Leggete il diario!

http://matteo.vaccari.name/so/diario.php

Dopo ogni lezione trovate:
- gli argomenti
- le sezioni del testo corrispondenti
- i lucidi
- documentazione addizionale
- esercizi

Tutte le letture riportate nel diario **sono parte del programma di esame**, a meno che non siano esplicitamente dichiarate *facoltative*

Deadlock

Processi: accesso esclusivo alle risorse

Due tipi di risorse:
- prelazionabili (preemptable)
  il processo può sopportare la sottrazione della risorsa
- non prelazionabili
  il processo cade se la risorsa è sottratta

Risorse

Ciascun processo:
0. richiede la risorsa
1. la usa
2. la rilascia

Se la risorsa non è disponibile, il processo
- resta bloccato, oppure
- riceve un codice di errore

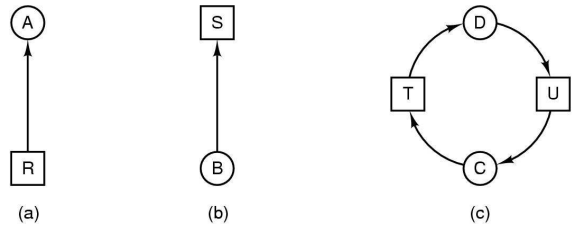
Definizione di deadlock

A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause

Condizioni necessarie per un deadlock

1. Mutua esclusione
   risorsa assegnata a *un* processo oppure disponibile
2. Hold and wait
   un processo ha una risorsa e ne può chiedere altre
3. Senza prelazione
   le risorse possono essere restituite solo volontariamente
4. Attesa circolare
   ci deve essere una catena di processi

Modellazione dei deadlock con grafi

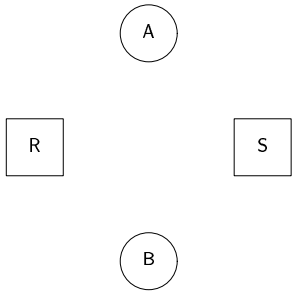


(a) il processo *A* ha assegnata la risorsa *R*

(b) il processo *B* chiede la risorsa *S*

(c) deadlock

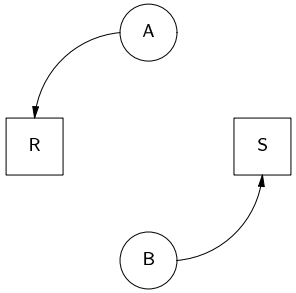
Come avviene un deadlock (i)



Processo A	**Processo B**
richiedi R	richiedi S
richiedi S	richiedi R
usa R e S	usa R e S
rilascia S	rilascia S
rilascia R	rilascia R

Come avviene un deadlock (ii)

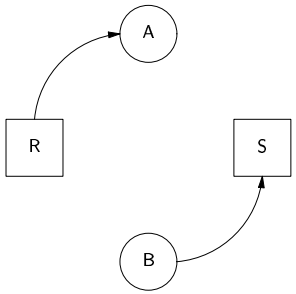
Processo A	**Processo B**
• richiedi R	• richiedi S
richiedi S	richiedi R
usa R e S	usa R e S
rilascia S	rilascia S
rilascia R	rilascia R



Il sistema operativo deve decidere quale risorsa assegnare per prima

Come avviene un deadlock (iii)

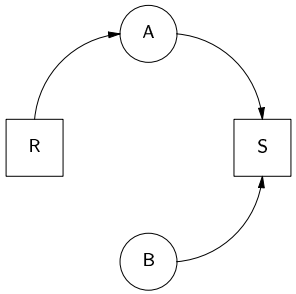
Processo A	**Processo B**
• richiedi R	• richiedi S
richiedi S	richiedi R
usa R e S	usa R e S
rilascia S	rilascia S
rilascia R	rilascia R



Il sistema operativo ha accordato la richiesta di *A*

Come avviene un deadlock (iv)

