Sistemas de Informação

Conceitos Básicos

Programação Concorrente e Distribuída

Prof. Dr. Aldo Henrique

Roteiro

- Concorrência e Linguagens de Programação
  - Introdução
  - Threads
  - Motivação
- Conceitos Básicos
  - Tarefa
  - Sincronização
  - Exclusão Mútua (Mutex)
  - Deadlock
  - Estado de Tarefas
- Java: multithread
  - Modelo
  - Classe Thread
- Bibliografia

Introdução

- Mecanismos de controle de concorrência foram desenvolvidos originalmente para problemas de sistemas operacionais.
- No entanto, podem ser usados para resolver problemas naturalmente concorrentes.
- Do ponto de vista de um programador, a concorrência lógica é igual a concorrência física.

Thread

- Uma linha de controle é a sequencia de pontos de um programa alcançados conforme o controle flui pelo programa.
- Programas podem ter múltiplas linhas de controle, cada uma executada em um processador diferente.
- Programas com concorrência lógica podem ser projetados com múltiplas linhas de controle.

Thread

- Um programa projetado para ter mais de uma *thread* é chamado de *multithread*.

- O que acontece quando um programa *multithread* é executado em uma máquina com um único processador?
  - Programa virtualmente *multithread*.

- Exemplo de concorrência em nível de comando:
  - Um laço que opera em elementos de um arranjo.
    - Cada elemento pode ser calculado de maneira independente.
    - O processamento pode ser distribuído por vários processadores.

Motivação

- Diversos domínios de problemas incluem mais do que uma entidade que co-existem e executam atividades simultaneamente.
  - Voo de aeronaves em uma área controlada; estações em uma rede de comunicações; moléculas em uma solução; etc.
- Atualmente, a grande maioria dos computadores possuem múltiplos processadores.

Tarefa

- É uma unidade de um programa (similar a um subprograma) que pode estar em execução concorrente com outras unidades do mesmo programa.

- Cada tarefa pode prover uma *thread* controle.

- São também conhecidas como processos.

Tarefa

- O programa não precisa esperar uma tarefa terminar para continuar sua execução.
- Quando uma tarefa termina, o controle pode ou não retornar a unidade que iniciou sua execução.
- Tarefas **disjuntas** não se comunicam ou não afetam a execução de qualquer outra tarefa.
- Em muitos casos as tarefas trabalham juntas, necessitando de uma forma de **comunicação**.

Sincronização

- Mecanismo que controla a ordem na qual as tarefas executam.
- **Sincronização de cooperação**: requerida entre duas tarefas A e B, quando A precisa esperar B completar alguma atividade para que possa continuar.
  - ➔ Uma unidade produz algum valor de dados ou recurso e outra usa.

