



Maestría en Ciencias de la Computación

Sistemas Operativos Avanzados

Profesor: Francisco Torres Rojas

Apuntes Lunes 13 de Febrero, 2017

Priscilla Piedra Hidalgo – 200822508

Quiz 1

1. Defina detalladamente los siguientes conceptos:
 - a. ENIAC Y EDVAC
 - b. Unidad de control
 - c. Ciclo de fetch
 - d. Microprogramación
 - e. Productividad y Eficiencia
 - f. Maquina dedicada
 - g. IBM 701
2. Explique las contribuciones principales de
 - a. John von Neumann
 - b. Maurice Wilkes
 - c. John Backus
 - d. John Eckert

Administrativo

- Tarea del 26 de febrero si es para esa fecha, es la primera tarea y la intención es que sea para ese domingo antes de la media noche (**media noche queda a discreción del profe, no mandar a las 11:56 pm**).
- Papers a leer: 0201, 0202, 0203, 0207

¿Dónde habíamos quedado? ¡Mejorando la maquina dedicada!

Los operadores fueron creados para reducir el tiempo que se consumía por los programadores cuando cada uno utilizaban las maquinas. Para mejorar los tiempos de productividad ya que, al ser maquinas dedicadas todos los programadores repetían tareas y no tenían un estándar en general.

El operador recolecta los trabajos y se factorizan los pasos, además ya exige una configuración estándar y es el nacimiento del procesamiento batch, esto incrementa rendimiento en centros de cómputo.

La cantidad de trabajos terminados por unidad de tiempo (throughput)



¿Se necesita un operador? El trabajo de él es muy mecánico, eran trabajos aburridos y fáciles de hacer, ¿cómo quitar el operador? ¡Haciendo un software! De acá el nombre de Sistema Operativo (despedir a los operadores.)

Sistemas operativos primitivos

Hay más compiladores disponibles, además menos fallas en el hardware ¿por qué no hacer un software de operación? Que tiene que hacer el SO: saber seleccionar el compilador, tenía que distinguir donde empezaba y terminaban el programa fuente (las tarjetas perforadas), compilar el programa, ejecutarlo y guardar resultados.

¿Cómo distinguir donde empieza un programa y donde otro y como saber que lenguaje es?

Pasos:

1. Va a correr el programa 1.
2. Se compila.
3. Se ejecuta.
4. Se generan resultados.
5. Así con el programa 2, 3, etc.



JCL

Anécdota de Torres

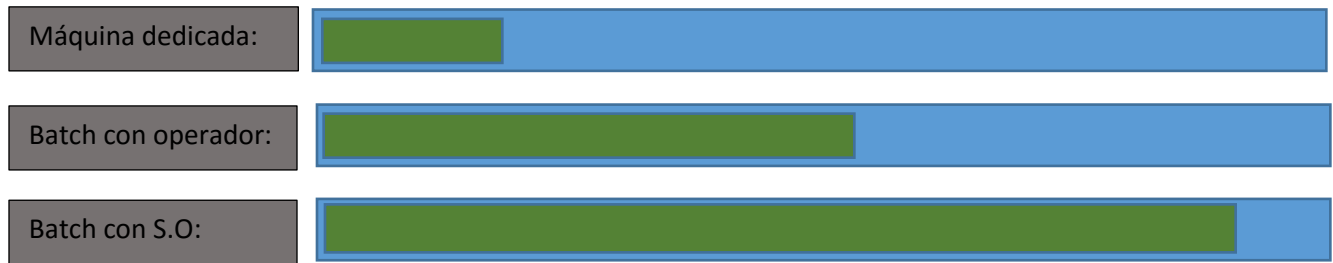
Cuando programaba tarjetas y el operador solo le daba un solo rollo de papel con los resultados, quería decir que no funcionó el JCL y perdía una tarjeta dorada☹. Un rollito con muchos papeles: por lo menos pasó el JCL se hizo algo. Antes todo era muy cruel.

Job Control Language, lenguaje de control, este lenguaje es el que sabía que programa era el que venía, el sistema operativo entendía el lenguaje y como interpretar lo que venía. Fue considerado el primer interfaz, en la actualidad el JCL (con crack☺) es la interfaz gráfica. Había que llevar cursos de JCL, una sintaxis critica con muchos asteriscos, si se equivocaba el SO daba error, JCL el primero fue hecho en lenguaje ensamblador, era muy estricto no toleraba nada fuera de lo que entendía (era un lenguaje primitivo).

