TCP concorrente
in ambiente Windows
(2)

Sezione critica

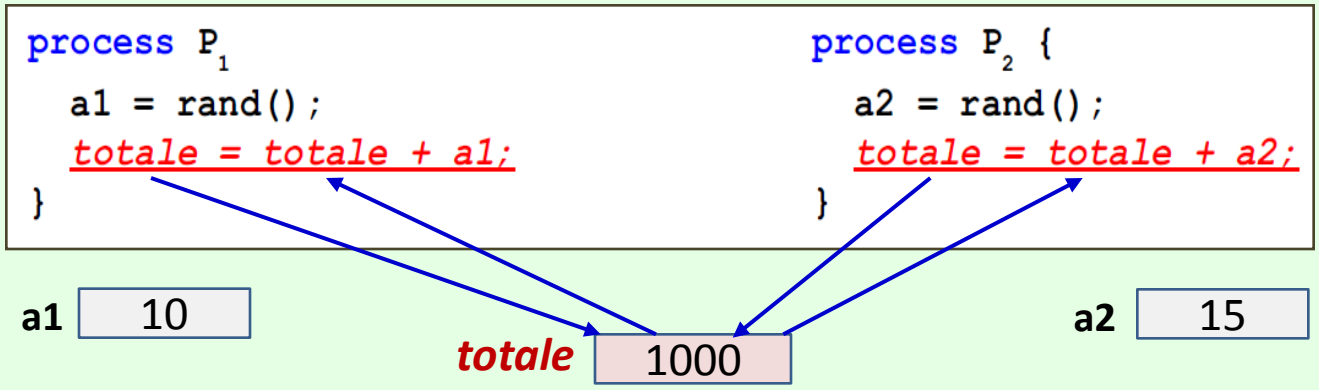
La parte di programma che utilizza **una o più risorse condivise** viene detta **sezione critica** (*critical section*).

L'esecuzione di sezioni critiche da parte di programmi concorrenti o cooperanti può portare a **risultati non attendibili.**

Per evitare questo:

- è necessario (ma non sufficiente) **evitare che due processi si trovino contemporaneamente all'interno di una sezione critica** relativa allo stesso dato condiviso
- la **sezione critica deve essere protetta bloccando** temporaneamente uno dei processi

Esempio:
La variabile ***totale*** è **globale** per i due processi e quindi è **condivisa**
mentre **a1** e **a2** sono variabili locali e quindi **non condivise**

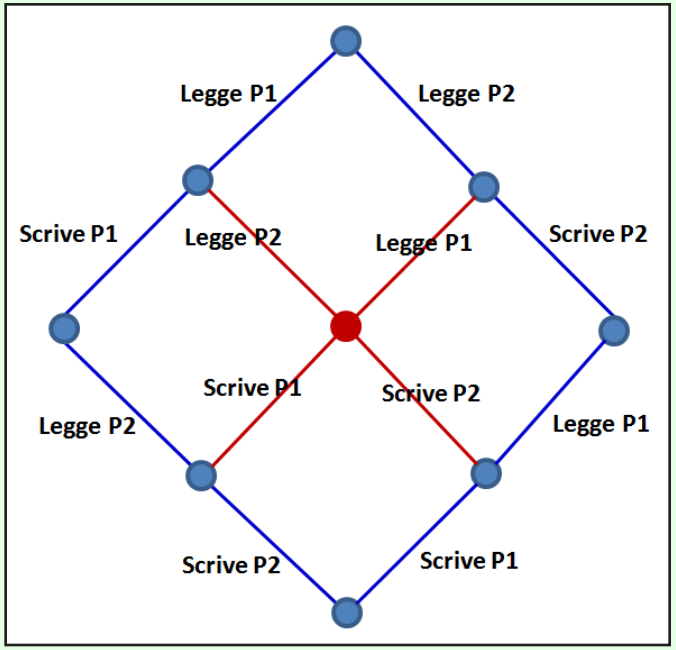


Al termine dell'esecuzione dei due processi ***totale*** 1010 oppure 1015 oppure 1025 ?

