Código:

```c
/*
Faça um programa que gere 3 processos para alterar um valor de uma var
iável
na memoria compartilhada inicializada com zero.

* Cada programa executa um loop de 5000 execuções
* O processo 1 soma 1, o processo 2 soma 2, e o processo 3 soma 3

Execute o programa e verifique se houve problema de concorrência

Depois, utilize semáforos para possibilitar alterar a variável
(na sua região crítica). Descreva o que ocorreu
*/


# include <unistd.h>    // simbolos
# include <stdio.h>     // I/O
# include <stdlib.h>    // geral
# include <string.h>


# include <sys/shm.h>   // memoria compartilhada
# include <sys/wait.h>
# include <sys/stat.h>

# include <sys/sem.h>   // semaforos

# define EXECUCOES 500
# define COD_MEMORIA 1334
# define COD_SEMAFORO 1337

int *shared_int;         // variavel compartilhada

union semun {
    int val;
    struct semid_ds *buf;
    unsigned short *array;
};

int criar_memoria_compartilhada() {
    // criacao da memoria compartilhada
```

```
    int segmento, status;
    segmento = shmget(COD_MEMORIA, sizeof(int), IPC_CREAT | IPC_EXCL |
 S_IRUSR | S_IWUSR);
    if ((int) segmento == -1) {
        printf("Erro na alocacao de memoria\n");
        exit(-1);
    }
    // anexando memoria
    shared_int = (int *)shmat(segmento, 0, 0);
    if ((int) shared_int == -1) {
        printf("Erro na anexacao da memoria\n");
        exit(-2);
    }

    // inicializando a variavel
    *shared_int = 0;

    return segmento;
}

int criar_semaforo() {
    int s = semget(COD_SEMAFORO,              // codigo do semaforo
                   1,                         // somente um semaforo
                   0666 | IPC_CREAT | IPC_EXCL // criar e dar erro
    );
    union semun semUnion;
    semUnion.val = 1;

    if (semctl(s, 0, SETVAL, semUnion) == -1) {
        printf("Erro no set do semaforo\n");
        exit(-1);
    }
    return s;  // retorna o id do semaforo
}

void deletar_semaforo(int semaforo) {
    semctl(semaforo, 0, IPC_RMID, NULL);
}

void alterar_variavel(int x, int semaforo) {
    // setando o semaforo para P
    struct sembuf semB;
```

```c
    semB.sem_num = 0;
    semB.sem_op = -1;
    semB.sem_flg = SEM_UNDO;
    semop(semaforo, &semB, 1);

    // escrevendo
    *shared_int += x;

    // setando o semaforo para V
    semB.sem_op = 1;
    semop(semaforo, &semB, 1);
    return;
}

int main_sem_semaforo() {
    int seg = criar_memoria_compartilhada();
    int i;

    // criando os processos
    for (i = 0; i < 3; i ++) {
        if (fork() == 0) {
            // dentro de um processo filho.
            // printf("Iniciado o processo %d\n", i);
            for (int x = 0; x < EXECUCOES; x++) {
                *shared_int += i + 1;
            }
            // printf("Fechando o processo %d\n", i);
            exit(0);
        }
    }

    // espera os filhos
    for (i = 0; i < 3; i ++) {
        wait(NULL);
    }

    // exibindo resposta final:
    printf("Total final: %d\nEsperado: %d\n", *shared_int,
        EXECUCOES * 6);

    // fecha a memoria compartilhada
    shmdt(shared_int);
```

```
    shmctl(seg, IPC_RMID, 0);
    return 0;
}

int main_com_semaforo() {
    printf("Criando a memoria\n");
    int segmento = criar_memoria_compartilhada();
    printf("Criando o semaforo\n");
    int semaforo = criar_semaforo();
    int i, status;

    // criando os processos
    for (i = 0; i < 3; i ++) {
        if (fork() == 0) {
            printf("Criando processo %d\n", i);
            // dentro de um processo filho
            for (int x = 0; x < EXECUCOES; x++) {
                alterar_variavel(i + 1, semaforo);
            }
            printf("Fechando o processo %d\n", i);
            exit(0);
        }
    }

    // espera os filhos
    for (i = 0; i < 3; i++) {
        wait(&status);
    }
    // exibindo resposta final:
    printf("Total final: %d\nEsperado: %d\n", *shared_int,
        EXECUCOES * 6);

    // fecha o semaforo
    deletar_semaforo(semaforo);
    // fecha a memoria compartilhada
    shmdt(shared_int);
    shmctl(segmento, IPC_RMID, 0);
    return 0;
}

int main(int argc, char* argv[]) {
    if (argc == 1) {
```

```
        printf("Chame com parametro 1 para executar com semaforo\n");
        printf("Chame com parametro 0 para executar sem semaforo\n");
        exit(0);
    }

    if (strcmp(argv[1], "0")) {
        return main_sem_semaforo();
    } else if (strcmp(argv[1], "1")) {
        return main_com_semaforo();
    } else {
        return -1;
    }
}
```

Execução:

Sem semáforos:

```
Iniciado o processo 0
Iniciado o processo 1
Iniciado o processo 2
Fechando o processo 0
Fechando o processo 1
Fechando o processo 2
Total final: 227399
Esperado: 300000
```

É possível notar que houve concorrência ao rodar com **50000** execuções por processo.

Com semáforos:

```
Criando a memoria
Criando o semaforo
Criando processo 0
Criando processo 1
Criando processo 2
Fechando o processo 0
Fechando o processo 1
Fechando o processo 2
Total final: 3000
Esperado: 3000
```

Não houve concorrência, todos os processos aguardaram sua vez corretamente, devido ao uso dos semáforos.