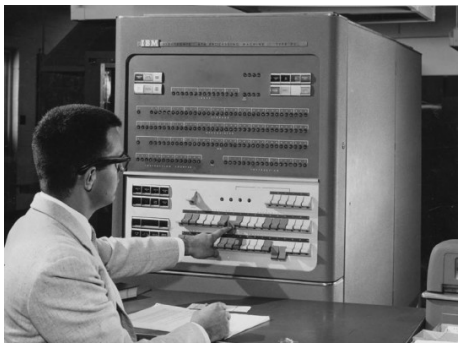
**Instituto Tecnológico de Costa Rica**

**Sistemas Operativos Avanzados**

# Apuntes 13 de Febrero



**Kelvin Jiménez - 200969475**

**Profesor: Dr. Francisco Torres Rojas**

Programa de Maestría en Ciencia de Computación
San José, 2017

# Índice general

# 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) Máquina dedica
   g) IBM 701
2. Explique las contribuciones principales de
   a) John von Neumann
   b) Maurice Wilkes
   c) John Backus
   d) John Eckert

# Sistemas Operativos

## 2.1. Sistemas Operativos Primitivos

En esta época hay más compiladores disponibles: Fortran, Algol, Cobol entre otros. La manera de proceder con los trabajos era mediante la clasificación y compilación **Batch**. Con esto se reducían las fallas de hardware, y se da la aparición de más periféricos.

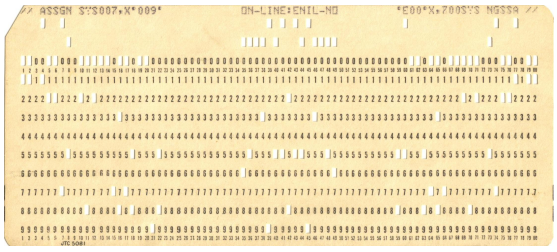
**Idea: Sistema monitor** un software que remplace la ejecución de estas tareas mecánicas.

- Seleccionar Compilador.
- Compilar.
- Ejecutar.
- Guardar Resultados.

Algo como un monitor o un "sistema operador.º un **"Sistema Operativo"**. Sin Embargo, ¿Cómo se sabe hasta donde llega cada trabajo?.

### 2.1.1. Job Control Language (JCL)

Se crea este lenguaje, y el sistema operativo conoce sobre este. Es la primera interfaz **Usuario - Sistema Operativo**. La manera en que se utilizaba era intercalando tarjetas entre los trabajos y sus diferentes secciones. Se puede decir que este tenía una sintaxis primitiva. Además es el antecesor de los **"Shell Languages"**.

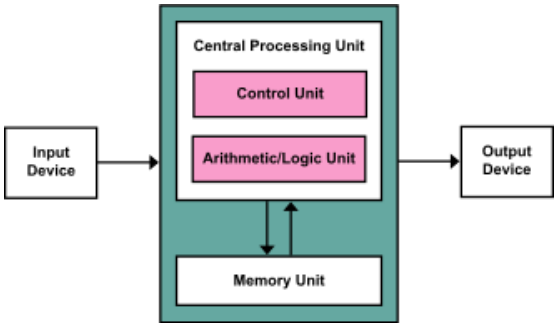


### 2.1.2. Productividad de Centros de Cómputo.

Cantidad de trabajos terminados por unidad de tiempo **throughput**.

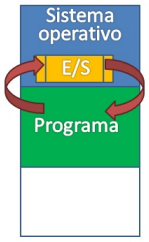

### 2.1.3. Entradas/Salidas (E/S) Primitivas

La arquitectura de von Neumann sugiere un espacio de direcciones para E/S. Se tenía que escribir rutinas específicas para los diferentes dispositivos. Pero esto no era fácil, además que los dispositivos nuevos eran más complejos. Lo que generó fue un fenómeno entre los programadores, ya que si alguna persona había escrito estas rutinas satisfactoriamente, los programadores las compartían y estas se agregaban como bibliotecas de E/S a cada programa.




