

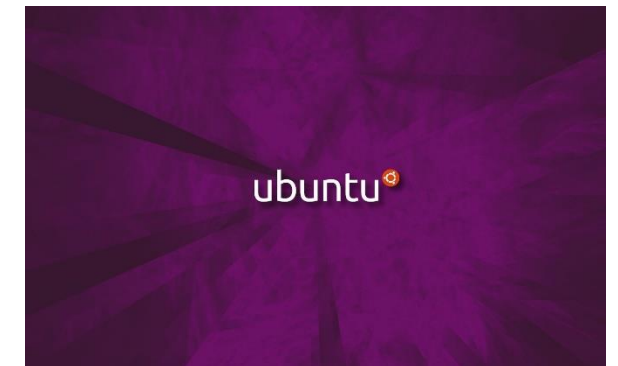
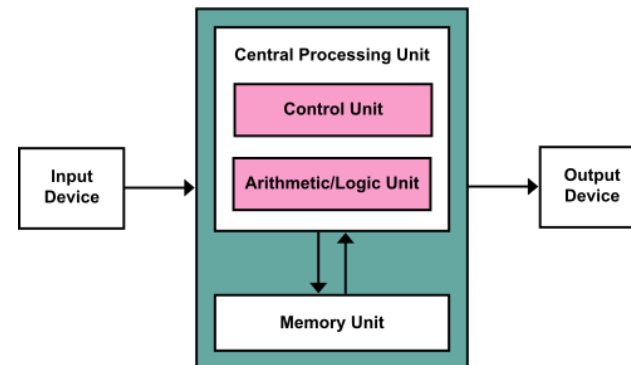
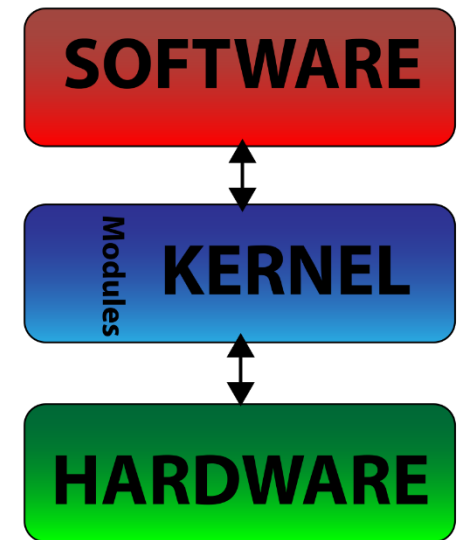
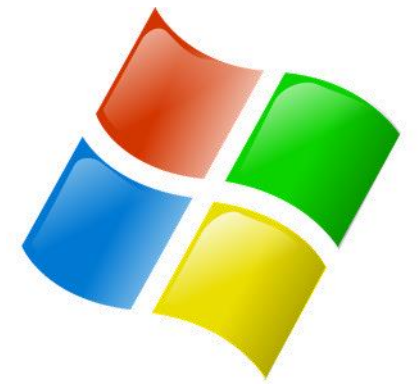


**UNIVERSITÀ DEGLI STUDI  
DELLA BASILICATA**

## *Corso di Sistemi Operativi*

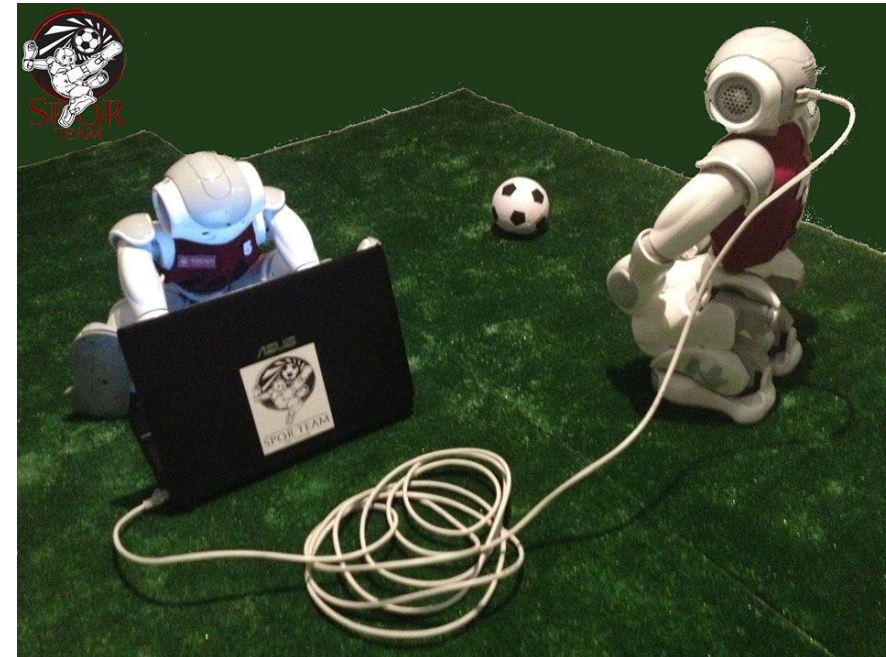
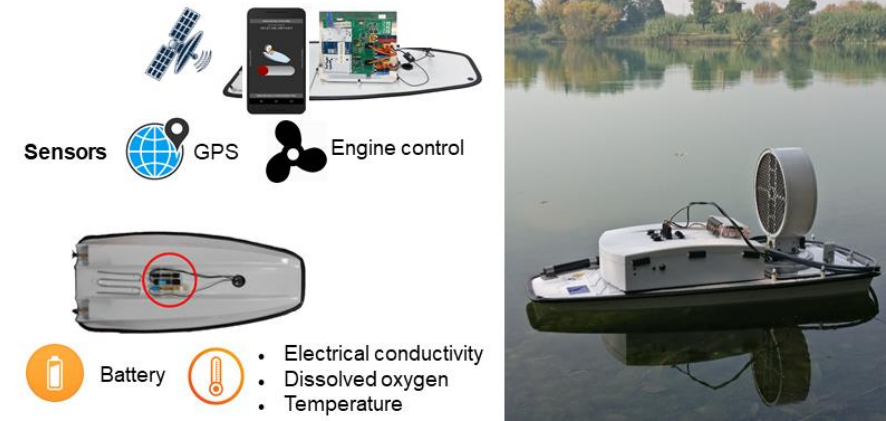
# File System

Docente:  
**Domenico Daniele  
Bloisi**



# Domenico Daniele Bloisi

- Ricercatore RTD B  
Dipartimento di Matematica, Informatica  
ed Economia  
Università degli studi della Basilicata  
<http://web.unibas.it/bloisi>
- SPQR Robot Soccer Team  
Dipartimento di Informatica, Automatica  
e Gestionale Università degli studi di  
Roma “La Sapienza”  
<http://spqr.diag.uniroma1.it>



# Informazioni sul corso

---

- Home page del corso:  
<http://web.unibas.it/bloisi/corsi/sistemi-operativi.html>
- Docente: Domenico Daniele Bloisi
- Periodo: I semestre ottobre 2021 – febbraio 2022
  - Lunedì dalle 15:00 alle 17:00 (Aula A18)
  - Martedì dalle 12:30 alle 14:00 (Aula 1)

# Ricevimento

---

- Durante il periodo delle lezioni:  
Martedì dalle 10:00 alle 11:30 → Edificio 3D, II piano, stanza 15  
**Si invitano gli studenti a controllare regolarmente la [bacheca degli avvisi](#) per eventuali variazioni**
- Al di fuori del periodo delle lezioni:  
da concordare con il docente tramite email

Per prenotare un appuntamento inviare  
una email a  
[domenico.bloisi@unibas.it](mailto:domenico.bloisi@unibas.it)



# Programma – Sistemi Operativi

---

- Introduzione ai sistemi operativi
- Gestione dei processi
- Sincronizzazione dei processi
- Gestione della memoria centrale
- Gestione della memoria di massa
- **File system**
- Sicurezza e protezione

# File System

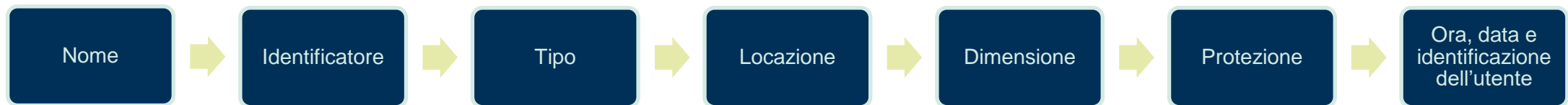
---

- Il **file system** fornisce il meccanismo per la **memorizzazione in linea** di dati e programmi appartenenti al sistema operativo
- Il **file system** è composto da:
  - un insieme di **file** (contenenti dati)
  - una struttura di **directory** (per organizzare i file)
- Il **file system** risiede, nella maggior parte dei casi, in **memoria secondaria**

# File

---

- Un **file** è un insieme di informazioni correlate, registrate in memoria secondaria, cui è stato assegnato un **nome**.
- Un file **ha attributi** che possono variare secondo il sistema operativo, ma che tipicamente comprendono i seguenti:





# Attributi dei file

La Figura 13.1 illustra una **finestra di informazioni** di un file su macOS nella quale sono visualizzati gli **attributi di un file**.

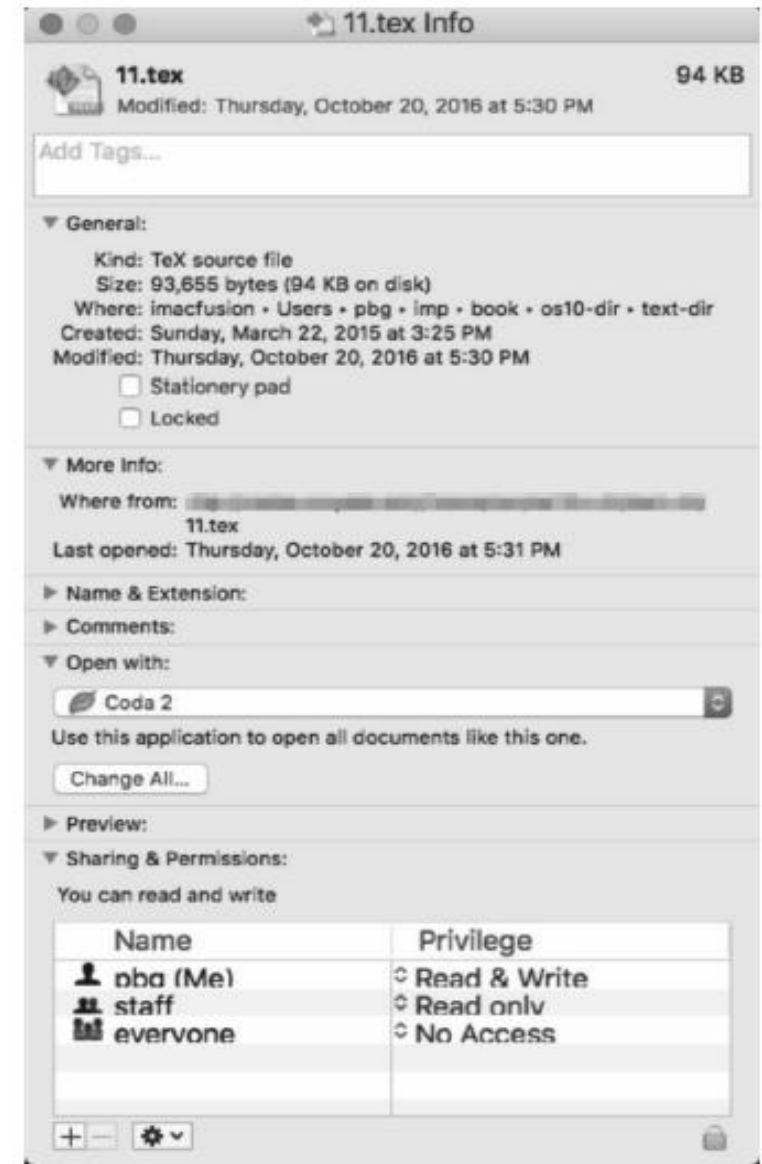
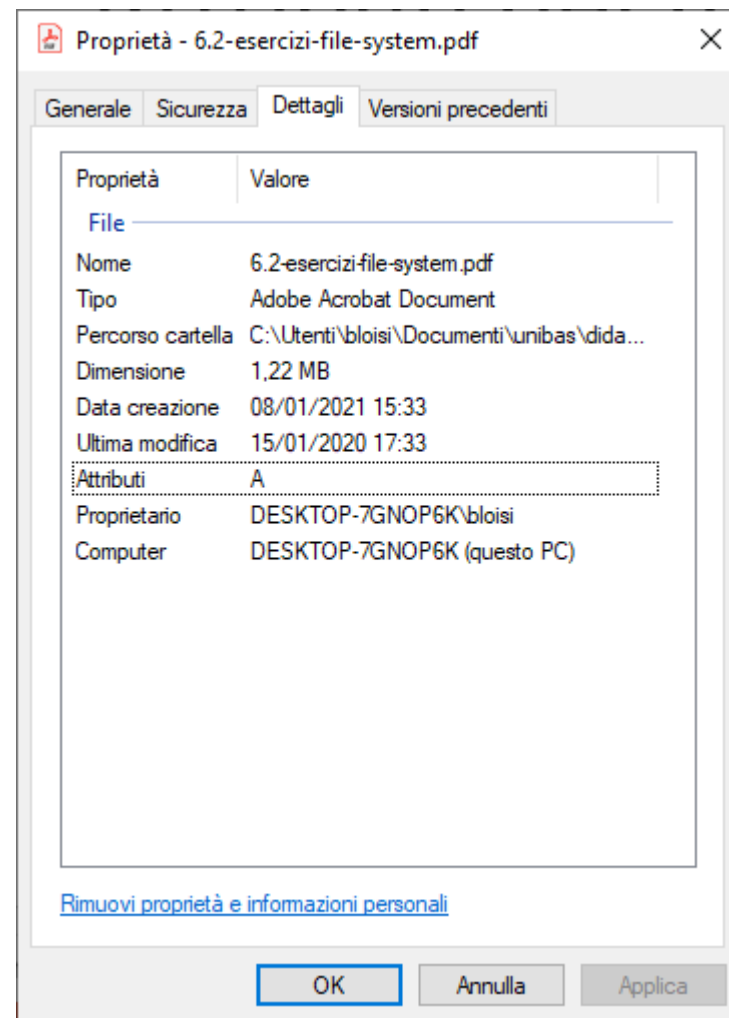
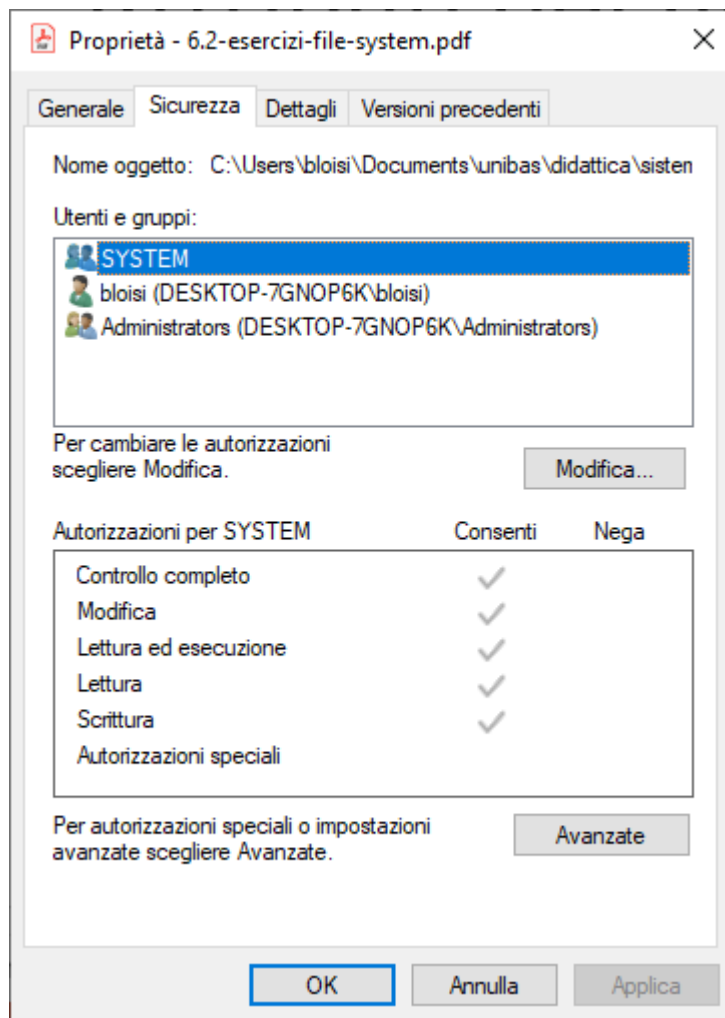
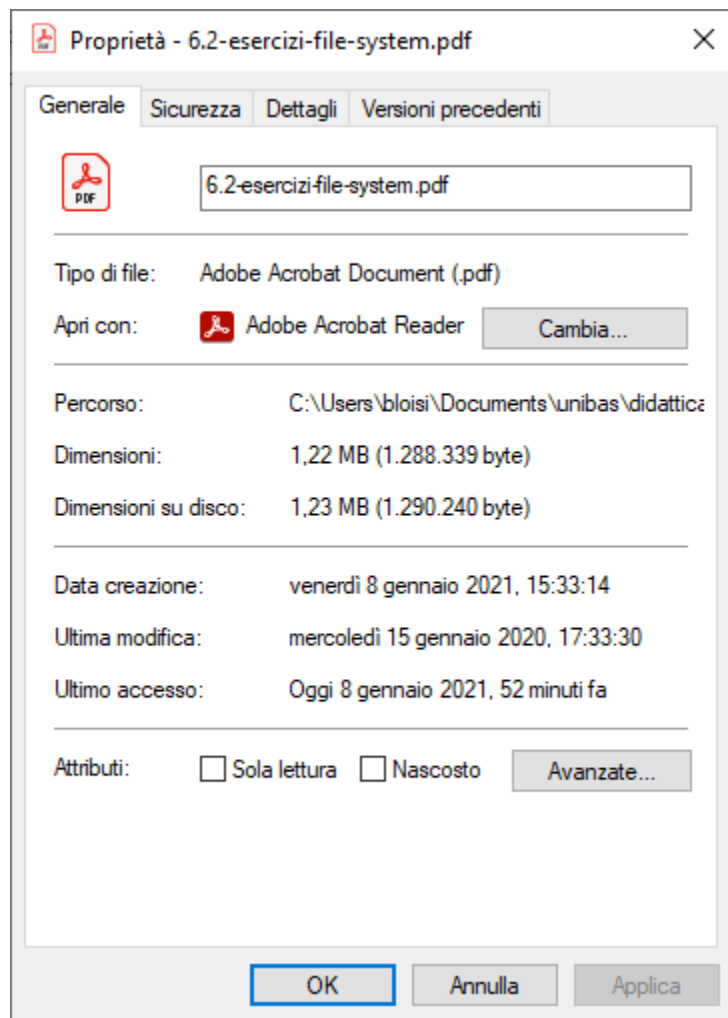