Sincronização

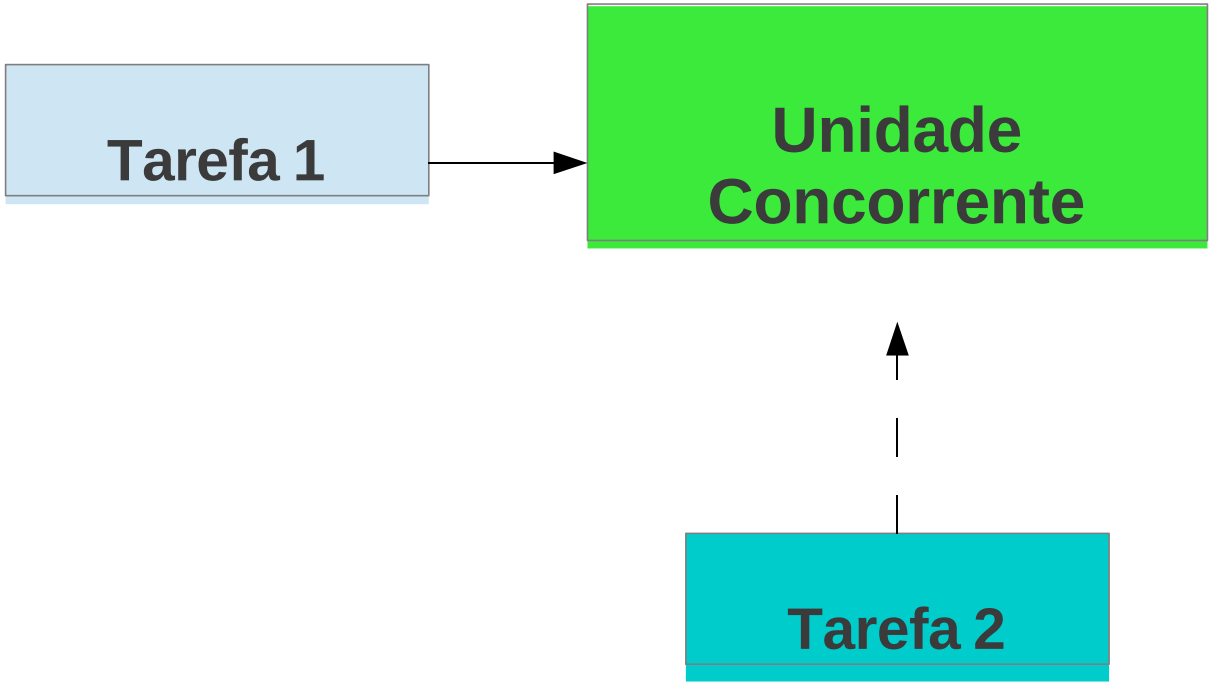
- **Sincronização de competição**: quando duas tarefas requerem o uso de um mesmo recurso que não pode ser usado simutaneamente. Exemplo:

  - A tarefa A precisa somar 1 a variável compartilhada interna TOTAL, quem tem valor inicial 3;
  - A tarefa B precisa multiplicar o valor de TOTAL por 2.

  - Cada tarefa termina a operação em TOTAL executando os seguintes passos:
    1. Obter o valor de TOTAL
    2. Realizar a operação aritmética
