

Arquitectura de Sistemas

Historia de los sistemas operativos

Gustavo Romero López

Updated: 20 de marzo de 2025

Departamento de Ingeniería de Computadores, Automática y Robótica

1. Definición
2. Historia
3. Estructura
4. Ejemplos
5. Comparativa

Lecturas recomendadas

Jean Bacon	Operating Systems (2, 26)
Abraham Silberschatz	Fundamentos de Sistemas Operativos (2)
William Stallings	Sistemas Operativos (2)
Andrew Tanuenbaum	Sistemas Operativos Modernos (1,12)

Motivación

- ⦿ La **arquitectura** de un SO marca de forma vital su funcionamiento.
- ⦿ Cada posible elección tendrá **consecuencias** ineludibles.
- ⦿ Ejemplo: el **compromiso** velocidad/espacio:

```
unsigned bits(char elem)
{
    unsigned count = 0;
    do count += elem & 1;
    while (elem >>= 1);
    return count;
}
```

```
unsigned bits(char elem)
{
    unsigned count;
    asm("popcnt %1, %0" : "=r"(count)
        : "r"(elem));
    return count;
}
```

```
#define bits(b) (b&1 + b&2 + b&4 + b&8 + b&16 + b&32 + b&64 + b&128)
```

```
char bits[256] = {0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, ... 8};
```

¿Qué es un sistema operativo?

- ⊙ Programa o conjunto de programas encargado de **gestionar los recursos** de la máquina y **proveer servicios** al resto de programas.
- ⊙ Se ejecuta en modo **privilegiado**.
- ⊙ Suelen venir junto con multitud de otros programas, que sin formar parte del sistema operativo, resultan de suma utilidad: intérprete de órdenes, editor de texto, gestor de ficheros, navegador,...

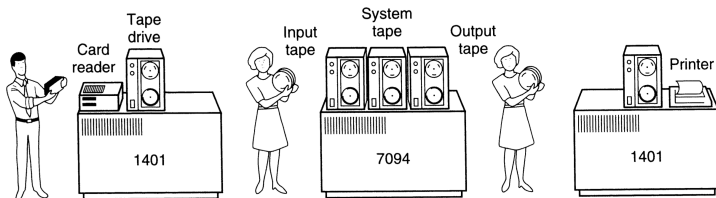
- ⊙ Primera generación (1945-55)
- ⊙ Segunda generación (1955-65)
- ⊙ Tercera generación (1965-80)
- ⊙ Cuarta generación (1980-hoy)

Primera generación (1945-55): Tubos de vacío y paneles

- ⊙ Utilidad: máquinas de cálculo.
- ⊙ Tecnología: dispositivos mecánicos $\Rightarrow$ tubos de vacío y paneles.
- ⊙ Método de programación: cables $\Rightarrow$ interruptores y tarjetas perforadas.
- ⊙ Diseño/construcción/operación/-programación/mantenimiento: genios como Aiken, von Newman o Mauchley.

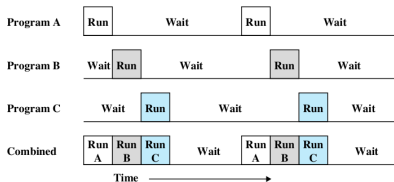
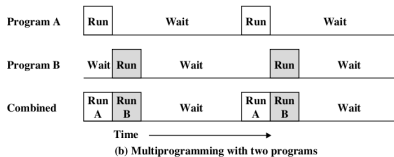
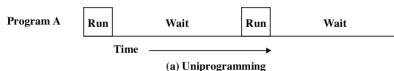
Segunda generación (1955-65): Transistores y sistemas por lotes

- ⦿ Utilidad: cálculo científico e ingeniería.
- ⦿ Tecnología: la invención del transistor redujo su tamaño y precio y los popularizó ⇒ **mainframes/IBM 1401/7094**.
- ⦿ Método de programación: **ensamblador** y lenguajes de alto nivel (**FORTRAN**) sobre tarjetas perforadas.
- ⦿ Paso de procesamiento **secuencial** a procesamiento **por lotes**.
- ⦿ Ejemplos: **FMS** y **IBSYS**.