Figura 13.1 La finestra di informazioni di un file su macOS.



# Attributi dei file – Windows 10



# Operazioni sui file

---

Creazione  
di un file

Scrittura  
di un file

Lettura  
di un file

Riposizionamento  
in un file

Cancellazione  
di un file

Troncamento  
di un file

# Tipi di file

---

Tipo di file	Estensione usuale	Funzione
Eseguibile	exe, com, bin, o nessuna	Programma eseguibile, in linguaggio macchina
Oggetto	obj, o	Compilato, in linguaggio di macchina, non linkato
Codice sorgente	c, cc, java, perl, asm	Codice sorgente in vari linguaggi di programmazione
Batch	bat, sh	Comandi per l'interprete dei comandi
Markup	xml, html, tex	Dati testuali, documenti
Word processor	xml, rtf, docx	Vari formati di word processor
Libreria	lib, a, so, dll	Librerie di procedure per la programmazione
Stampa o visualizzazione	gif, pdf, jpg	File ASCII o binari in formato per la stampa o la visualizzazione
Archivio	rar, zip, tar	File contenenti più file tra loro correlati, talvolta compressi, per archiviazione o memorizzazione
Multimediali	mpeg, mov, mp3, mp4, avi	File binari contenenti informazioni audio o A/V

**Figura 13.3** Comuni tipi di file.

# Esempio Markdown .md

The image displays two side-by-side browser windows illustrating a GitHub repository page and its source code editor.

**Left Window (Repository View):** The browser address bar shows `github.com/dbloisi/detectball`. The page title is `dbloisi/detectball: detectball is a`. The main content area displays the `README.md` file. The title **detectball** is prominently shown. Below the title, the text reads: "detectball is a software for detecting the RoboCup SPL black and white ball in images captured by NAO robots - Copyright 2017 Domenico Daniele Bloisi". Further down, it states: "detectball is part of the tutorial 'How to Use OpenCV for Ball Detection' by Domenico D. Bloisi ([http://web.unibas.it/bloisi/tutorial/balldetection.html](\"http://web.unibas.it/bloisi/tutorial/balldetection.html\")) and it is distributed under the terms of the GNU Lesser General Public License (Lesser GPL)". The section **Prerequisites** follows, with the text: "You need to have OpenCV 4.3 installed to compile and run the code". The **Build the code** section contains a code block with the following commands: 

```
mkdir build
cd build && cmake ..
make
```

. The **Run the code** section contains a code block with the command: 

```
detectball test
```

.

**Right Window (Source Code Editor):** The browser address bar shows `github.com/dbloisi/detectball/edit/master/README.md`. The page title is `Editing detectball/README.md`. The navigation bar includes links for `<> Code`, `! Issues`, `🔗 Pull requests`, `🔄 Actions`, and `📁 Projects`. The breadcrumb navigation shows `detectball / README.md`. The editor interface has two tabs: `<> Edit file` (active) and `👁 Preview changes`. The code editor displays the following content: 

```
1 # detectball
2 detectball is a software for detecting the RoboCup SPL black and white ball in images
  captured by NAO robots - Copyright 2017 Domenico Daniele Bloisi
3
4 detectball is part of the tutorial "How to Use OpenCV for Ball Detection" by Domenico D.
  Bloisi (http://web.unibas.it/bloisi/tutorial/balldetection.html) and it is distributed
  under the terms of the GNU Lesser General Public License (Lesser GPL)
5
6 ### Prerequisites
7
8 You need to have OpenCV 4.3 installed to compile and run the code
9
10 ### Build the code
11
12 ```
13 mkdir build
14 cd build && cmake ..
15 make
16 ```
17
18 ### Run the code
19
20 ```
21 detectball test
22 ```
```

# Metodi di accesso

---

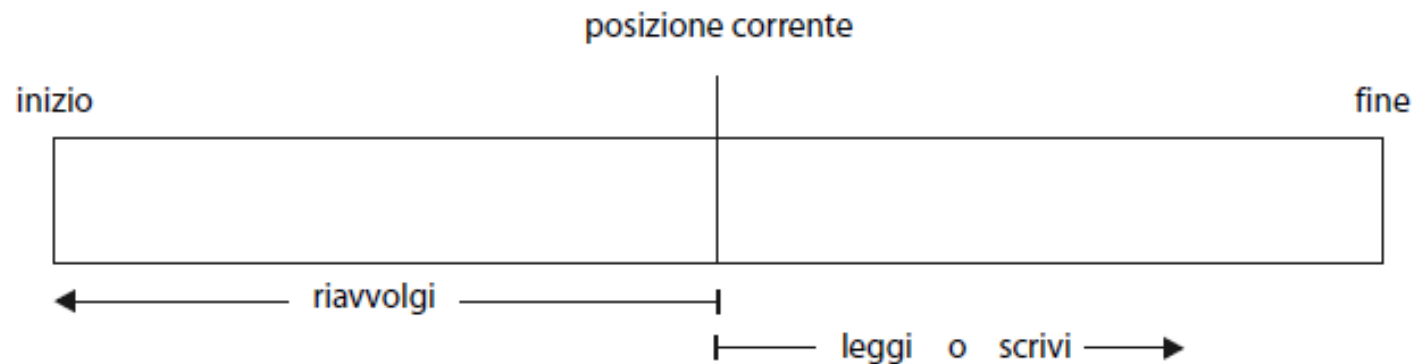
Accesso  
sequenziale

Accesso diretto

Metodo ad  
accesso  
sequenziale  
indicizzato

# Accesso sequenziale

Il più semplice metodo d'accesso è l'**accesso sequenziale**: le informazioni del file si elaborano ordinatamente, un record dopo l'altro; questo metodo d'accesso è di gran lunga il più comune, ed è usato, per esempio, dagli editor e dai compilatori.



**Figura 13.4** File ad accesso sequenziale.

# Accesso diretto

---

- Nel caso dell'**accesso diretto**, un file è formato da elementi logici (**record**) di lunghezza fissa; ciò consente ai programmi di leggere e scrivere rapidamente tali elementi senza un ordine particolare.
- I **file ad accesso diretto** sono molto utili quando è necessario accedere immediatamente a grandi quantità di informazioni (ad es. in un sistema di prenotazione dei voli).



# Accesso sequenziale simulato

---

Non tutti i sistemi operativi gestiscono ambedue i tipi di accesso:  
Tuttavia si può facilmente *simulare* l'accesso sequenziale a un file ad accesso diretto

Accesso sequenziale	Realizzazione nel caso di accesso diretto
<code>reset</code>	<code>cp = 0;</code>
<code>read_next()</code>	<code>read cp;</code> <code>cp = cp + 1;</code>
<code>write_next()</code>	<code>write cp;</code> <code>cp = cp + 1;</code>

**Figura 13.5** Simulazione dell'accesso sequenziale a un file ad accesso diretto.

# Accesso sequenziale indicizzato

L'**indice** (*index*) contiene *puntatori* ai vari blocchi; per trovare un elemento del file occorre prima cercare nell'indice, e quindi usare il *puntatore* per accedere direttamente al file e trovare l'elemento desiderato.

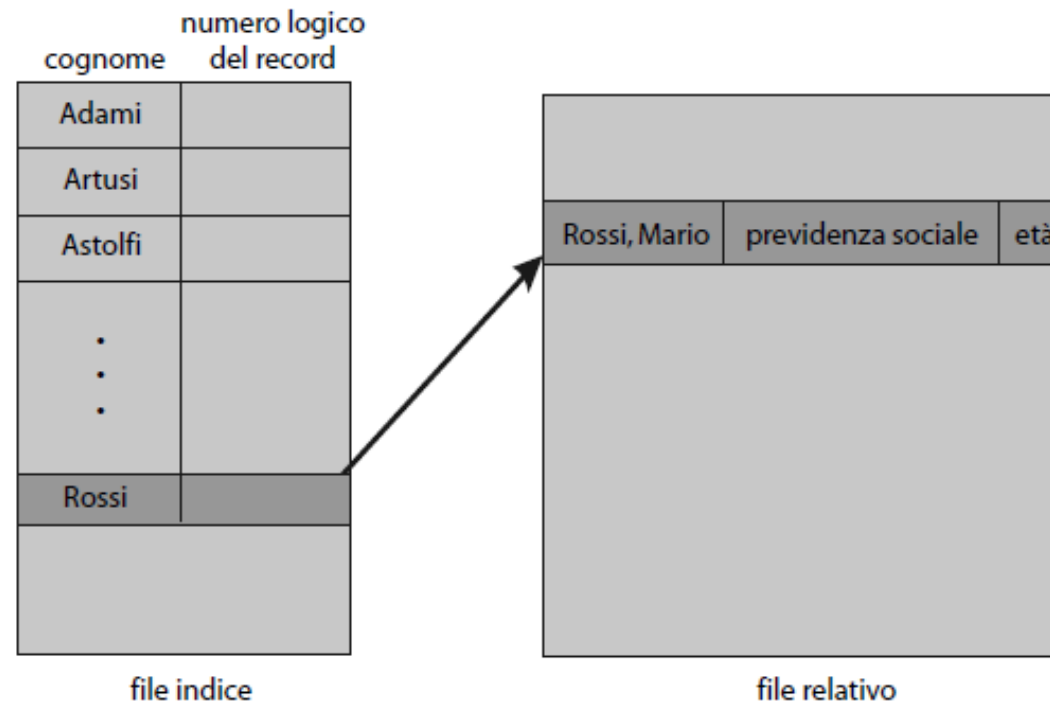


Figura 13.6 Esempio di indice e relativi file.