    3. Atribuir o novo valor a TOTAL

Sincronização

- Sincronização de competição:



Exclusão Mútua (Mutex)

- Caso particular de competição por recurso, onde apenas se pode permitir o acesso individual ao recurso.

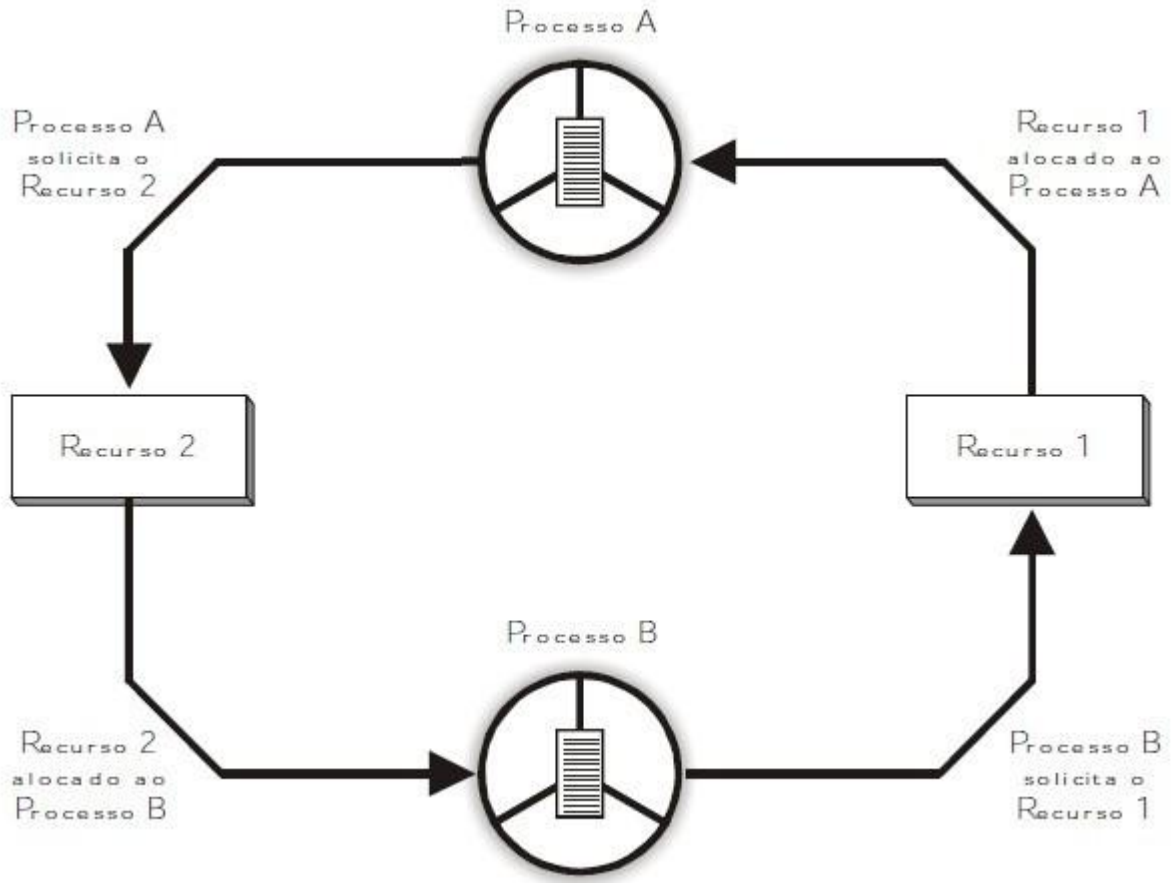
- Os primeiros mecanismos de exclusão mútua faziam com que os processos testassem continuamente o valor de uma variável que guarda o estado do recurso, até sua liberação, podendo causar um ***deadlock***.

