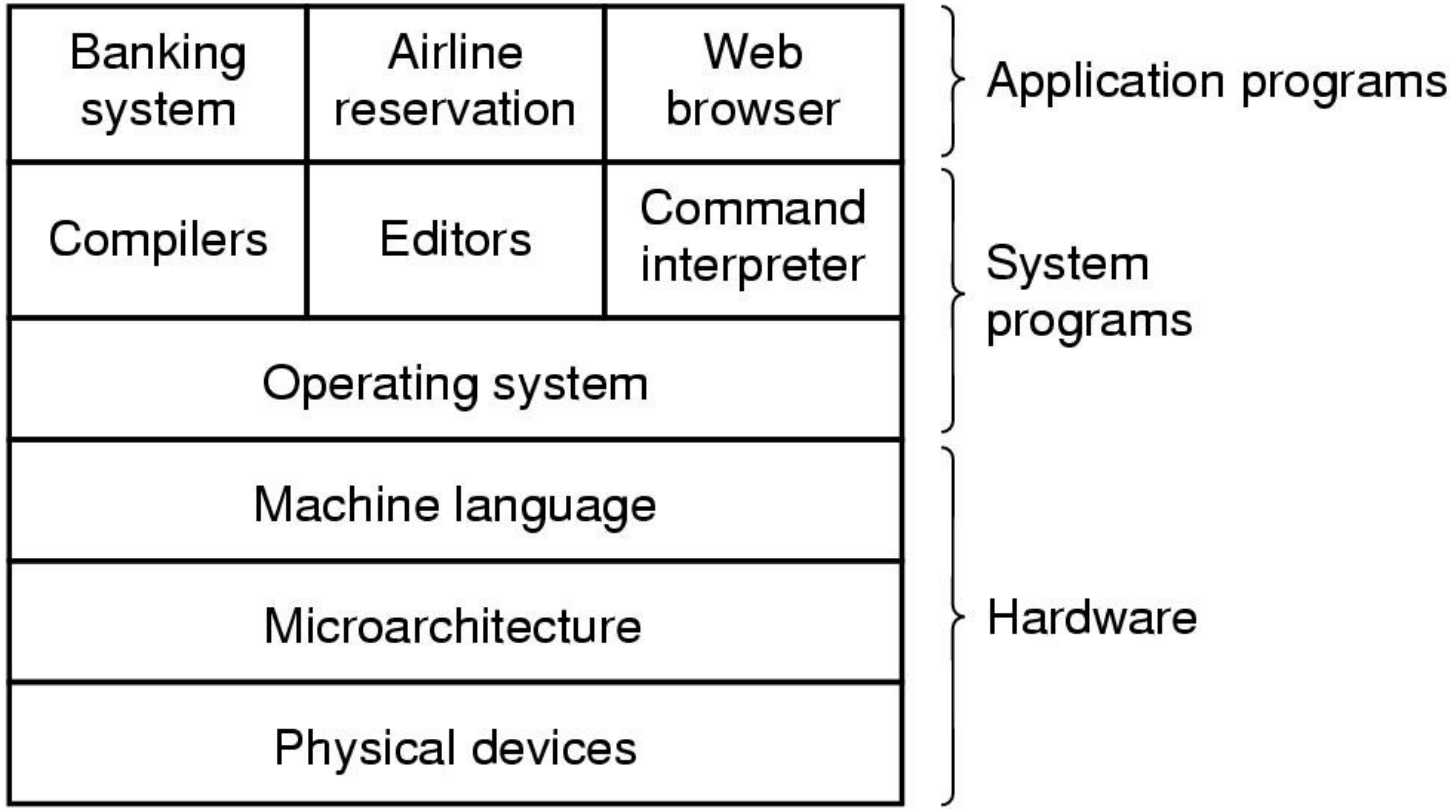
Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative
Istituto sistemi informativi e networking

