Universidade Federal do Rio de Janeiro

Figure 1: Logo Ufrj

Lista 2 computacao concorrente

Lucas Tatsuya Tanaka
Prof:Silvana Rosseto
Dre:118058149

May 17, 2021

1 Questao 1

- **A)** Apos a verificação do código se observa que sendo a principal condição do programa a alternância entre as Threads, se averiguá que com o início das Threads a Thread Foo tomara prioridade na execução devido à presença do pthread_cond_wait(&condBar, &mutex) que bloqueara as Threads Bar ate o termino de execução de todas as Threads Foo, apos o termino da execução a última Thread Foo ira adentrar o condicional e como contaFoo sera igual a M ocorrera o broadcast das Threads bloqueadas por condBar, ocorrera então a entrada no While da Thread Bar no qual ela se bloqueara ate que a última Thread adentre o condicional e desbloqueia as Threads bloqueadas. por condFoo e que assim por diante irão alternar a execução das Threads.

- **B)** O código apresentados apos ser verificado, garante a inexistência de deadlocks e ausência de condição de corrida isto ocorre, pois, devido as condições do programa serem atendidas os recursos são acessados através de exclusão mutua e assim garantem os resultados esperados, com relação

aos deadlocks não a possibilidade do mesmo, pois sempre que a última Thread chegar ao final da execução ele atendera a condição de ser igual ao M ou N e ira executar o broadcast liberando as Threads.

2 Questao 2

- **A)** O que causou o deadlock foi a situacão ao qual a Thread apresentava mais Threads remove do que as Threads insere
- **B)** Uma possivel ordem de execucao seria o total bloqueio de todas as Threads Remove inicialmente e em seguida realizar uma Thread insere que vai desbloquear uma Thread remove e assim repedidamente caso existam mais Threads remove o programa entrara entrara em deadlock
- **C)** Uma possivel correcao seria

```
class Buffer{
static final int N = 10; //qtde de elementos no buffer
private int[] buffer = new int[N]; //area de dados compartilhada
private int count=0; //qtde de posicoes ocupadas no buffer
private int in=0; //proxima posicao de insercao
private int out=0; //proxima posicao de retirada

//construtot
Buffer() {... //inicializa o buffer }

// Insere um item
public synchronized void Insere (int item) {
    while (count==N) {
        wait();
    }
    buffer[in%N] = item;
    in++;
    count++;
    notifyAll();
}

// Remove um item
public synchronized int Remove (int id) {
    int aux;
    while (count==0) {
        wait();
    }
    aux = buffer[out%N];
    out++;
    count--;
```

```
        notifyAll();
        return aux;
    }
}
```

3 Questao 3

- **A)** Apos a averiguacao do programa se entende que a Thread B avia se bloqueado e depois de 10 execucoes do While da Thread A a Thread b ganha execucao pois essa ja avia sido desbloqueada pelo signal anteriormente executado pela Thread A.
- **B)** O codigo utilizado pode ser:

```
    int x = 0; pthread_mutex_t x_mutex; pthread_cond_t x_cond;
void *A (void *tid ){
    for (int i=0; i<100; i++) {
        pthread_mutex_lock(&x_mutex);
        x++;
        if(!(x%10))
            pthread_cond_signal(&x_cond);
        pthread_mutex_unlock(&x_mutex);
    }
}

void *B (void *tid) {
    pthread_mutex_lock(&x_mutex);
    if(x%10){
        pthread_cond_wait(&x_cond, &x_mutex);
        printf("X=%d\n", x);
    }
    pthread_mutex_unlock(&x_mutex);
}
```

4 Questao 4

- **A)** A solução proposta atende a todos os requisitos apresentados mais de um leitor pode realizar a leitura como averiguado no metodo EntraLEitor na qual nao a nenhum bloqueio, o segundo requisito é atendido quando analizado o synchronized do metodo Escrita na qual permite apenas a execucao de um dos metodos de cada vez e ocorre a garantia de que quando

ocorre a escrita nao existe mais leitores pois se atende a condicao do While no metodo.

- **B)** Apos a averiguação do programa se entende que com a troca do notifyAll por notify ocorre a garantia ainda da execucao do programa sem afetar a corretude da solução isso ocorrendo pois com a chegada na linha 11 o notify poderia estar em duas situacoes principais a primeira sendo o bloqueio de todas as Threads escritoras no momento em que o notify fosse executado a Thread escritor iria se liberar e assim poderia se executar e devido a presença do notify no final da Thread Escrita este iria liberar em efeito domino as Threads seguintes na segunda existe parcialidade ou nenhuma Thread bloqueada devido ao fato da Thread ter this.leit iguais a zero a Thread Ecritores não iria se bloquear e assim ocorreria a execução do programa.

- **C)** A chamada do metodo notify() poderia ser retirada pois devido a presença do metodo notifyAll presente na Thread leitora todas as Threads qua aviam sido previamente bloqueadas seriam liberadas fazendo assim a retomada de execução das Threads Escriotras e assim terminando a execução do programa.