Tercera generación (1965-80): Circuitos integrados y multiprogramación

- ⊙ 2 usos principales:
 - cálculo científico e ingeniería.
 - procesamiento de caracteres.
- ⊙ **Circuito integrado** $\Rightarrow$ +barato $\Rightarrow$ +popular $\Rightarrow$ IBM 360, GE-645, DEC PDP-1.
- ⊙ Logros destacables:
 - **multiprogramación.**
 - **spooling.**
 - **tiempo compartido.**
- ⊙ Ejemplos: OS/360, CTSS, MULTICS, UNIX.



Cuarta generación (1980-hoy): Ordenador personal (era μ)

- ⊙ (V)LSI $\Rightarrow$ ++barato $\Rightarrow$ ++popular $\Rightarrow$ IBM PC.
- ⊙ μ P: 8080, Z80, 80x86, Alpha, Ultrasparc, ARM, RISC-V.
- ⊙ Logros destacables:
 - GUI.
 - SO de red.
 - SMP.
 - SO distribuidos.
 - IoT.
- ⊙ Ejemplos: UNIX, CP/M, MS-DOS, Linux, MacOS, Windows.



Figura: Amstrad CPC 472



Figura: IBM PC

Clasificación de SO según su estructura interna

⊙ Clasificación:

- Estructura simple:
 - monolíticos
 - capas
 - modulares
- Estructura cliente/servidor:
 - micronúcleo
 - exonúcleo
- Máquina virtual.
- Híbridos.

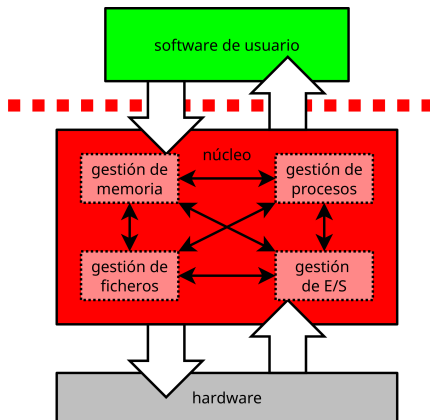
⊙ Tendencias:

- Núcleos extensibles.
- Multiservidores sobre un micronúcleo.
- Núcleos híbridos.

Monolítico

- ⦿ El SO completo se ejecuta en modo protegido.
- ⦿ Nula protección entre los componentes.
- ⦿ Ventajas:
 - Economía de cambios de contexto $\Rightarrow$ **+eficiente.**
- ⦿ Inconvenientes:
 - Falta de protección $\Rightarrow$ **-fiabilidad** (controladores).
 - Manejo de la complejidad: Es más

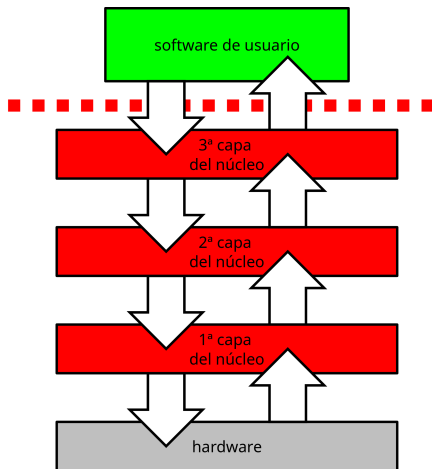
sencillo escribir 10^3
programas de 10^3
líneas que uno de 10^6 .



Capas/Niveles

- ⊙ El SO completo se ejecuta en modo protegido.
- ⊙ Escasa protección entre los componentes.
- ⊙ Ventajas:
 - Economía de cambios de contexto ⇒ **+eficiente.**
 - Menor complejidad.
- ⊙ Inconvenientes:
 - Falta de protección ⇒ **-fiabilidad** (controladores).
 - Menos flexible que monolítico.