Sezione critica

```
process P1                          process P2 {
  a1 = rand();                        a2 = rand();
  totale = totale + a1;               totale = totale + a2;
}                                   }
```

a1 10 — **totale** 1000 — **a2** 15



totale 1025 — *totale* 1015 — *totale* 1010 — *totale* 1025

Gestione sezione critica in C

```
CRITICAL_SECTION  mutex;          // si crea una struttura condivisa  di tipo CRITICAL_SECTION
.....
.....

InitializeCriticalSection (&mutex);
.....
.....

    EnterCriticalSection (&mutex);

      // inizio sezione critica
      ..........
      ..........
      ..........
      // fine sezione critica

    LeaveCriticalSection (&mutex);


.....
.....
DeleteCriticalSection (&mutex);
```

Accesso alle risorse condivise

Gestione sezione critica in C

Un **oggetto CRITICAL_SECTION** fornisce la sincronizzazione per l'accesso ad una sezione critica che può essere utilizzato solo dai **thread di un singolo processo**, a differenza di un oggetto mutex.
Oggetti mutex e semafori possono essere utilizzati anche in applicazioni a processo singolo, ma gli oggetti sezione critica consentono meccanismi più veloci e efficienti per la sincronizzazione e la mutua esclusione.
Un oggetto **sezione critica** può **essere di proprietà di un solo thread per volta**, e questo consente di proteggere una risorsa condivisa dall'accesso simultaneo di più thread.

Un oggetto **CRITICAL_SECTION** prima di essere utilizzato da un thread di un processo deve essere inizializzato.

```
void WINAPI  InitializeCriticalSection( _Out_  LPCRITICAL_SECTION lpCriticalSection );
```

Puntatore all'oggetto **sezione critica**.

Vengono riservate tutte le risorse di sistema che servono per gestire l'oggetto **CRITICAL_SECTION**.

Gestione sezione critica in C

Questa funzione serve ad abilitare l'**accesso mutuamente esclusivo a una risorsa condivisa**.

```
void WINAPI EnterCriticalSection ( _Inout_ LPCRITICAL_SECTION lpCriticalSection );
```

```
BOOL WINAPI TryEnterCriticalSection ( _Inout_ LPCRITICAL_SECTION lpCriticalSection );
```

Puntatore all'oggetto **sezione critica**.

Ogni thread chiama la funzione **EnterCriticalSection (…)** o **TryEnterCriticalSection (…)** per richiedere di **avere la proprietà esclusiva** della sezione critica prima di eseguire qualsiasi porzione di codice che accede alle **risorse protette**.

La differenza tra le due funzioni è la seguente:

- **EnterCriticalSection(…)** blocca fino a quando il thread non ottiene la proprietà della sezione critica.
- **TryEnterCriticalSection(…)** termina immediatamente, indipendentemente dal fatto che abbia ottenuto la proprietà della sezione critica oppure no e restituisce: 0 se un altro thread possiede già il diritto esclusivo alla sezione critica, altrimenti restituisce un valore diverso da 0;

```
void WINAPI LeaveCriticalSection ( _Inout_ LPCRITICAL_SECTION lpCriticalSection );
```

Quando ha finito di eseguire il codice protetto, il thread utilizza la funzione **LeaveCriticalSection(…)** per **cedere la proprietà**, consentendo ad un altro thread di diventarne proprietario e accedere alla risorsa protetta.

Non vi è alcuna garanzia circa **l'ordine** in cui i thread in attesa acquisiranno la proprietà della sezione critica.

```
void WINAPI DeleteCriticalSection( _Inout_ LPCRITICAL_SECTION lpCriticalSection );
```

Ogni thread del processo utilizza la funzione **DeleteCriticalSection** (…) per liberare le risorse di sistema che sono state assegnate quando l'oggetto sezione critica è stato inizializzato.

Dopo che questa funzione è stata richiamata, l'oggetto sezione critica non può più essere utilizzato per la sincronizzazione.

