(msqid,&msg,sizeof(msg.mtext),IPC_NOWAIT) == -1)
        {
            perror("Envio de mensagem impossivel") ;
            exit(-1) ;
        }
        printf("-->envio da mensagem: %d\n",msg.mtext) ;
        sleep(t);

        i++ ;
    }
    return;
}


int main( int argc, char *argv[] )
{
```

```
int i = 1 ;
int msqid ;
pid_t pid;

/*Criando fila de mensagens*/
if( (msqid = msgget(KEY, IPC_CREAT|0600)) == -1 ) {
    fprintf(stderr,"Impossivel criar a fila de mensagens!\n");
    exit(1);
}
struct msqid_ds buf;
msgctl(msqid, IPC_STAT , &buf );
buf.msg_qbytes = 128;
if(msgctl(msqid, IPC_SET , &buf ) == 0)
    printf("tamanho da fila alterado para 128\n");

for( i = 0; i < NUMPROC; i++ ) {
    if (fork() == 0){
        if(getpid() % 2){
            printf("Receptor iniciado 1 ...\n");
            receive(msqid, 1);

        }
        else {

            printf("Receptor iniciado 2 ...\n");
            receive(msqid, 3);

        }
        exit(0);
    }
}

// Processo pai
printf("\n");
sender(msqid,1);

for(i = 0 ; i<NUMPROC ; i++){
    wait(NULL);
}
// Removendo a fila de mensagens
if( msgctl(msqid,IPC_RMID,NULL) != 0 ) {
```

```
        fprintf(stderr,"Impossivel remover a fila!\n");
        exit(1);
    }
    exit(0);

}
```

1.2 Resultado da Execução

```
tamanho da fila alterado para 128
Receptor iniciado 1 ...

-->envio da mensagem: 1
Receptor iniciado 2 ...
pid 99 leu mensagem: 1
-->envio da mensagem: 2
pid 99 leu mensagem: 2
-->envio da mensagem: 3
pid 100 leu mensagem: 3
-->envio da mensagem: 4
pid 99 leu mensagem: 4
-->envio da mensagem: 5
pid 99 leu mensagem: 5
-->envio da mensagem: 6
pid 99 leu mensagem: 6
-->envio da mensagem: 7
pid 100 leu mensagem: 7
-->envio da mensagem: 8
pid 99 leu mensagem: 8
-->envio da mensagem: 9
pid 99 leu mensagem: 9
-->envio da mensagem: 10
pid 100 leu mensagem: 10
-->envio da mensagem: 11
pid 99 leu mensagem: 11
-->envio da mensagem: 12
pid 99 leu mensagem: 12
-->envio da mensagem: 13
pid 100 leu mensagem: 13
```

Figura 1: Caption

1.3 Comentário

O programa executa exatamente o que foi pedido pelo enunciado.

O processo pai cria uma fila de mensagens com 128 bytes e envia uma mensagem por segundo. O filho que tiver executando no momento recebe a mensagem e a remove da fila.

O Filho cujo pid é 99 lê uma mensagem a cada 1 segundo, equanto o outro Filho, com pid 100, lê uma mensagem a cada 3 segundos. Cada um lê a primeira mensagem da fila.

2 Problema 2

Implemente a troca de mensagens síncrona entre 2 processos. Serão geradas e consumidas 64 mensagens (números inteiros). Novamente descreva no relatório como funcionam os processos, suas ações e o resultado de suas ações

2.1 Código Fonte

```
/*
Tharik Lourenço
Suemy Inagaki
Lab 05 Troca de Mensagens
Entrega Final
Sistemas Operacionais
*/

#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

#define KEY 123
#define MSG_SIZE_TEXT 256
#define NUMPROC 1
#define numMsg 64

/* estrutura msg associada as mensagens */
struct msgtext {
```

```
    long mtype ;
    int  mtext ;
};

void receive (int msqid){

    struct msgtext msg;
    int count = 0;
    while(count < numMsg){
        while (msgrcv(msqid,&msg,sizeof(msg.mtext),1,IPC_NOWAIT)==-1);
        printf("pid %d leu mensagem: %d\n",getpid(), msg.mtext) ;
        count++;