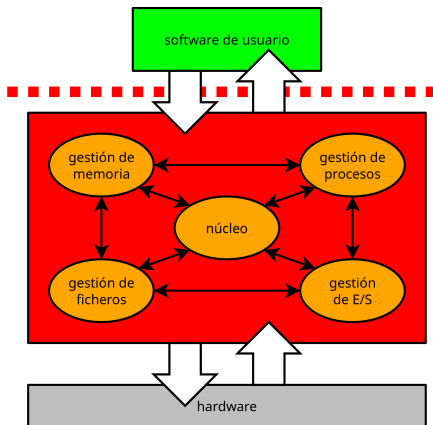
⊙ ¿Cómo subdividir?



Modular

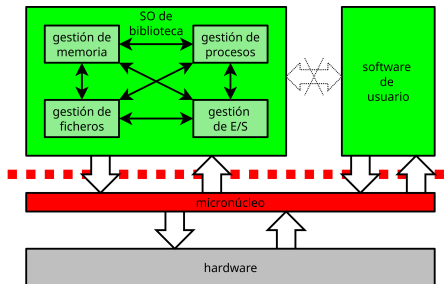
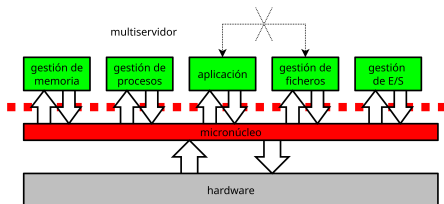
- ⊙ El SO completo se ejecuta en modo protegido.
- ⊙ Escasa protección entre los componentes.
- ⊙ Ventajas:
 - Economía de cambios de contexto ⇒ **+eficiente.**
 - Menor complejidad.
- ⊙ Inconvenientes:
 - Falta de protección ⇒ **-fiabilidad** (controladores).
 - Menos flexible que monolítico.

- ⊙ ¿Qué colocar en el núcleo y qué en módulos?



Micronúcleo

- Una mínima parte del SO se ejecuta en modo protegido.
- Ventajas:
 - Perfecta protección entre componentes $\Rightarrow$ **+fiabilidad.**
 - Manejo de la complejidad.
 - Facilidad de programación.
- Inconvenientes:
 - Sobrecarga en las comunicaciones $\Rightarrow$ **-eficiencia.**

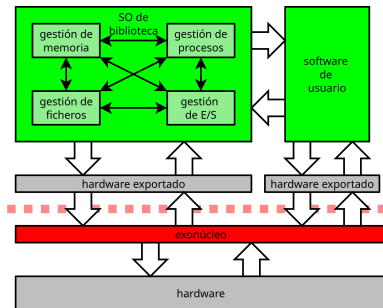


Exonúcleo

- ⊙ Apenas existe el SO, sólo un gestor de recursos.
- ⊙ Dejamos que el software acceda directamente al hardware.
- ⊙ Ventajas:
 - Perfecta protección entre componentes ⇒ **+fiabilidad.**
 - Acceso directo al hardware ⇒ **máxima eficiencia**

- ⊙ Inconvenientes:

- Pobre reutilización de código.

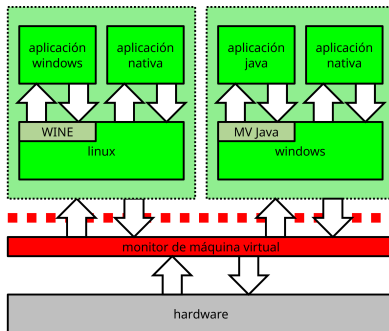


Máquina virtual

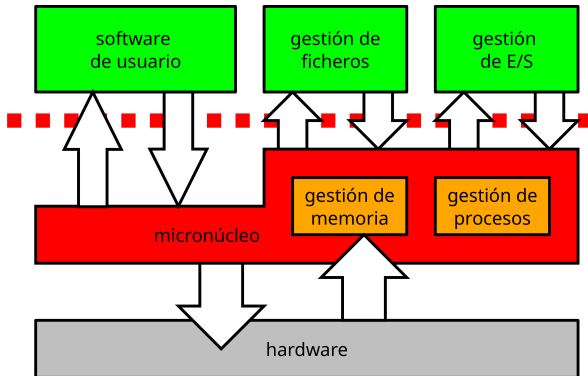
- ⊙ **Copias virtuales** de la máquina real:
 - SW: Bochs, Qemu, VMWare, Xen.
 - HW: VMWare, Xen.
- ⊙ IBM VM/370 (1972).
- ⊙ **Ventajas:**
 - Perfecta protección entre componentes ⇒ **+fiabilidad.**
 - Mejor aprovechamiento del hardware.
 - Máxima reutilización de código.
- ⊙ **Inconvenientes:**