Era una tarjeta perforada con lenguaje JCL y después venían las tarjetas con los programas, tal vez en fortran. Eran tarjetas intercaladas entre los trabajos de los usuarios.

Es el antecesor del “Shell lenguaje”.

Con JCL aún no se podía quitar al operador, alguien tenía que barrer el centro de cómputo, sacar las cintas y recolectar los trabajos procesados.



Entrada/ salida primitiva

Muy limitada, leer tarjetas y perforar tarjetas, después se podía leer tarjetas, leer de cintas, tal vez un plotter, cada dispositivo nuevo había que usarlo. Alguien escribía las rutinas para leer de estos dispositivos entonces se compartían las tarjetas que por ejemplo leían bien la impresora, lo ideal era que todos usaran la misma rutina y tenerla en algún lado (¿biblioteca?) Este código era considerado bibliotecas que iban incluidas en cada programa.

Entonces... por que no combinar las rutinas de entrada/salida y el sistema operativo siempre está en memoria... ¿por qué no combinar ambas? El SO puede entender la I/O, que siempre estén disponibles en memoria como parte del código del SO.

¿Cómo se usan ahora las I/O? Nada más se le pide al programa que dispositivo usar, los programas se comunican con el SO para pedirle operaciones.

¿Cuáles son las ventajas?

Menos trabajo, rutinas están buenas, son eficientes.

¿Desventajas?

No conoce todos los dispositivos, quitan los accesos a los dispositivos, si hay uno nuevo hay que escribir la rutina.

Pero ganaron las ventajas😊

- Fue todo un éxito y se generalizo, se crean los servicios de SO. Es lo que se conoce en la actualidad como device driver (librería que sabe cómo usar un dispositivo, es software), la responsabilidad se fue pasando a los creadores de hardware, cualquier hardware tiene su device driver que entiende el SO. Estos servicios dan paso a los **system calls**.
- Los programas ahora ocupaban tanto de la maquina como del sistema operativo, el programa tiene que saber que corre un frotran en un Linux, esto crea una capa nueva que rodea la maquina el SO donde los sistema calls son instrucciones virtuales. El sistema operativo va aumentando su complejidad, se fueron agregando mas system calls (nos hace la vida simple).
- Entre más system calls más recursos son consumidos.

Virtual vs transparente

Virtual: lo que se ve pero no existe físicamente, es de mentiras.



Transparente: lo que existe pero no se ve. Cuando se manda un email nadie piensa si la tarjeta de red va a funcionar o no, hay todo un protocolo de redes... Pero nadie se preocupa por eso.



Computación: nos dedicamos a crear cosas virtuales a través de cosas transparentes.

Real: lo que existe y lo que se ve, como el teclado.

El sistema operativo se convierte en la primera máquina virtual.

Ahora los programadores piensan en los SO, ya no en la maquina donde va a correr.



A todos nos gustan los SO, ahora no hay compu sin SO, nadie sale a comprar el SO, antes si, se compraba o se pirateaba. Si la compu no tenía SO, se quedaba ahí botada. El SO quita el hardware pero a cambio da conveniencia y servicios.

Generalización Maquinas Multinivel

La aplicación en última instancia es una maquina virtual, como SAP (la aplicación de conta), toda aplicación puede ser considerada una máquina virtual y gano conveniencia y pierdo eficiencia.



Multiprogramación

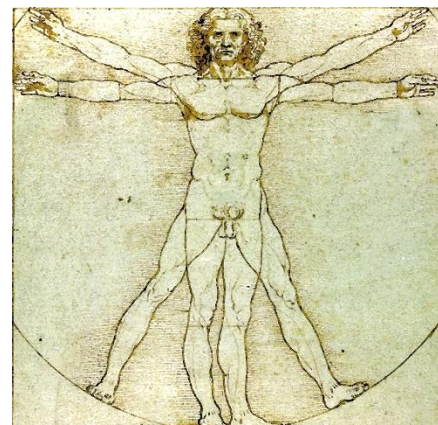
Diferencia de velocidad: CPU siempre ha sido más rápida que la memoria y la I/O. Entrada salida es más lento que RAM y RAM más lento que CPU, con forme se avanza la brecha es más grande entre todos ellos. Con esto el CPU se vuelve ociosa, es uno de nuestros enemigos en SO, este es el recurso más caro de una compu. ¿Cómo se ve esto en escala humana?