SUPSI

Ambienti Operativi: Progettazione dei sistemi operativi

Amos Brocco, Ricercatore, DTI / ISIN

Organizzazione di un sistema operativo



Da A. Tanenbaum, "Modern Operating Systems", 2ª Edizione

Progettazione di sistemi operativi

- Requisiti
  - Astrazioni
  - Funzionalità di base
  - Isolazione (es. protezione della memoria)
  - Gestione dell'hardware

Progettazione dei sistemi operativi

- **Astrazioni**
  - processi, thread
  - spazi di indirizzamento (memoria)
  - files
  - segnali
  - ....
- **Operazioni di base**
  - Chiamate di sistema
    - Permettono di interagire e manipolare le astrazioni

Perché è difficile progettare un sistema operativo?

- **I sistemi operativi sono "programmi" grandi e complessi**
  - **Molte linee di codice**: difficile capire tutto
  - **Molte dipendenze**: interazione complessa tra le componenti
  - **Concorrenza**: utenti multipli, più processi in esecuzione, più periferiche da gestire
  - **Sicurezza**: proteggere gli utenti...
    - ... senza impedirgli di gestire, modificare e condividere i dati

Perché è difficile progettare un sistema operativo?

- **I sistemi operativi sono "programmi" grandi e complessi**
  - È necessaria una visione a **lungo termine** (es. Y2K bug)
  - **Bisogna essere "generalisti"**
    - I progettisti non possono prevedere tutto quello che farà l'utente
  - Devono essere **portabili su più sistemi**
  - Spesso devono essere **compatibili con versioni precedenti (backward-compatible)**

- (da http://www.heise-online.co.uk/open/Kernel-Log-Linux-2-6-27-Released--/features/111671/6)

Linee di codice del kernel Linux

Facts and figures about the latest versions of the Linux kernel

Linux version	number of files[1]	lines of source code[2]	development time	number of commits[3]	diffstat[4]
2.6.20	21280	8102486 (7400843)	66 days	4768	5825 files changed, 262475 insertions(+), 136162 deletions(-)
2.6.21	21614	8246470 (7522286)	80 days	5016	6568 files changed, 319232 insertions(+), 175247 deletions(-)
2.6.22	22411	8499363 (7744727)	74 days	6526	7620 files changed, 519591 insertions(+), 266699 deletions(-)
2.6.23	22530	8566554 (7818168)	93 days	6662	7203 files changed, 406268 insertions(+), 339071 deletions(-)
2.6.24	23062	8859629 (8082358)	107 days	9836	10209 files changed, 776107 insertions(+), 483031 deletions(-)
2.6.25	23810	9232484 (8396250)	83 days	12243	9738 files changed, 777371 insertions(+), 404514 deletions(-)
2.6.26	24270	9411724 (8535933)	88 days	9941	8676 files changed, 595393 insertions(+), 416143 deletions(-)
2.6.27	24354	9709868 (8690888)	88 days	10628	15127 files changed, 1131171 insertions(+), 912939 deletions(-)