# Directory

---

La directory è una sorta di **tabella di simboli** che viene impiegata per tradurre i nomi dei file negli elementi in essa contenuti.

```
training_demo/  
├── annotations/  
├── exported-models/  
├── images/  
│   ├── test/  
│   │   ├── 0.png  
│   │   └── train/  
│   │       └── 234.png  
├── models/  
├── pre-trained-models/  
└── README.md
```

# Operazioni sulle Directory

---

Operazioni che si possono eseguire su una **directory**

Ricerca  
di un file

Creazione  
di un file

Cancellazione  
di un file

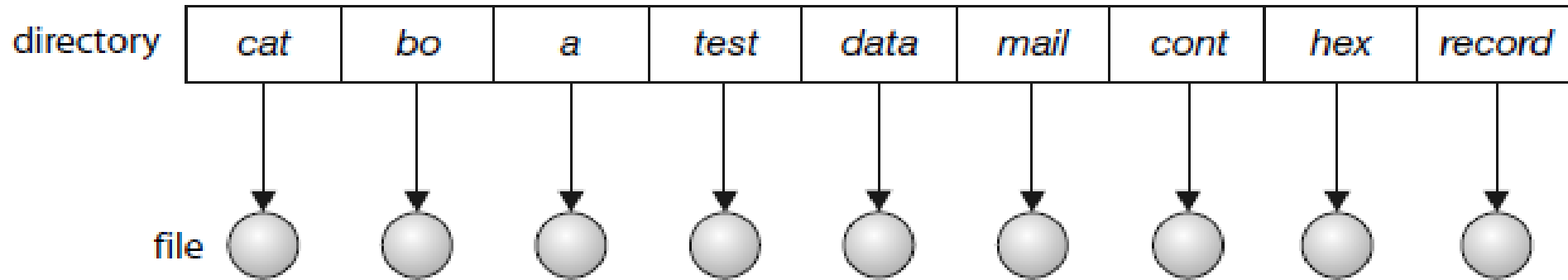
Elencazione  
di una directory

Ridenominazione  
di un file

Attraversamento  
del file system

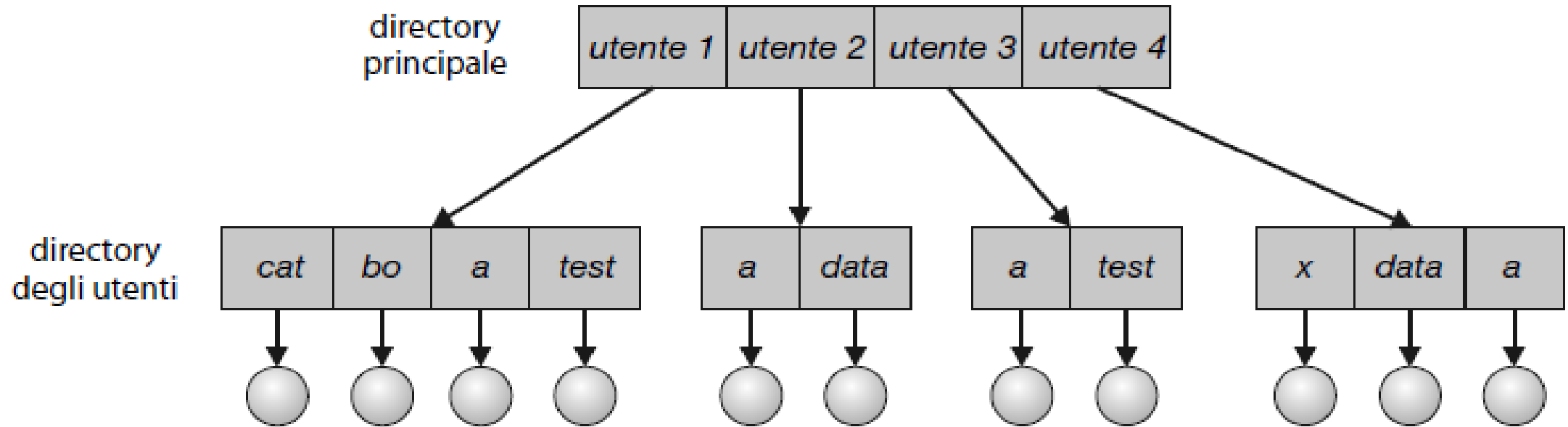
# Directory a un livello

---



**Figura 13.7** Directory a livello singolo.

# Directory a due livelli



**Figura 13.8** Struttura della directory a due livelli.

# Directory con struttura ad albero

Questo tipo di struttura permette a un utente di creare sottodirectory in cui organizzare i file

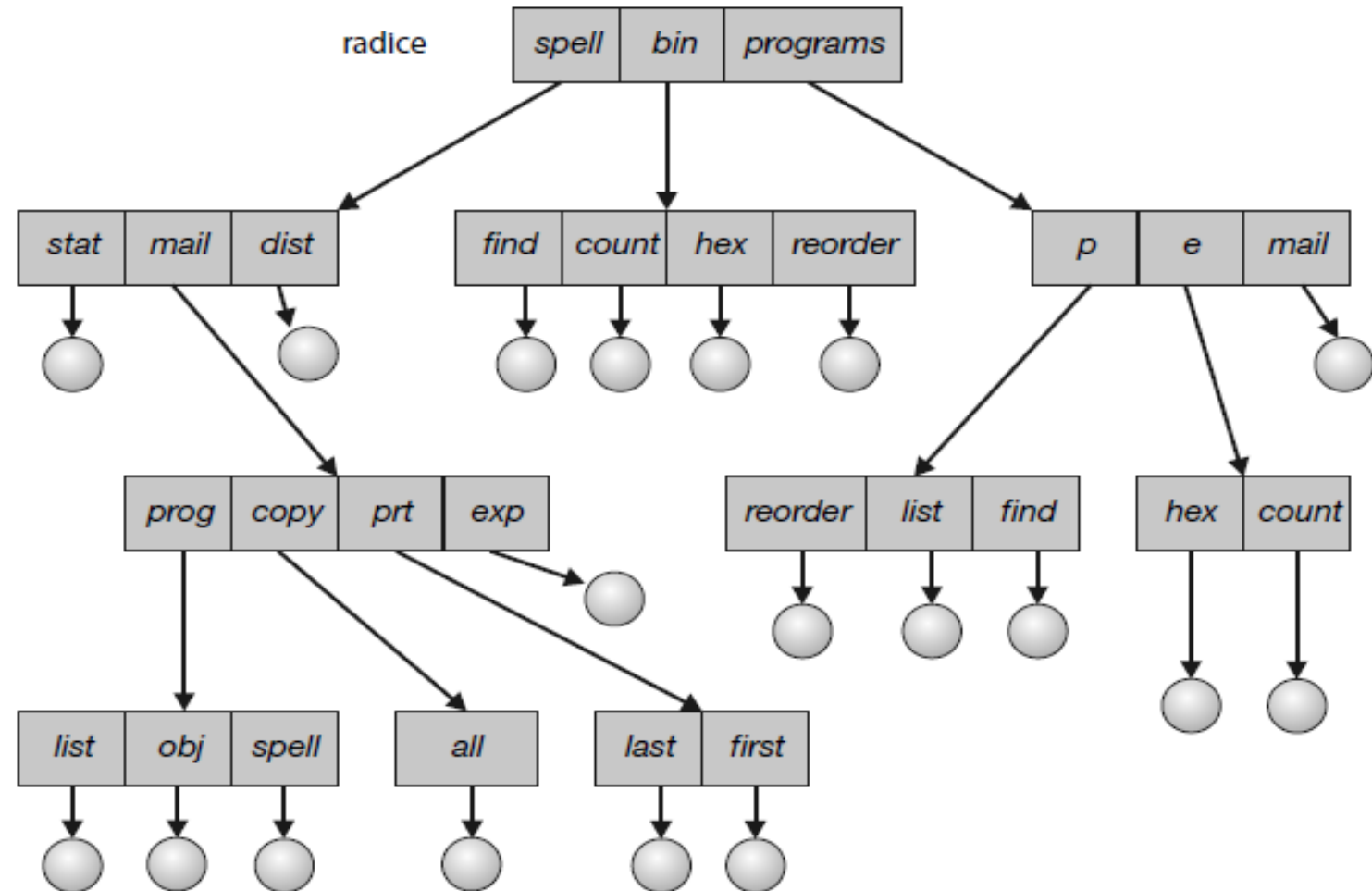
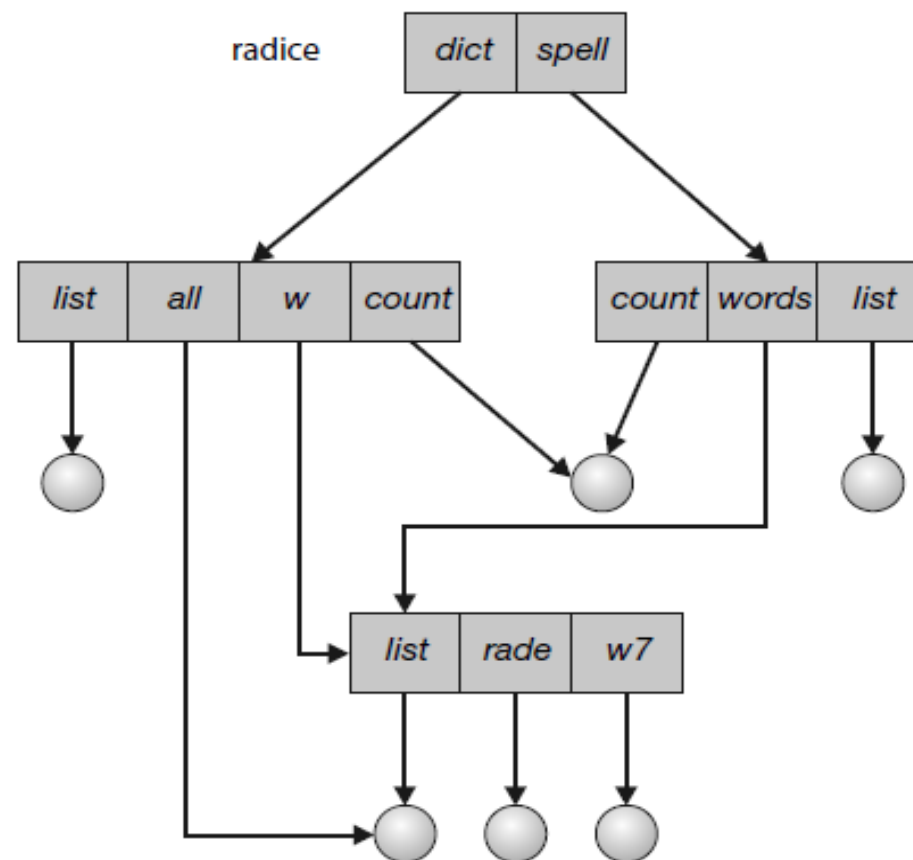


Figura 13.9 Struttura della directory ad albero.



# Directory con struttura a grafo aciclico

Le strutture delle **directory a grafo aciclico** permettono la condivisione di sottodirectory e file, ma complicano le funzioni di ricerca e cancellazione.



**Figura 13.10** Struttura della directory a grafo aciclico.

# Directory con struttura a grafo generale

Una **struttura a grafo generale** permette la massima flessibilità nella condivisione dei file e delle directory, ma talvolta richiede operazioni di “ripulitura” (*garbage collection*) per recuperare lo spazio inutilizzato nei dischi

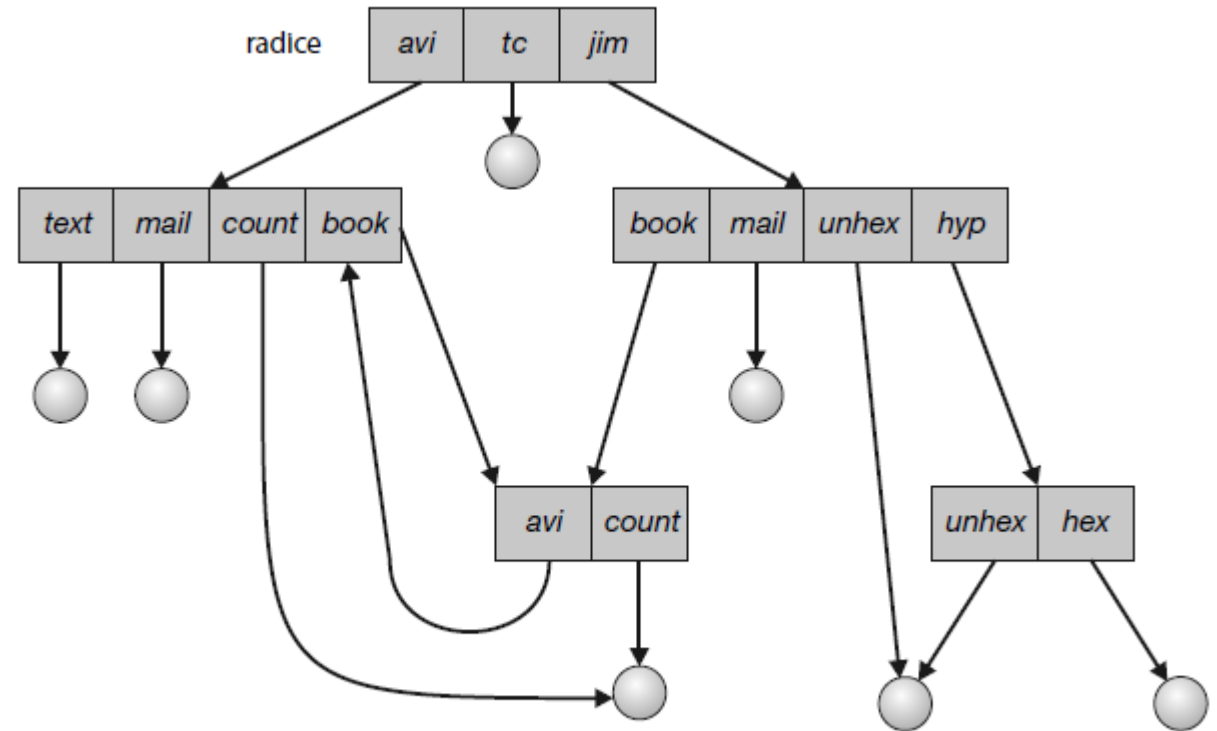


Figura 13.11 Directory a grafo generale.

# Protezione

---

- Le informazioni contenute in un sistema elaborativo devono essere protette dai danni fisici (la questione della *affidabilità*) e da accessi impropri (la questione della *protezione*).
- La necessità di proteggere i file deriva direttamente dalla possibilità di accedervi → **accesso controllato**

# Tipi di accesso

---

Lettura

Scrittura

Esecuzione

Aggiunta

Cancellazione

Elencazione

# Controllo degli accessi

Nella **lista di controllo degli accessi** (*access-control list, ACL*) sono specificati i **nomi degli utenti** e i **relativi tipi d'accesso consentiti**

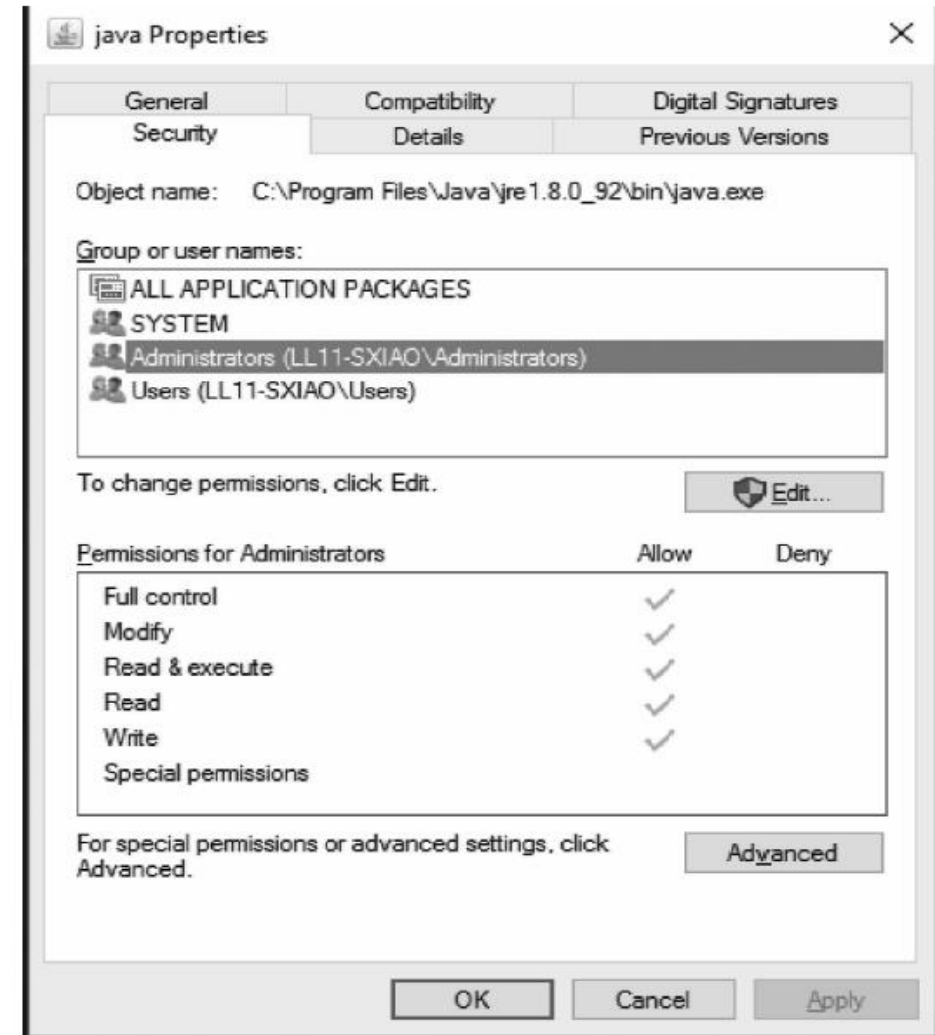


Figura 13.12 Gestione della lista di controllo degli accessi in Windows 10.