Escala Humana

Los humanos no manejamos el concepto de grandes espacios de tiempo.

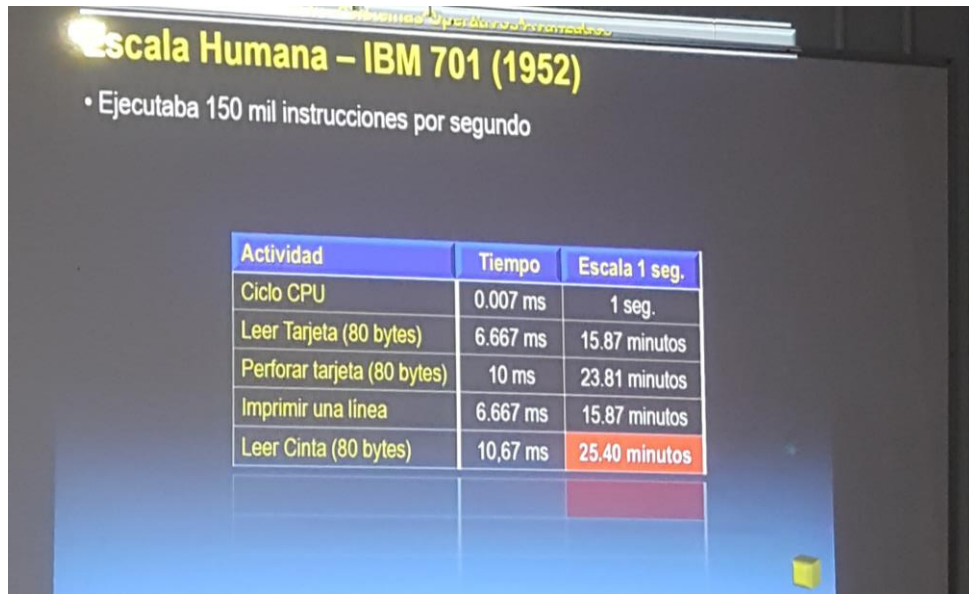
Nos cuesta mucho entender el tiempo, o son pequeños a grandes.

Supongamos que cada ciclo de CPU toma 1 segundo.



Escala humana IBM 701

Podía ejecutar 150mil instrucciones por segundo.

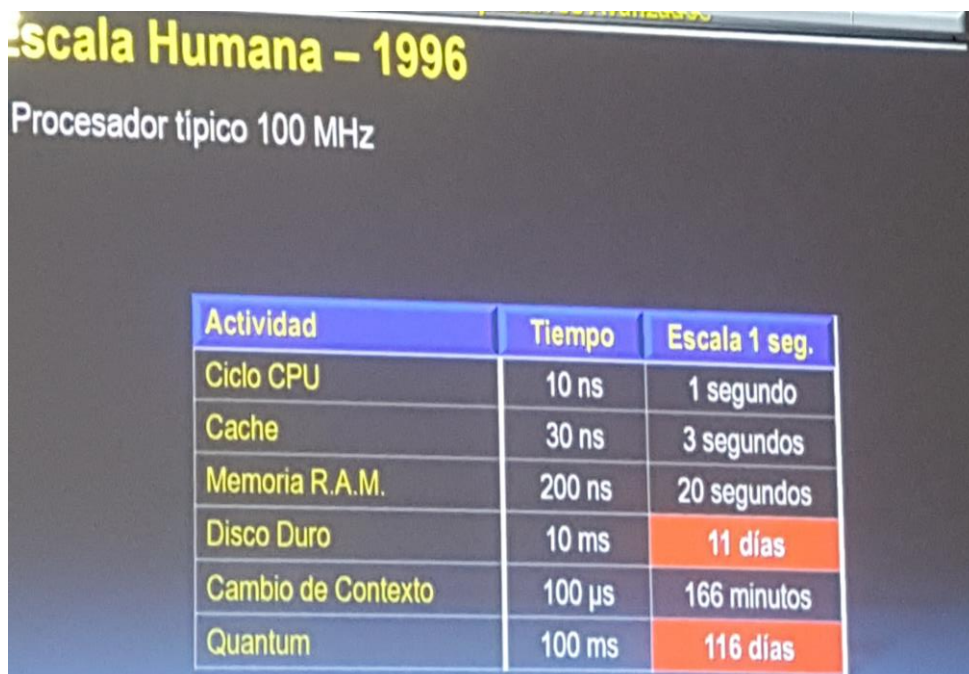


Escala Humana – IBM 701 (1952)