- La simulación del hardware real puede ser costosa ⇒ **poco eficiente**

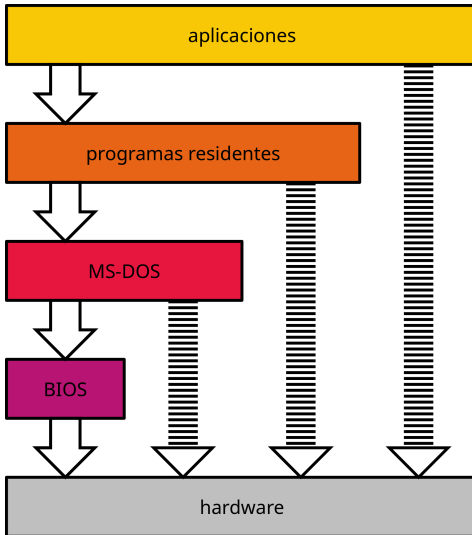
- ⊙ Éxito por mejoras HW.



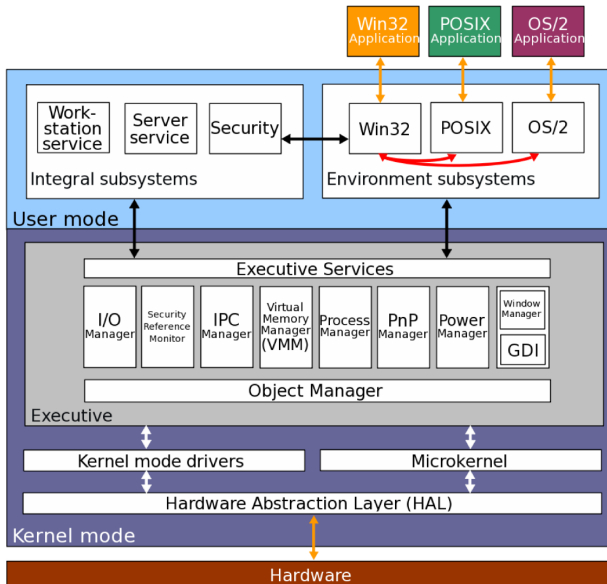
- ⊙ Mezcla más frecuente: **micronúcleo + monolítico**.
- ⊙ Ventaja: $\Rightarrow$ ganamos velocidad respecto a micronúcleo.
- ⊙ Inconveniente: $\Rightarrow$ perdemos protección entre componentes.

