**Estudo 3ª Frequência de SO**

* Concorrência Sincronismo

**Multiple Processes:** Design do SO preocupa-se com gestão de processos e threads

MultiProgramming

MultiProcessing

Distributed Processing

**Aponta em 3 Contextos Diferentes:**

Multiple Applications

Structured Applications

OS Structure

**Termos-Chave de concorrência:**

Atomic Operation: Função ou ação implementada em sequencia de uma ou mais instruções, que apresenta ser indivisível

Critical Section: Secção de código dentro de um processo que requer acesso a recursos partilhados

Deadlock: Situação onde 2 ou mais processos são incapaz de proceder porque um tem que esperar que o outro faça algo

Livelock: Situação onde 2 ou mais processos mudam os seus estados, continuamente como resposta a mudanças em outros processos

Mutual Exclusion: Requerimento onde 1 processo está numa zona critica onde acede a recursos partilhados

Race Condition: Situação onde múltiplos threads ou processos leem ou escrevem dados partilhados e o resultado final depende no seu tempo relativo de execução

Starvation: Situação onde um processo incapaz de correr é esquecido pelo escalonador

**Principios da Concorrência:**

Interleaving e Overlapping

Uniprocessor

**Dificuldades da Concorrência:**

Partilha de recursos globais

Dificuldade do SO em gerir a alocação optimizada dos recursos

Dificuldade de localizar erros de programação onde os resultados não são determinísticos

**Preocupações do SO:**

Design e gestão de problemas criados pela existência de concorrência

O SO deve:

Ser capaz de observar os vários processos

Alocar e Desalocar recursos para cada processo ativos

Proteger dados e recursos físicos de cada processos contra a interferência de outros processos

Assegurar que os processos e os outputs são independentes do tempo de processamento

**Resource Competition:**

Processos concorrentes entram em conflito quando estão a competir pelo mesmo recurso. Ex.: I/O, memory, etc…

**Requirements for Mutual Exclusion:**

Deve ser imposto

Sem deadlock ou starvation

Garantir acesso a secções criticas

Um processo permanece num secção critica por um tempo finito

**Mutual Exclusion -> Hardware Support:**

Interrupt Disabling: sistema uniprocessador ; desabilitar interrupções garante mutual exclusion

Disadvantages: decrementação notória na eficiência ; a aproximação não ira funcionar numa arquitetura multiprocessador

**Special machine instructions:**

Special machine instructions: Compare(feito entre o valor de memoria e o valor de teste) & Swap(se os valores forem iguais o “swap” ocorre)

Advantages:

Simples e fácil de verificar;

Pode ser usado para suporte de múltipla secções criticas, cada secção pode ser definida pela sua própria variável;

Aplicável a qualquer numero de processo num único processador ou em múltiplos processadores partilhando a memoria principal

Desvantagens:

Starvation é possível

Deadlock é possível

Busy-waiting é empregado

**Semaphore:**

1. Deve ser inicializado para um valor inteiro não negativo
2. A operação semWait decrementa o valor
3. A operação semSignal incrementa o valor

Consequências:

Incapacidade de saber se um processo apos descrementação do semaphore será block ou não

Quando damos signal ao semaphore , não precisamos de saber quando outro processos está a espera

Após incrementação do semaphore e outro processo “acorda”, ambos continua a correr conrrentemente. Não há forma de saber qual irá continuar imediatamente num sistema uniprocessador

Strong/Weak Semaphores:

Uma quele é usada para guardar os processos em espera no semaphore

Politica FIFO -> semaphore forte

Sem politica-> semaphore fraco

Implementação de Semaphores

semWait e semSignal serão implementados como primitivas atómicas(?)

Podem ser implementados em hardware ou firmware

**Alternativa aos Semaphores:**

Monitores

Rendez-vous

Hardware-supported: swap, text&set

Dekker’s or Peterson’s algorithms