- Ejecutaba 150 mil instrucciones por segundo

Actividad	Tiempo	Escala 1 seg.
Ciclo CPU	0.007 ms	1 seg.
Leer Tarjeta (80 bytes)	6.667 ms	15.87 minutos
Perforar tarjeta (80 bytes)	10 ms	23.81 minutos
Imprimir una línea	6.667 ms	15.87 minutos
Leer Cinta (80 bytes)	10,67 ms	25.40 minutos

Escala humana en 1996



Escala Humana – 1996

Procesador típico 100 MHz

Actividad	Tiempo	Escala 1 seg.
Ciclo CPU	10 ns	1 segundo
Cache	30 ns	3 segundos
Memoria R.A.M.	200 ns	20 segundos
Disco Duro	10 ms	11 días
Cambio de Contexto	100 μ s	166 minutos
Quantum	100 ms	116 días

Escala humana- 2014

procesador típico 3.9 GHz

Actividad	Tiempo	Escala 1 seg.
Ciclo CPU	0.256 ns	1 segundo
Cache – L1	1.026 ns	4 segundos
Cache – L2	3.077 ns	12 segundos
Cache – L3	6.154 ns	24 segundos
Memoria R.A.M.	8.4 ns	32 segundos
Disco Duro – mejor caso	2.9 ms	132 días
Disco Duro –peor caso	12 ms	1.5 años
SDD	85 µs	3 días y 20 horas
Cambio de Contexto	10 µs	10.8 horas
Quantum	100 ms	12.4 años

Conclusión

Es una crueldad que el CPU se quede sin hacer nada... ¿qué hacer? Mientras se espera un dispositivo conteste ponemos a correr otro programa. ¿Cuál fue el problema? Hacer multiprogramación. Varios programas coexisten con el SO. Alguien corre pero algún día hay que volver al proceso que deje corriendo. Este es uno de los retos más grandes de la computación.

El grado de multiprogramación era fijo y bajo, además habían particiones estáticas y había una garantía: **que los programas no interfirieran entre ellos y que den los mismos resultados que sin multiprogramación.**

Protección



permite que el SO corra cuando sea necesario y garantizar protección.

- Exceptuando el tiempo de ejecución un programa se ejecuta de manera idéntica si corre solo en una maquina **monoprogramada** o si corre en una maquina **multiprograma** con cualquier combinación de otros programas.

- Se garantiza que cada programa este protegido de los otros, en principio solo hay un CPU, **¿quién es el responsable de dar protección?** El SO. **¿Como?** Por software o por hardware cuando es muy complicado, se necesita un mecanismo de hardware que me

Alteración del flujo de ejecución (A.F.E)

1. Programa reside en memoria.
2. PC contiene dirección de instrucción actual que se está ejecutando.
3. Ciclo fetch.
4. Establece un flujo de ejecución.

¿Cómo se altera? **A.F.E:**

A.F.E

1. El hardware va a ejecutar lo que diga el PC.
2. Si se fuerza un valor en el PC se ejecuta otro código.
3. Alteración del flujo de ejecución:
 - a. Guardar el valor actual del PC.
 - b. Cargar un nuevo valor en el PC.
 - c. Ejecutar el nuevo flujo.
 - d. Restaurar el valor previo del PC.
 - e. Regresar al flujo previo.

La A.F.E es un flujo transparente pues el código no se da cuenta que fue alterado el A.F.E, esto es un concepto de hardware o **EXEPCIONES**.

Detalles de las excepciones

Hay muchos tipos de excepciones, es un número único para cada situación.

¿A dónde vamos a dar? A una rutina especialista en atender la excepción dada.

¿Quién proporciona la rutina? El sistema operativo,

¿De dónde viene el nuevo valor del PC? Del vector o tabla de direcciones indexado por número único de excepciones.

¿Quién carga los valores de la tabla o vector? El sistema operativo.

