Laboratório 2

Hanna Epelboim Assunção 2310289
Erica Oliveira Regnier 2211893

Ex1: Paralelismo
Criar um vetor a de 10.000 posições inicializado com valor 5. Criar 10 processos trabalhadores que utilizam áreas diferentes do vetor para multiplicar a sua parcela do vetor por 2 e somar as posições do vetor retornando o resultado para um processo coordenador que irá apresentar a soma de todas as parcelas recebidas dos trabalhadores. Obs: O 1º trabalhador irá atuar nas primeiras 1.000 posições, o 2º trabalhador nas 1.000 posições seguintes e assim sucessivamente.

```c
//Érica Oliveira Regnier
//Hanna Epelboim Assunção

#include <stdio.h>

#include <stdlib.h>

#include <sys/shm.h>

#include <sys/wait.h>

#include <sys/types.h>

#include <unistd.h>

#define TAM_VETOR 10000 // Tamanho total do vetor

#define N_PROCESSES 10 // Número de processos

#define SEGMENT_SIZE (TAM_VETOR / N_PROCESSES) // Tamanho do segmento
para cada processo

int main() {

   int *p, segmento;
```

```
  // Criação da memória compartilhada

  segmento = shmget(IPC_PRIVATE, TAM_VETOR * sizeof(int), IPC_CREAT
| 0666);

  if (segmento == -1) {

      perror("Erro ao alocar memória compartilhada");

      exit(1);

  }

  // Associação da memória compartilhada ao processo principal

  p = (int *)shmat(segmento, NULL, 0);


  if (p == (void *)-1) {

      perror("Erro ao associar a memória compartilhada");

      exit(1);

  }

  // Preenchendo o vetor

  for (int i = 0; i < TAM_VETOR; i++) {

      p[i] = 5;

  }



  // Exibindo o vetor para referência
```

```
// printf("Vetor: ");
// for (int i = 0; i < TAM_VETOR; i++) {
//     printf("%d ", p[i]);
// }
// printf("\n");

// Criando processos para buscar a chave no vetor
for (int i = 0; i < N_PROCESSES; i++) {
    pid_t pid = fork();
    if (pid < 0) {
        perror("Erro na criação do novo processo");
        exit(2);
    } else if (pid == 0) {  // Processo filho
        for (int j = 0; j < SEGMENT_SIZE; j++) {
            int index = i * SEGMENT_SIZE + j;
            p[index] *= 2;
        }
        shmdt(p); // Desanexa a memória compartilhada
        exit(0);  // Termina o processo filho
```

```
        }

    }

    // Processo pai espera todos os filhos terminarem

    for (int i = 0; i < N_PROCESSES; i++) {

        int soma = 0;

        waitpid(-1,NULL,0);

        for (int j = 0; j < SEGMENT_SIZE; j++) {

                int index = i * SEGMENT_SIZE + j;

                soma += p[index];

            }

        printf("Processo: %d Soma: %d\n", i, soma);

    }

    // Libera a memória compartilhada

    shmdt(p);

    shmctl(segmento, IPC_RMID, NULL);

    return 0;

}
```

Comandos:

```
[c2310289@pantera-negra lab2sei]$ gcc -Wall -o ex1 ex1.c
[c2310289@pantera-negra lab2sei]$ ./ex1
```

Saída:

```
Processo: 0 Soma: 10000
Processo: 1 Soma: 10000
Processo: 2 Soma: 10000
Processo: 3 Soma: 10000
Processo: 4 Soma: 10000
Processo: 5 Soma: 10000
Processo: 6 Soma: 10000
Processo: 7 Soma: 10000
Processo: 8 Soma: 10000
Processo: 9 Soma: 10000
```

Explicação:

O programa usa a função shmget para criar um segmento de memória compartilhada, que será acessado por vários processos. O tamanho total da memória é baseado na constante TAM_VETOR, e a memória é dividida entre 10 processos. O vetor, alocado na memória compartilhada, é preenchido com o valor 5 em cada posição. Usando a função fork, o programa cria 10 processos filhos. Cada processo filho é responsável por multiplicar por 2 os valores de uma parte do vetor, calculando seu índice com base no segmento que o processo deve manipular. O processo pai espera a finalização de todos os processos filhos e, em seguida, calcula e imprime a soma dos valores de cada segmento do vetor para cada processo.