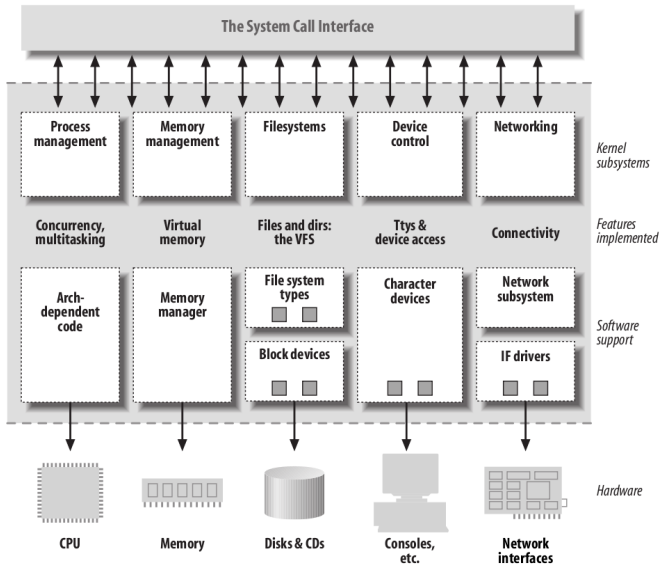


MS-DOS

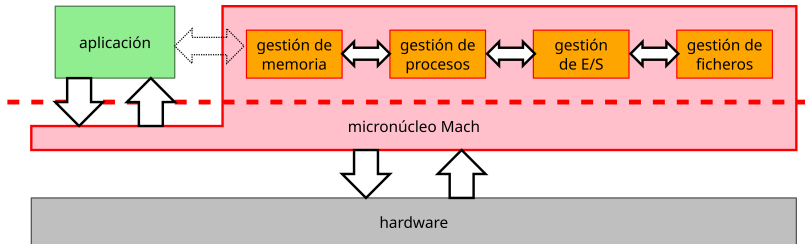


Windows 2000

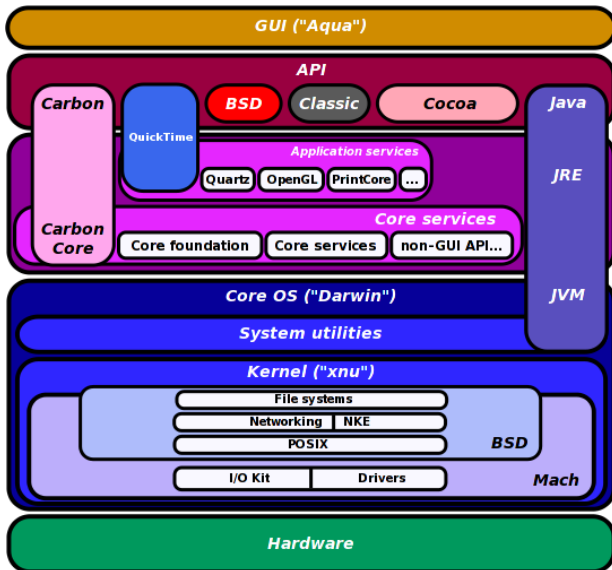


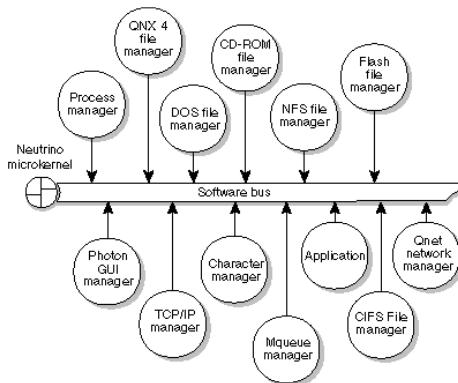
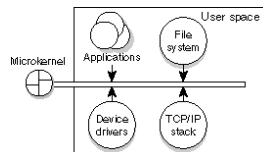
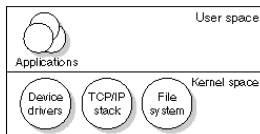
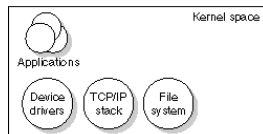


Mach



MacOS X

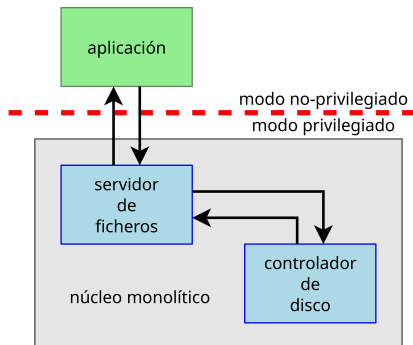




⊙ 1 llamada al sistema:

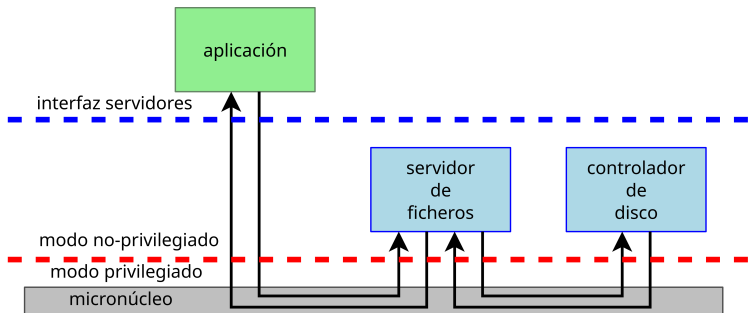
- entrada al núcleo.
- cambio al espacio de direcciones del núcleo.
- recuperar el espacio de direcciones original.
- salida del núcleo.

