UNIVERSIDADE FEDERAL DE UBERLÂNDIA
UFU

Matheus Cunha Reis - 11521BCC030

Bônus II - IPC

Uberlândia
2017

:

Exercicios

1)

P1:
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <unistd.h>     /* Symbolic Constants */
#include <semaphore.h>  /* Semaphore */
#include <pthread.h>    /* POSIX Threads */
#include <signal.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/wait.h>

#define MEM_SZ 100
#define BUFF_SZ MEM_SZ-sizeof(int)

struct shared_area{
        int num;
        int vet[15];
};

main(int argc, char *argv[])
{
        int pidP4 = atoi(argv[1]);
        int i, j;
        key_t key=1234;
        srand( (unsigned)time(NULL) );
        struct shared_area *shared_area_ptr;
        void *shared_memory = (void *)0;
        int shmid;

        sem_t *mutex = sem_open("semaphore1", O_CREAT, 0644, 3);

        shmid = shmget(key,MEM_SZ,IPC_CREAT);
        if ( shmid == -1 )
        {
                printf("shmget falhou\n");
                exit(-1);
        }

        printf("shmid=%d\n",shmid);

        shared_memory = shmat(shmid,(void*)0,0);

        if (shared_memory == (void *) -1 )
        {
                printf("shmat falhou\n");
                exit(-1);
        }

        printf("Memoria compartilhada no endereco=%x\n",(int) shared_memory);

        shared_area_ptr = (struct shared_area *) shared_memory;


        sem_wait(mutex);                                // ABRE
```

```
        shared_area_ptr->num=0;

        sem_post(mutex);            // FECHA

        int h;
        while(1)
        {
                sem_wait(mutex);                                // ABRE
                //sleep(3);
                if ( shared_area_ptr->num < 10 )
                {
                        int numAleatorio = rand()%1000 + 1;
                        printf("Numero Aleatorio %d\n", numAleatorio);

                        shared_area_ptr->vet[shared_area_ptr->num] = numAleatorio;
                        shared_area_ptr->num++;
                }

                if( shared_area_ptr->num == 10 )
                {
                        //Pid do processo 4
                        sem_post(mutex);            // FECHA
                        kill(pidP4, SIGUSR1);
                }
                else
                        sem_post(mutex);            // FECHA
        }

        sem_close(mutex);
   exit(0);
}


P2 e P3:
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <unistd.h>     /* Symbolic Constants */
#include <semaphore.h>  /* Semaphore */
#include <pthread.h>    /* POSIX Threads */
#include <signal.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/wait.h>

#define MEM_SZ 100
#define BUFF_SZ MEM_SZ-sizeof(int)

struct shared_area{
        int num;
        int vet[15];
};

main(int argc, char *argv[])
{
        int pidP4 = atoi(argv[1]);
        int i, j;
        key_t key=1234;
```

```
        srand( (unsigned)time(NULL) );
        struct shared_area *shared_area_ptr;
        void *shared_memory = (void *)0;
        int shmid;

        sem_t *mutex = sem_open("semaphore1", O_RDWR);

        shmid = shmget(key,MEM_SZ,0666|IPC_CREAT);
        if ( shmid == -1 )
        {
                printf("shmget falhou\n");
                exit(-1);
        }

        printf("shmid=%d\n",shmid);

        shared_memory = shmat(shmid,(void*)0,0);

        if (shared_memory == (void *) -1 )
        {
                printf("shmat falhou\n");
                exit(-1);
        }

        printf("Memoria compartilhada no endereco=%x\n",(int) shared_memory);

        shared_area_ptr = (struct shared_area *) shared_memory;

        int h;
        while(1)
        {
                sem_wait(mutex);                                // ABRE
                if ( shared_area_ptr->num < 10 )
                {
                        int numAleatorio = rand()%1000 + 1;
                        printf("Numero Aleatorio %d\n", numAleatorio);

                        shared_area_ptr->vet[shared_area_ptr->num] = numAleatorio;
                        shared_area_ptr->num++;
                }

                if( shared_area_ptr->num == 10 )
                {
                        //Pid do processo 4
                        sem_post(mutex);            // FECHA
                        kill(pidP4, SIGUSR1);
                }
                else
                        sem_post(mutex);            // FECHA
        }