TCP concorrente in C e sezione critica

Phonebook.c (rif.: esempio pagg. 102-109 libro di testo)

Realizzazione di una rubrica di numeri telefonici interni di una grande azienda, da rendere disponibile come servizio di rete al quale è possibile accedere per richiedere un numero a partire da un nominativo, o viceversa il nominativo a partire dal numero. È prevista la possibilità di aggiungere una nuova voce della rubrica, o di modificarne una già esistente.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <process.h>
#include <winsock2.h>


struct ENTRY   // struttura di una singola voce della rubrica
{
 char name[32];
 char phone[16];
};

struct PHONEBOOK
{
 int N_entry;                    // numero di voci presenti in rubrica
 struct ENTRY phonebook[1000];   // Tabella di max 1000 voci che viene caricata
                                 // con i dati  presenti nel file rubrica.csv
};


struct PHONEBOOK shared;         // struttura globale condivisa per rubrica
CRITICAL_SECTION mutex;          // oggetto mutex di tipo CRITICAL_SECTION
                                 // per l'accesso esclusivo alla rubrica
```

shared

shared.N_entry

shared.phonebook [ ]

name	phone
Rossi	0187450093
Verdi	34878867758
....	....
....	....

...

TCP concorrente in C e sezione critica

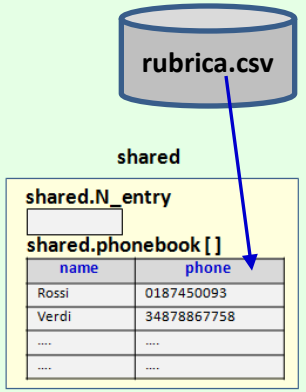
. . . Phonebook.c - Programma main(…)

```
int main(int argc, char* argv[])
{
 WSADATA wsaData;
 SOCKET request_socket_id; // socket richieste di connessione
 SOCKET communication_socket_id; // socket comunicazione con client
 struct sockaddr_in server_add; // struttura per indirizzo server
 struct sockaddr_in client_add; // struttura per indirizzo client
 int port; // numero di porta server
 size_t client_add_size;

 if (argc != 2)
   {
   printf("Errore argomenti!\r\n");
   return -1;
   }
 port = atoi(argv[1]);
 if (WSAStartup(0x0202, &wsaData) != 0)
   {
   printf("Errore inizializzazione WinSock DLL!\r\n");
   return -1;
   }

 InitializeCriticalSection(&mutex);

 // caricamento rubrica da file
 if (loadFromFile("rubrica.csv") < 0)
   {
   printf("Errore caricamento file!\r\n");
   WSACleanup();
   DeleteCriticalSection(&mutex);
   return -1;
   }
```

// riserva le risorse di sistema per l'oggetto mutex (sezione critica)

funzione **loadFromFile(…)**

. . .



TCP concorrente in C e sezione critica

. . . Phonebook.c - Programma main(…)

```
// apertura del socket
if ( (request_socket_id = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == INVALID_SOCKET)
  {
  printf("Errore apertura socket!\r\n");
  WSACleanup();
  DeleteCriticalSection(&mutex);
  return -1;
  }

memset(&server_add, 0, sizeof(server_add));
server_add.sin_family = AF_INET;
server_add.sin_addr.s_addr = 0; // local host
server_add.sin_port = htons((short)(port));

if (bind( request_socket_id, (struct sockaddr *)&server_add, sizeof(server_add)) == SOCKET_ERROR)
  {
  printf("Errore associazione socket!\r\n");
  closesocket(request_socket_id);
  WSACleanup();
  DeleteCriticalSection(&mutex);
  return -1;
  }

// impostazione del socket per ricevere le richieste di connessione
if (listen(request_socket_id, 1) == SOCKET_ERROR)
  {
  printf("Errore impostazione socket!\r\n");
  closesocket(request_socket_id);
  WSACleanup();
  DeleteCriticalSection(&mutex);
  return -1;
  }
printf("Servizio attivo...\r\n");
```

. . .

TCP concorrente in C e sezione critica

. . . Phonebook.c - Programma main(…)

```
while (1)
  {
  client_add_size = sizeof(client_add);
  communication_socket_id = accept(request_socket_id, (struct sockaddr*)&client_add, &client_add_size);
  if (communication_socket_id != INVALID_SOCKET)
     {
     printf("Ricevuta richiesta di connessione.\r\n");
     // creazione thread di comunicazione con il client
     CreateThread(NULL, 8192, &client_service, &communication_socket_id, 0, NULL);
     }
  }
closesocket(request_socket_id);
WSACleanup();
DeleteCriticalSection(&mutex);
return 0;
}
```

// libera le risorse di sistema che sono state assegnate quando l'oggetto sezione critica è stato inizializzato.

TCP concorrente in C e sezione critica

. . . Phonebook.c - Programma client_service(…)

