



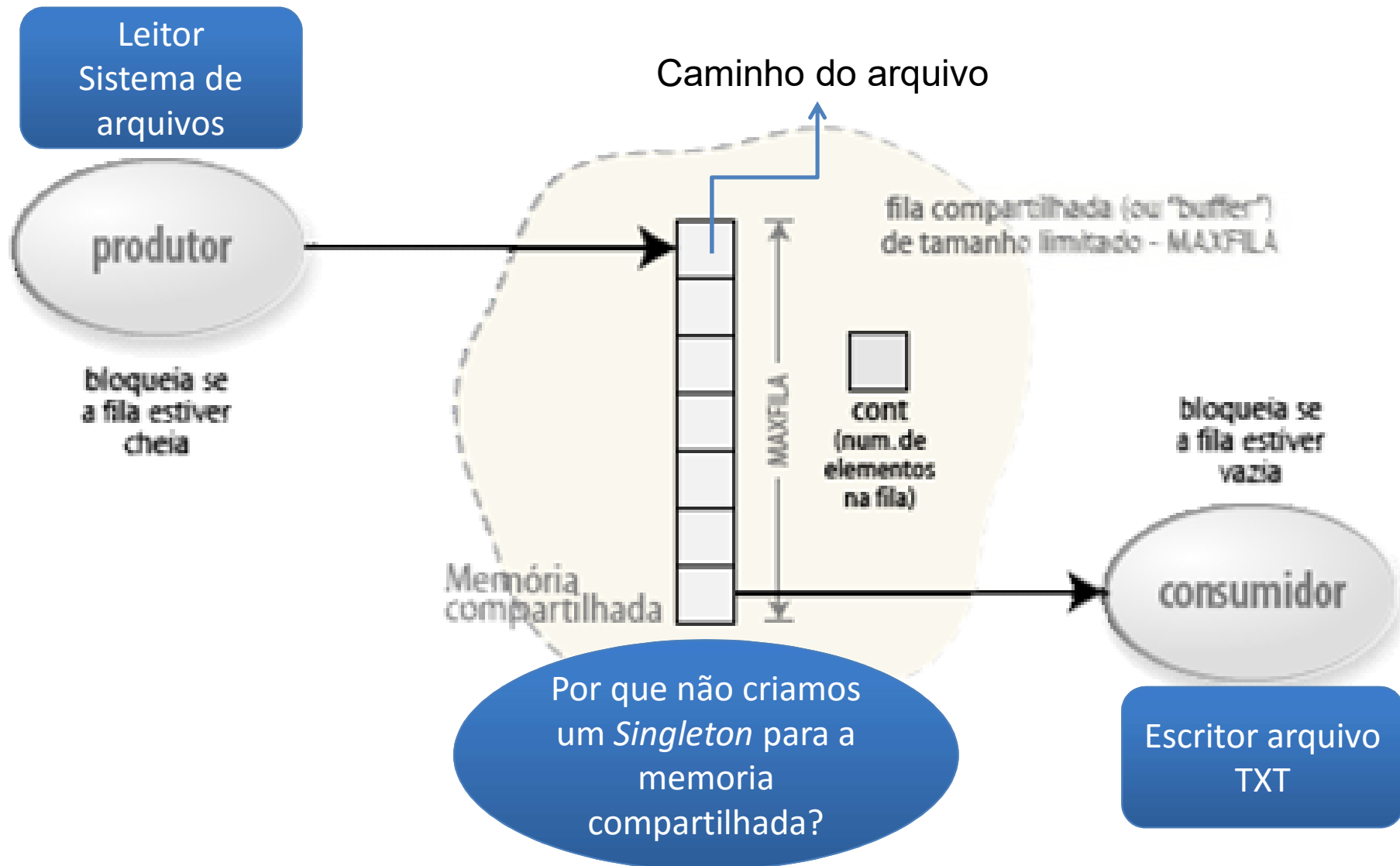
Tópicos avançados em Programação

Programação paralela - *Threads*

<http://dl.dropbox.com/u/3025380/prog2/aula15.pdf>

flavio.ceci@unisul.br

Vamos rever o estudo de caso















Exercício

- Desenvolva uma estratégia para corrigir o problema entre a thread produtora e consumidora.
- Baixe o projeto do eclipse disponível no link abaixo:

<http://dl.dropbox.com/u/3025380/prog2/aula14.zip>

Exercício

- ▼  Aula14-Prog2-Thread
 - ▼  src
 - ▼  br.unisul.prog2.aula14
 - ▶  exemplos
 - ▶  serial
 - ▼  thread
 - ▶  EscritorArquivoTexto.java
 - ▶  LeitorSistemaArquivo.java
 - ▶  PrincipalMigracaoThread.java
 - ▶  ServicoMemoriaCompartilhada.java
 - ▶  JRE System Library [javaSE-1.6]
 -  conteudo

Analizando o problema

- O consumidor (escritor) estava terminando antes que o produtor (leitor);
- Motivo;
 - Consumidor estava consumindo muito rápido e finalizando os itens da memória compartilhada;
 - Como a memória estava vazia a thread consumidora morria antes de processar os demais elementos.