# File mappati in memoria

In alternativa alla lettura sequenziale di un file sul disco possiamo avvalerci delle **tecniche di memoria virtuale** per trattare l'I/O dei file come l'accesso ordinario alla memoria



**mappatura dei file in memoria**



una parte dello spazio degli indirizzi virtuali può essere associata logicamente al file

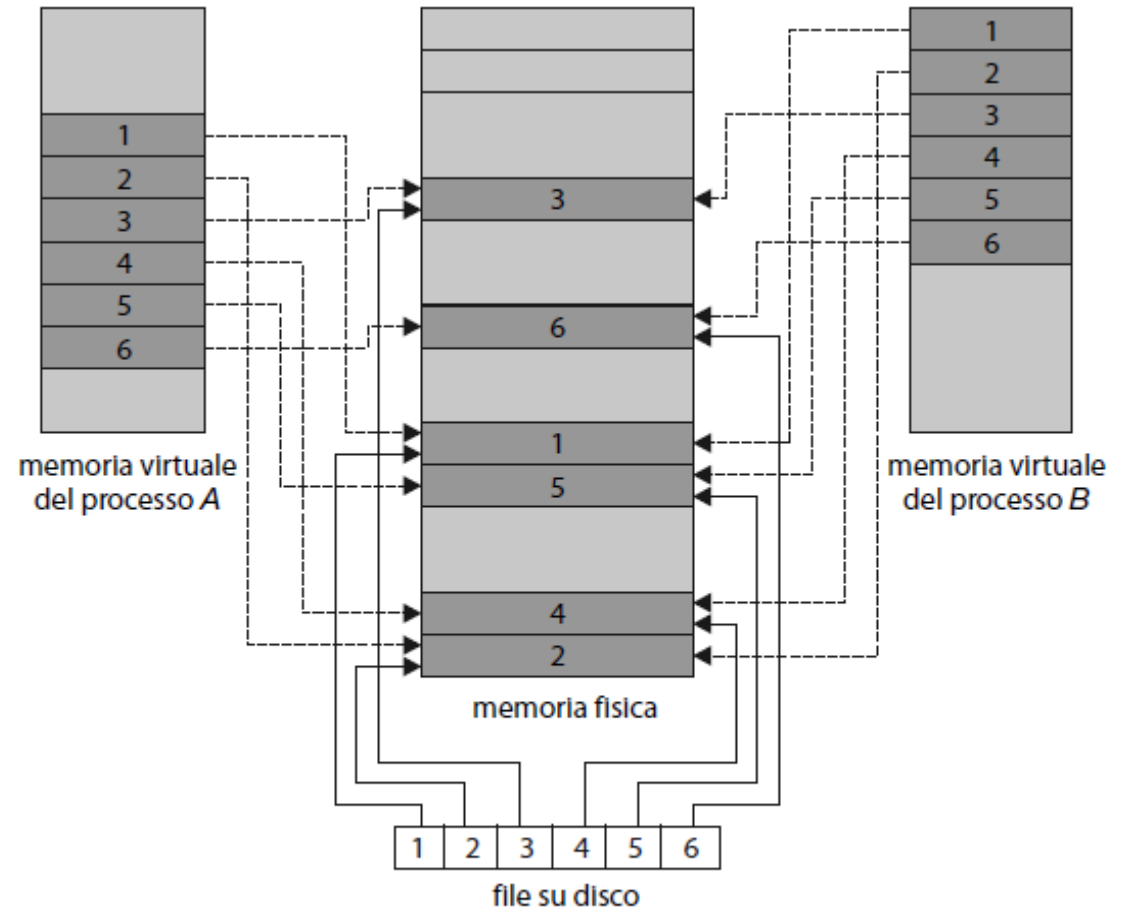
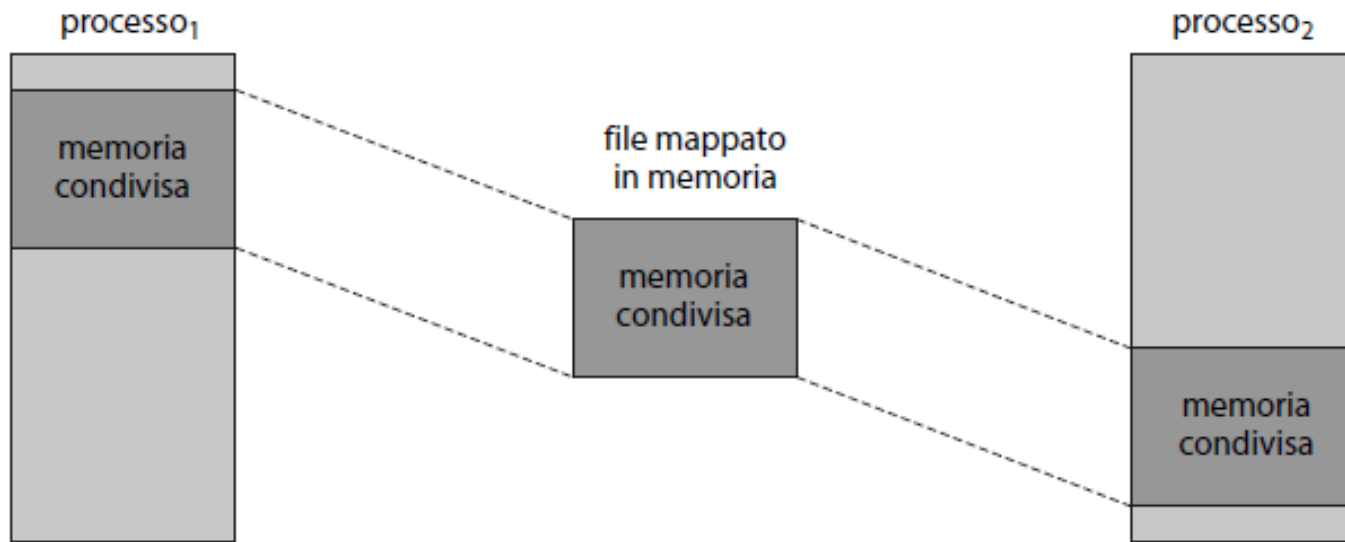


Figura 13.13 File mappati in memoria.

# File mappati in memoria

Spesso la memoria condivisa viene implementata utilizzando i **file mappati in memoria**. La **comunicazione fra processi** si ottiene in questi casi mappando in memoria uno stesso file negli spazi degli indirizzi virtuali dei processi coinvolti. Il file mappato in memoria funge da **area di memoria condivisa tra i processi comunicanti** (Figura 13.14)



**Figura 13.14** Condivisione della memoria tramite I/O mappato in memoria.



# Memoria condivisa in Windows

---

Il **processo produttore** crea dapprima un oggetto di memoria condiviso, sfruttando le funzionalità per la **mappatura di memoria** disponibili nella API Windows. Il produttore scrive quindi un messaggio nella memoria condivisa.

Il **processo consumatore** in seguito (Figura 13.16) crea a sua volta una mappatura del file, e legge il messaggio scritto dal produttore.

*Inizio listato Figura 13.15.*

```
#include <windows.h>
#include <stdio.h>

int main(int argc, char *argv[])
{

    HANDLE hFile, hMapFile;
    LPVOID lpMapAddress;

    hFile = CreateFile("temp.txt", /* nome del file */
        GENERIC_READ | GENERIC_WRITE, /* accesso R/W */
        0, /* nessuna condivisione del file*/
        NULL, /* sicurezza di default */
        OPEN_ALWAYS, /* apre il file (nuovo o esistente) */
        FILE_ATTRIBUTE_NORMAL, /* attributi del file ordinari */
        NULL); /* niente template del file*/
```

# Memoria condivisa in Windows - producer

---

*Completamento listato Figura 13.15.*

```
hMapFile = CreateFileMapping(hFile, /* riferimento al file */
    NULL, /* sicurezza di default */
    PAGE_READWRITE, /* accesso R/W alle pagine mappate */
    0, /* mappa l'intero file */
    0,
    TEXT("OggettoCondiviso")); /* oggetto condiviso con nome */

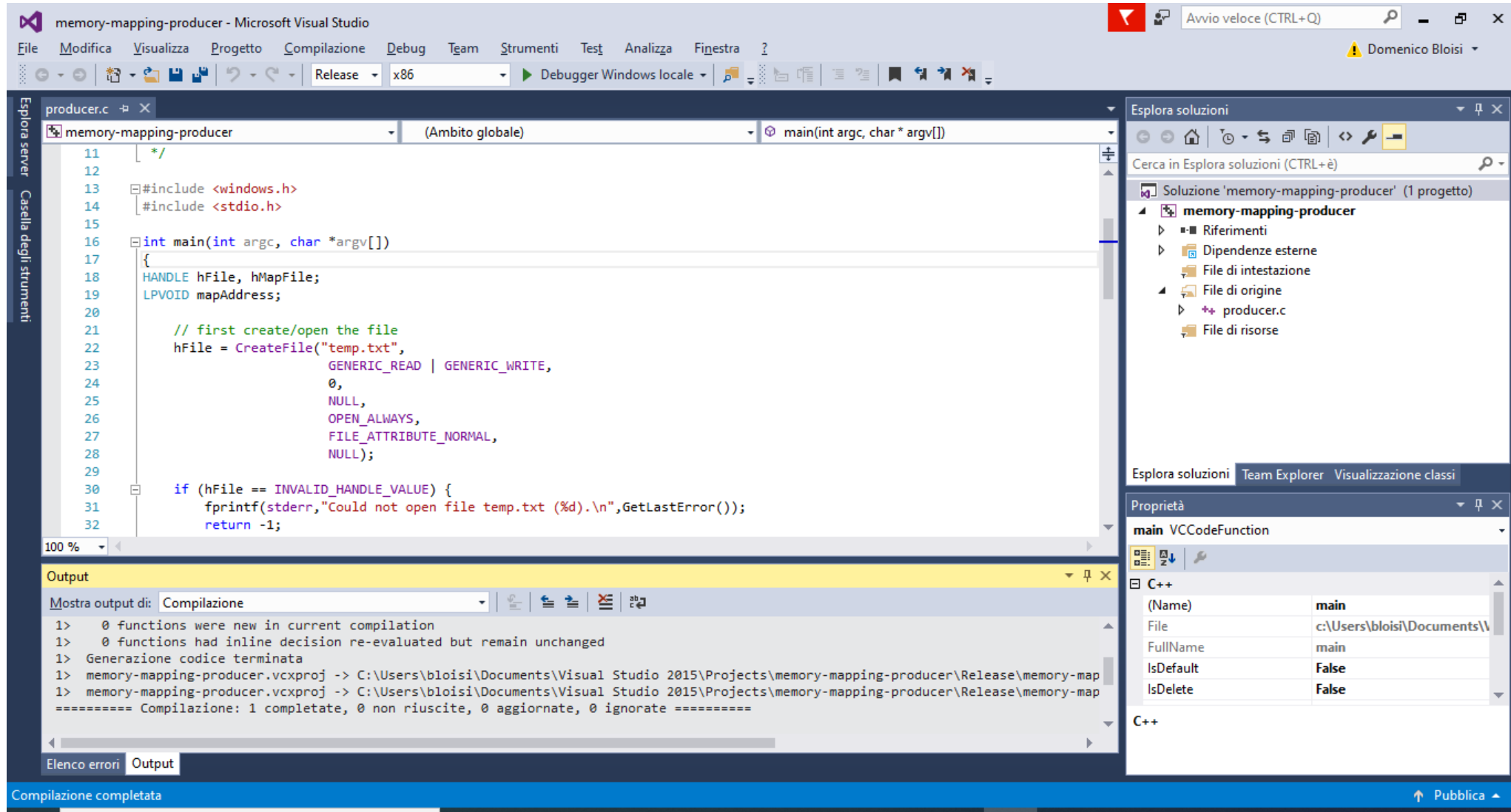
lpMapAddress = MapViewOfFile(hMapFile, /* riferimento al file */
    FILE_MAP_ALL_ACCESS, /* accesso R/W */
    0, /* vista dell'intero file */
    0,
    0);
/* scrive nella memoria condivisa */
sprintf(lpMapAddress, " Messaggio nella memoria condivisa ");

UnmapViewOfFile(lpMapAddress);
CloseHandle(hFile);
CloseHandle(hMapFile);

}
```

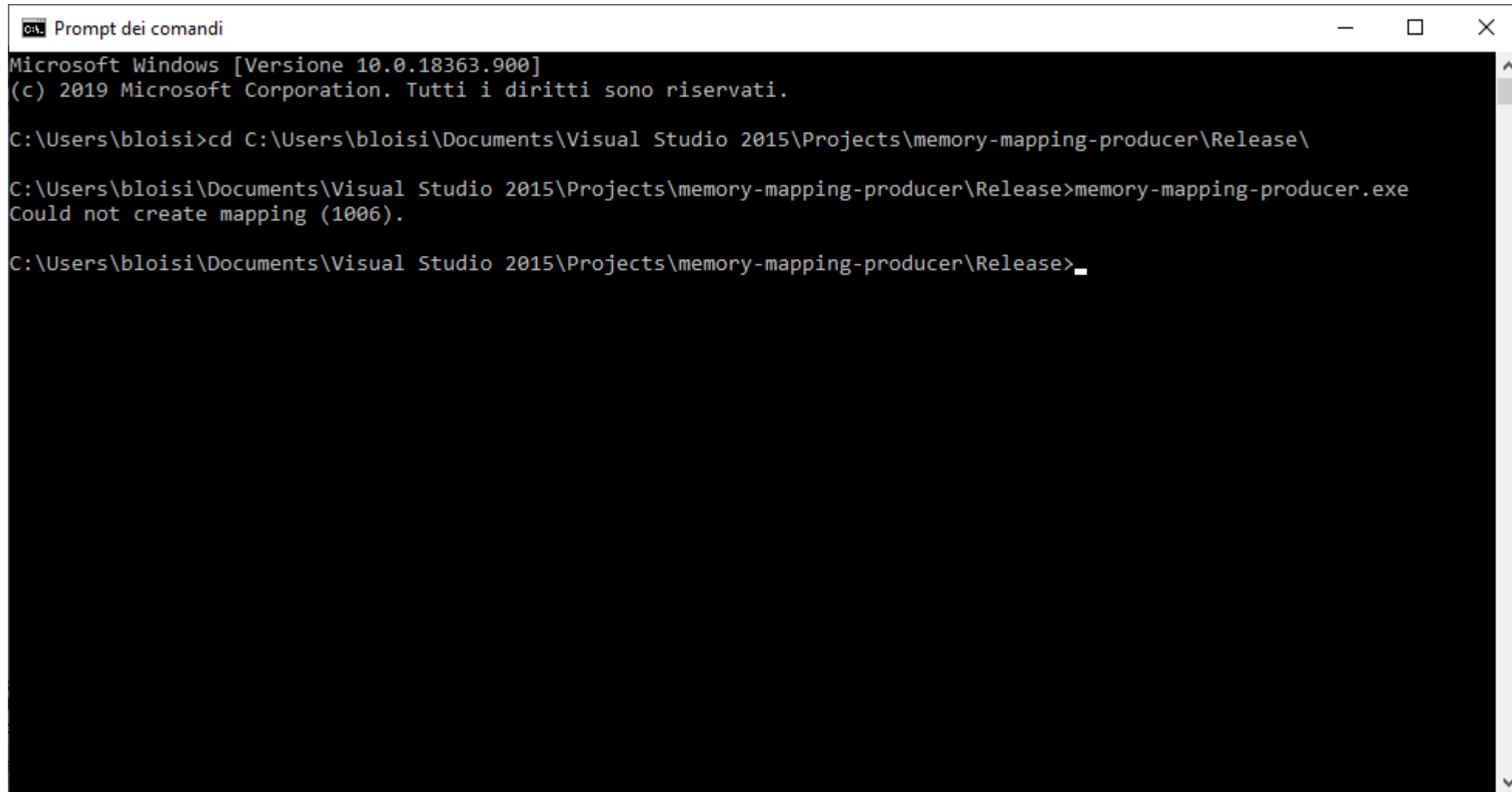
**Figura 13.15** Produttore che scrive nella memoria condivisa tramite la API Windows.