```
// funzione di gestione della comunicazione con il client
unsigned long WINAPI   client_service (void* arg)
{
  char *request, *name, *phone, *name_or_phone;
  unsigned char buffer[1024];
  char command[64], answer[64];
  int index, error;
  int i, n;
  SOCKET socket_id   =  *((SOCKET*) (arg));
  index = 0;

  while (1)
      {
        if ((n = recv(socket_id, (void *)buffer, sizeof(buffer), 0))   <= 0 )
             {
              // chiusura della connessione da parte del client
              printf("Chiusura connessione.\r\n");
              break;
             }
        else
           {
              .......
              .......
           }
      }
  closesocket(socket_id);
  ExitThread(0);
}
```

. . .

TCP concorrente in C e sezione critica

. . . Phonebook.c - Programma client_service(…)

```
else
 {
   // ricezione caratteri
  for (i=0; i<n; i++)
    if (buffer[i] == '\r' || buffer[i] == '\n' || buffer[i] == '\0')
      {
        // ricezione completata
        command[index] = '\0';
        if (strlen(command) > 0)
          {
            error = 1;
            request = strtok(command, ",\r\n\0");
            if (request != NULL)
              {
                // confronta  se la prima stringa ricevuta nella riga
                // di comando è GET o SET
                if (strcmp(request, "GET") == 0)
                  {
                    // commando GET
                    name_or_phone = strtok(NULL, ",\r\n\0");
                    if (name_or_phone != NULL)
                      {
                        if ( find (name_or_phone, answer) > 0)
                          {
                            // invio risposta al client
                            strcat(answer, "\r\n");
                            send( socket_id, answer, strlen(answer), 0);
                            error = 0;
                          }
                      }
                  }
        ……
        ……  //comando SET
      }
    else
      {
        // copia del carattere nel  buffer command
        if (index >= sizeof(command))
        index = 0;
        command[index] = buffer[i];
        index++;
      }
 }
```

funzione find(…)

. . .

TCP concorrente in C e sezione critica

. . . Phonebook.c - Programma client_service(…)

```
if (strlen(command) > 0)
  {
   error = 1;
   request = strtok(command, ",\r\n\0");
   if (request != NULL)
       {
          ……..    //comando GET

          if (strcmp(request, "SET") == 0)
           {
            // commando SET
            name = strtok(NULL, ",\r\n\0");
            if (name != NULL)
                {
                 phone = strtok(NULL, ",\r\n\0");
                 if (phone != NULL)
                     {
                        if (strlen(name)>0 && strlen(phone)>0)
                             {
                                 if ( update(name, phone) > 0)
                                     {
                                       // invio risposta al client
                                       send(socket_id, "OK\r\n", 4, 0);
                                       // salvataggio su file
                                       saveToFile("rubrica.csv");
                                       error = 0;
                                     }
                             }
                     }
                }
           }
       }
  }
if (error)     // invio messaggio di errore al client   perché non è stato ricevuto  né  GET  né  SET
    send (socket_id, "ERROR\r\n", 7, 0);
index = 0;
break;
```

funzione update(…)

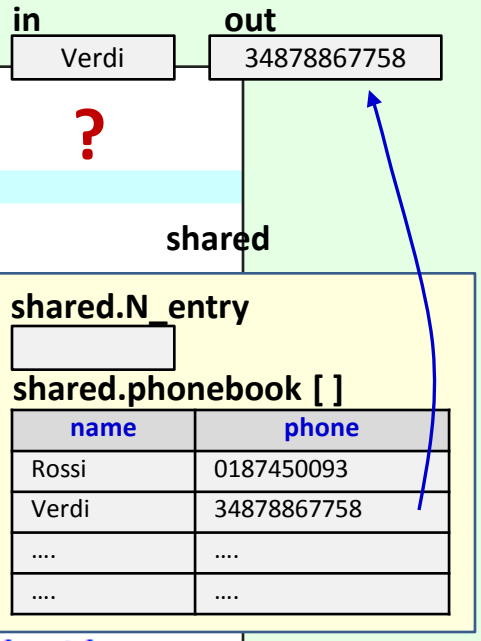
funzione saveToFile(…)

TCP concorrente in C e sezione critica

. . . Phonebook.c - funzione find(…)

```
// ricerca di un nominativo o di un numero nella rubrica
// fornisce il corrispondente numero o nominativo
// restituisce 0 in caso di ricerca fallita, 1 altrimenti
int find(char in[], char *out)
{
 int i, n;