**Idea : E/S + S.O.** Todos ocupan el acceso a las entradas y salidas, entonces siempre tiene que estar disponible. La idea consiste en colocar en el sistema operativo estas librerías, y que los programas se comunique con el sistema operativo para solicitar operaciones de E/S.

**Ventajas**




- Menos trabajo
- Eficiente
- No se tiene acceso al dispositivo por completo.
- Se generaliza a servicios del S.O.

### 2.1.4. Servicios del sistema operativo.


**Device driver** es un Software Especialista en manejar un dispositivo.
**Idea:** Se genera una nueva capa como máquina virtual, los system calls son instrucciones de ese sistema operativo (Instrucciones virtuales), de hecho cuando los compiladores generan código lo hacen a esta "máquina virtual". Se agregan más servicios y el sistema operativo crece en tamaño y complejidad.

## 2.2. Sistema operativo como máquina Virtual

**Virtual:** se ve pero no existe.
**Transparente:** no se ve, pero existe.
**Real:** Lo que existe y se ve.

En los sistemas operativos se crean cosas virtuales a través de medios transparentes. Esto quiere decir que es como una máquina que vemos pero no existe. De esta manera el sistema operativo nos quita el hardware pero da conveniencia y servicios. Además es más fácil de usar pero menos eficiente.

## 2.3. Diferencias de velocidad

La CPU siempre ha sido más rápido que la memoria y E/S. Cada vez esta brecha se hace más grande.

Entonces el C.P.U. está ocioso y este es el recurso más caro de una computadora. Por lo que es un desperdicio de recursos.


### 2.3.1. Escala Humana

Desafortunadamente nos cuesta mucho entender tiempos muy pequeños o muy largos.

**por ejemplo:** ¿10ns? ¿65 millones de años?

Por esa razón a continuación se muestra una conversión de los ciclos de CPU a una escala humana, o más comprehensible. CPU cada ciclo de CPU toma 1 segundo.

#### Escala Humana - IBM 701

Ejecutaba 150 mil instrucciones po segundo.

| Actividad | Tiempo (ms) | Escala 1 seg. |
|---|---|---|
| Ciclo CPU | 0.007ms | 1s |
| Leer Tarjeta (80 bytes) | 6.667ms | 15.87m |
| Perforar tarjeta (80 bytes) | 10ms | 23.81m |
| Imprimir una linea | 6.667ms | 15.87m |
| Leer Cinta (80 bytes) | 10.67ms | 25.40m |

#### Escala Humana - 1996

Procesador típico de 100MHz.

| Actividad | Tiempo (ms) | Escala 1 seg. |
|---|---|---|
| Ciclo CPU | 10ns | 1s |
| Cache | 30ns | 3s |
| Memoria R.A.M. | 200ns | 20s |
| Disco Duro | 10ns | 11 días |
| Cambiar de Contexto | $100\mu s$ | 166m |
| Quantum | 100ms | 116 días |

**Escala Humana - 2014**

Procesador típico de 3.9GHz.

| Actividad | Tiempo (ms) | Escala 1 seg. |
|---|---|---|
| Ciclo CPU | 0.256ns | 1s |
| Cache L1 | 1.026ns | 4s |
| Cache L2 | 3.077ns | 12s |
| Cache L3 | 6.154ns | 24s |
| Memoria R.A.M. | 8.4ns | 32s |
| Disco Duro mejor caso | 2.9ms | 132 días |
| Disco Duro peor caso | 12ms | 1.5 años |
| SDD | $85\mu s$ | 3 dias y 20h |
| Cambiar de Contexto | $10\mu s$ | 10.8h |
| Quantum | 100ms | 12.4 años |

### 2.3.2. Multiprogramación

**Correr otro programa...**

Como se vio en las tablas anteriores entonces se empezo a tratar de reduciendo el desperdicio de recursos.

**idea** Mientras se espera que un dispositivo lento responda, se pone a correr otro programa en el CPU y a esto se le conoce como **multiprogramación**.

