UFCG UASC/CEEI

Disciplina: Programação Concorrente Prova 1 Repo
Aluno:
Matrícula:

1. Considere a API abaixo, para controle requisições de IO em um sistema operacional

   func iop(Request req)
   func submit(Request req)

   Considere que múltiplas threads podem chamar a função iop. O tipo Request é definido da seguinte forma:

```
type Request {
      //indicar se a requisição é para uma op de leitura ou escrita
      enum op_type {R, W};
      //indica o id do bloco para o qual a req será feita
      int block_id;
      //conteúdo a ser lido/escrito para cada o bloco
      byte[] buffers;
}
```

   Considere que não é bacana submeter, num curto espaço de tempo (vamos chamar de MERGE_INTERVAL), operações de um determinado tipo para um mesmo bloco. Ou seja, é ruim ter duas ou mais operações de escrita para o mesmo bloco dentro de um MERGE_INTERVAL.

   Escreva a função iop (e funções auxiliares) que controla o acesso à função submit e aglutina requisições de um mesmo tipo para um mesmo bloco. Ou seja, você deve juntar Requests para um mesmo bloco em uma Request única. Isso pode ser feito através da função auxiliar **merge** abaixo:

   func merge(Request one, Request two) Request

   Você pode usar também a função **now** que retorna o instante de tempo atual (tal como unix epoch, unidades de tempo desde 1970)

   func now() Timestamp

   Ou seja, se precisa calcular tempo decorrido, você pode fazer algo do tipo: **(now() - oldTimeStamp) > MERGE_INTERVAL**

   Dica uma vez que você não quer que duas requests para um mesmo bloco sejam submetidas dentro de um mesmo MERGE_INTERVAL, isso significa que você pode atrasar a submissão de requests (mantendo requests em uma estrutura de dados auxiliar pelo tempo que for necessário). Ainda, você é livre para implementar a função iop de maneira bloqueante ou não. Ou seja, a thread que chama iop não precisa esperar até que a execução da request seja terminada. Você também pode criar novas threads (com a API parecida com qualquer linguagem vista em sala).

2. Considere a API abaixo. A função **gateway** deve criar **nthreads** threads diferentes. O código executado por cada thread deve ser o da função request**.** A função request deve sortear um número aleatório **n** e dormir **n** segundos. Após criar as threads, a função **gateway** deve esperar que até **wait_nthreads** terminem. Após a espera, a função gateway deve retornar a soma dos valores **n** sorteados nas funções request.

```
int gateway(int nthreads, int wait_nthreads)
void request()

//ao chamar a função, uma thread bloqueará por nseconds segundos
void sleep(int nseconds)

//a função random retorna um número aleatório positivo
int random()

//cria uma thread que executará a função f quando for escalonada. retorna o
id da //thread. Assuma que no máximo podem existir 1024 threads (ids entre 0
e 1023)
int create_thread(func() f)
```

API Semáforos/Variáveis condicionais

```
Semaphore (int initialValue)
wait()
signal()


CV()
wait()
signal()
broadcast()
```