⊙ 1 llamada a *procedimiento*: llamada y retorno en el interior del espacio de direcciones del núcleo y pudiendo compartir información.

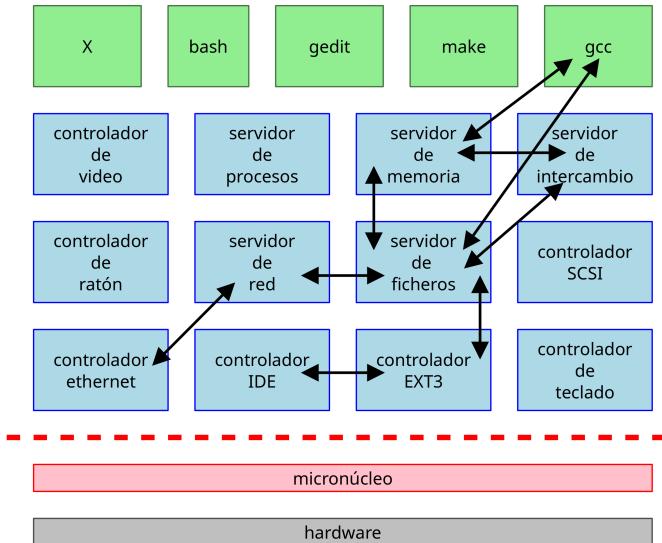


⊙ 4 llamadas al sistema:

- entrada al micronúcleo.
- cambio al espacio de direcciones del micronúcleo.
- transferencia del mensaje.
- recuperar el espacio de direcciones original.
- salida del micronúcleo.

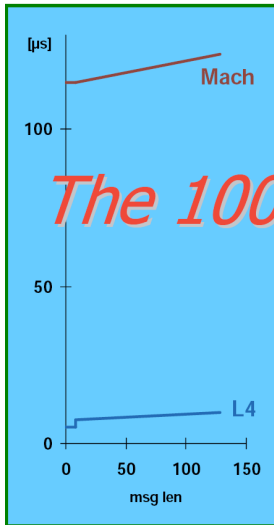


Coste estructural: multiservidor

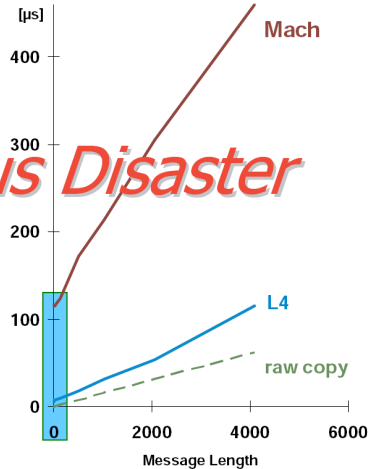


El desastre de los 100 μ s (micronúcleos de 1ª generación)