        sem_close(mutex);
   exit(0);
}
```

P4:
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/shm.h>
```

```
#include <unistd.h>     /* Symbolic Constants */
#include <semaphore.h>  /* Semaphore */
#include <pthread.h>    /* POSIX Threads */
#include <signal.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/wait.h>

#define MEM_SZ 100
#define BUFF_SZ MEM_SZ-sizeof(int)

int numeros[11], cont, flag, canal1[2], canal2[2], contP5, contP6;
sem_t *mutex;

struct shared_area{
        int num;
        int vet[15];
};

struct shared_area2{
        int num;
        int wait;
        int vet[15];
};

struct shared_area *shared_area_ptrF1;
struct shared_area2 *shared_area_ptrF2;

void handler();
void *thread1(void *p);
void *thread2(void *p);
void insereF2(int num, int opt);

void configF1()
{
        key_t key=1234;
        void *shared_memory = (void *)0;
        int shmid;
        shmid = shmget(key,MEM_SZ,0666|IPC_CREAT);
        if ( shmid == -1 ){
                printf("shmget falhou\n");
                exit(-1);
        }
        printf("shmid=%d\n",shmid);

        shared_memory = shmat(shmid,(void*)0,0);

        if (shared_memory == (void *) -1 ){
                printf("shmat falhou\n");
                exit(-1);
        }

        shared_area_ptrF1 = (struct shared_area *) shared_memory;
}

void configF2()
{
        key_t key=5678;
        void *shared_memory = (void *)0;
        int shmid, i;
```

```
        shmid = shmget(key,MEM_SZ,0666|IPC_CREAT);
        if ( shmid == -1 ){
                printf("shmget falhou\n");
                exit(-1);
        }
        printf("shmid=%d\n",shmid);

        shared_memory = shmat(shmid,(void*)0,0);

        if (shared_memory == (void *) -1 ){
                printf("shmat falhou\n");
                exit(-1);
        }

        shared_area_ptrF2 = (struct shared_area2 *) shared_memory;
        shared_area_ptrF2->num=0;
        shared_area_ptrF2->wait = 0;
}

main()
{
        signal(SIGUSR1, handler);
        key_t key=1234;

        mutex = sem_open("semaphore1", O_RDWR);

        if ( pipe(canal1) == -1 ){ printf("Erro pipe()"); exit(0); }
        if ( pipe(canal2) == -1 ){ printf("Erro pipe()"); exit(0); }

        cont = 0;
        flag = 0;

        configF1();
        configF2();

        while(1){ }

        sem_close(mutex);

   exit(0);
}

void handler()
{
        int i, j;
        sem_wait(mutex);

        printf("%d ", cont);
        cont++;
        for(i=0;i<10;i++)
        {
                numeros[i] = shared_area_ptrF1->vet[i];
                printf("%d ", numeros[i]);
        }

        printf("\n");
        shared_area_ptrF1->num = 0;

        pthread_t t1, t2;
        pthread_create(&t1,NULL,thread1,NULL);
```

```
	pthread_create(&t2,NULL,thread2,NULL);

	pthread_join(t1,NULL);
	pthread_join(t2,NULL);

	if(cont == 1000){
		printf("Parou\n");
		while (1){ }
	}
	//sleep(2);
	sem_post(mutex);
}

void * thread1(void *p)
{
	pid_t pid;
	int i;

	for(i = 0; i < 5; i++)
		write(canal1[1], &numeros[i], sizeof(int));

	pid=fork();

	if ( pid == 0 )
	{
		for(i = 0; i < 5; i++){
			int num;
			read(canal1[0], &num, sizeof(int));

			//Insere na F2
			while(1){
				if(shared_area_ptrF2->wait == 0){
					if(shared_area_ptrF2->num >= 10){
						shared_area_ptrF2->wait = 2;
						break;
					}

					shared_area_ptrF2->vet[shared_area_ptrF2->num] = num;
					shared_area_ptrF2->num = shared_area_ptrF2->num + 1;

					if(shared_area_ptrF2->num >= 10){
						shared_area_ptrF2->wait = 2;
						break;
					}

					shared_area_ptrF2->wait = (shared_area_ptrF2->wait + 1)%3;
						break;
					}
				}
			}

			fflush(stdout);
			exit(0);
	}

	wait(&pid);
	pthread_exit(0);
}

void * thread2(void *p)
```

```
{
	pid_t pid;
	int i;

	for(i = 5; i < 10; i++)
		write(canal2[1], &numeros[i], sizeof(int));

	pid=fork();

	if ( pid == 0 )
	{
		for(i = 5; i < 10; i++){
			int num;
			read(canal2[0], &num, sizeof(int));

			//Insere na F2
			while(1){
				if(shared_area_ptrF2->wait == 1){

				if(shared_area_ptrF2->num >= 10){
					shared_area_ptrF2->wait = 2;
					break;
				}

				shared_area_ptrF2->vet[shared_area_ptrF2->num] = num;
				shared_area_ptrF2->num = shared_area_ptrF2->num + 1;

				if(shared_area_ptrF2->num >= 10){
					shared_area_ptrF2->wait = 2;
					break;
				}

				shared_area_ptrF2->wait = (shared_area_ptrF2->wait + 1)%3;
					break;
				}
			}
		}

		fflush(stdout);
		exit(0);
	}

	wait(&pid);
	pthread_exit(0);
}
```

