Nome: Pedro Gabriel Garcia Ribeiro Balestra		Matrícula: 1551
Curso: GEC	Período: P8	Matéria: C012

Cap 6 – Sincronização de Processos

1. **Race condition** é o acesso descontrolado a recursos compartilhado por vários processos ao mesmo tempo, causando inconsistências no sistema. Podemos prevenir isso utilizando da seção crítica, onde se utiliza de protocolos para que os processos possam utilizar o recurso compartilhado de maneira controlada.

2. **Seção crítica (CS)**, é um recurso computacional que utiliza de protocolos para orientar os processos a utilizarem do recurso compartilhado de maneira controlada. Para controlar a seção crítica, deve-se seguir as seguintes regras:

   - **Exclusão Mútua:** Um único processo é executado por vez na CS.
   - **Progresso:** Processo que não esteja utilizando da CS não pode impedir que outro processo a utilize.
   - **Espera Limitada:** Todo processo tem um tempo de limite de espera.

3. **Espera em ação** é quando um processo que está na espera para poder usar a CS, entra em um estado de loop, assim desperdiçando ciclos de CPU que algum outro processo poderia utilizar de maneira mais produtiva.

4. **Semáforos** é uma alternativa para controle de Seções Críticas, podendo controlar o acesso de N processos simultaneamente, utilizando-se de duas funções **wait()** e **signal()**. Temos dois tipos de semáforos:

   - **Semáforos Binários:** Valores variam entre 0 e 1, conhecido como locks mutex, sendo locks de exlusão mútua.
   - **Semáforos de contagem:** Funcionamento parecido como semáforos binários, porém utilizado para controlar acesso de uma quantidade finita de instâncias.

5. **Problema do buffer limitado:** Utilizando semáforos podemos controlar a entrada de dados já que o processo consumidor só pode consumir quando o processo produtor estiver produzindo algo dentro do buffer.

   Problema dos leitores/gravadores: Utilizando semáforos podemos controlar a escrita em um recurso compartilhado.

   Problema do jantar dos filósofos: Utilizando um semáforo controlamos o consumo e a ordem de disponibilidade, já que o jantar se trata de uma mesa com 5 filósofos, que desejam jantar, porém só podem quando ambos os palitos ao seu lado estiverem disponíveis.

6. **Monitor** é uma classe que fornece ao programador o controle de uma seção crítica, e somente um processo pode estar em execução dentro de um monitor. Pode ser implementado em várias linguagens.

7. Para prover uma exclusão mútua em Java basta colocar **synchronized** em um monitor.