**teoria 1**

falta diapo 16

**cosas que quedaron colgadas:**

**teoria 1**

Evolución Historia:

60’: Evolución del SO, no determinismo, problema de la SC.

70’: Formalización de concurrencia en los lenguajes.

80´: Redes, procesamiento distribuido.

90’: MPP, internet, Cliente/servidor, web computing

2000’: procesamiento masivo de datos distribuidos, clusters, computación móvil.

Concurrencia a nivel hardware:

**Multiprocesadores de memoria compartida:** La interacción se da modificando datos almacenados en la MC. Problema de consistencia

* Esquemas UMA con bus o crossbar switch (SMP, multiprocesadores simétricos). Problema de sincronización y consistencia.
* Esquemas NUMA para mayor número de procesadores distribuidos.

**Multiprocesadores con memoria distribuida.**

Procesadores conectados por una red. Cada uno tiene memoria local y la interacción es sólo por pasaje de mensajes.

Grado de acoplamiento de los procesadores:

* Multicomputadores *(tightly coupled machine*). Procesadores y red físicamente cerca. Pocas aplicaciones a la vez, cada una usando un conjunto de procesadores. Alto ancho de banda y velocidad.
* Redes (*loosely coupled multiprocessor*)*.*
* NOWs / Clusters.
* Memoria compartida distribuida.
* La Programación Distribuida es un “caso” de concurrencia con múltiples procesadores y sin memoria compartida.

resumen.pdf

Clases de aplicaciones

clase 5 monitores, no puse a partir de la 32.

**Lenguaje Occam**

Hoare introdujo CSP como lenguaje formal que sigue el modelo de PMS, pero nunca fue implementado en forma completa. OCCAM es un lenguaje real, que implementa lo esencial de CSP sobre la arquitectura “modelo” de los transputers.

Transputers *(multiprocesador de memoria distribuida de bajo costo)* + OCCAM = “sistema multiprocesador p/ procesamiento concurrente”, donde tanto la arquitectura como el lenguaje son simples y adecuadas a la programación concurrente PMS. OCCAM es un lenguaje simple con una sintaxis rígida. Los procesos y los caminos de comunicación entre ellos son estáticos (cantidades fijas definidas en compilación). Modelo de comunicación sincrónico por canales half dúplex.

Las sentencias básicas (asignación y comunicación) son vistas como procesos primitivos.

No soporta recursión, ni creación o nombrado dinámico → algunos algoritmos son difíciles de programar, aunque el compilador puede determinar cuántos procesos tiene un programa y cómo se comunican. Permite mapear procesos a procesadores del transputer.

Las Unidades básicas de programa son las declaraciones y 3 "procesos" primitivos: asignación, input (receive), output (sync\_send). Los procesos primitivos se combinan en procesos convencionales usando “constructores”: secuencial (SEQ), paralelo (PAR, similar a la sentencia co), y sentencia de comunicación guardada.

**Programación Paralela con el concepto de Bag of Tasks**

Idea: tener una “bolsa” de tareas que pueden ser compartidas por procesos “worker”.

Cada worker ejecuta un código básico

*while (true) {*

*obtener una tarea de la bolsa*

*if (no hay más tareas)*

*BREAK; # exit del WHILE*

*ejecutar tarea (incluyendo creación de tareas);*

*}*

Puede usarse p/ resolver problemas con un n° fijo de tareas y p/ soluciones recursivas con nuevas tareas creadas dinámicamente. El paradigma de “bag of tasks” es sencillo, escalable (aunque no necesariamente en performance) y favorece el balance de carga entre los procesos.

**Linda**

Aproximación distintiva al procesamiento concurrente que combina aspectos de MC y PMA. NO es un lenguaje de programación, sino un conjunto de 6 primitivas que operan sobre una MC donde hay “tuplas nombradas” (tagged tuples) que pueden ser pasivas (datos) o activas (tareas). Puede agregarse como biblioteca a un lenguaje secuencial. El núcleo de LINDA es el espacio de tuplas compartido (TS) que puede verse como un único canal de comunicaciones compartido, pero en el que no existe orden:

* Depositar una tupla (OUT) funciona como un SEND.
* Extraer una tupla (IN) funciona como un RECEIVE.
* RD permite “leer”como un RECEIVE pero sin extraer la tupla de TS.
* EVAL permite creación de procesos (tuplas activas) dentro de TS.
* Por último INP y RDP permiten hacer IN y RD no bloqueantes.