De modo que varios programas coexisten en la memoria con el sistema operativo. Algún día se regresa al proceso que esperaba. En sus inicios el grado de multiprogramación era fijo y bajo. Además que las particiones eran estáticas.

Aunque esto suena fácil de imaginar que de construir, de hecho es uno de los retos tecnológicos más grandes de la historia de la computación.

**Garantía:**

- Excepto el tiempo de ejecución, un programa se ejecutará de manera **idéntica** en máquina monoprogramada o multiprogramada con **cualquier** combinación de otros programas.

**Protección**

Se garantiza que cada programa esta protegido de los otros. En principio solo hay un CPU.

> ¿**Quién** es el responsable de protección?
> El sistema operativo.¿**Cómo** logra el sistema operativo dar protección?
> ¿Software? ¿Hardware? Se necesita un mecanismo de hardware que permita al sistema operativo corra cuando sea necesario para garantizar la protección.

## Resúmenes: 201*,202,203,207*

## 2.4. Alteración de flujo de ejecución (A.F.E)

EL hardware va a ejecutar lo que diga el P.C. Si se fuerza un valor en el P.C. se ejecuta otro código a esto se le conoce como alteración del flujo de ejecución. Y se logra mediante los siguientes pasos:

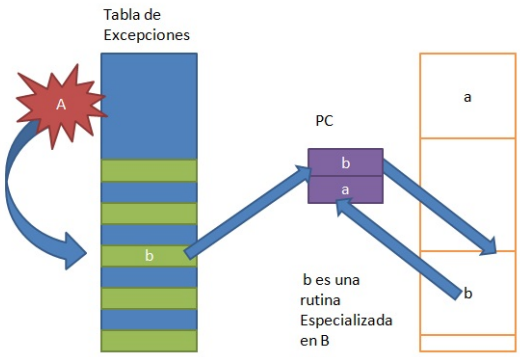
1. Guardar el actual P.C.
2. Cargar un nuevo valor
3. Ejecutar el nuevo flujo.
4. Restaurar el P.C
5. Regresar el flujo previo.

### 2.4.1. Detalle de Excepciones

Hay muchos tipos de Excepciones, y existe un número único para cada situación.

- ¿A dondé va a dar?
  Rutina especialista en atender la excepción dada.
- ¿Quién proporciona esta rutina?
  El sistema operativo.
- ¿De donde viene el nuevo valor del PC?
  Vector o tabla con direcciones indexada por número único.

**Tabla de Excepciones**



### 2.4.2. Taxonomía de Excepciones

Muchos de los nombres que reciben son: Excepción, Interrupción, Trap, Fault Etc. y todas estas son alteraciones al flujo de ejecución.