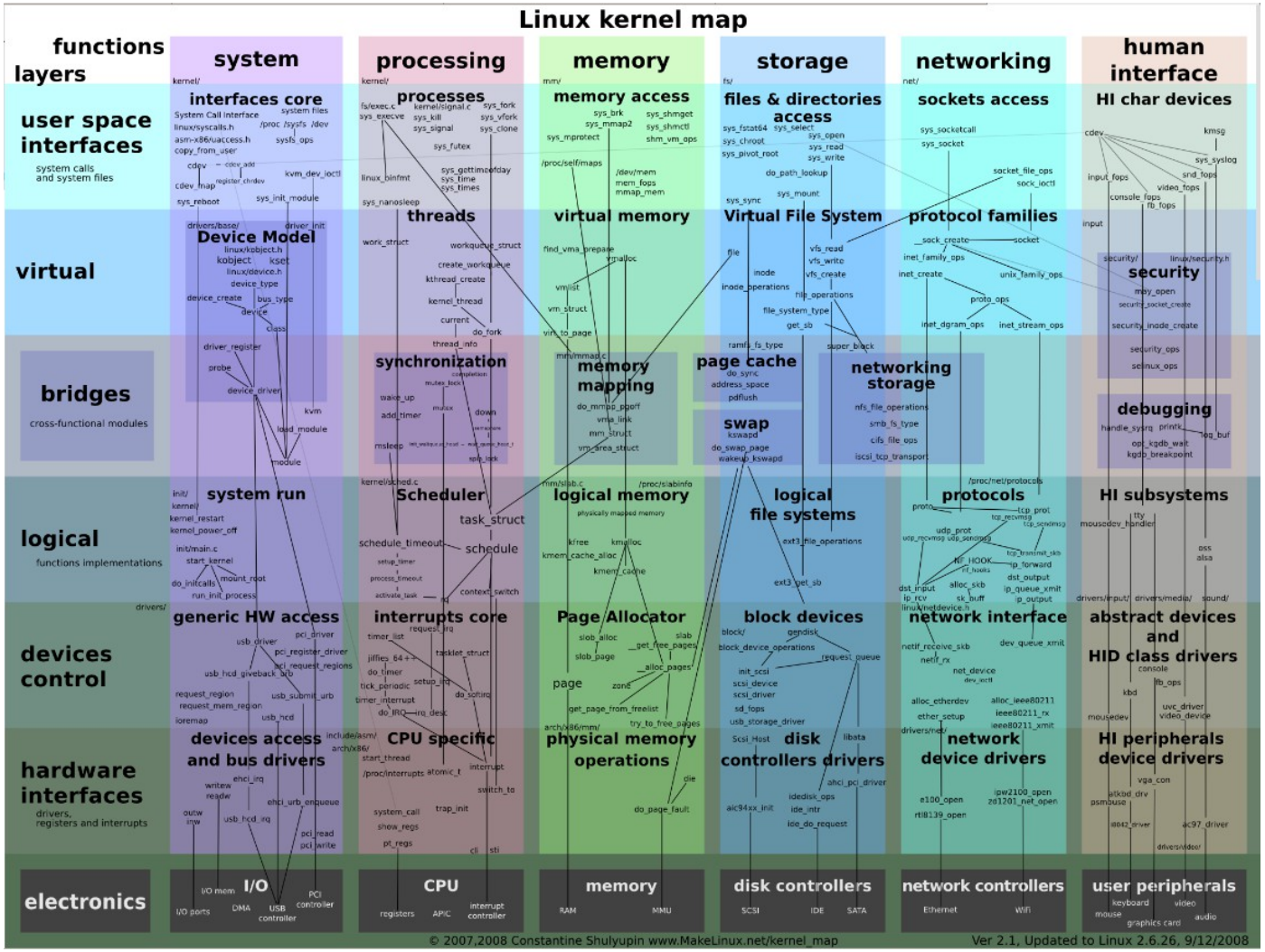
[1] find . -type f -not -regex '\./\.git/.*' | wc -l

[2] find . -type f -not -regex '\./\.git.*' | xargs cat | wc -l (find . -name *.[hcS] -not -regex '\./\.git.*' | xargs cat | wc -l)