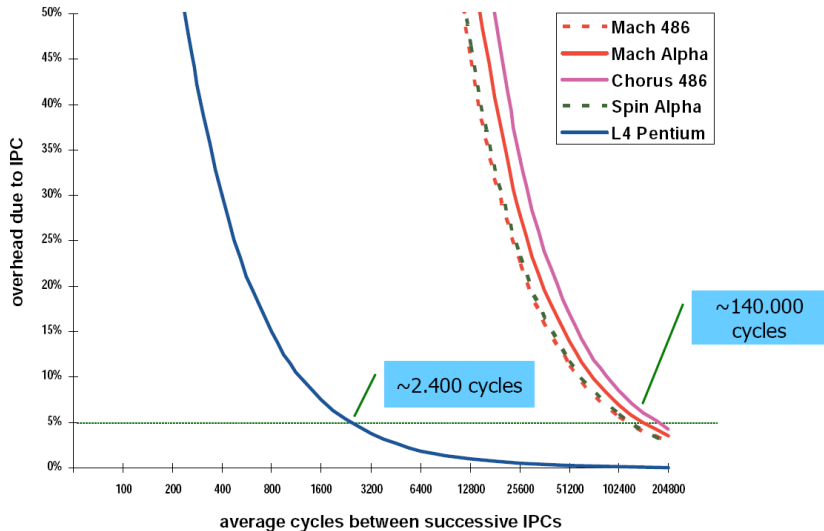
⊙ $SYSCALL \leftrightarrow RPC \approx 2 \times IPC$: $Mach_{IPC}=115\mu s$ $Linux_{IPC}=20\mu s$ $L4_{IPC}=5\mu s$ (486 50MHz)

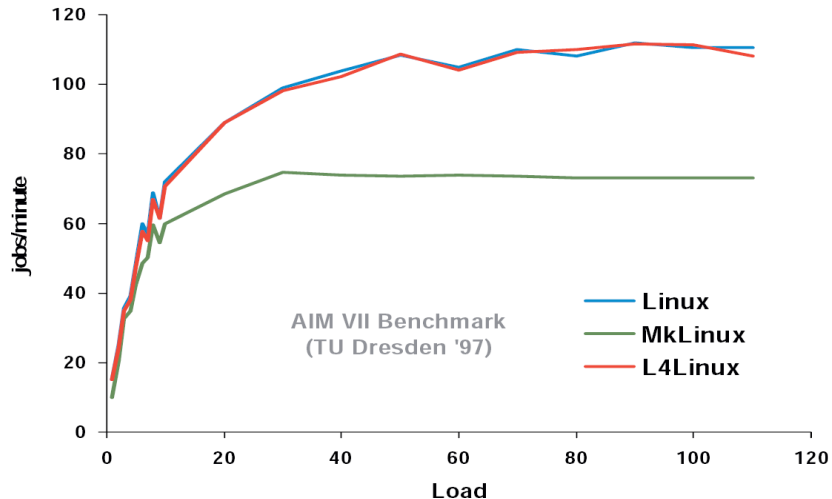


The 100- μs Disaster



Sobrecarga por comunicación entre procesos





Coste estructural: cambio de contexto

- ⊙ Linux 2.4.21: 13200 ciclos/ $5.4\mu s$ en un Pentium 4 a 2.4GHz
- ⊙ L4 (Liedtke, Achieved IPC performance):

	Pentium		R4600		Alpha	
	<i>instructions</i>	<i>cycles</i>	<i>instructions</i>	<i>cycles</i>	<i>instructions</i>	<i>cycles</i>
enter kernel mode (trap)	1	52	23	25	1	5
ipc code	43	23	47	50	60	38
segment register reload	4	16	–	–	–	–
exit kernel mode (ret)	1	20	9	11	1	2
total	50	121	79	86	62	45
	166 MHz: $0.73\ \mu s$		100 MHz: $0.86\ \mu s$		433 MHz: $0.10\ \mu s$	

	Pentium		R4600		Alpha	
	<i>cache lines</i>	<i>L1 cache usage</i>	<i>cache lines</i>	<i>L1 cache usage</i>	<i>cache lines</i>	<i>L1 cache usage</i>
kernel code (I-cache)	6	2.3 %	14	2.7 %	13	5.1 %
global kernel data (D-cache)	2	0.8 %	1	0.2 %	0	0.0 %
thread kernel data (D-cache)	2×2	1.6 %	2×2	0.8 %	2×2	1.6 %
total (I+D-cache)	12	2.3 %	19	1.9 %	17	3.3 %

- ◎ Barrelfish
- ◎ Fuchsia
- ◎ $L4 \Rightarrow \begin{cases} seL4 \\ PikeOS \end{cases}$
- ◎ Singularity $\Rightarrow$ Midori