# Memoria condivisa in Windows - producer



# Memoria condivisa in Windows - producer

---



```
Ca. Prompt dei comandi
Microsoft Windows [Versione 10.0.18363.900]
(c) 2019 Microsoft Corporation. Tutti i diritti sono riservati.

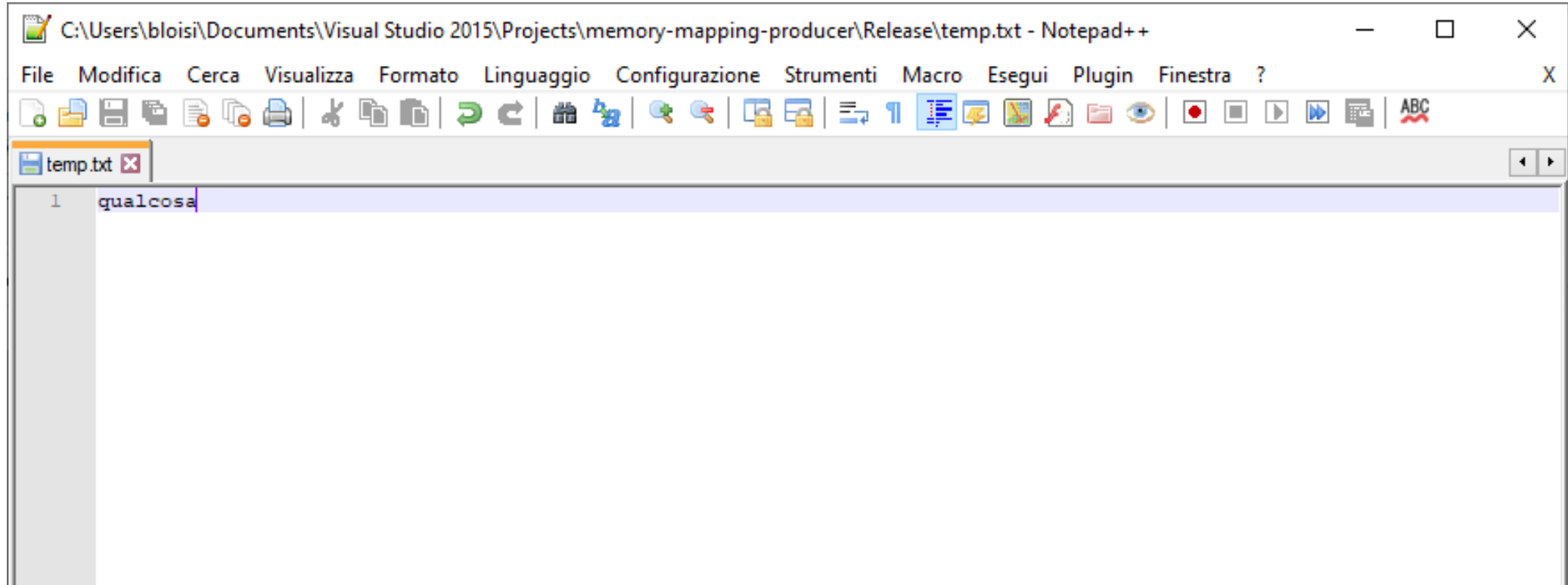
C:\Users\bloisi>cd C:\Users\bloisi\Documents\Visual Studio 2015\Projects\memory-mapping-producer\Release\

C:\Users\bloisi\Documents\Visual Studio 2015\Projects\memory-mapping-producer\Release>memory-mapping-producer.exe
Could not create mapping (1006).

C:\Users\bloisi\Documents\Visual Studio 2015\Projects\memory-mapping-producer\Release>_
```

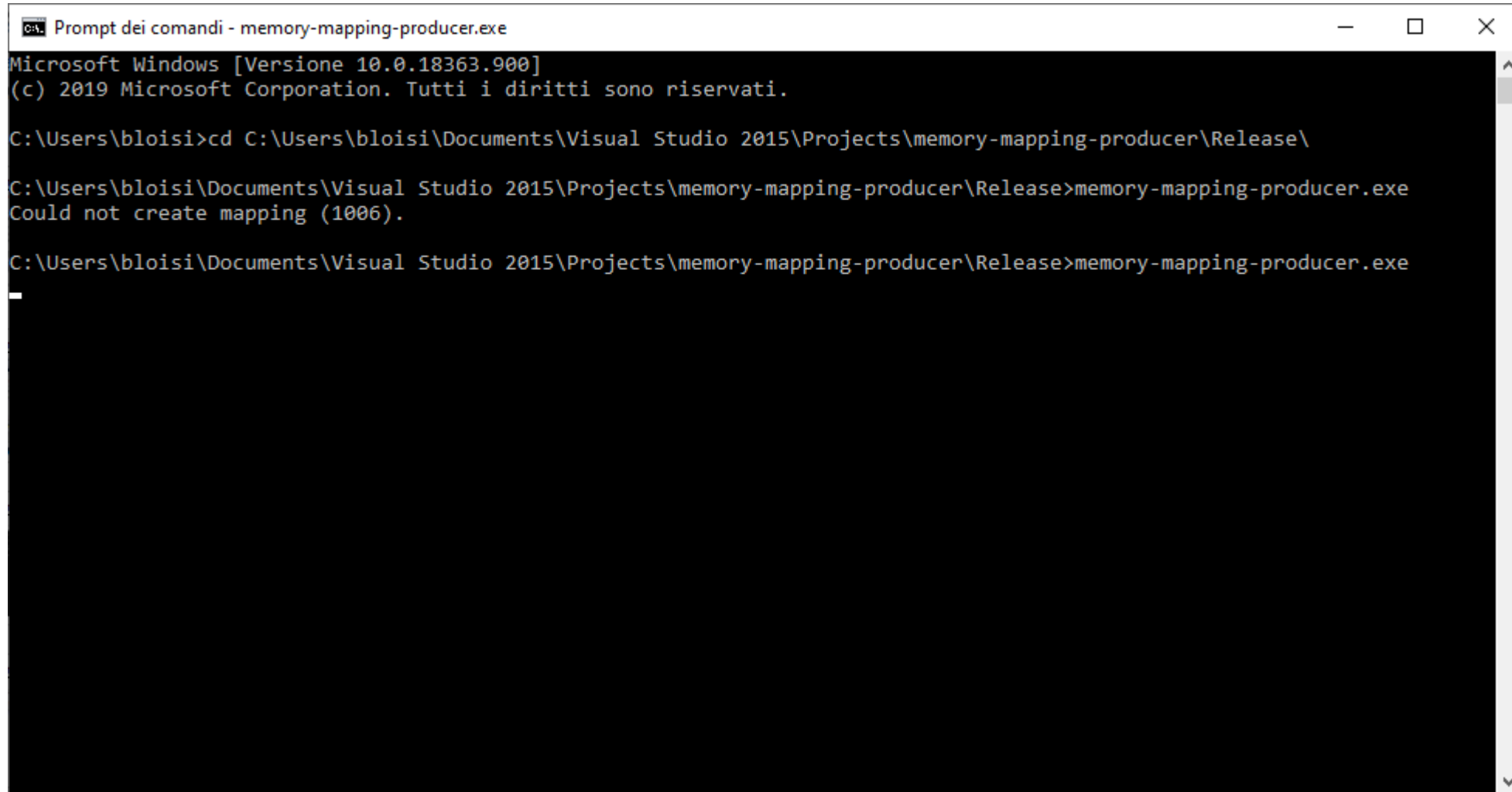
# Memoria condivisa in Windows - producer

---



# Memoria condivisa in Windows - producer

---



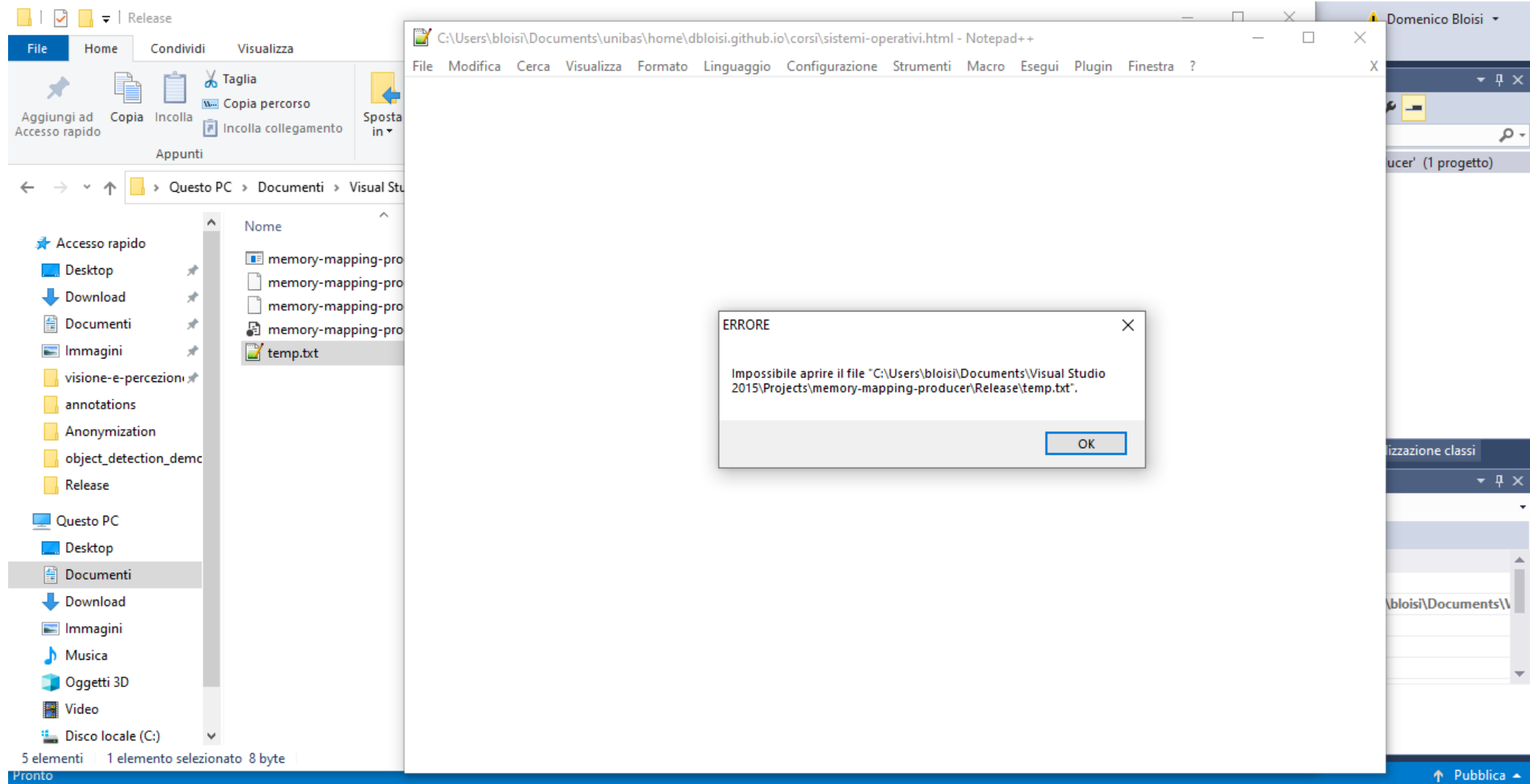
```
Prompt dei comandi - memory-mapping-producer.exe
Microsoft Windows [Versione 10.0.18363.900]
(c) 2019 Microsoft Corporation. Tutti i diritti sono riservati.

C:\Users\bloisi>cd C:\Users\bloisi\Documents\Visual Studio 2015\Projects\memory-mapping-producer\Release\

C:\Users\bloisi\Documents\Visual Studio 2015\Projects\memory-mapping-producer\Release>memory-mapping-producer.exe
Could not create mapping (1006).

C:\Users\bloisi\Documents\Visual Studio 2015\Projects\memory-mapping-producer\Release>memory-mapping-producer.exe
_
```