P7:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <unistd.h>     /* Symbolic Constants */
#include <semaphore.h>  /* Semaphore */
#include <pthread.h>    /* POSIX Threads */
#include <signal.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/wait.h>
```

```
#define MEM_SZ 100
#define BUFF_SZ MEM_SZ-sizeof(int)

struct shared_area2{
	int num;
	int wait;
	int vet[15];
};

int cont, res[1005], min, max;
struct shared_area2 *shared_area_ptrF2;

void *tira_f2(void * p)
{
	int i;
	int *opt = (int*)p;

	while(1)
	{
		if(cont == 10000)
			break;
		if(shared_area_ptrF2->wait == *opt)
		{
			if(shared_area_ptrF2->num == 0){
			shared_area_ptrF2->wait = (shared_area_ptrF2->wait + 1)%6 ;
			}
			else{

				int num = shared_area_ptrF2->vet[0];

				res[num]++;
				if(num > max) max = num;
				else if(num < min) min = num;
				if(*opt)

				for(i = 0 ; i < shared_area_ptrF2->num; i++)
				shared_area_ptrF2->vet[i] = shared_area_ptrF2->vet[i+1];

				shared_area_ptrF2->num--;
				cont++;
				printf("Recebendo numero %d:  %d\n", cont, num);
				shared_area_ptrF2->wait    =    (shared_area_ptrF2->wait    +
1)%6 ;
			}
		}
	}

	pthread_exit(0);
}

main()
{
	time_t inicio, fim;
	inicio = time(NULL);

	int i;
	key_t key=5678;
	pthread_t th1, th2, th3;
	void *shared_memory = (void *)0;
```

```
	int shmid;

	cont = 0;
	min = 1005;
	max = 0;
	memset(res, 0, sizeof(res));

	shmid = shmget(key,MEM_SZ,0666|IPC_CREAT);
	if ( shmid == -1 )
	{
		printf("shmget falhou\n");
		exit(-1);
	}

	printf("shmid=%d\n",shmid);

	shared_memory = shmat(shmid,(void*)0,0);

	if (shared_memory == (void *) -1 )
	{
		printf("shmat falhou\n");
		exit(-1);
	}

	printf("Memoria compartilhada no endereco=%x\n",(int)shared_memory);

	shared_area_ptrF2 = (struct shared_area2 *) shared_memory;

	int t1 = 3, t2 = 4, t3 = 5;
	pthread_create(&th1, NULL, tira_f2,(void*)&t1);
	pthread_create(&th2, NULL, tira_f2,(void*)&t2);
	pthread_create(&th3, NULL, tira_f2,(void*)&t3);

	char aux[5];
	while(1)
	{
		if(cont == 10000)
			break;

		if(shared_area_ptrF2->wait == 2 && shared_area_ptrF2->num > 0)
			shared_area_ptrF2->wait = shared_area_ptrF2->wait + 1;
	}

	int indice, best = 0;
	for(i = 1; i <= 1000; i++){
		if(res[i] > best){
			best = res[i];
			indice = i;
		}
	}

	fim = time(NULL);
	printf("Tempo total de execucao do programa: %f\n", difftime(fim, inicio));
	printf("Quantidade de valores processados por P5: %d\n", 5000);
	printf("Quantidade de valores processados por P6: %d\n", 5000);
	printf("Moda: %d\n", indice);
	printf("Maior valor: %d\n", max);
	printf("Menor Valor: %d\n", min);
   exit(0);
}
```

PARA EXECUTAR:

1. Compilar todos os codigos
2. Executar ./P4 depois ./P7
3. Pegar pid do P4
4. Executar P1 passando o pid do P4 como argumento. Ex: ./P1 1205
5. Fazer a mesma coisa para P2 e P3

2)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <unistd.h>     /* Symbolic Constants */
#include <semaphore.h>  /* Semaphore */
#include <pthread.h>    /* POSIX Threads */
#include <signal.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/wait.h>

void *acao(void *p)
{
        //0 - comer
        //1 - pensar
        int *num = (int*)p;

        while(1)
        {
                int escolha = rand()%2;

                if(escolha == 0){

                        printf("Filosofo %d esta comendo com os garfos %d e %d\n",
(*num), (*num), (*num + 1)%5);
                        int tempo = rand()%3;
                        sleep(tempo);
                        printf("Filosofo %d parou de comer\n", (*num));
                }
                else if(escolha == 1)
                {
                        printf("Filosofo %d esta pensando\n", (*num));
                        int tempo = rand()%3;
                        sleep(tempo);
                }
        }

        pthread_exit(0);
}

main()
{
        srand( (unsigned)time(NULL) );
        int i;
        pthread_t filosofos[5];

        int f[5];
        for(i = 0; i < 5; i++)
                f[i] = i;
```

```
	for(i = 0; i < 5; i++)
		pthread_create(&filosofos[i],NULL,acao,(void*)&f[i]);

	for(i = 0; i < 5; i++)
		pthread_join(filosofos[i], NULL);

	exit(0);
}
```