[3] git-log --no-merges --pretty=oneline v2.6.*(x-1)*..v2.6.*(x)* | wc -l

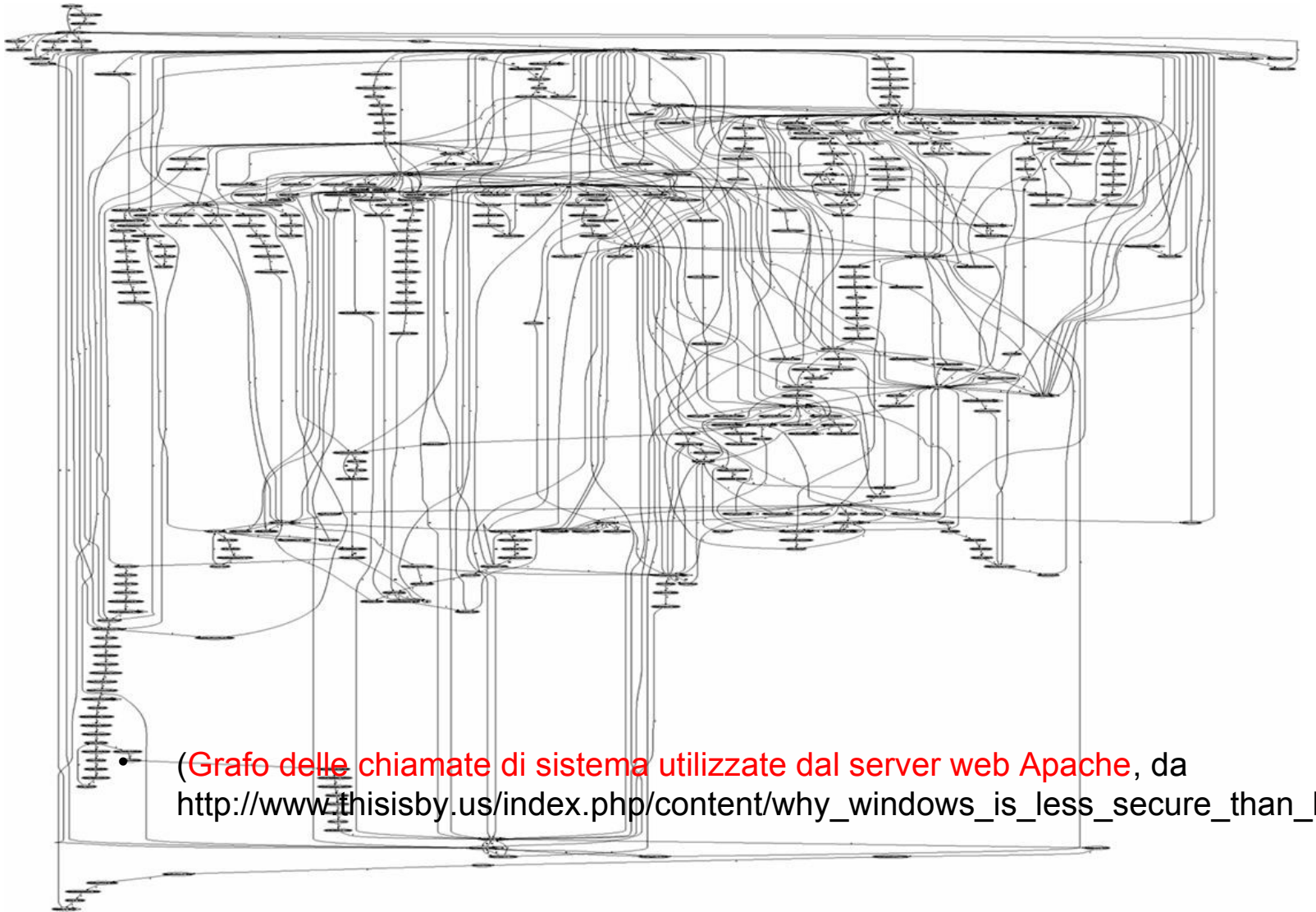
[4] ;git diff --shortstat v2.6.*(x-1)*..v2.6.*(x)*