- *Deadlock* é uma situação onde um processo ou um conjunto de processos estão bloqueados, a espera de um evento que nunca vai acontecer.

Deadlock

- Espera circular de um recurso.
- Exemplo: o processo P1 está de posse do recurso r1 e precisa do recurso r2; o processo P2 está de posse do recurso r2 e precisa do recurso r1.

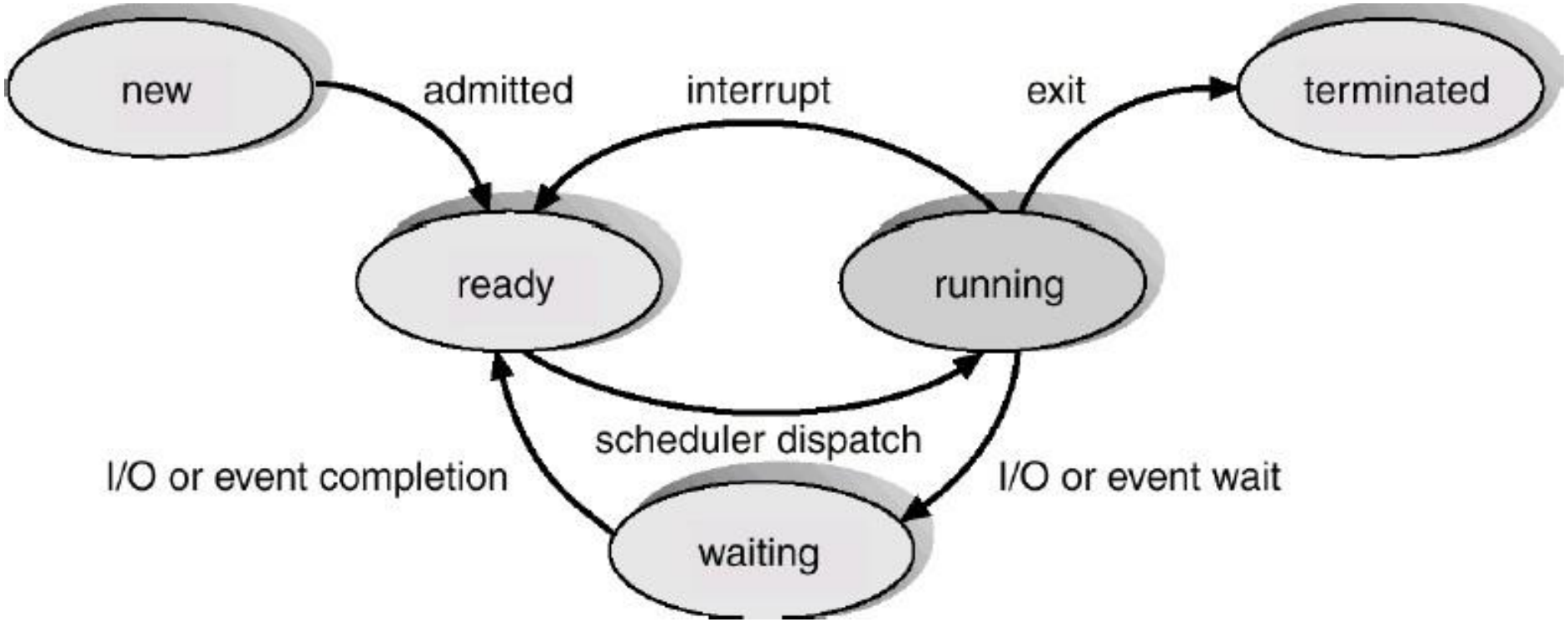
Deadlock


Processo A
Processo A
solicita o
Recurso 2
Recurso 1
alocado ao
Processo A
Recurso 2
Recurso 1
Processo B
Recurso 2
alocado ao
Processo B
Processo B
solicita o
Recurso 1

Estados de tarefas

- ***New***: quando foi criada mas ainda não foi executada;
- ***Ready***: pronta para executar, mas ainda não executando ou ainda não conseguiu tempo no processador ou estava executando, foi bloqueada e está novamente pronta. Tarefas prontas são armazenadas numa fila de tarefas prontas (*task ready queue*);
- ***Running***: atuamente em execução, ou seja, foi atribuída a um processador e seu código está sendo executado;
- ***Blocked***: estava executando mas foi interrompida por um evento, geralmente uma operação de entrada e saída, que são mais lentas geralmente;
- ***Dead***: não está mais ativa, tendo completado sua execução ou sido explicitamente morta pelo programa.

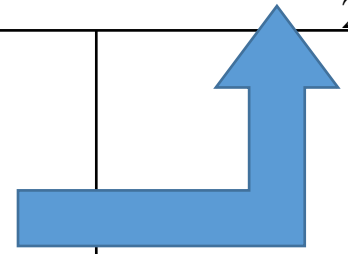
Estados de tarefas



```
import threading


def processar_parte(proceso,lista, inicio, fim):

    for i in range(inicio, fim):
        print(str(proceso)+' '+str(i))
        lista[i] = lista[i] ** 2

def processamento_multitarefa(lista):


    tamanho = len(lista)
    meio = tamanho // 2


    t1 = threading.Thread(target=processar_parte, args=(1,lista, 0, meio))
    t2 = threading.Thread(target=processar_parte, args=(2, lista, meio, tamanho))




    print('Start 1')
    t1.start()
    print('Start 2')
    t2.start()


    t1.join()
    t2.join()
```

```
# Criando uma lista grande de números
lista_numeros = list(range(1, 101))


# Processamento utilizando multitarefas
processamento_multitarefa(lista_numeros[:])


print('Processamento multitarefa concluído')
```