# Memoria condivisa in Windows - producer





# Memoria condivisa in Windows - consumer

---

```
#include <windows.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    HANDLE hMapFile;
    LPVOID lpMapAddress;

    hMapFile = OpenFileMapping(FILE_MAP_ALL_ACCESS, /* R/W */
        FALSE, /* nessuna ereditarietà */
        TEXT("OggettoCondiviso")); /* nome del file mappato */

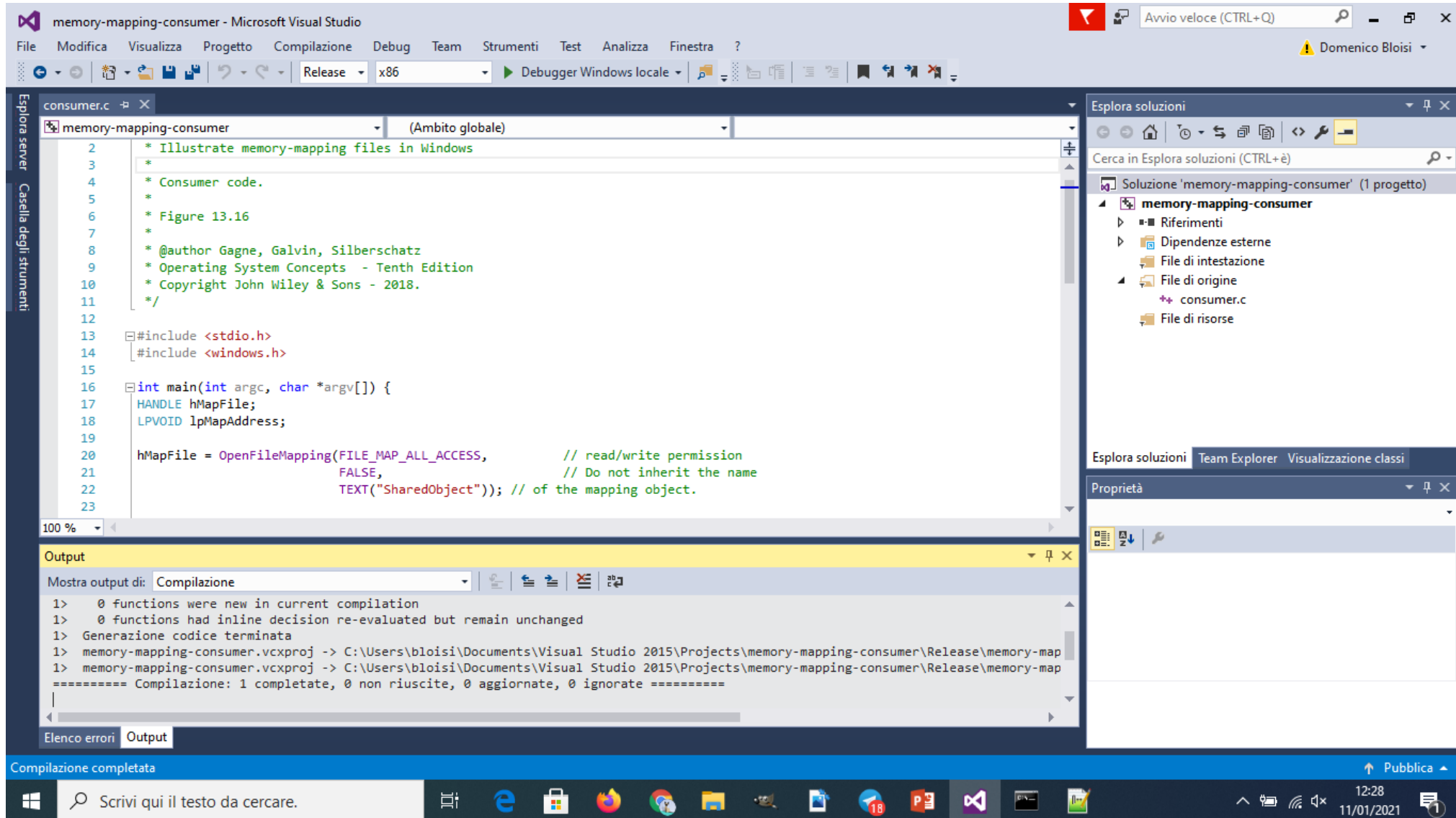
    lpMapAddress = MapViewOfFile(hMapFile, /*riferimento al file */
        FILE_MAP_ALL_ACCESS, /* accesso in lettura/scrittura */
        0, /* vista dell'intero file */
        0,
        0);

    /* lettura dalla memoria condivisa */
    printf("Messaggio letto: %s", lpMapAddress);

    UnmapViewOfFile(lpMapAddress);
    CloseHandle(hMapFile);
}
```

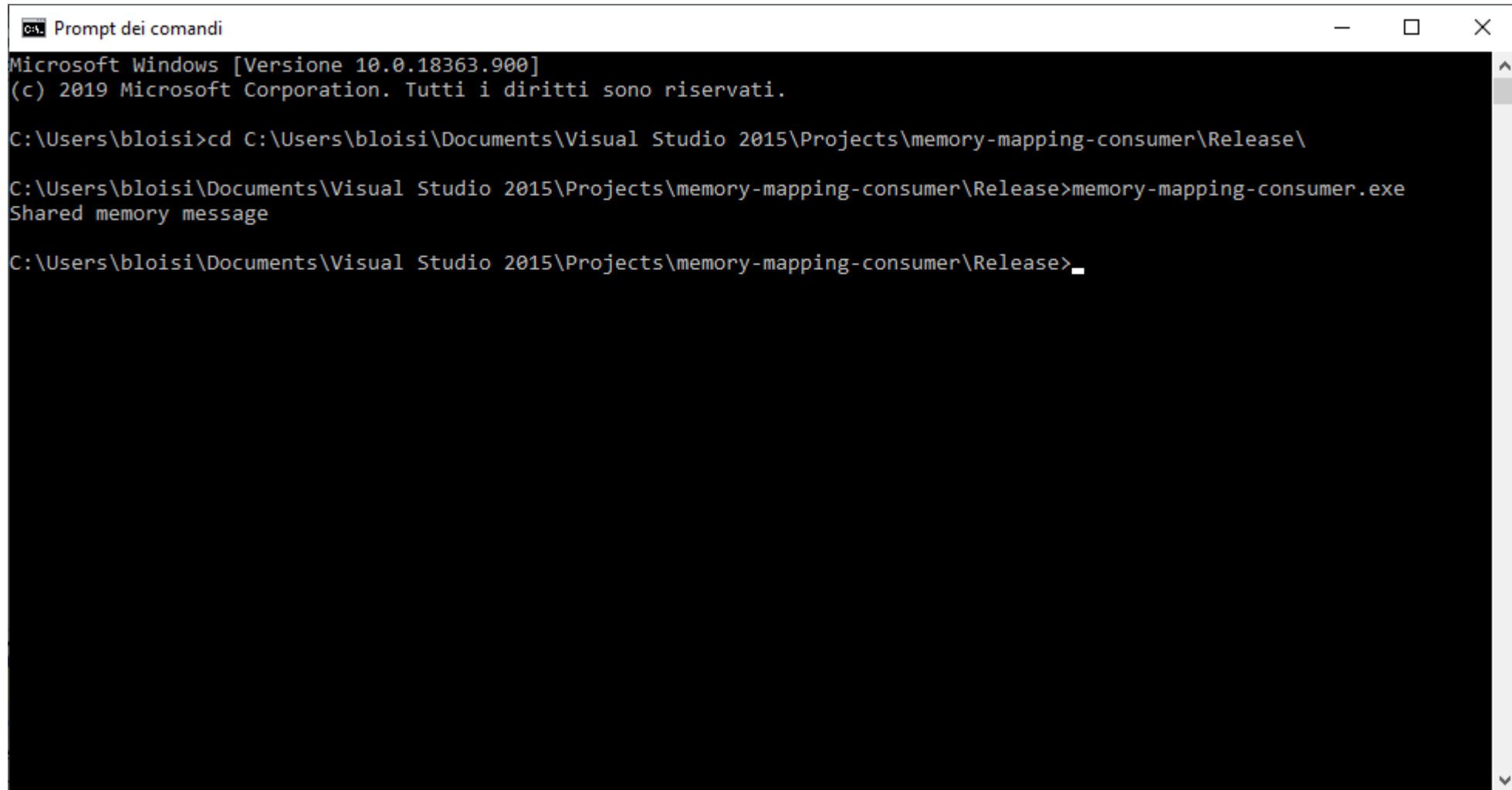
**Figura 13.16** Consumatore che legge dalla memoria condivisa tramite la API Windows.

# Memoria condivisa in Windows - consumer



# Memoria condivisa in Windows - consumer

---



```

C:\> Prompt dei comandi
Microsoft Windows [Versione 10.0.18363.900]
(c) 2019 Microsoft Corporation. Tutti i diritti sono riservati.

C:\Users\bloisi>cd C:\Users\bloisi\Documents\Visual Studio 2015\Projects\memory-mapping-consumer\Release\

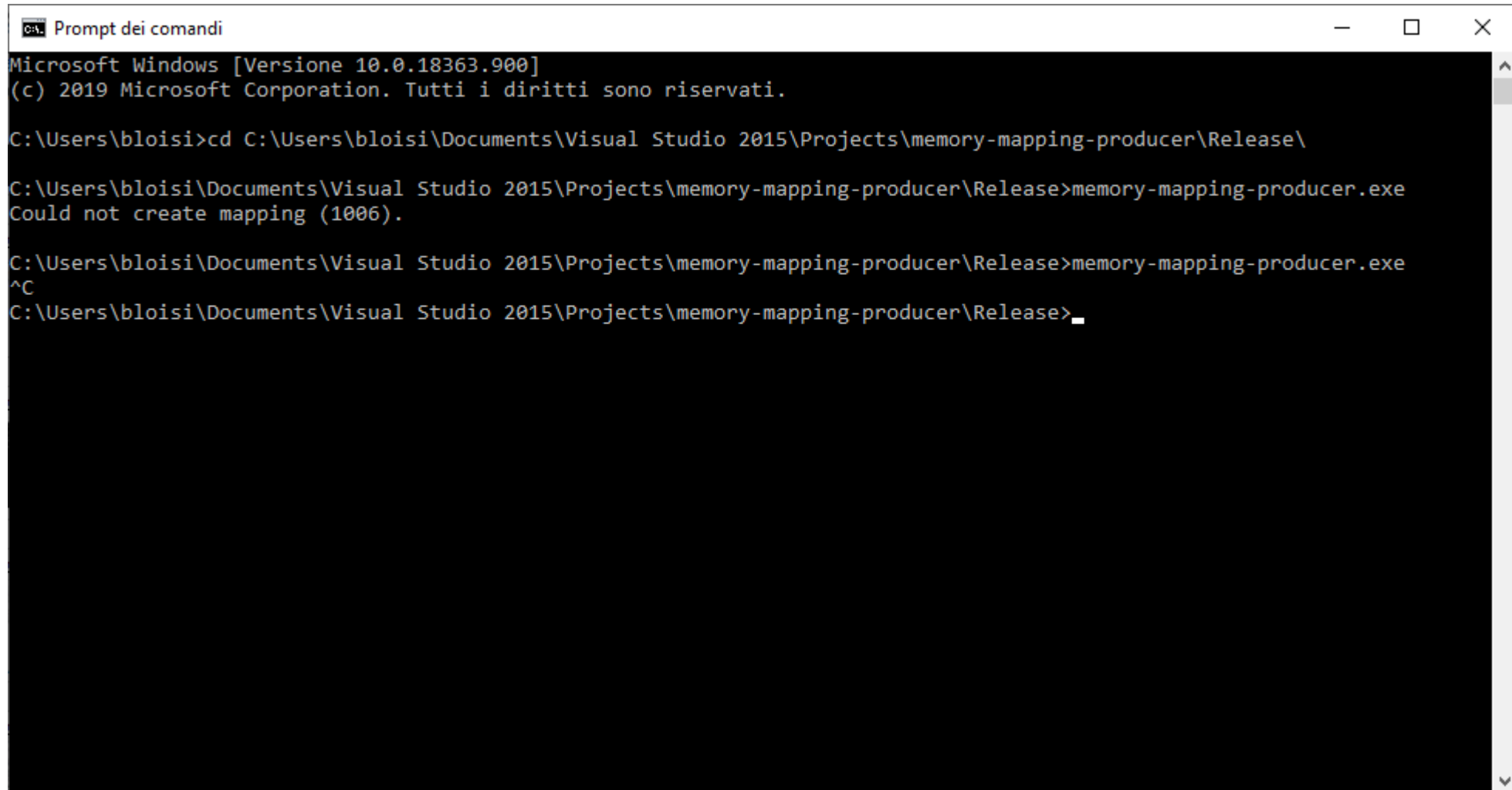
C:\Users\bloisi\Documents\Visual Studio 2015\Projects\memory-mapping-consumer\Release>memory-mapping-consumer.exe
Shared memory message

C:\Users\bloisi\Documents\Visual Studio 2015\Projects\memory-mapping-consumer\Release>_

```

# Memoria condivisa in Windows - producer

---



```
C:\> Prompt dei comandi
Microsoft Windows [Versione 10.0.18363.900]
(c) 2019 Microsoft Corporation. Tutti i diritti sono riservati.

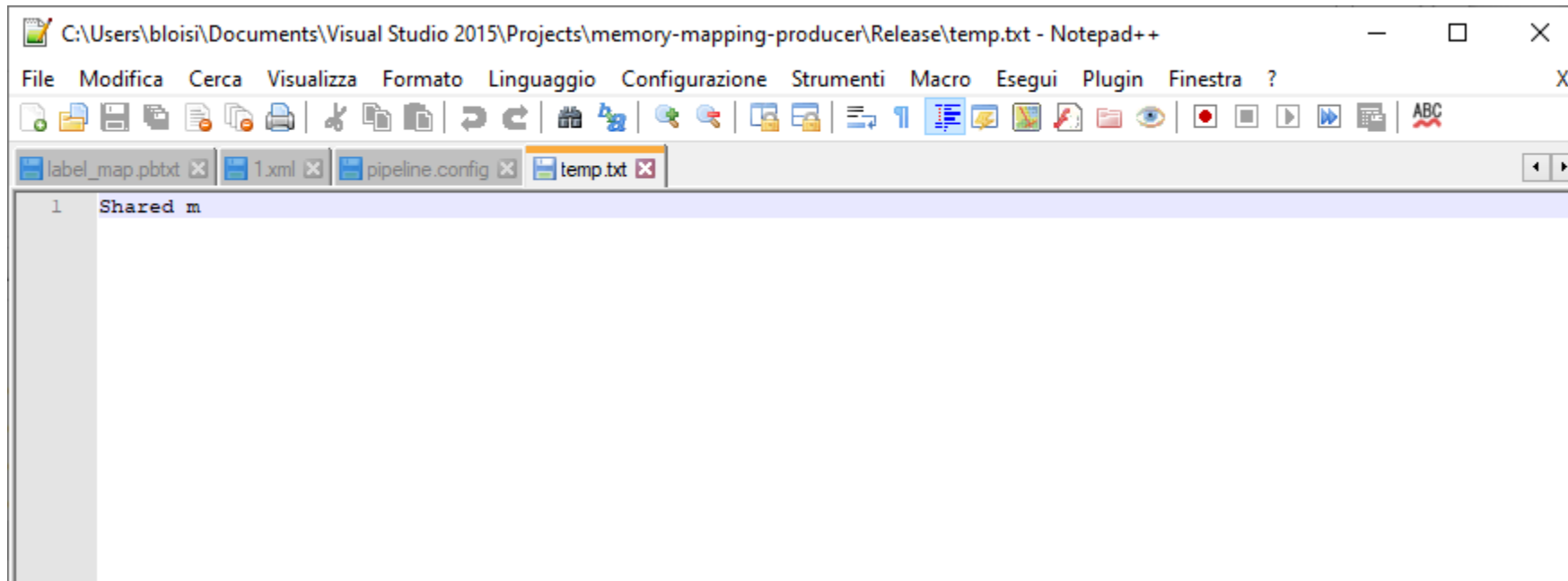
C:\Users\bloisi>cd C:\Users\bloisi\Documents\Visual Studio 2015\Projects\memory-mapping-producer\Release\

C:\Users\bloisi\Documents\Visual Studio 2015\Projects\memory-mapping-producer\Release>memory-mapping-producer.exe
Could not create mapping (1006).

C:\Users\bloisi\Documents\Visual Studio 2015\Projects\memory-mapping-producer\Release>memory-mapping-producer.exe
^C
C:\Users\bloisi\Documents\Visual Studio 2015\Projects\memory-mapping-producer\Release>_
```

# Memoria condivisa in Windows - producer

---



# Realizzazione del file system

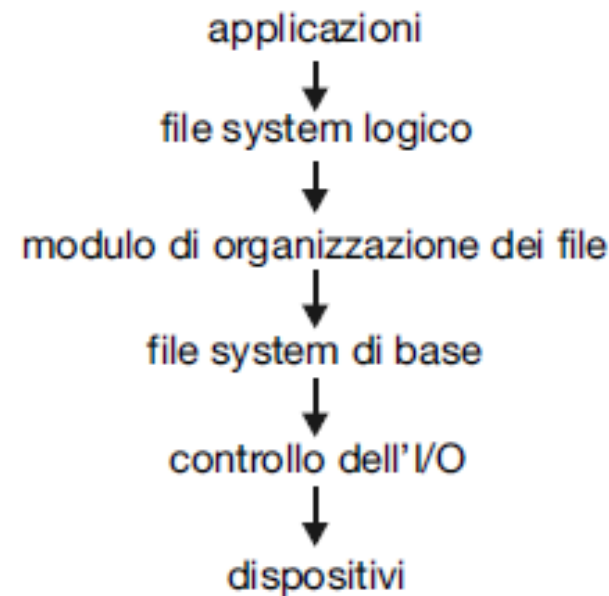
- Descriveremo, per semplicità, i file sytem che risiedono sul più comune tipo di memoria secondaria, cioè il disco.
- Per fornire un efficiente e conveniente accesso al disco, il sistema operativo fa uso di uno o più **file system** che consentono di *memorizzare, individuare e recuperare* facilmente i dati.