Complessità del kernel Linux



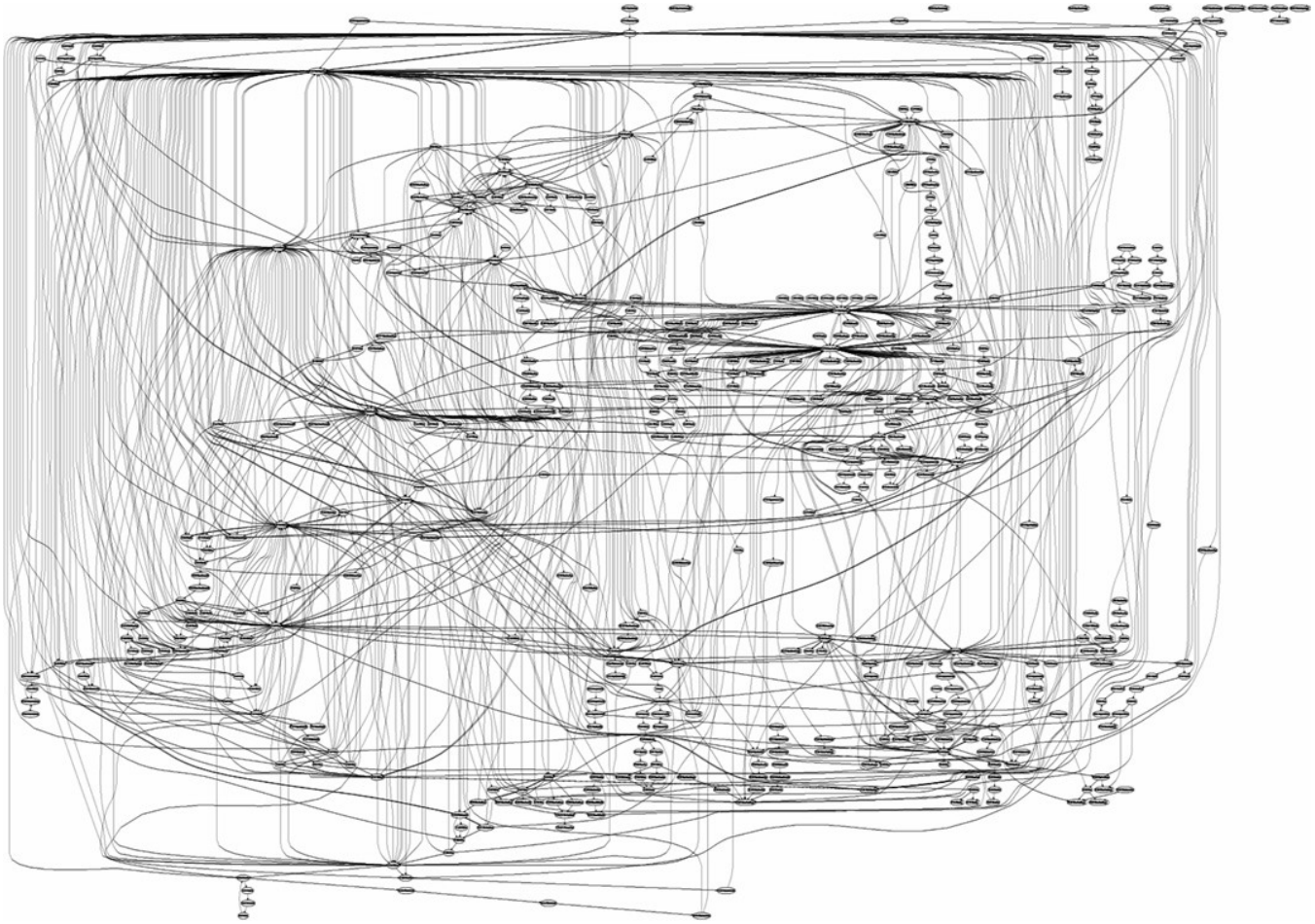
- (da http://www.makelinux.net/kernel_map)

Una giungla di dipendenze: Linux



(Grafo delle chiamate di sistema utilizzate dal server web Apache, da http://www.thisisby.us/index.php/content/why_windows_is_less_secure_than_linux)