3)
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <unistd.h>     /* Symbolic Constants */
#include <semaphore.h>  /* Semaphore */
#include <pthread.h>    /* POSIX Threads */
#include <signal.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/wait.h>

int garfos[5];
sem_t mutex[5];

void *acao(void *p)
{
	//0 - comer
	//1 - pensar
	int *num = (int*)p;

	while(1)
	{
		int escolha = rand()%2;

		if(escolha == 0){

			int j = *num;
			sem_wait(&mutex[j]);
			printf("Filosofo %d pegou o garfo %d\n", j, j);
			sleep(1);
			sem_wait(&mutex[(j+1)%5]);
			printf("Filosofo %d pegou o garfo %d\n", j, j+1);
			int tempo = rand()%3;
			sleep(tempo);
			printf("Filosofo %d parou de comer\n", j);

			sem_post(&mutex[j]);
			sem_post(&mutex[(j+1)%5]);
		}
		else if(escolha == 1)
		{
			printf("Filosofo %d esta pensando\n", (*num));
			int tempo = rand()%3;
			sleep(tempo);
		}
	}
```

```
        pthread_exit(0);
}

main()
{
        int i;
        pthread_t filosofos[5];
        srand( (unsigned)time(NULL) );

        for(i = 0; i < 5; i++){
                garfos[i] = i;
                sem_init(&mutex[i], 0, 1);
        }

        for(i = 0; i < 5; i++)
                pthread_create(&filosofos[i],NULL,acao,(void*)&garfos[i]);

        for(i = 0; i < 5; i++)
                pthread_join(filosofos[i], NULL);

        for(i = 0; i < 5; i++)
                sem_destroy(&mutex[i]);
        exit(0);
}
```

4)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <unistd.h>     /* Symbolic Constants */
#include <semaphore.h>  /* Semaphore */
#include <pthread.h>    /* POSIX Threads */
#include <signal.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/wait.h>
#define N 5
#define LEFT (i+N-1)%N
#define RIGHT (i+1)%N
#define NOTHING 0
#define HUNGRY 1
#define EATING 2

int state[N];
sem_t mutex;
sem_t s[N];

void test(int i)
{
        if (state[i] == HUNGRY && state[RIGHT] != EATING && state[LEFT] != EATING)
        {
                state[i] = EATING;
                sem_post(&s[i]);
        }
}

void take_forks(int i)
{
```

```
        sem_wait(&mutex);
        state[i] = HUNGRY;
        test(i);
        sem_post(&mutex);
        sem_wait(&s[i]);
}

void put_forks(int i)
{
        sem_wait(&mutex);
        state[i] = NOTHING;
        test(LEFT);
        test(RIGHT);
        sem_post(&mutex);
}

void *acao(void *p)
{
        //0 - comer
        //1 - pensar
        int *num = (int*)p;

        while(1)
        {
                int escolha = rand()%2;

                if(escolha == 0){
                        take_forks(*num);
                        int tempo = rand()%3;
                        printf("Filosofo %d esta comendo\n", (*num));
                        sleep(tempo);
                        put_forks(*num);

                }
                else if(escolha == 1)
                {
                        printf("Filosofo %d esta pensando\n", (*num));
                        int tempo = rand()%3;
                        sleep(tempo);
                }
        }

        pthread_exit(0);
}

main()
{
        int i ,f[5];
        pthread_t filosofos[5];
        srand( (unsigned)time(NULL) );
        sem_init(&mutex, 0, 1);

        for(i = 0; i < 5; i++){
                sem_init(&s[i], 0, 1);
                f[i] = i;
        }

        for(i = 0; i < 5; i++)
                pthread_create(&filosofos[i],NULL,acao,(void*)&f[i]);
```

```
	for(i = 0; i < 5; i++)
		pthread_join(filosofos[i], NULL);

	for(i = 0; i < 5; i++)
		sem_destroy(&s[i]);
	sem_destroy(&mutex);
	exit(0);
}
```