**PREGUNTAS
IMPORTANTES
PARA SOA!!!**

Taxonomía de excepciones

Hay muchos nombres: excepción, interrupción, trap, fault etc todas son alteraciones al flujo de ejecución, se pueden clasificar de diferentes maneras: **temporal, detección y origen**.

a) Temporal

Una clasificación de naturaleza temporal: sincrónicas y asincrónicas.

Sincrónicas: tienen un patrón que es repetible, siempre se van a dar en el mismo instante relativo, esto casi siempre es provocado por software, esa excepción va a ocurrir n veces, un error o causas externas sincrónicas, son predecibles, el nombre más aceptado es trap, **también son llamadas interrupciones de software**, por ejemplo **system call**, división por cero, page fault, etc. Normalmente cuando se divide por cero el SO mata el proceso.

Asincrónicas: no son predecibles, si corro el programa un millón de veces no sé cuándo se van a dar, son causadas por eventos externos, como tocar el mouse, tocar el teclado (provocado por hardware)... no sé cuándo van a ocurrir, son difíciles de reproducir, **se llaman interrupciones**.





b) Por detección

¿Cuándo me doy cuenta que paso cierta situación? Puede ser durante la ejecución o después. Durante quiere decir que es el programa el causante, de lo contrario quiere decir que viene externo. Pueden ocurrir en el MAR o en MDR o cuando se ejecuta.

Foto 11

Detección antes o durante la ejecución: Cuando la ejecución de la instrucción provoca la excepción, por ejemplo la instrucción podría ser por: página no presente, código de operación inexistente, instrucción privilegiada, falla falta de hardware. Por argumento: violación de memoria, división entre cero, pagina no

presente, falla falta de hardware... Todos estos **se conocen como Errores o faults.**

Detección después de la ejecución: La instrucción se completó, puede ser intencional como un sistema call, puede ser un trap, puede ser por causa externa también como el hardware, en este caso no hay que repetir la instrucción, el PC señala a la siguiente instrucción.

c) Por origen

Interno: por intención, como un system call, o bien no intencional como un error (división por cero, overflow, instrucciones privilegiadas), transparente (memoria virtual, máquina virtual, D.S.M).

Externo: por una falla de hardware (motherboard, memoria, etc.) o un dispositivo que ya completo su tarea (E/S, timer, G.P.U).

Continuación de protección



Para poder proteger algo es necesario una alteración. La protección se va a ver en tres áreas claves: protección de CPU, memoria y I/O.

Protección del cpu: Programas deben terminar en un ciclo finito, siempre tiene que haber CPU disponible para todos los programas, no que haya un programa que se lleve toda la CPU. Por software no hay forma de saber que un programa esta ciclado, para eso fue necesario el

hardware y así nació el timer que genere una alteración del flujo de ejecución y revise que todo esté en orden.

Protección de memoria: Que no escriba en otro espacio de memoria de otro programa. ¿Cómo se evita que un programa genere una dirección fuera del rango? Solo por hardware con registros de frontera. La CPU manda operación, dirección y el dato sin protección, con protección el cpu manda la operación, dirección y dato, en medio hay hardware adicional que son los registros de inicio y final para que se dé cuenta que la dirección este contenida en el rango que le corresponde, si esta fuera del rango va a disparar una excepción que va a ser resuelta por el SO que va a llamar a la rutina encargada en atender esa excepción. El hardware va a disparar un FAULT, fue algo que se detectó durante la ejecución de la excepción. Lo normal es que el SO mate la rutina.

Protección de I/O: Impide que un programa use los dispositivos de i/o de manera directa. ¿Cómo se hace para frenar el problema que un programa use una instrucción de i/o? con privilegios en el hardware, se necesita distinguir entre los procesos.

Instrucciones privilegiadas: la arquitectura tiene un conjunto finito de instrucciones, algunas son peligrosas o privilegiadas, la unidad del control tiene que distinguir el modo y que tipo de instrucción se esta haciendo. El modo del CPU.

Modo privilegiado o no privilegiado: Hay un bit dentro de la instrucción que dice si es privilegiada o no, una no privilegiada siempre se ejecuta (como shift a la izquierda), una privilegiada se puede hacer si el hardware esta en modo privilegiado. Cundo el SO corre (boot) está en modo privilegiado, cuando corre cualquier otro programa está en modo no privilegiado. La alteración del flujo de ejecución cambia solo para que corra el SO si esta en modo privilegiado.