Analizando o problema

- Uma possível solução:
 - A thread consumidora deve levar em consideração não só a quantidade de elementos disponíveis na memóriaCompartilhada, mas também o estado (rodando ou não) da thread produtora.
 - Adicionar uma flag na memória compartilhada para se identificar se a produção foi finalizada ou não.

Flag de notificação

```
public class ServicoMemoriaCompartilhada {  
    private List<String> memoriaCompartilhada = new ArrayList<String>();  
    private boolean produzindo;  Atributo produzindo representa a flag de notificação  
    private static ServicoMemoriaCompartilhada instancia = new ServicoMemoriaCompartilhada();  
    private ServicoMemoriaCompartilhada(){  
    };  
  
    public static ServicoMemoriaCompartilhada getInstancia() {  
        return instancia;  
    }  
  
    public void adiciona(String elemento) {  
        this.memoriaCompartilhada.add(elemento);  
    }  
  
    public String recupera() {  
        return this.memoriaCompartilhada.remove(0);  
    }  
  
    public boolean temElemento() {  
        return !this.memoriaCompartilhada.isEmpty();  
    }  
  
    public void setProduzindo(boolean produzindo) {  
        this.produzindo = produzindo;  
    }  
  
    public boolean estaProduzindo() {  
        return produzindo;  
    }  
}
```

  Métodos para acesso a flag

Flag de notificação (produtor)

```
public class LeitorSistemaArquivo extends Thread {  
  
    private String caminhoRaiz;  
  
    private ServicoMemoriaCompartilhada memoria = ServicoMemoriaCompartilhada.getInstance();  
  
    @Override  
    public void run() {  
        this.recuperarListaArquivo();  
    }  
  
    public LeitorSistemaArquivo(String caminhoRaiz) {  
        this.memoria.setProduzindo(true);  
        this.caminhoRaiz = caminhoRaiz;  
    }  
  
    private void recuperarListaArquivo() {  
        memoria.adiciona(caminhoRaiz);  
  
        this.lerRaizPasta(caminhoRaiz);  
    }  
}
```

A thread produtora notifica o inicio do seu processamento.

//Mantem-se o restante do código

Flag de notificação (consumido)

```
public void escreverConteudoArquivo() {
    OutputStream escritorByte = null;
    OutputStreamWriter escritorCaracter = null;
    BufferedWriter escritorPalavras = null;

    try {
        escritorByte = new FileOutputStream("conteudo/arquivos.txt", true);
        escritorCaracter = new OutputStreamWriter(escritorByte);
        escritorPalavras = new BufferedWriter(escritorCaracter);

        while(memoria.temElemento() || memoria.estaProduzindo()) {
            if(memoria.temElemento()) {
                String linha = memoria.recupera();

                if(linha != null && !linha.isEmpty()) {
                    escritorPalavras.write(linha);
                    escritorPalavras.newLine();
                }
            } else {
                Thread.sleep(500);
            }
        }
        escritorPalavras.flush();
    } catch (FileNotFoundException | InterruptedException e) {
```

Verificado se a thread está produzindo ainda...

Caso não tenha elementos para consumir mas a thread está produzindo ainda espere 500ms

Flag de notificação

```
public class PrincipalMigracaoThreadNotify {  
    public static void main(String[] args) throws InterruptedException {  
        ServicoMemoriaCompartilhada memoria = ServicoMemoriaCompartilhada.getInstancia();  
        LeitorSistemaArquivo leitor = new LeitorSistemaArquivo("/Users/flavioceci/Documents");  
        EscritorArquivoTexto escritor = new EscritorArquivoTexto();  
        long inicio = System.currentTimeMillis();  
  
        leitor.start();  
  
        while(memoria.temElemento() == false) {  
            System.out.println("Esperando preencher adicionar elementos na memoria");  
        }  
  
        escritor.start();  
  
        while(leitor.isAlive() || escritor.isAlive()) {  
            Thread.sleep(500);  
  
            if(!leitor.isAlive()) {  
                memoria.setProduzindo(false);  
            }  
        }  
  
        System.out.println("TEMPO TOTAL: " + (System.currentTimeMillis() - inicio));  
    }  
}
```

Análise da proposta de solução

- O processo funciona corretamente;
- Quando a thread consumidora “dormi” pelo tempo determinado pode desperdiçar tempo esperando;
- O ideal seria a thread consumidora ser notificada de imediato para voltar ao trabalho.