2



https://www.onlinegdb.com/online_python_compiler

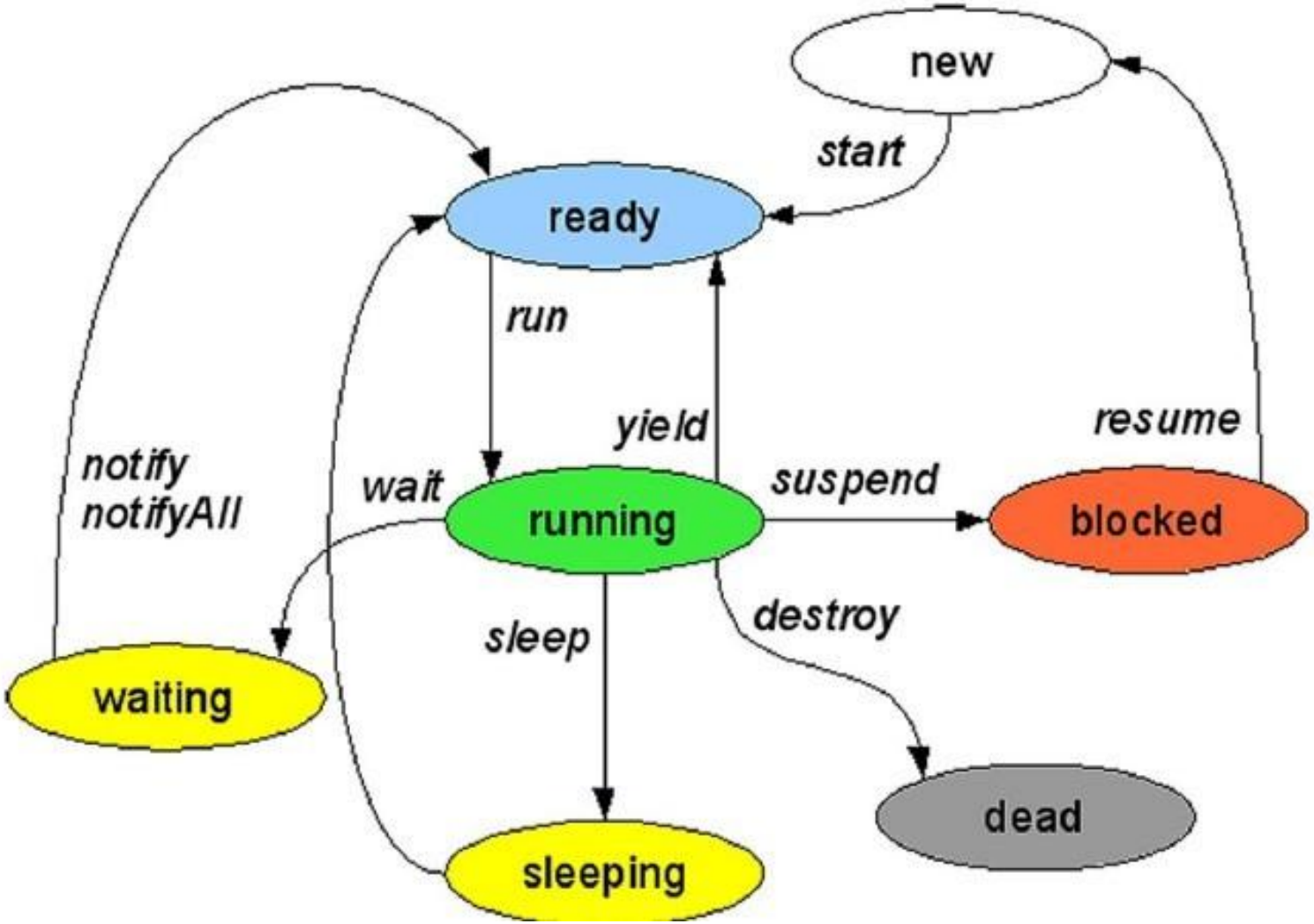
Classe Thread

- *Thread* encapsula todos os controles das linhas e execução.
- Um objeto *thread* é o procurador de uma linha em execução.
- Quando um programa Java é iniciado já existe uma linha sendo executada. O programa principal (*main*) de Java é uma linha em execução.

