Trabalho 1 - Passagem de bastão - Produtor/Consumidor

Mauro André Murari

1. Funcionamento

Ao iniciar é solicitado a quantidade de produtores, consumidores e mensagens que serão geradas. São criados os semáforos para os produtores e todas as threads (Leitores+Escritores). Ao finalizar as threads, todas são destruídas. Usei pthreads e semaphores para abstrair a parte mais complexa e já resolvida por essas bibliotecas. Abaixo está uma definição do funcionamento de cada parte do algoritmo.

Final: O método *is_done* checa se a execução chegou ao final, primeiro verifica se todos os escritores produziram até a última mensagem, depois válida se todos os leitores leram as mensagens, caso os dois forem verdadeiros, o as threads chegam ao final e a mensagem de *done* é gerada.

Consumidor: Leitor checa se ainda existem valores para serem lidos, se existir, verifica quais produtores tem valor e se o valor foi lido, quando encontra um que não foi lido, solicita o *lock* daquele semáforo, gera a mensagem "C{*me*} - Lendo P{*produtor*}: *i*", lê o valor, marca como lido, gera a mensagem "C{*me*} - Liberando P{*produtor*}" e libera o semáforo.

Produtor: Recebe um parâmetro inteiro chamado *me*, se todos os consumidores tiverem lido o valor anterior, o produtor pede o *lock*, ao conseguir o *lock,* gera a mensagem "P{*me*} - Produzindo...", marca todas as linhas do *read[me][0~consumers-1]* como não lidas, gera um novo valor *values[me]=i*, imprime a mensagem "P{*me*} - Produzido: *i*" e por fim é liberado o lock deste semáforo. Fica nesse ciclo até conseguir gerar todas mensagens.

Importante notar que o *lock* só é solicitado pelo escritor, após todos os leitores terem lido, isso garante que a preferência é sempre dos leitores, ou seja, um leitor nunca vai ficar sem receber um valor.

O código está rodando em clang-802.0.42 no MacOSX, usando Posix como modelo de threads. Pode ser baixado pelo anexo ou pelo github.

Use o comando abaixo para executar, a ordem dos argumentos é: quantidade de produtores, quantidade de consumidores e quantidade de mensagens:

```
clang trab-1.c -o trab && ./trab 2 3 4
```

Para rodar e validar

```
clang trab-1.c -o trab && ./trab 2 3 4 > output.txt &&
python validador.py
```

2. Validação

Escrevi um código em python 2.7 para fazer a validação do código. Fiz alguns checks como:

- Os produtores geraram todas as mensagens?
- Os leitores conseguiram ler todos os valores?
- Programa chegou ao final?

3. Problemas

Quando existe um valor alto (+200) de threads, pode acontecer de o valor lido estar incorreto, tentei resolver isso de várias formas (Sleep, outros semáforos, outras estruturas, mas não resolveu), acredito que seja algo no nível de SO/output do console pois sempre que usei debug, funcionou.

Ao rodar o algoritmo vários *warnings* de "*deprecated*" são gerados, tentei atualizar para a versão atual do método, mas parou de funcionar. Como não altera o funcionamento do programa e este código não vai ser usado em produção, optei por dedicar muito tempo para isto.

4. Documentação

```
int values[999]; // Lista dos valores, cada posição representa um produtor
int reads[999][999]; // Matriz de valores vs leitores, identifica que o valor já foi lido pelo leitor


void *write(int me) {
    for (int i = 1; i <= MESSAGES; i++) { // Gerando todas as mensagens
        while (is_all_read(me) == FALSE) {} // Esperando todo mundo ler o valor produzido anteiormente
        sem_wait(&semaphores[me]); // espera o semaforo ser liberado (Algum leitor ainda pode estar lendo)
        set_not_reads(me); values[me] = i; // seta valor
        sem_post(&semaphores[me]); // signal, libera o semaforo para os leitores
    }
 }

void *read(int me) {
    while (is_done() == FALSE) { // Verifica que tudo foi produzido e lido
        for (int producer = 0; producer < PRODUCERS; producer++) { // Checando todos os produtores
            if (values[producer] > 0 && reads[producer][me] == FALSE){ // Verifica sem tem valor pra ser lido
                sem_wait(&semaphores[me]); // await, espera os demais leitores liberarem o semaforo
                reads[producer][me] = TRUE; // Marca como lido
                sem_post(&semaphores[producer]); // signal, libera para as outras threads
            }
        }
    }
}
```