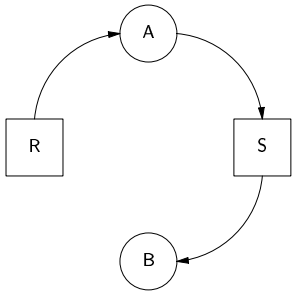
Processo A	**Processo B**
richiedi R	• richiedi S
• richiedi S	richiedi R
usa R e S	usa R e S
rilascia S	rilascia S
rilascia R	rilascia R



A prosegue e richiede la risorsa *S*

Come avviene un deadlock (v)

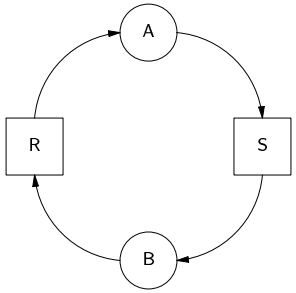
Processo A	**Processo B**
richiedi R	• richiedi S
• richiedi S	richiedi R
usa R e S	usa R e S
rilascia S	rilascia S
rilascia R	rilascia R



Il sistema operativo ha accordato la richiesta di *B*
brutta mossa!

Come avviene un deadlock (vi)

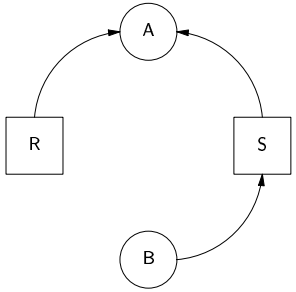
Processo A	**Processo B**
richiedi R	richiedi S
• richiedi S	• richiedi R
usa R e S	usa R e S
rilascia S	rilascia S
rilascia R	rilascia R



L'unico esito possibile ora è il deadlock

Potevamo evitarlo al passo v

Processo A	**Processo B**
richiedi R	• richiedi S
richiedi S	richiedi R
• usa R e S	usa R e S
rilascia S	rilascia S
rilascia R	rilascia R



Il sistema operativo accorda l'altra richiesta di *A*

0. L'algoritmo dello struzzo

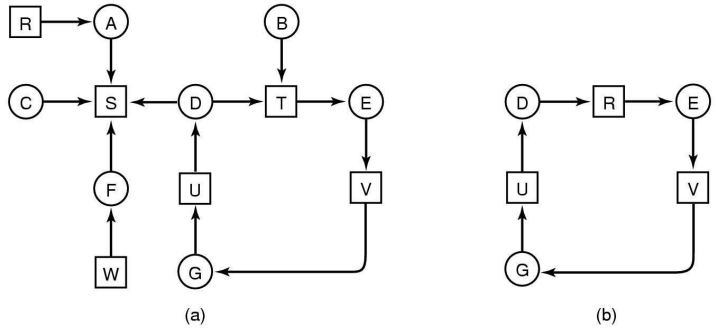
Facciamo finta che non ci sia problema

Ragionevole se:
- il deadlock è molto improbabile
- il costo della prevenzione è molto alto

Strategie per gestire i deadlock

1. ignorare il problema (!)
2. detection and recovery
3. algoritmi di allocazione risorse "prudenti"
4. prevenzione (neghiamo una delle 4 condizioni necessarie)

1. Detection and recovery: detection



Tenere traccia di allocazioni e richieste

Trovare i deadlock è un semplice problema di teoria dei grafi

1. Detection and recovery: recovery

Abbiamo trovato un deadlock; e ora? Tre possibilità

Preemption
: togliamo la risorsa; il processo riceve un codice di errore

Rollback
: lo stato del processo viene fatto "retrocedere" a prima della acquisizione