Ex2: concorrência
Considere o vetor de 10.000 posições inicializado com o valor 5. Crie 2 trabalhadores, ambos multiplicam por 2 e somam 2 em todas as posições do vetor. Verifique automaticamente se todas as posições têm valores iguais e explique o que ocorreu

Código fonte:

```
//Érica Oliveira Regnier
//Hanna Epelboim Assunção

#include <stdio.h>

#include <stdlib.h>

#include <sys/shm.h>

#include <sys/wait.h>

#include <sys/types.h>

#include <unistd.h>

#define TAM_VETOR 10000 // Tamanho total do vetor

#define N_PROCESSES 100 // Número de processos

int main() {

   int *p, segmento;

   // Criação da memória compartilhada

   segmento = shmget(IPC_PRIVATE, TAM_VETOR * sizeof(int), IPC_CREAT
| 0666);

   if (segmento == -1) {
```

```
    perror("Erro ao alocar memória compartilhada");
    exit(1);
}
// Associação da memória compartilhada ao processo principal
p = (int *)shmat(segmento, NULL, 0);

if (p == (void *)-1) {
    perror("Erro ao associar a memória compartilhada");
    exit(1);
}
// Preenchendo o vetor
for (int i = 0; i < TAM_VETOR; i++) {
    p[i] = 5;
}


// Exibindo o vetor para referência
printf("Vetor: ");
for (int i = 0; i < TAM_VETOR; i++) {
    printf("%d ", p[i]);
}
```

```
printf("\n");

// Criando processos para buscar a chave no vetor

for (int i = 0; i < N_PROCESSES; i++) {

    pid_t pid = fork();

    if (pid < 0) {

        perror("Erro na criação do novo processo");

        exit(2);

    } else if (pid == 0) {  // Processo filho

        for (int j = 0; j < TAM_VETOR; j++) {

            p[j] *= 2;

            p[j] += 2;


        }

        // Exibindo o vetor para referência

// printf("Vetor: ");

// for (int i = 0; i < TAM_VETOR; i++) {

//     printf("%d ", p[i]);

// }

printf("\n");
```

```
            shmdt(p); // Desanexa a memória compartilhada

            exit(0);  // Termina o processo filho

        }

    }

    // Processo pai espera todos os filhos terminarem

    for (int i = 0; i < N_PROCESSES; i++) {

        waitpid(-1,NULL,0);

    }

    int valor = p[0];

    int igual = 1;

    for (int j = 0; j < TAM_VETOR; j++) {

        if(p[j]!=valor){

            printf("Valores diferentes!!!\n Vetor[0] = %d\n Vetor[%d]
= %d\n", valor, j, p[j]);

            igual = 0;

        }

    }

    if(igual){
        printf("Todos os valores iguais à %d!!!\n", valor);
    }
```

```
    // Libera a memória compartilhada

    shmdt(p);

    shmctl(segmento, IPC_RMID, NULL);

    return 0;

}
```

Comandos:

```
eriquita@batatatriste:~$ gcc -Wall -o lab lab2.c
eriquita@batatatriste:~$ ./lab
```

Saída:

```
Valores diferentes!!!
 Vetor[0] = -2
 Vetor[9092] = -393218
eriquita@batatatriste:~$
```

Explicação:

O programa usa shmget para criar um segmento de memória compartilhada com a chave IPC_PRIVATE e tamanho suficiente para armazenar TAM_VETOR inteiros.
shmat associa o segmento de memória compartilhada ao espaço de endereçamento do processo principal. O vetor é inicialmente preenchido com o valor 5. Em seguida, 100 processos filhos são criados, e cada um modifica o vetor (multiplicando por 2 e adicionando 2). O processo pai espera que todos os filhos terminem, verifica se todos os valores no vetor são iguais e, finalmente, libera a memória compartilhada.
Os valores podem ser diferentes, como nesse caso, pois há a concorrência, assim, como vários processos atuam sobre a mesma área de memória, eles modificam os resultados uns dos outros, de forma que entre as operações, neste exemplo de multiplicar por 2 e somar 2, algum processo entrou no meio, dando resultados diferentes.

Ex3: Troca de mensagens via memória compartilhada
Mensagem do Dia
Faça um programa que: Leia da entrada uma mensagem do dia. Crie uma memória compartilhada
com a chave 7000. Salve a mensagem na memória compartilhada.
Faça um outro programa que utilize o mesmo valor de chave da memória compartilhada e dê attach na memória gerada pelo programa anterior. Em seguida este processo deve exibir a mensagem do dia para o usuário e deve liberar a memória compartilhada.
Observação: Atenção com os flags

ex3.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <unistd.h>
#include <sys/wait.h>
int main (int argc, char *argv[])
{
   int segmento;
   char mensagem[100], *p;

   printf("Digite sua mensagem: ");
   fgets(mensagem,sizeof(mensagem),stdin);