# Struttura del file system

---

Per migliorare l'efficienza dell'I/O, i trasferimenti tra memoria centrale e dischi si eseguono per **blocchi**. Ciascun blocco nel disco è composto da uno o più **settori**.

Il **file system** è generalmente composto da più **livelli** distinti



**Figura 14.1** File system stratificato.

# Operazioni del file system

---

Per realizzare un **file system** si usano diverse strutture dati, sia nei dischi sia in memoria. Queste strutture variano a seconda del sistema operativo e del file system, ma esistono dei principi generali.

Fra le strutture presenti nei dischi ci sono le seguenti:

blocco di  
controllo  
dell'avviamento

blocco di  
controllo  
del volume

struttura  
della directory

blocco di  
controllo del file  
(FCB)



# Blocco di controllo del file

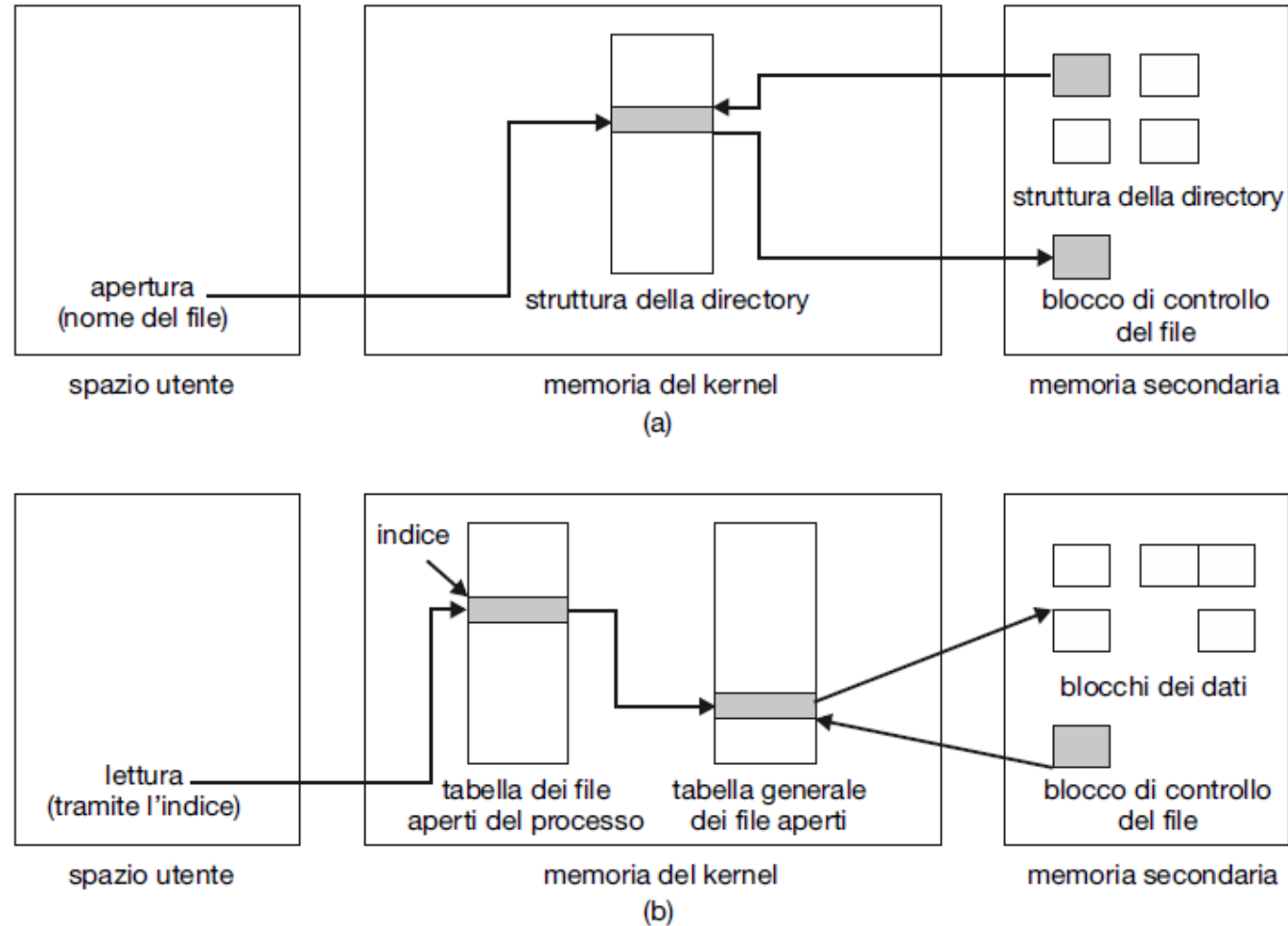
---

Il **blocco di controllo del file (FCB)** contiene molti dettagli del relativo file. Ha un identificatore unico per poterlo associare a una voce della directory.

permessi per il file
data e ora di creazione, di ultimo accesso e di ultima scrittura
proprietario del file, gruppo, ACL
dimensione del file
blocchi di dati del file o puntatori a blocchi di dati del file

**Figura 14.2** Tipica struttura di FCB (*file-control block*).

# Apertura e lettura di file



**Figura 14.3** Strutture del file system che si mantengono nella memoria; (a) apertura di file; (b) lettura di file.

# Realizzazione delle directory

---

Lista  
lineare

Tabella  
hash

Il più semplice metodo di realizzazione di una directory è basato sull'uso di una **lista lineare** contenente i nomi dei file con puntatori ai blocchi di dati.

Questo metodo è di facile programmazione, ma la sua esecuzione è onerosa in termini di tempo.

Un'altra struttura dati che si usa per realizzare le directory è la **tabella hash**, che riceve un valore calcolato a partire dal nome del file e riporta un puntatore al nome del file nella lista lineare. Offre diminuzione del tempo di ricerca nella directory, ma presenta il problema delle **collisioni**.

# Metodi di allocazione

---

Esistono tre metodi principali per l'allocazione dello spazio di un disco:

Allocazione  
contigua

Allocazione  
concatenata

Allocazione  
indicizzata

# Allocazione contigua

L'**allocazione contigua** può risentire di **frammentazione esterna**. Lo spazio contiguo si può allargare attraverso delle estensioni allo scopo di aumentare la flessibilità e ridurre la **frammentazione esterna**.

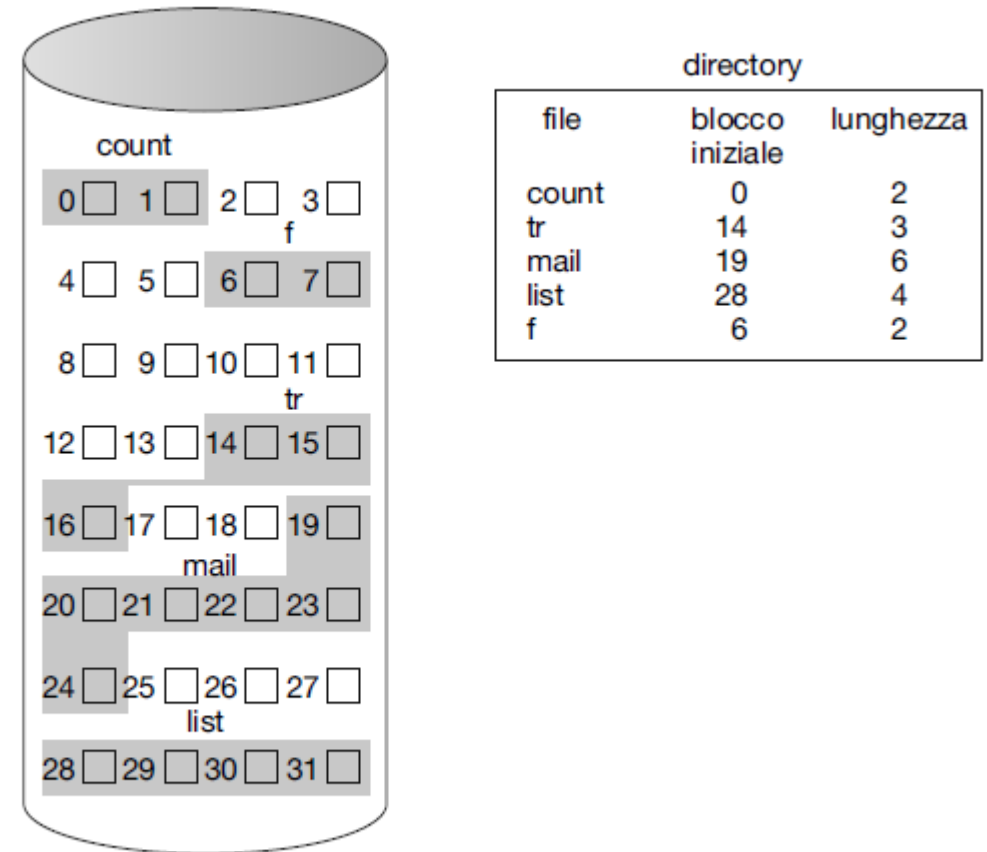


Figura 14.4 Allocazione contigua dello spazio dei dischi.

# Allocazione concatenata

L'accesso diretto ai file è molto inefficiente con **l'allocazione concatenata**

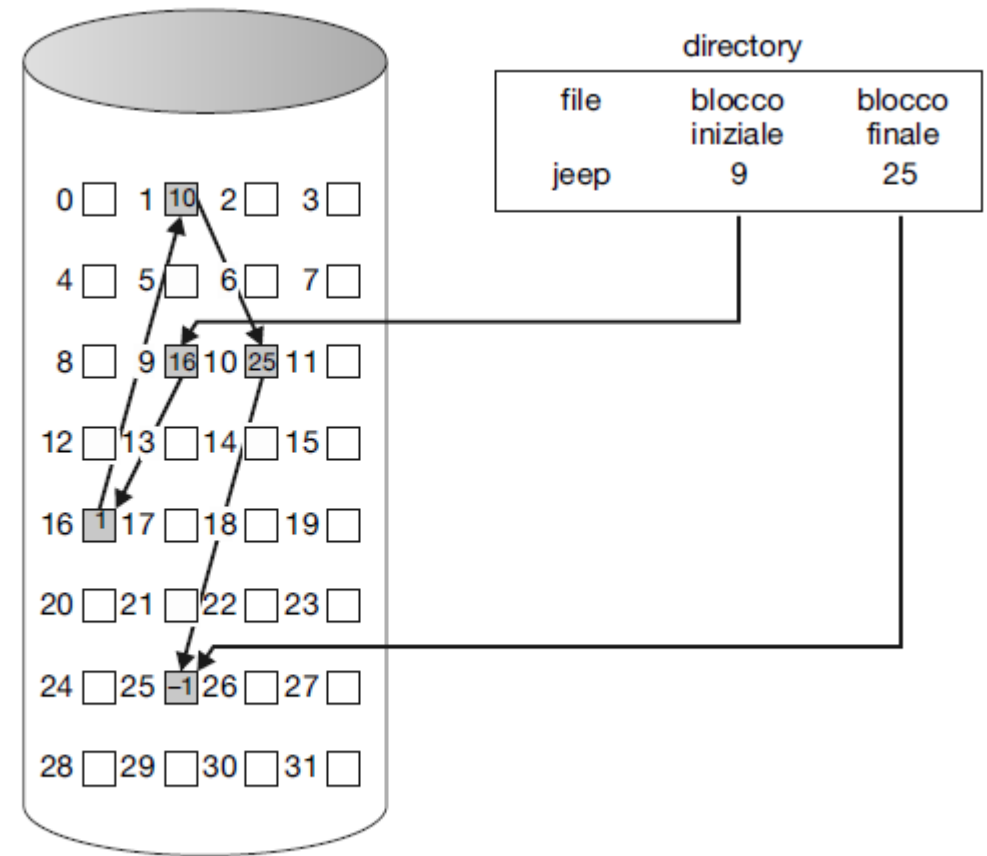


Figura 14.5 Allocazione concatenata dello spazio dei dischi.

# Allocazione concatenata

Una variante importante del metodo di allocazione concatenata consiste nell'uso della **tabella di allocazione dei file** (*file allocation table, FAT*).

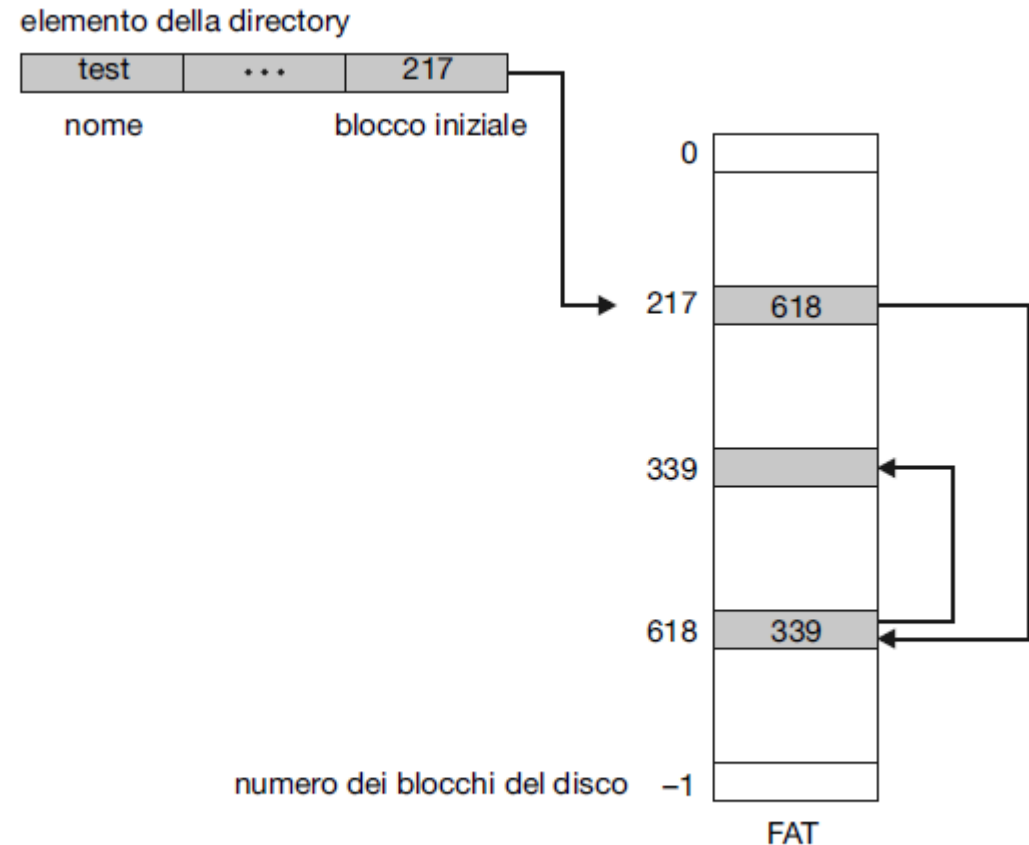


Figura 14.6 Tabella di allocazione dei file.

