Trabalho 1: Barreira e fila de uso FIFO para processos

Luiz Henrique Murback Wiedmer
GRR20221234

Barreira e fila de uso FIFO para processos

1.Introdução

O programa cria um número n de filhos(sendo n decidido pelo usuário) e controla o uso de um recurso que deve ser utilizado por todos os processos. Esse controle é feito por meio de uma estrutura de barreira e uma estrutura de fila FIFO.

2.Forma de uso

Use "make" para compilar, e rode "./mainProc n", sendo n o número de processos a ser gerado(incluindo o pai).

3.Estruras de dados

Para a implementação desse programa, foram utilizadas três estruturas, sendo a primeira a seguinte estrutura de barreira:

```
typedef struct barrier_s {
    int maxProcNum;
    int procNumCount;
    sem_t sem;
    sem_t mutex;
} barrier_t;
```

Sendo que maxProcNum é o número de processos que devem chegar na barreira para que ela "abra", procNumCount é o número atual de processos esperando na barreira, sem é um semáforo para bloquear o processo quando necessário, e mutex é um semáforo que é bloqueado antes de fazer ações críticas para evitar condições de corrida.

A segunda estrutura utilizada foi uma estrutura de nodo para guardar os dados do processo:

```
typedef struct nodoProc {
    int Pi;
    sem_t sem;
} nodoProc_t;
```

Pi é o pid lógico do processo(0 é o pai, 1 é o primeiro filho, 2 é o segundo e assim segue), e sem é o semáforo que que controla se o processo deve esperar ou pode usar o recurso. Pi não foi utilizado na implementação, mas foi necessário para fazer o debug.

A terceira estrutura é uma estrutura de fila FIFO:

```
typedef struct fifoQ_s {
    int head;
    int tail;
    int recursoLivre;
    nodoProc_t nodos[MAX_NODES];
    sem_t mutex;
} FifoQT;
```

Head é o índice da cabeça da fila(a posição a ser retirada), e tail a posição em que deve ser enfileirado o próximo elemento, e recursoLivre é uma variável que mostra se o recurso está livre para uso ou não. Nodos é um vetor de tamanho MAX_NODES que guarda os dados dos processos(foi escolhido um vetor para a fila porque é necessário o uso de memória compartilhada). Mutex novamente é um semáforo para garantir com a função de evitar condições de corrida.

4.Funções

Tanto o módulo da barreira quanto o da FIFO possuem suas própria funções, sendo elas:

barreira.c:

```
void init_barr( barrier_t *barr, int n );
```

Essa função seta as váriaveis da struct passada em barr e cria os semáforos com sem_init, para que eles possam ser utilizados com shared memory.

```
void process_barrier( barrier_t *barr );
```

Essa função faz com que o chamador espere até que todos os processos esperados cheguem na barreira(maxProcNum), e quando o último faz a chamada, o mesmo libera todos os processos que estavam esperando.

myFifo.c:

```
void init_fifoQ(FifoQT *fila);
```

Essa função seta as váriaveis da struct passada em fila e cria os semáforos com sem_init, para que eles possam ser utilizados com shared memory.

```
void enfila(FifoQT* fila, int Pi);
```

Essa função cria um nodo para o processo que chamou a funcao, e o coloca na fila.
Obs: A funcao nao tem mutex interno, entao eh necessario utilizar bloqueio externo ao fazer a chamada.

```
nodoProc_t* desenfila(FifoQT* fila);
```

Essa função retorna um ponteiro para o nodo na cabeça da fila, e move a cabeça.
Obs: A funcao nao tem mutex interno, entao eh necessario utilizar bloqueio externo ao fazer a chamada.

5.Limitações

O fato de o processo pai funcionar como os processos filhos, faz com que seja possível que o programa entre em deadlock caso a fila de processos fique cheia. Isso pode ser resolvido com um programa watchdog externo ou fazendo com que o pai seja apenas um supervisor. Por conta da simplicidade da implementação, foi feita apenas uma trava, impedindo que o programa funcione para n's maiores que MAX_NODES.