   // aloca a memória compartilhada
   segmento = shmget (7000, sizeof (mensagem), IPC_CREAT | IPC_EXCL |
S_IRUSR | S_IWUSR);
   if (segmento == -1) {
       perror("Erro ao alocar memória compartilhada");
       exit(1);
   }

   // associa a memória compartilhada ao processo
   p = (char *) shmat (segmento, 0, 0); // comparar o retorno com -1
   if (p == (void *)-1) {
       perror("Erro ao associar a memória compartilhada");
```

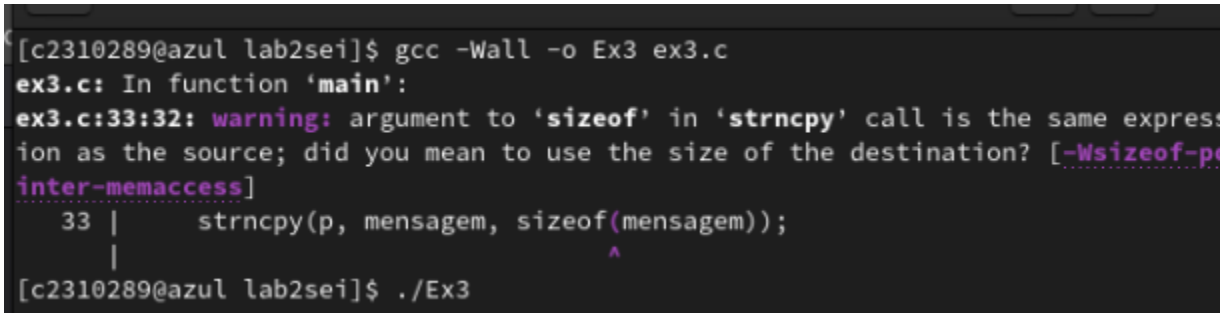
```
        exit(1);

    }

    mensagem[strcspn(mensagem, "\n")] = '\0';
    strncpy(p, mensagem, sizeof(mensagem));


    shmdt (p);

    return 0;
}
```

ex3Cliente.c

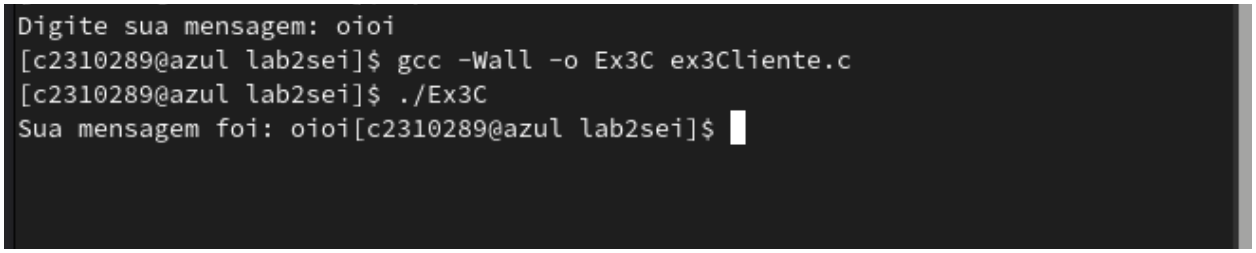
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <unistd.h>

int main(void){
    int segmento;
    char *p;

    segmento = shmget (7000,100,0);
    if (segmento == -1) {
        perror("Erro ao alocar memória compartilhada");
        exit(1);
    }

    p = (char *) shmat (segmento, 0, 0);
    if (p == (void *)-1) {
        perror("Erro ao associar a memória compartilhada");
        exit(1);
```

```
    }

    printf("Sua mensagem foi: %s",p);

    shmdt(p);

    return 0;
}
```

Comandos:



Saída:



Explicação:

ex3Servidor.c

Este programa é responsável por criar um segmento de memória compartilhada, escrever uma mensagem nele, e então desvincular o segmento de memória compartilhada. Ele cria um segmento de memória compartilhada com a chave `7000` e um tamanho igual ao tamanho da string (`mensagem`). Depois, ele associa o segmento de memória compartilhada ao processo. Se a operação falhar, o programa exibe uma mensagem de erro e termina. Em seguida, o programa lê uma mensagem e remove o caractere de nova linha, se presente. Por fim, ele copia a mensagem para o segmento de memória compartilhada.

ex3Cliente.c

Este programa é responsável por acessar o segmento de memória compartilhada, ler a mensagem e exibi-la. Primeiramente, ele acessa o segmento de memória compartilhada com a chave 7000 e tamanho 100. O terceiro argumento 0 indica que o segmento deve existir, mas não cria um novo. Em seguida, ele associa o segmento de memória compartilhada ao processo. Ao final, ele exibe a mensagem lida da memória compartilhada.