Universidade Federal de Campina Grande
Departamento de Sistemas e Computação
Disciplina: Programação concorrente 2019.2
Professor: Thiago Emmanuel Pereira

Prova 1

obs 1) Pequenas mudanças na API indicada podem ser realizadas, para que seja possível adaptar o código para as linguagens indicadas. Contacte o professor caso queira realizar alguma mudança.

1. (java) A interface BlockingQueue de java é uma fila que adiciona um comportamento de espera, caso se queira retirar um elemento da fila e esta esteja vazia. De maneira análoga, também se espera que espaço seja liberado quando se queira adicionar um novo elemento e a fila tenha atingido sua capacidade máxima. Considere aspectos de desempenho na sua solução (por exemplo, evite esperas ocupadas). A fila deve ter capacidade estática definida no construtor. Implemente os métodos abaixo:
    a. void **put**(Object E) - Insere o elemento **E** na fila, esperando, caso necessário, para que espaço seja disponibilizado.
    b. Object **take**() - Recupera e remove o elemento na cabeça da fila, esperando, se necessário, até que um elemento esteja disponível na fila.
    c. int **remainingCapacity**() - Retorna o número de elementos que ainda podem ser adicionados na fila, sem bloqueio, considerando sua capacidade.

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

int gateway(int nthreads, int wait_nthreads)

void* request(void*)

3. (clang) Abaixo, temos um esboço de implementação de lista encadeada. Esta implementação tem problemas de concorrência. Detecte e corrija os problemas detectados.

```
// basic node structure
typedef struct __node_t {
      int key;
      struct __node_t  *next;
} node_t;

// basic list structure (one used per list)
typedef struct __list_t {
      node_t     *head;
} list_t;

void List_Init(list_t *L) {
      L->head = NULL
}

int List_Insert(list_t *L, int key) {
      node_t* new = malloc(sizeof(node_t));
      if (new == NULL) {
            perror("malloc");
            return -1; // fail
      }
      new->key = key;
      new->next = L->head;
      L->head = new;
      return 0; // success
}

int List_Lookup(list_t*L, int key) {
      pthread_mutex_lock(&L->lock);
      node_t*curr = L->head;
      while (curr) {
            if (curr->key == key){
                  return 0; // success
            }
            curr =curr->next;
      }
      return -1; // failure
}
```