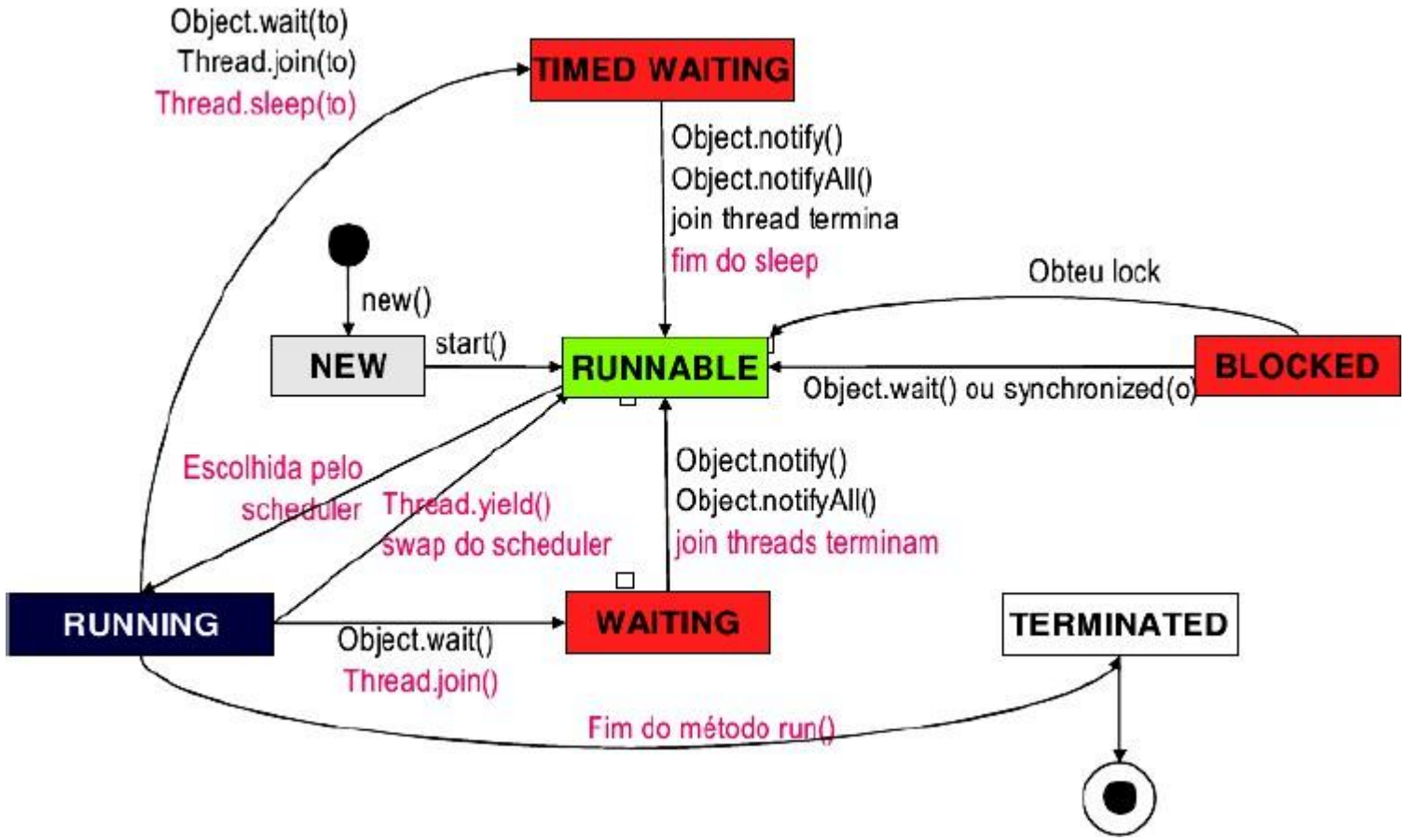
Classe Thread: principais métodos

- ***sleep***: passa como parâmetro um número inteiro de milissegundos que a *thread* deve ser interrompida.
- ***suspend***: interrompe temporariamente a *thread*, que pode ser reiniciada posteriormente.
- ***resume***: reinicia a *thread*.
- ***yield***: passa o controle para outra *thread*.
- ***notify***: libera um recurso para a próxima *thread*.
- ***notifyAll***: libera um recurso para todas as *threads*.
- ***wait***: aguarda pela liberação de um recurso.

