//intro//
(slide 2)
Mutexes
-Definição
Mutexes (ou exclusão mútua) é uma técnica utilizada em programação concorrente para evitar que duas threads tenham acesso simultâneo compartilhado, acesso esse denominado por seção crítica. Em muitos casos é necessário que processos precisem acessar os mesmos recursos, e é preciso evitar que os arquivos fiquem corrompidos ou inconsistentes.

(slide 3)
Monitores
-Definição
Monitor é um mecanismo de sincronização de processos/threads que provê programação concorrente estruturada que concentra a responsabilidade para correção dentro de poucos módulos. Seções críticas tais como alocação de dispositivos de I/O e memória, requisições de enfileiramento para I/O, e assim por diante, são centralizados em um programa privilegiado. Programas ordinários requisitam serviços que são realizados pelo monitor central.

Simplificando, monitores basicamente tem o objetivo de impedir o acesso concorrente inadequado, obrigando uma thread a aguardar a outra terminar o que está fazendo.

(slide 4)
Semáforos
-Definição
Semáforo é uma variável especial protegida, do tipo inteiro que pode assumir valores maiores ou iguais a zero, sua função é o controle de acesso a recursos compartilhados (exemplo, um espaço de armazenamento) num ambiente multitarefa. Os semáforos podem ser classificados como binários ou contadores. Um semáforo contador representa simplesmente a alocação de determinado recurso, fornecendo a estes uma maneira de conversação que os possibilita sincronizar suas atividades. Enquanto os semáforos binários (também chamados de mutex) avisa para os processos se determinado recurso já está sendo utilizado.
A variável semáforo foi inventada em 1965, pelo holandês Edsger Wybe Dijkstra e foi usada inicialmente no sistema operacional THEOS. Essa função é comumente usada para controlar casos de Mutexes, travando antes de utilizar o recurso, e, após o uso, liberando o recurso.

Problema Produtor/Consumidor (slide 5-8)

(slide 5)
1-Definição:
É basicamente um processo que produz dados que serão consumidos por outro processo,os dados são armazenados temporariamente no buffer enquanto esperam sua vez de ser utilizado.
O produtor faz os dados e os envia para o **consumidor,** que recebe os dados e processa.

O problema ocorre pois o produtor só pode enviar cada dado quando o consumidor estiver preparado para recebê-lo, logo o produtor fica esperando ocioso enquanto o consumidor não estiver apto a receber os dados. Da mesma forma o consumidor sempre precisa esperar um dado enviado pelo produtor para dar continuidade o que também gera um tempo ocioso.
Portanto, como transferir dados do produtor para o consumidor sem que nenhum fique muito tempo ocioso?

(slide 6)
2-Solução:
Para eliminar o obstáculo de um processo ter que esperar o outro antes de enviar ou receber, a solução seria a implementação de um buffer, onde os dados ficam armazenados temporariamente. Desse jeito, o produtor passa os dados que produz para o buffer e o consumidor retira do dele, agora eles podem operar de forma independente.

(slide 7)
Problema relacionado ao buffer
Se o buffer ficar cheio o produtor não pode mais enviar dados para o consumidor, pois não teria onde alocar. No caso do buffer ficar vazio o problema seria do consumidor já que ele não teria o que retirar.

Solução do problema do buffer
A maneira de impedir esses problemas seria fazer com que o produtor faça uma pausa quando o buffer estiver cheio, dando tempo para que o consumidor retire elementos do buffer e o mesmo deve parar quando o buffer estiver vazio, a fim de que o produtor possa colocar elementos no buffer. Isso pode ser feito com o uso de mutexes controladas por um semáforo

(slide 8)
Variáveis semáforas
Para saber se o buffer está cheio ou vazio deve se utilizar as variáveis semáforas que são:

Full: para contar o número de espaços ocupados no buffer. Inicia com o valor 0. Espaços ocupados

Empty: para contar o número de espaços vazios no buffer. Inicia com o valor n. – Onde n é o número de espaços no buffer.

Mutexes:para garantir que consumidor e produtor não acessem o buffer ao mesmo tempo, ou seja, garantir a exclusão mútua. Começa com valor 1.

Exemplo (slide 9-11)
(slide 9)
Estas são algumas funções do problema produtor/consumidor escrito em linguagem de programação C.

(slide 10)

Esse trecho mostra o produtor escrito na mesma linguagem de programação.
(slide 11)
Já aqui vemos o trecho do consumidor

(slide 12-13)
aqui vemos como ficam quando juntos os processos produtor e consumidor.