Una giungla di dipendenze: Windows



- (Grafo delle chiamate di sistema di MS IS su Windows da http://www.thisisby.us/index.php/content/why_windows_is_less_secure_than_linux)

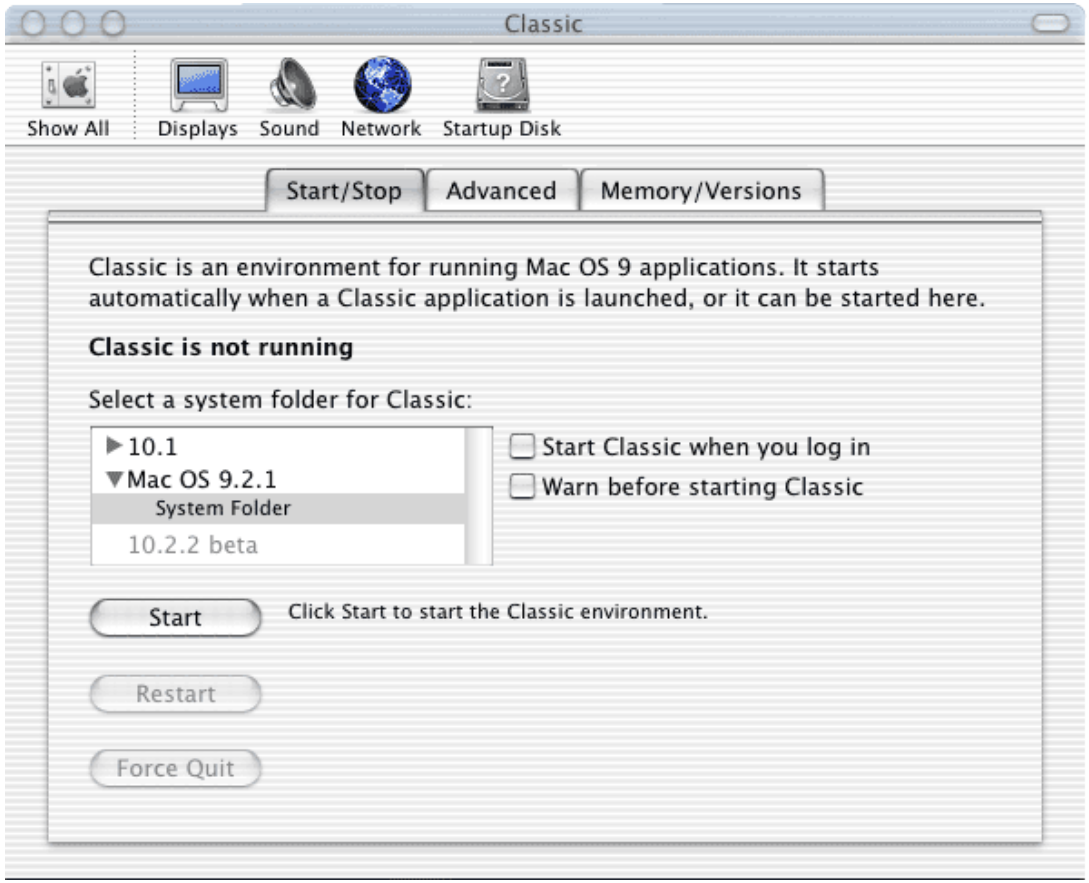
Portabilità

Supported CPU architectures [edit]