### 2.4.3. Clasificación: Temporal, Origen, Detección

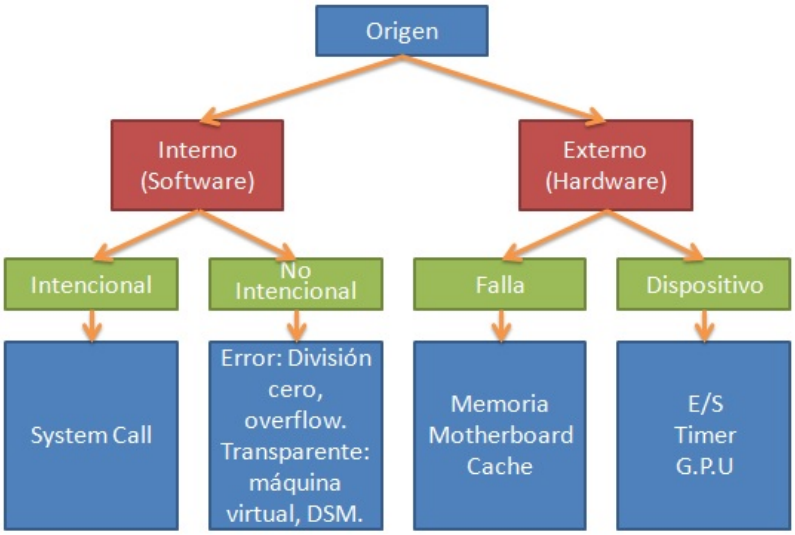
**Tiempo**

| Sincrónico | Asincrónico |
|---|---|
| • Siempre se da en el mismo instante relativo. Si se corre N veces, ocurre N veces, en el mismo instante. | • Ninguna relación temporal con el programa que está corriendo. |
| • Puede ser intencional, por error o por causas externas sincrónicas. | • Impredecibles, muy difíciles de reproducir. Causas externas. |
| • **Trap:** interrupción de software Ej. System Calls, división por cero, page fault. | • **Interrupciones:** Hardware externo. Ej. Dispositivo de E/S, timer. |

**Detección**

| Durante | Despúes |
|---|---|
| • La ejecución de la instrucción provoca la excepción. | • La instrucción se completo. |
| • **Instrucciones:** Página no presente, código de operación inexistente, instrucción privilegiada, falla fatal de hardware. | • **Intencional:** Hay instrucciones cuyo propósito es generar una excepción, son sincrónicas, eje. Traps. |
| • **Argumentos:** Violación de memoria, división por cero, página no presente o falla fatal de hardware. | • **Externa:** Hay una causa totalmente externa a la instrucción, son asincrónicas, relacionadas con el hardware. |
| • **Errores o fault.** ¿Repetir la instrucción? o No avanzar el PC. | • **Interrupción.** No hay que repertir la instrucción, el PC señala a la siguiente instrucción. |

**Origen**




## 2.5. Protección

Cada programa esta protegido mediante hardware en los siguientes componente: Protección CPU, Memoria, E/S.

### 2.5.1. Protección de CPU

Los programas deben terminar en un tiempo finito, sin ciclos infinitos, el sistema operativo es software que necesita la CPU para correr por lo que este no puede garantizarlo, por lo que este hace uso de hardware específicamente un **timer** y una interrupción que va a ser procesada por una rutina especialista en en atender el timer y esta toma alguna de las siguientes decisiones.

- Regresar al programa y continuar.
- Cancelar el programa.
- Suspender el programa.

### 2.5.2. Protección de memoria

La memoria de cada programa (incluido el S.O.) no debe ser afectado por otro programa. Hay que evitar que un programa genere

direcciones fuera de sus espacio asignado, pero no hay forma de hacer por software, por lo que se logra con hardware nuevamente, se le conoce como **registros de Frontera.**

- **Uso de memoria sin protección**


- **Uso de memoria con protección**


### 2.5.3. Protección de E/S

La arquitectura von Neuman sugiere un espacio de direcciones de E/S, Se usaban instrucciones de lenguaje máquina para ese espacio. El manejo de operaciones de E/S migró hacia el S.O. Pero aún alguien puede escribir esas rutinas directas de E/S. No queremos que los programas usen indebidamente los dispositivos. Menos si hay varios programas que coexisten.

Hay que generar un mecanismo para evitar que los programas ejecuten esas instrucciones "peligrosas", no se puede confiar ni el programador ni en los compiladores. Además esto no se puede lograr mediante software. Se tiene que hacer uso del hardware.

**Instrucciones privilegiadas**


La arquitectura tiene un conjunto finito de instrucciones. Un conjunto de ellas son peligrosas y especiales. Se desea que solo el S.O. las pueda ejecutar. Nuevamente hay que utilizar el hardware y se introduce el concepto de **modo del CPU**.

Para la implementación del modo privilegiado y no privilegiado. Se tiene que imaginar un bit que indica el modo del procesador, además tiene que existir un bit dentro de las instrucciones para clasificarlas. Una instrucción privilegiada se puede ejecutar siempre, pero una privilegiada sólo se puede si el hardware esta en modo privilegiado.

No existe una instrucción para el cambio de modo por lo que las A.F.E provocan ese efecto secundario, al inicio la máquina esta en modo privilegiado para que el sistema operativo pueda organizar todo lo necesario.