    }
    return;
}

void sender (int msqid){

    struct msgtext msg;
    int i =1;
    msg.mtype = 1 ;           /* tipo das mensagens */
    while(i<=numMsg){

        /* envia a mensagem a fila */
        msg.mtext = i;
        if(msgsnd(msqid,&msg,sizeof(msg.mtext),IPC_NOWAIT) == -1)
        {
            perror("Envio de mensagem impossivel") ;
            exit(-1) ;
        }
        printf("-->envio da mensagem: %d\n",msg.mtext) ;

        i++ ;
    }
    return;
}


int main( int argc, char *argv[] )
{
```

```
    int i = 1 ;
    int msqid ;
    pid_t pid;

    /*Criando fila de mensagens*/
    if( (msqid = msgget(KEY, IPC_CREAT|0600)) == -1 ) {
        fprintf(stderr,"Impossivel criar a fila de mensagens!\n");
        exit(1);
    }
    struct msqid_ds buf;
    msgctl(msqid, IPC_STAT , &buf );
    buf.msg_qbytes = 128;
    if(msgctl(msqid, IPC_SET , &buf ) == 0)
        printf("tamanho da fila alterado para 128\n");

    for( i = 0; i < NUMPROC; i++ ) {
        if (fork() == 0){
            printf("Receptor iniciado 1 ...\n");
            receive(msqid);
            exit(0);
        }
    }

    // Processo pai
    printf("\n");
    sender(msqid);

    for(i = 0 ; i<NUMPROC ; i++){
        wait(NULL);
    }
    // Removendo a fila de mensagens
    if( msgctl(msqid,IPC_RMID,NULL) != 0 ) {
        fprintf(stderr,"Impossivel remover a fila!\n");
        exit(1);
    }
    exit(0);

}
```

2.2 Resultado da Execução

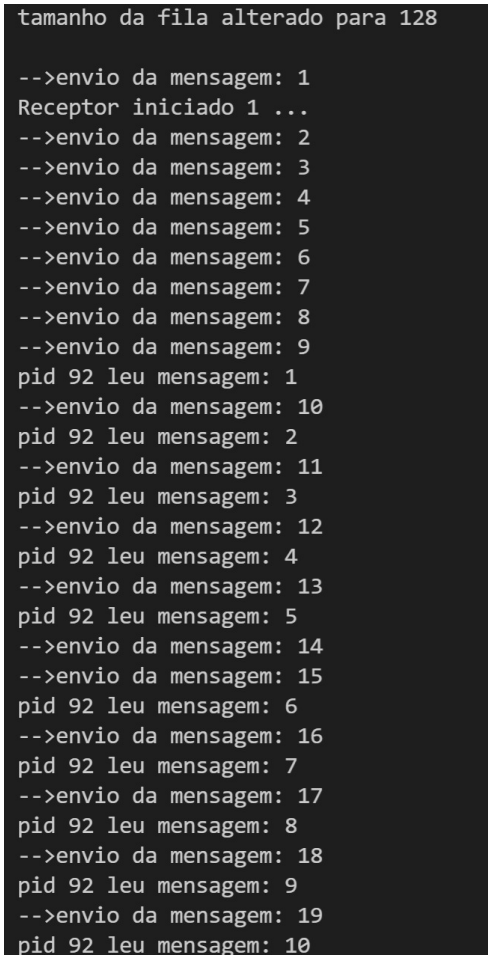

Figura 2: Caption

2.3 Comentário

No problema 2, o programa também executa como pedido pelo enunciado.

O processo pai cria a fila de mensagens e envia uma mensagem por segundo. E quando o filho inicia a execução, lê a primeira mensagem e a remove da fila. Os dois executam em concorrência, mas como o filho iniciou a execução depois do processo pai, existiam algumas mensagens na fila e o filho foi removendo uma por uma, ao mesmo tempo em que o pai enviava uma mensagem por segundo.