 EnterCriticalSection(&mutex);    // Inizio sezione critica

 n = shared.N_entry;

 LeaveCriticalSection(&mutex);    // Uscita dalla sezione critica

 for (i=0; i<n; i++)
   {
   EnterCriticalSection(&mutex); // Inizio sezione critica

   if (strcmp(in, shared.phonebook[i].name) == 0)  // TROVATO nominativo
      {
      strncpy(out, shared.phonebook[i].phone, 15);
      LeaveCriticalSection(&mutex);   // Uscita dalla sezione critica
      return 1;
      }
   if (strcmp(in, shared.phonebook[i].phone) == 0) // TROVATO num.telefonico
      {
      strncpy(out, shared.phonebook[i].name, 31);
      LeaveCriticalSection(&mutex);   // Uscita dalla sezione critica
      return 1;
      }

   LeaveCriticalSection(&mutex);  // Uscita dalla sezione critica
   }
 return 0;
}
```

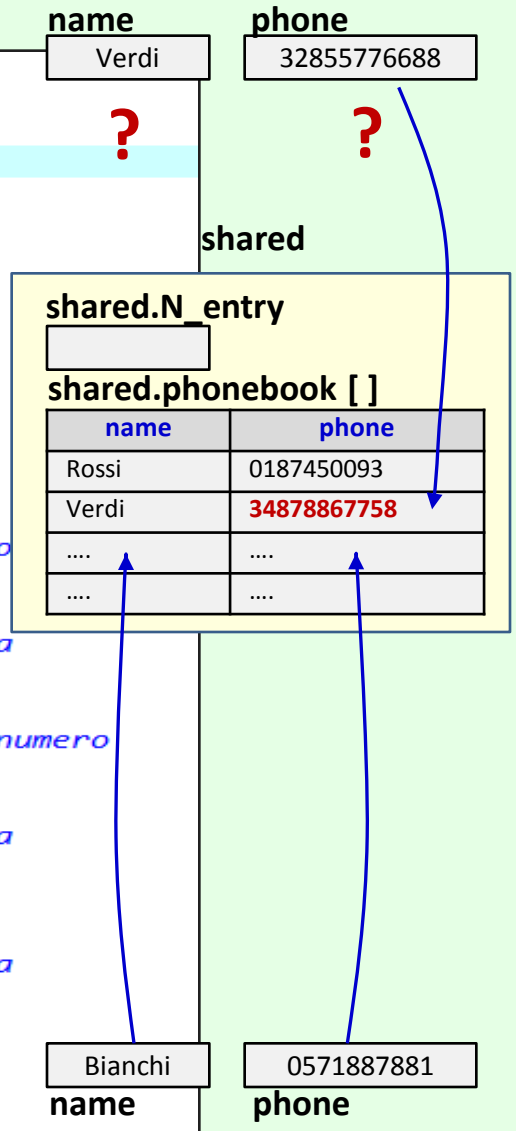
. . .



TCP concorrente in C e sezione critica

. . . Phonebook.c - funzione update(...)

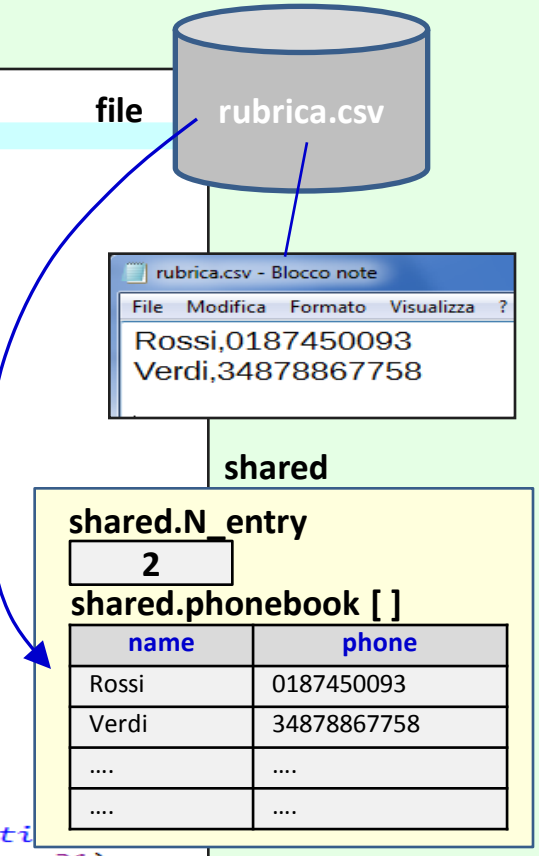
```
// aggiornamento del nominativo o numero di una voce della rubrica
// aggiunge una voce alla rubrica se non esistono nominativo/numero
// restituisce 1 in caso di aggiornamento, 2 in caso di aggiunta
int update(char name[], char phone[])
{
 int i, n;