Termination
: il processo viene ucciso

2. Algoritmi specializzati

Algoritmi per allocazione di risorse

Non cedere mai una risorsa se questo comporta la possibilità di un deadlock

Questi algoritmi dipendono da assunzioni sul comportamento futuro dei processi

3. Prevenzione

Neghiamo una delle 4 condizioni necessarie

0. Mutua esclusione
: risorsa assegnata a *un* processo oppure disponibile

Alcune risorse possono essere rese condivisibili

Es. spooling; code di eventi

Non tutte le risorse si prestano a questa soluzione

3. Prevenzione

neghiamo
1. Hold and wait
: un processo ha una risorsa e ne può chiedere altre, una per volta

tutte le risorse sono acquisite alla partenza di un processo (es. sistemi batch)
- inefficiente
- ma efficace

3. Prevenzione

neghiamo
2. Senza prelazione
   le risorse possono essere restituite solo volontariamente

- poco pratico

3. Prevenzione

neghiamo
3. Attesa circolare
   ci deve essere una catena di processi

idea: imponiamo un ordine sulle risorse

i processi devono richiedere le risorse nell'ordine

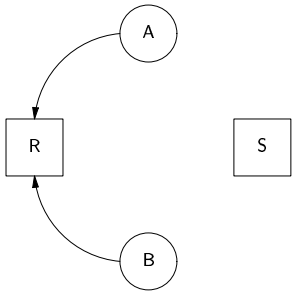
impossibile creare un circolo

es. il kernel di Linux

Le richieste vanno fatte in ordine; supponiamo $R < S$

Processo A
- richiedi R
  richiedi S
  usa R e S
  rilascia S
  rilascia R

Processo B
- richiedi R
  richiedi S
  usa R e S
  rilascia S
  rilascia R

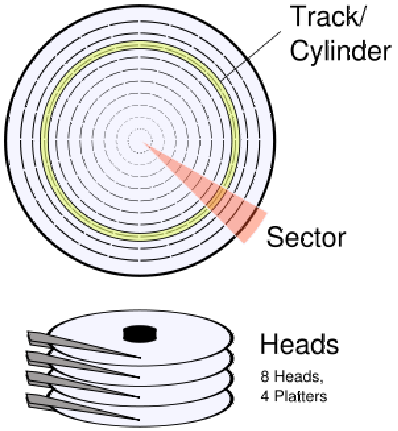


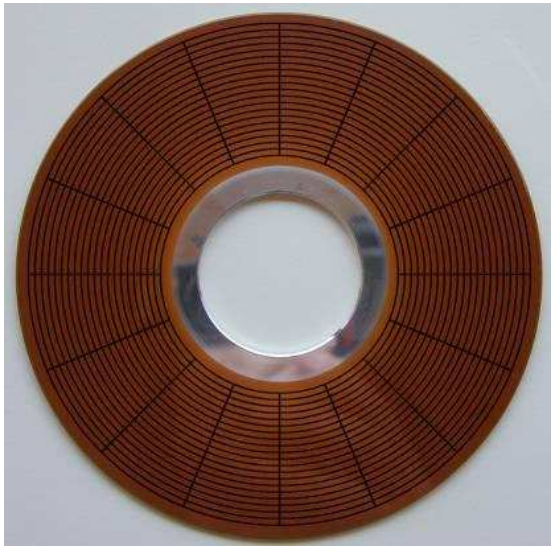
A e B competono per R; solo uno dei due potrà proseguire

Dischi magnetici

La geometria:
- heads
- cylinders
- sectors

Ogni settore può essere identificato dalla terna (head, cylinder, sector)