# Allocazione indicizzata

L'**allocazione indicizzata** può richiedere un notevole overhead per il proprio **blocco indice**. L'allocazione indicizzata si può realizzare in **cluster** per incrementare il throughput e ridurre il numero di elementi dell'indice necessari. L'**indicizzazione in cluster** di grandi dimensioni è simile all'**allocazione contigua con estensioni**.

L'**allocazione indicizzata** raggruppa tutti i puntatori in una sola locazione: il **blocco indice**.

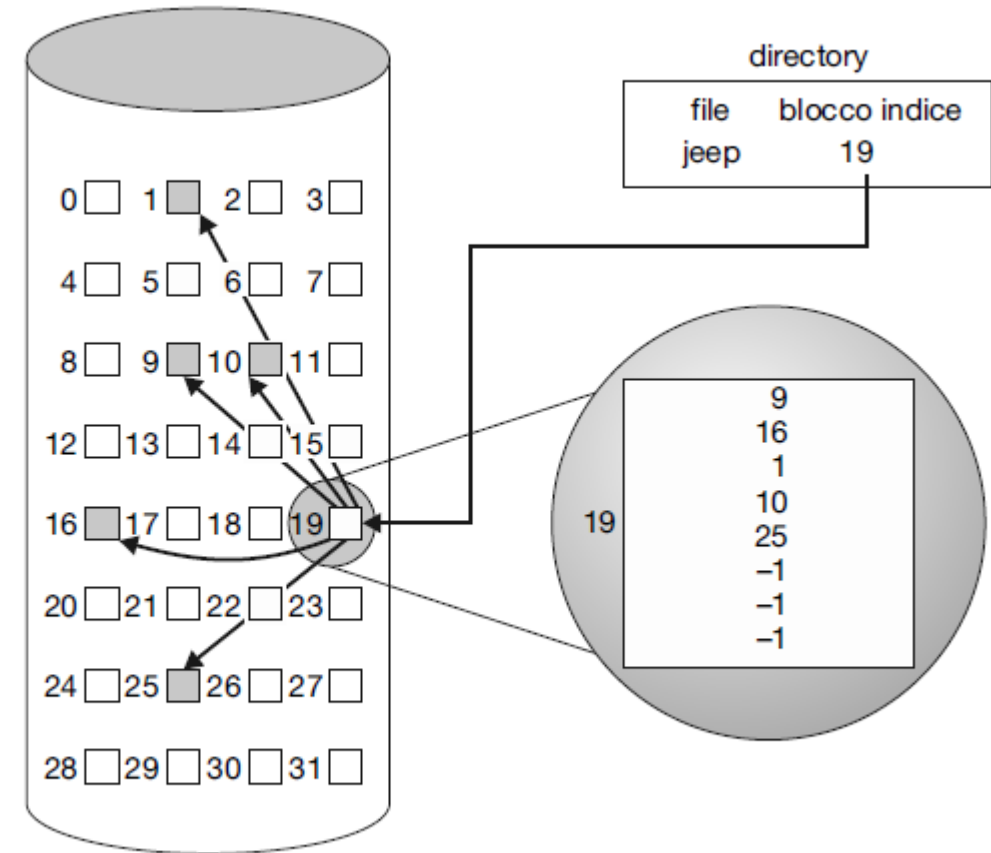


Figura 14.7 Allocazione indicizzata dello spazio dei dischi.



# iNode

Fra i possibili meccanismi per gestire la questione della dimensione del **blocco indice** vi è il così detto **schema combinato** utilizzato nei sistemi basati su UNIX.

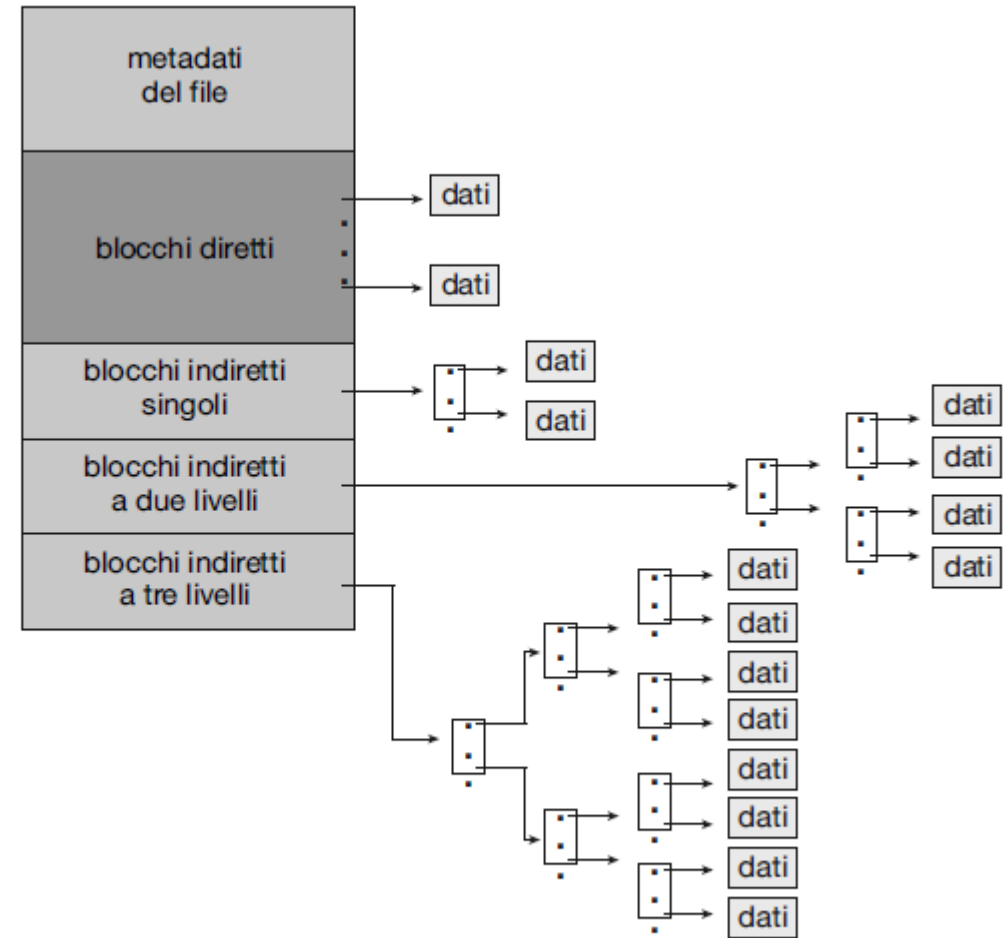


Figura 14.8 Inode di UNIX.

# Gestione dello spazio libero

---

Per tener traccia dello **spazio libero** in un disco, il sistema conserva una **lista dello spazio libero**.

Metodi di allocazione dello spazio libero:

Vettori di bit

Liste  
concatenate

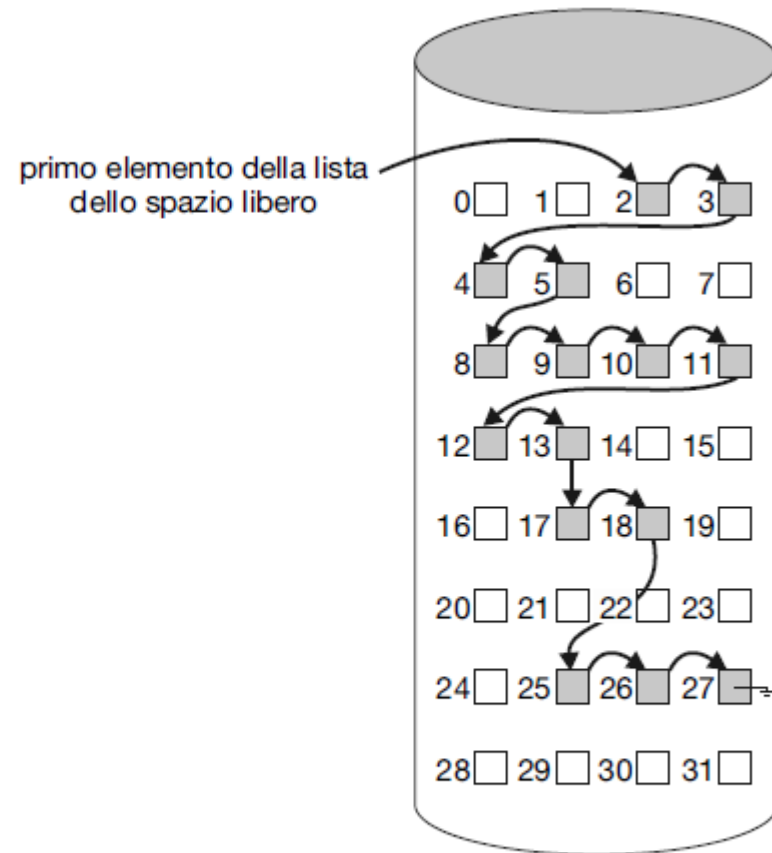
Ottimizzazioni:

Raggruppamento

Conteggio

FAT (colloca  
la lista concatenata  
in una singola area  
contigua)

# Gestione dello spazio libero



**Figura 14.9** Lista concatenata degli spazi liberi su disco.

# Efficienza e prestazioni

- Le procedure di gestione delle directory devono tener conto dell'efficienza, delle prestazioni e dell'affidabilità.
- Buffer cache:** sezione separata della memoria centrale dove tenere i blocchi in previsione di un loro riutilizzo entro breve tempo.

**Memory-mapping:** prevede la lettura dei blocchi di disco dal file system e la loro memorizzazione nella buffer cache

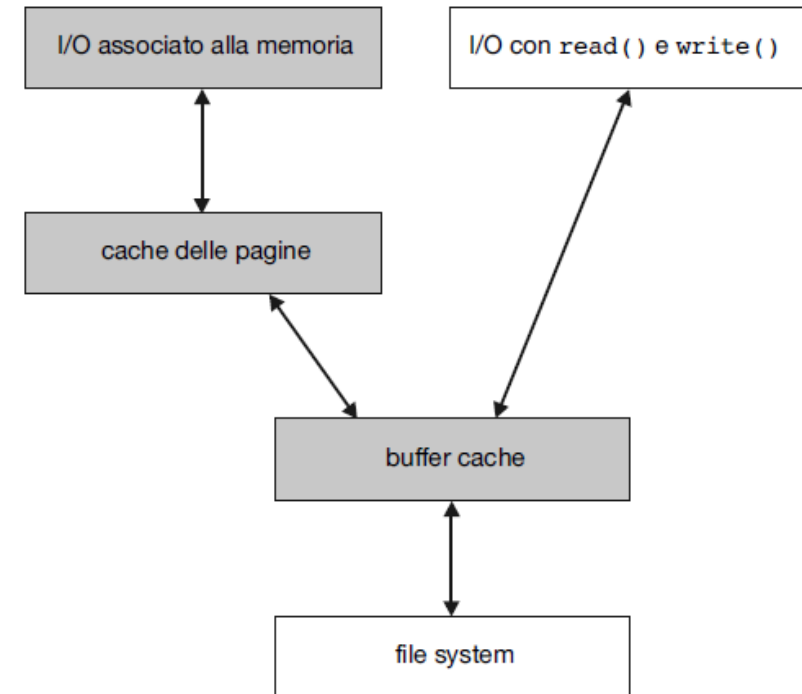
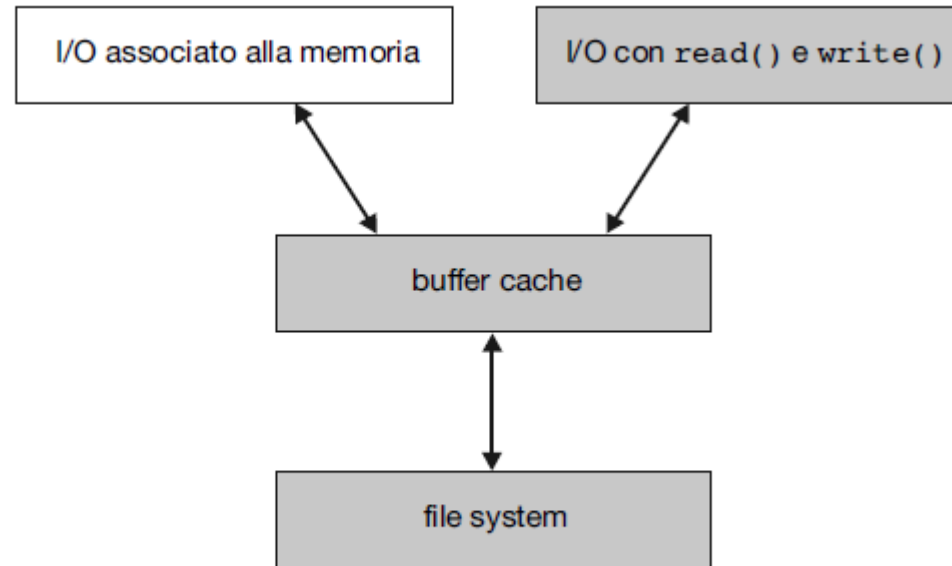


Figura 14.10 I/O senza una buffer cache unificata.

# Buffer cache unificata

Con una **buffer cache unificata**, sia il memory-mapping sia le chiamate di sistema `read()` e `write()` usano la stessa cache delle pagine, con il vantaggio di evitare *// double caching* e di permettere al sistema di memoria virtuale di gestire dati del file system.



**Figura 14.11** I/O con una buffer cache unificata.

# Ripristino – verifica della coerenza

---

- Il **verificatore della coerenza** – un programma di sistema come fsck in UNIX – confronta i dati delle directory con i blocchi dati dei dischi, tentando di correggere ogni incoerenza.
- Può essere utilizzato per riparare la struttura danneggiata di un file system.
- Gli strumenti del sistema operativo per la creazione di **copie di backup** consentono la copiatura nelle unità a nastro dei dati contenuti nei dischi allo scopo di poterli **ripristinare** in seguito a perdite dovute a malfunzionamenti dei dispositivi fisici, errori del sistema operativo, o a errori degli utenti.



**UNIVERSITÀ DEGLI STUDI  
DELLA BASILICATA**

## *Corso di Sistemi Operativi*

# File System

Docente:  
**Domenico Daniele  
Bloisi**