 EnterCriticalSection(&mutex);  // Inizio sezione critica
 n = shared.N_entry;
 LeaveCriticalSection(&mutex);  // Uscita dalla sezione critica

 for (i=0; i<n; i++)
   {
     EnterCriticalSection(&mutex);  // Inizio sezione critica

     if (strcmp(name, shared.phonebook[i].name) == 0)  // TROVATO no
      {
      strncpy(shared.phonebook[i].phone, phone, 15);
      LeaveCriticalSection(&mutex);  // Uscita dalla sezione critica
      return 1;
      }
     if (strcmp(phone, shared.phonebook[i].phone) == 0) // TROVATO numero
      {
      strncpy(shared.phonebook[i].name, name, 31);
      LeaveCriticalSection(&mutex);  // Uscita dalla sezione critica
      return 1;
      }

    LeaveCriticalSection(&mutex);     // Uscita dalla sezione critica
   }
 // aggiunge una voce in rubrica
 EnterCriticalSection(&mutex);
  strncpy(shared.phonebook[i].name, name, 31);
  strncpy(shared.phonebook[i].phone, phone, 15);
  shared.N_entry++;
 LeaveCriticalSection(&mutex);
 return 2;
}
```



TCP concorrente in C e sezione critica

. . . Phonebook.c - funzione loadFromFile(...)

```
// carica le voci della rubrica dal file specificato
// restituisce -1 in caso di errore, il numero di voci altrimenti
int loadFromFile(char filename[])
{
 FILE* file;
 char line[1024];
 char *name, *phone;
 int n;

 file = fopen(filename, "r");
 if (file == NULL)
   return -1;

 EnterCriticalSection(&mutex);    // Inizio sezione critica
   shared.N_entry = 0;
 LeaveCriticalSection(&mutex);    // Uscita dalla sezione critica

 do {
   fgets(line, sizeof(line)-1, file);
   name = strtok(line, ",\r\n\0");
   if (name != NULL)
      {
      if (strlen(name) > 0)
         {
         phone = strtok(NULL, ",\r\n\0");
         if (phone != NULL)
            if (strlen(phone) > 0)
               {
               EnterCriticalSection(&mutex);  // Inizio sezione criti
                 strncpy(shared.phonebook[shared.N_entry].name, name, 31);
                 strncpy(shared.phonebook[shared.N_entry].phone, phone, 15);
                 shared.N_entry++;
               LeaveCriticalSection(&mutex);  // Uscita dalla sezione critica
               }
         }
      }
   } while (!feof(file));
 fclose(file);
 EnterCriticalSection(&mutex);  // Inizio sezione critica
   n = shared.N_entry;
 LeaveCriticalSection(&mutex);  // Uscita dalla sezione critica
 return n;
}
```



TCP concorrente in C e sezione critica

... Phonebook.c - funzione saveToFile(...)

```
// salva le voci della rubrica nel file specificato
// restituisce -1 in caso di errore, il numero di voci altrimenti
int saveToFile(char filename[])
{
 FILE* file;
 int i, n;

 file = fopen(filename, "w");
 if (file == NULL)
   return -1;

 EnterCriticalSection(&mutex);  // Inizio sezione critica
   n = shared.N_entry;
 LeaveCriticalSection(&mutex);  // Uscita dalla sezione critica

 for (i=0; i<n; i++)
   {
   EnterCriticalSection(&mutex);  // Inizio sezione critica
     fprintf( file, "%s,%s\r\n", shared.phonebook[i].name, shared.phonebook[i].phone);
   LeaveCriticalSection(&mutex);  // Uscita dalla sezione critica
   }
 fclose(file);
 return n;
}
```