Classe Thread



Estados de uma Thread



```python
import threading
import time

class Impressora:
    def __init__(self, nome, tempo_impressao):
        self.nome = nome
        self.tempo_impressao = tempo_impressao
        self.mutex = threading.Condition()
        self.imprimindo = False

    def imprimir(self, documento):
        with self.mutex:
            while self.imprimindo:
                self.mutex.wait()
            self.imprimindo = True
            print(f"{self.nome} iniciou a impressão do documento: {documento}")
            time.sleep(self.tempo_impressao)
            print(f"{self.nome} concluiu a impressão do documento: {documento}")
            self.imprimindo = False
            self.mutex.notify_all()


# Função para simular a impressão
def simular_impressao(documento, impressora):
    impressora.imprimir(documento)



# Criando as impressoras
impressora1 = Impressora("Impressora 1", 5)
impressora2 = Impressora("Impressora 2", 2)


# Criando as threads para os documentos
documento1 = threading.Thread(target=simular_impressao, args=("Documento 1", impressora1))
documento2 = threading.Thread(target=simular_impressao, args=("Documento 2", impressora2))
documento3 = threading.Thread(target=simular_impressao, args=("Documento 3", impressora1))
documento4 = threading.Thread(target=simular_impressao, args=("Documento 4", impressora2))

# Iniciando as threads
documento1.start()
documento2.start()
documento3.start()
documento4.start()

# Aguardando as threads terminarem
documento1.join()
documento2.join()
documento3.join()
documento4.join()

print("Todos os documentos foram impressos")
```

Importações de módulos: O código começa importando os módulos threading e time. O módulo threading é usado para criar e manipular threads em Python, enquanto time é usado para introduzir atrasos ou pausas no programa.

Definição da classe Impressora:
__init__: Este é o método construtor da classe Impressora. Ele inicializa os atributos de uma instância de Impressora com o nome da impressora (nome), o tempo de impressão de um documento (tempo_impressao), um objeto de bloqueio (mutex), que é utilizado para sincronizar o acesso concorrente ao recurso (a impressora), e uma variável booleana imprimindo, que indica se a impressora está atualmente ocupada ou não.
imprimir: Este método simula o processo de impressão. Ele adquire o bloqueio (mutex) da impressora e verifica se a impressora está atualmente imprimindo. Se estiver, ele espera até que a impressora esteja disponível. Quando a impressora está livre, ela é marcada como ocupada, o documento é impresso (apenas uma mensagem é impressa aqui, já que estamos simulando), e depois de um tempo de espera (representado por time.sleep(self.tempo_impressao)), a impressão é concluída. Finalmente, a impressora é marcada como livre novamente e uma notificação é emitida para qualquer thread em espera ser notificada.
Explicação das funções fora da classe:

selecionar_impressora: Esta função auxiliar percorre a lista de impressoras passada como argumento e verifica se alguma delas está livre (ou seja, não está imprimindo). Se encontrar uma impressora livre, ela a retorna. Caso contrário, ela aguarda um segundo antes de verificar novamente.
simular_impressao: Esta função é chamada pelas threads para simular o processo de impressão de um documento. Ela recebe como parâmetros o nome do documento a ser impresso e uma lista de impressoras. Utiliza a função selecionar_impressora para escolher uma impressora disponível e, em seguida, solicita à impressora selecionada para imprimir o documento.
Criação de instâncias de impressoras e threads: Duas instâncias da classe Impressora são criadas (impressora1 e impressora2). Em seguida, quatro threads são criadas para simular a impressão de quatro documentos diferentes.

Iniciando e aguardando threads: As threads são iniciadas com o método start() e o programa principal aguarda até que todas as threads terminem com o método join().

Mensagem final: Após a conclusão de todas as impressões, uma mensagem indicando que todos os documentos foram impressos é exibida.

Principais referências

- Bibliografia Básica:
  - BEM-ARI, M. Principles of concurrent and distributed programming. 2.ed. Pearson Addison-Wesley, 2006.
  - DEITEL, H. M.; DEITEL, P. J. C++: como programar. 5.ed. Pearson Prentice Hall, 2006.
  - TANENBAUM, A. S. Sistemas distribuídos: princípios e paradigmas. 2.ed. Pearson Prentice Hall, 2007.

Principais referências

- Bibliografia Complementar:
  - COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. Sistemas distribuídos: conceitos e projeto. 4.ed. Bookman, 2007.
  - DEITEL, H. M.; DEITEL, P. J. Java: como programar. 6.ed. Pearson Prentice Hall, 2005.
  - ECKEL, B. Thinking in C++, vol. 2. Upper Saddle River: Prentice Hall, 2004.
  - SEBESTA, R. W. Concepts of programming languages. 8.ed. Addison Wesley, 2007.
  - ZIVIANI, N. Projeto de algoritmos: usando C++ e Java. Pioneira Thomson Learning, 2007.