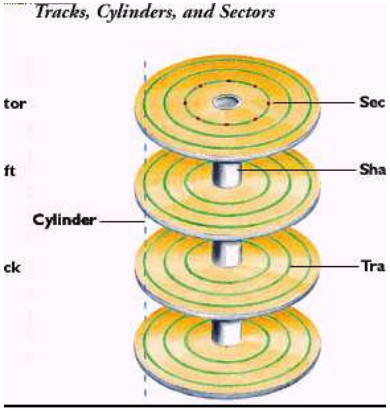




La differenza fra traccia e cilindro

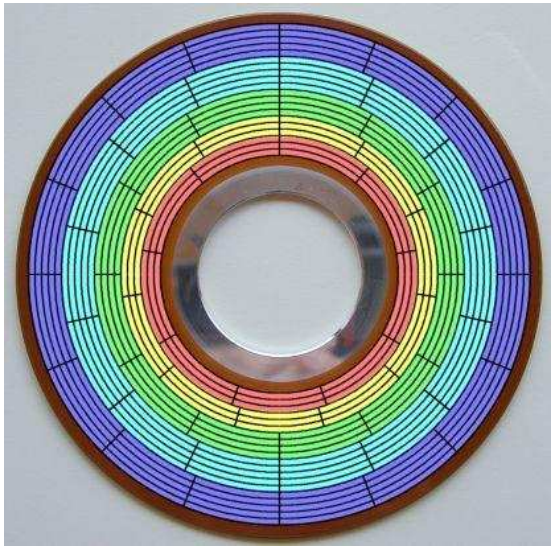
L'immagine mostra quattro piatti, con disegnate tre tracce

Il cilindro indicato è composto da otto tracce (due per superficie)



Zoned Bit Recording

20 tracce divise in 5 zone



Il BIOS non "capisce" i dischi formattati con ZBR

Questi dischi presentano al BIOS una geometria virtuale (es. 63 settori per traccia)

Il controller del disco esegue una traduzione da virtuale a fisico

Limite del BIOS:
- 16 bit per specificare il # di settori
- 4 bit per specificare il # di testine
- 6 bit per specificare il # di tracce $\Rightarrow$ max # settori = $(2^{16}-1)*2^4*(2^6-1) = 66,059,280 \Rightarrow$ max capacità = 66,059,280 * 512 = 31.499 GB

Per superare questo limite si usa il *Logical Block Addressing*: i settori sono numerati linearmente

La formattazione

0. Formattazione di basso livello
- marca la struttura fisica (tracce, settori) sulle superfici
- eseguita dal fabbricante
- richiede di conoscere con precisione la geometria fisica del disco

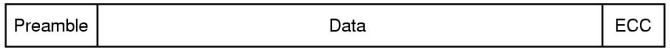
1. Partizionamento
- divide il disco in "pezzi" logici *basati su una convenzione*
- è una funzione del sistema operativo
- tutti i SO su PC rispettano la convenzione del PC IBM

2. Formattazione di alto livello
- è una funzione del sistema operativo
- (es. comando "format" di MS-DOS, comando mkfs(8) di Unix)
- costruisce un *filesystem* in una partizione

Formattazione di basso livello

- marca la struttura fisica (tracce, settori) sulle superfici

Un settore:



- eventuali blocchi difettosi vengono "rimpiazzati" con blocchi di scorta

- il rimpiazzo è fatto in maniera trasparente dal controller

Tempo necessario per leggere o scrivere un blocco

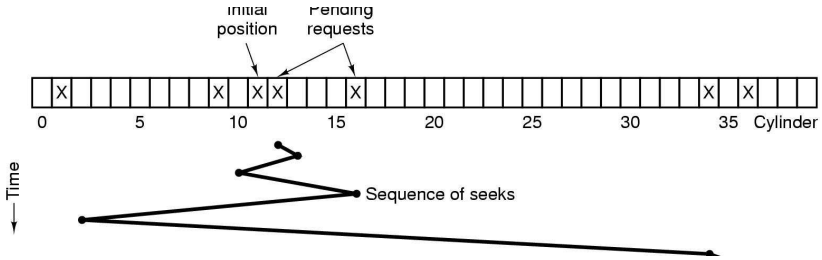
0. seek time
1. rotational delay
2. data transfer time

Il tempo di seek domina gli altri due

Se le richieste di I/O sono servite in modo First Come, First Served (FCFS) abbiamo pessima performance

Algoritmi per ottimizzare il tempo di seek (i)

Shortest Seek First (SSF)