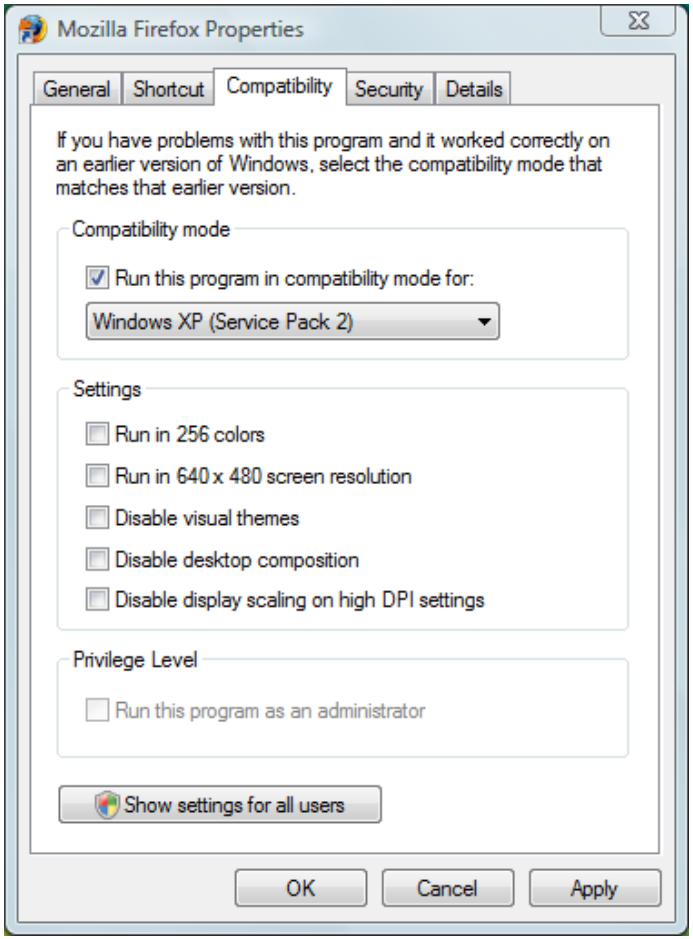
kernel	68k	DEC Alpha	ARM	HP PA-RISC	IA-64	MIPS	PowerPC	PowerPC 970	System/ 390	SuperH	SPARC 32-bit	SPARC 64-bit	x86	x86-64
Linux kernel	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
DragonFly BSD kernel	No	No	No	No	No	No	No	No	No	No	No	No	Yes	No
FreeBSD kernel	No	Yes	Yes	No	Yes	No	Yes	No	No	No	No	Yes	Yes	Yes
NetBSD kernel	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes
NetWare kernel	No	No	No	No	No	No	No	No	No	No	No	No	Yes	No
OpenBSD kernel	Yes	Yes	Yes	Yes	No	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes
Solaris kernel	No	No	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes
Windows NT kernel	No	NT 5.0 RC1 and below only	No	No	Yes	NT 4.0 and below only	NT 3.51 and NT 4.0 only	No	No	No	No	No	Yes	Yes
XNU	No	No	Yes	No	No	No	Yes	Yes[13]	No	No	No	No	Yes	Yes[13]
SPARTAN kernel	No	No	Yes	No	Yes	Yes	Yes	In Progress	No	No	No	Yes	Yes	Yes

- (da http://en.wikipedia.org/wiki/Comparison_of_kernels)