Filas bloqueantes do Java

- É uma extensão da Collections do Java para auxiliar no processamento paralelo.
- Vamos trabalhar com o conceito de filas bloqueantes:
 - **BlockingQueue**
<http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html>
 - Conhece o trabalho das duas threads e notifica sobre os seus estados.

Convertendo nosso projeto para o
uso de filas bloqueantes

BlockingQueue

```
public class PrincipalMigracaoThreadBlocking {  
    public static void main(String[] args) throws InterruptedException {  
        BlockingQueue<String> memoriaCompartilhada = new ArrayBlockingQueue<String>(1000);  
  
        Thread leitor = new Thread(new LeitorSistemaArquivo("/Users/flavioceci/Documents", memoriaCompartilhada));  
        Thread escritor = new Thread(new EscritorArquivoTexto(memoriaCompartilhada));  
        long inicio = System.currentTimeMillis();  
  
        leitor.start();  
        escritor.start();  
  
        while(leitor.isAlive() && escritor.isAlive()) {  
            Thread.sleep(500);  
        }  
  
        System.out.println("TEMPO TOTAL: "+(System.currentTimeMillis()-inicio));  
    }  
}
```

Vamos remover a classe ServicoMemoriaCompartilhada e criar uma BlockingQueue na classe principal, passando como parâmetro para as demais threads.

Vamos transformar as Threads em Runnables

BlockingQueue (produtora)

```
public class LeitorSistemaArquivo implements Runnable {
```

```
    private String caminhoRaiz;
```

```
    protected BlockingQueue<String> queue = null;
```

```
    @Override
```

```
    public void run() {  
        this.recuperarListaArquivo();  
    }
```

```
    public LeitorSistemaArquivo(String caminhoRaiz, BlockingQueue<String> queue) {  
        this.queue = queue;  
        this.caminhoRaiz = caminhoRaiz;  
    }
```

```
    private void recuperarListaArquivo() {  
        try {  
            queue.put(caminhoRaiz);  
  
            this.lerRaizPasta(caminhoRaiz);
```

```
            queue.put("DONE");  
        } catch (InterruptedException e) {  
            System.out.println(e);  
        }  
    }
```



Recebendo a fila bloqueante
por parâmetro no construtor.

Adicionou uma mensagem para notificar o fim
“dos trabalhos”.

BlockingQueue (produtora)

```
private void lerRaizPasta(String caminhoRaiz) throws InterruptedException {  
    File raiz = new File(caminhoRaiz);  
  
    if(raiz != null && raiz.exists()) {  
        File[] arquivos = raiz.listFiles();  
  
        if(arquivos != null) {  
            for(File arquivo : arquivos) {  
                if(arquivo != null && arquivo.isHidden() == false) {  
                    String caminho = arquivo.getAbsolutePath();  
                    queue.put(caminho);  
  
                    if(arquivo.isDirectory()) {  
                        lerRaizPasta(caminho);  
                    }  
                }  
            }  
        }  
    }  
}
```

Adiciona na fila bloqueante via método **put**.

BlockingQueue (consumidora)

```
public class EscritorArquivoTexto implements Runnable {  
    protected BlockingQueue<String> queue = null;  
  
    public EscritorArquivoTexto(BlockingQueue<String> queue) {  
        this.queue = queue;  
    }  
  
    @Override  
    public void run() {  
        this.escreverConteudoArquivo();  
    }  
}
```



Recebendo a fila bloqueante
por parâmetro no construtor.

BlockingQueue (consumidora)

```
public void escreverConteudoArquivo() {  
    OutputStream escritorByte = null;  
    OutputStreamWriter escritorCaracter = null;  
    BufferedWriter escritorPalavras = null;  
  
    try {  
        escritorByte = new FileOutputStream("conteudo/arquivos.txt", true);  
        escritorCaracter = new OutputStreamWriter(escritorByte);  
        escritorPalavras = new BufferedWriter(escritorCaracter);  
        String linha = null;  
        while (!(linha = queue.take()).equals("DONE")) {  
            escritorPalavras.write(linha);  
            escritorPalavras.newLine();  
        }  
        escritorPalavras.flush();  
    }  
  
    catch (FileNotFoundException | InterruptedException e) {
```



Enquanto não receber a notificação fica processando.



Exercício



Exercícios

- Foi solicitado que sejam extraídos dados do portal de dados abertos para a construção e carga de uma tabela (SGBD).
- O domínio são as obras do programa de aceleração do crescimento:

<http://bit.ly/1WPGQaJ>

Exercícios

- Dicionário de dados:
 - <http://bit.ly/1WPGVez>
- Dados:
 - <http://bit.ly/1WPH6Xd>
 - <http://bit.ly/1WPH9Co>