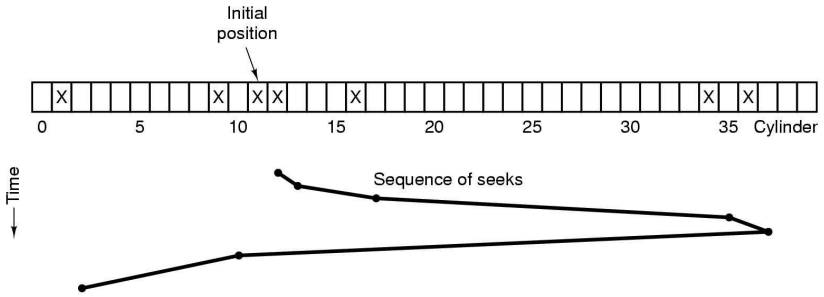


Mediamente è il doppio più veloce di FCFS

Ma ha un problema di *fairness*

Algoritmi per ottimizzare il tempo di seek (ii)

Algoritmo dell'ascensore



Ma se il disco usa una geometria virtuale...

... diversa dalla geometria fisica?

Il SO non può sapere se il cilindro 39 è più vicino al cilindro 40 o al cilindro 200

Comunque questi algoritmi vengono usati dal controller

Il sistema operativo può comunque *ordinare* le richieste di I/O

I/O in unix: buffer cache

Quando il SO riceve una richiesta di lettura, prima va a vedere se il blocco è presente nella cache

Quando il blocco viene letto, viene conservato nella cache

Quando il SO riceve una richiesta di scrittura, la modifica viene fatta sulla cache

A intervalli regolari, i blocchi *sporchi* vengono copiati dalla cache al disco

Clocks

Ogni PC ha due orologi:

Real Time Clock (RTC) (o "CMOS Clock")
- mantiene la data e l'ora quando il PC è spento
- non viene usato quando il PC è acceso

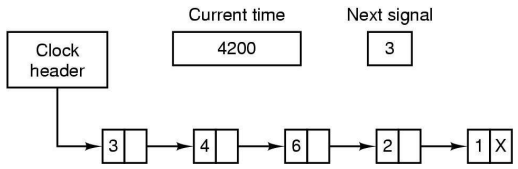
System clock (o "kernel clock" o "software clock")
- lancia interrupt a intervalli regolari
- la routine di interrupt viene usata per vari scopi

Clock interrupt handler

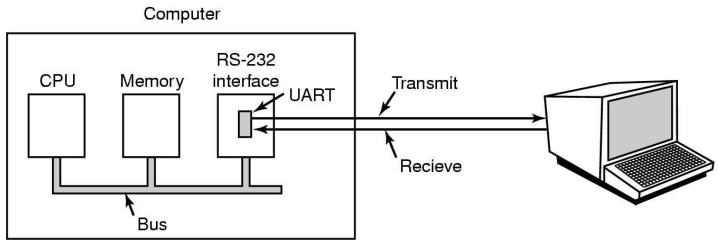
- mantiene l'ora corrente (sotto forma di "numero di secondi passati da 1/1/1970 0:00" ovvero "Unix timestamp")
- implementa lo scheduling con prelazione
- mantiene i contatori di CPU Usage dei processi
- gestisce le syscall alarm(2), sleep(2)
- gestisce vari timer interni al kernel
- mantiene statistiche, profilazione, ecc.

In Linux:
- incrementa la var globale `jiffies`
- incrementa la var globale `xtime`

Implementare i soft timer



Character terminals



TTY input: crudo o cotto?