Compatibilità con le versioni precedenti



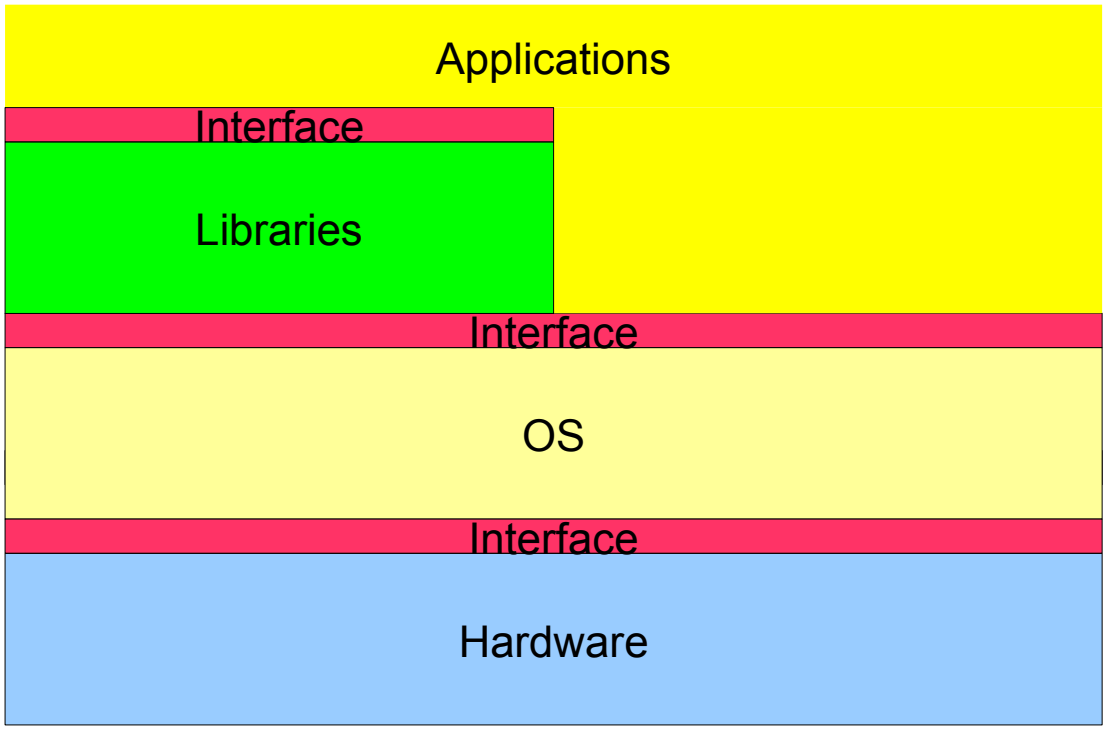
OSX Classic Mode



Windows compatibility options

Accedere alle astrazioni

- ...ogni astrazione ha un'interfaccia che ne permette l'accesso



Principi per la progettazione di un'interfaccia

- **Semplicità**
  - Facile da capire
  - Facile da implementare (senza bug)
    - *KISS (Keep It Simple, Stupid)*

Principi per la progettazione di un'interfaccia

- **Completezza**
  - Deve essere possibile fare tutto quello che è necessario all'utente (**né più, né meno**)
  - Ogni funzionalità ha un prezzo (linee di codice, complessità, ecc)
    - è veramente necessaria?
    - per ogni nuova funzionalità applicare il principio "Fai una cosa sola, ma falla bene"

Principi per la progettazione di un'interfaccia

- **Efficienza**
  - Una chiamata di sistema che richiede 1 minuto per completarsi non verrà mai usata
  - Le procedure che sono intuitivamente veloci devono esserlo

Paradigmi

- Paradigmi:
  - **Paradigmi di architettura**
    - L'utente come vedrà il sistema: bisogna garantire una certa coerenza di architettura: come ogni funzionalità si collega alle altre
  - **Paradigmi dell'interfaccia utente**
    - Come l'utente interagirà con il sistema
  - **Paradigmi di esecuzione**
    - Come il sistema si comporterà e eseguirà i propri compiti
  - **Paradigmi di gestione dei dati**
    - Come il sistema presenterà le strutture dati (es. “Tutto è un file” in UNIX)

Esempio: paradigmi di esecuzione

- Paradigma algoritmico (*hardwired*)

```
main( )
{
        int ... ;

        init( );
        do_something( );
        read(...);
        do_something_else( );
        write(...);
        keep_going( );
        exit(0);
}
```

Esempio: paradigmi di esecuzione

- Paradigma a eventi

```
main( )
{
        mess_t msg;

        init( );
        while (get_message(&msg)) {
                switch (msg.type) {
                        case 1: ... ;
                        case 2: ... ;
                        case 3: ... ;
                }
        }
}
```

Ortogonalità

- Bisogna permettere di combinare concetti separati in modo indipendente
  - Esempio: int, floats, struct → struct { int; float; }