Il driver della tastiera converte i codici dei tasti in codici ASCII

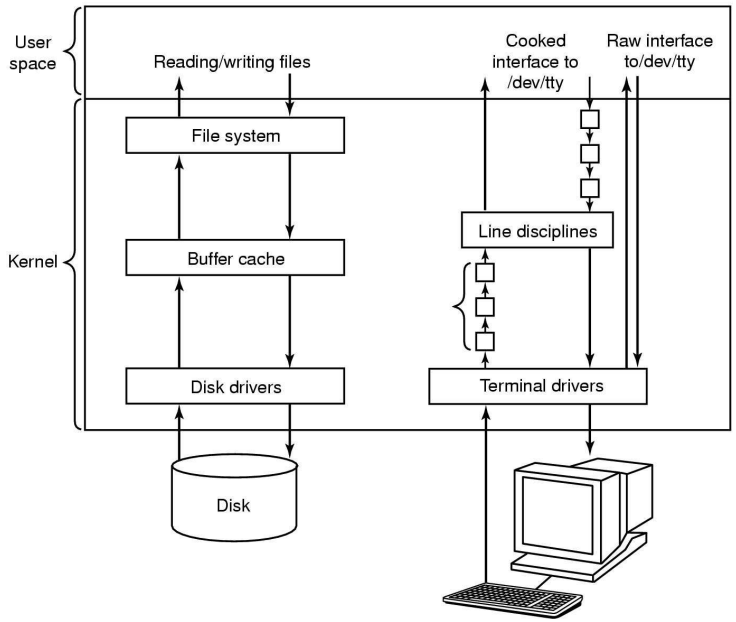
Chi gestisce il tasto per cancellare?

- raw mode: gestito dall'applicazione
  es. emacs, vi
  l'applicazione riceve i caratteri appena sono digitati
- cooked mode: gestito dal driver
  modo di default
  l'applicazione riceve i caratteri una linea per volta

Caratteri di controllo

Character	POSIX name	Comment
CTRL-H	ERASE	Backspace one character
CTRL-U	KILL	Erase entire line being typed
CTRL-V	LNEXT	Interpret next character literally
CTRL-S	STOP	Stop output
CTRL-Q	START	Start output
DEL	INTR	Interrupt process (SIGINT)
CTRL-\	QUIT	Force core dump (SIGQUIT)
CTRL-D	EOF	End of file
CTRL-M	CR	Carriage return (unchangeable)
CTRL-J	NL	Linefeed (unchangeable)

I/O in Unix



TTY Output

Esistono centinaia di terminali diversi (veri e virtuali)

Ciascun terminale
- ha diverse *capacità*
- usa *sequenze di escape* diverse

Problema: realizzare applicazioni *portabili*

Soluzione: il database *terminfo*

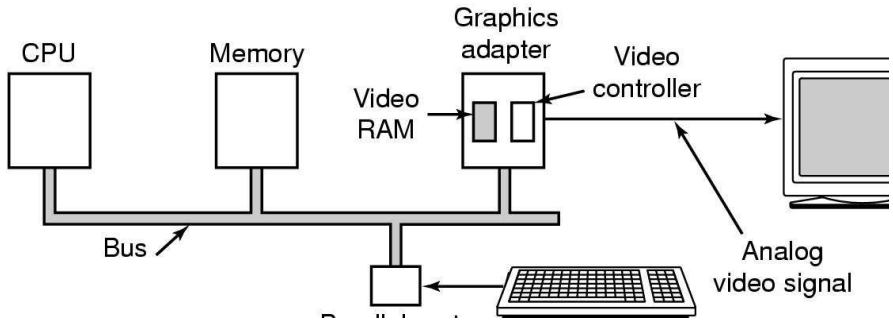
Un frammento dal terminfo master database

```
dumb|80-column dumb tty,
        am,
        cols#80,
        bel=^G, cr=^M, cud1=^J, ind=^J,
unknown|unknown terminal type,
        gn, use=dumb,
lpr|printer|line printer,
        hc, os, OTbs,
        cols#132, lines#66,
        bel=^G, cr=^M, cub1=^H, cud1=^J, ff=^L, ind=^J,
glasstty|classic glass tty interpreting ASCII control characters
        OTbs, am,
        cols#80,
        bel=^G, clear=^L, cr=^M, cub1=^H, cud1=^J, ht=^I,
        .kbs=^H, kcub1=^H, kcud1=^J, nel=^M^J,
vanilla,
        OTbs, bel=^G, cr=^M, cud1=^J, ind=^J,
```

Le sequenze di escape ANSI

Escape sequence	Meaning
ESC [*n*A	Move up *n* lines
ESC [*n*B	Move down *n* lines
ESC [*n*C	Move right *n* spaces
ESC [*n*D	Move left *n* spaces
ESC [*m*;*n*H	Move cursor to (*m*,*n*)
ESC [*s*J	Clear screen from cursor (0 to end, 1 from start, 2 all)
ESC [*s*K	Clear line from cursor (0 to end, 1 from start, 2 all)
ESC [*n*L	Insert *n* lines at cursor
ESC [*n*M	Delete *n* lines at cursor
ESC [*n*P	Delete *n* chars at cursor
ESC [*n*@	Insert *n* chars at cursor
ESC [*n*m	Enable rendition *n* (0=normal, 4=bold, 5=blinking, 7=reverse
ESC M	Scroll the screen backward if the cursor is on the top line

Memory-mapped displays



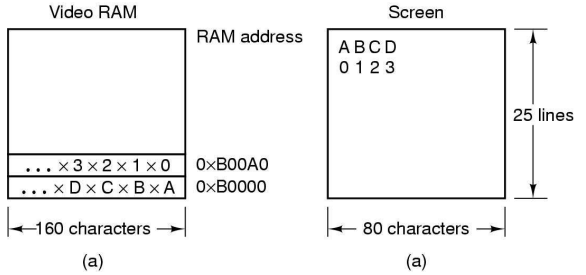
I due modelli di output grafico

raster: basato su una matrice di punti (bitmap)
- es. immagini gif, jpeg

vector: basato su una lista di oggetti (segmenti, punti, spline)
- es. immagini Flash, Scalable Vector Graphics

Video controllers

- text mode
- graphics mode



Windowing software

I programmi basati su GUI sono centrati su un *event loop*

```
while (1) {
   get event
   if event is mouse button down then ...
   if event is mouse button up then ...
   if event is key down then ...
   if event is key up then ...
   ...
}
```

Gli eventi possono essere *preprocessati*

I sistemi di programmazione più "facili" nascondono l'event loop (es. Visual Basic, TCL/Tk)

Esempio: Win32

L'elemento fondamentale è la *finestra*

La finestra ha una coda di eventi

Alla finestra viene associata una *WindowProc*

```
long MyWndProc(HWND hwnd, UINT message, UINT wParam,
  long lParam)
{
  switch (message) {
    case WM_PAINT: ...    /* disegna la finestra */
    case WM_DESTROY: ...  /* la finestra viene chiusa */
    ...
  }
  return DefWindowProc(hWnd, message, wParam, lParam);
}
```

Un'applicazione Windows inizia con WinMain

```
int WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
  LPSTR lpCmdLine, int nCmdShow)
{
    HWND hwnd;

    hwnd = CreateWindowEx(
        // parameters
        );

    ShowWindow(hwnd, SW_SHOW);
    UpdateWindow(hwnd);

    while (GetMessage(&msg, NULL, 0, 0)) {
      DispatchMessage(&msg);
    }
    return msg.wParam;
}
```

Disegnare in una finestra

Il SO manda l'evento WM_PAINT quando una parte della finestra deve essere ridisegnata

```
switch (message)
{
    case WM_PAINT:
        hdc = BeginPaint(hwnd, &ps);
        TextOut(hdc, 0, 0, Hello, Windows!, 15);
        EndPaint(hwnd, &ps);
        return 0L;
    ...
```

Rappresentare testo

I font possono essere rappresentati come bitmap

Ma i migliori risultati si ottengono con tecnologia vettoriale

Es. TrueType

20 pt: abcdefgh

53 pt: abcdefgh

81 pt: abcdefgh

The X Windows System

works in user mode

provides mechanism, not policy

- policy is left to Window Managers
- policy is left to widget libraries

works transparently over a network

The X Windows System