Lectia 15. RUTINELE DE BIBILIOTECĂ RUN-TIME (RUN-TIME LIBRARY ROUTINES) ŞI VARIABILE DE MEDIU (ENVIRONMENT VARIABLES)

15.1 Privire generală:

Standardul OpenMP defineste o interfată API pentru apeluri la bibiliotecă care execută urmatoarea varietate de functii:

- Află prin chestionare numărul de fire/procesoare, stabileste numărul de fire utilizate.
- Blocări de domeniu general prin rutine adecvate (semafoare).
- Rutine portabile pentru măsurarea timpului universal (wall clock time).
- Setarea funcțiilor de mediu de excuție: paralelism unul-în-altul, ajustarea dinamică a firelor.

Pentru C/C++ poate fi necesară specificarea fișierului de incluziune "omp.h".

Pentru functiile/rutinele de închidere (Lock):

- Variabiele de tip **`lock`** trebuie să fie accesate numai prin rutinele de închidere.
- Pentru C/C++, variabila de tip **`lock`** trebuie să aibă tipul **`omp_lock_t`** sau tipul **`omp_nest_lock_t`**, în functie de modul de utilizare.

Note de implemetare:

- Implementarea poate să suporte sau poate să nu suporte paralelismul unul-înaltul și/sau firele dinamice. Dacă paralelismul unul-în-altul este suportat, este adesea numai nominal prin acea că o regiune paralela una-în-alta poate avea numai un fir.
- Documentația implementării trebuie consultată pentru aceste detalii –sau se pot experimenta unele aspecte pentru a afla ceea ce nu este scris explicit în documentatie.

15.2 Rutine utilizate pentru setarea şi returnarea numarului de fire

15.2.1 OMP_SET_NUM_THREADS

Scop: Setează numărul de fire care vor fi utilizate în regiunea paralela. Trebuie să fie un întreg pozitiv.

Formatul în C/C++:

```
#include <omp.h>
void omp_set_num_threads(int num_threads)
```

Note şi restrictii: Mecanismul firelor dinamice modifică efectul acestei rutine.

Enabled: specifică numărul maxim de fire care pot fi utilizate pentru orice regiune paralelă prin mecanismul firelor dinamice.

Disabled: specifică numărul exact de fire de utilizat până la apelul următor la această rutină.

Această rutină poate fi apelată numai din porțiunile secventiale ale codului.

Acest apel are precedentă fată de variabila de mediu **`OMP_NUM_THREADS`**.

15.2.2 OMP_GET_NUM_THREADS

Scop: returnează numărul de fire care sunt în fasciculul curent şi execută regiunea pralela din care este apelată.

Format în C/C++:

```
#include <omp.h>
int omp_get_num_threads(void)
```

Note şi restrictii: Dacă acest apel este făcut dintr-o portiune secventială a programului sau dintr-o regiune paralela una-în-alta care este serializată, returul este 1. Numărul prin default al firelor este dependent de implementare.

Exemplu 15.2.1. *În acest exemlu se ilustrează modul de utilizare a rutinelor*

a) ***omp_get_num_threads()****-returnează numărul de fire care sunt în fasciculul curent*

b) ***omp_set_num_threads()-s****etează numărul de fire care vor fi utilizate în regiunea paralela.*

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#ifdef _OPENMP
 #include <omp.h>
 #define TRUE  1
 #define FALSE 0
#else
 #define omp_get_thread_num() 0
 #define omp_get_num_threads() 1
#endif
int main()
{
  int TID;
#ifdef _OPENMP
  (void) omp_set_dynamic(1);
  if (omp_get_dynamic()) {printf("Warning: dynamic adjustment of threads has been
     set\n");}
 //  (void) omp_set_num_threads(4);
#endif

  for (int n=5; n<11; n+=5)
  {
#pragma omp parallel if (n > 5) num_threads(n) default(none) \
      private(TID) shared(n)
  {
    TID = omp_get_thread_num();
#pragma omp single
    {
      printf("Value of n = %d\n",n);
      printf("Number of threads în parallel region: %d\n", omp_get_num_threads());
    }
sleep(omp_get_thread_num());
    printf("Print statement executed by thread %d\n",TID);
  }  /*-- End of parallel region --*/
  }

  return(0);
}
```

Rezultatele vor fi urmatoarele:

a) Cazul cand (void) omp_set_dynamic(0)- desabilitează ajustarea dinamică (de sistemul de execuție) a numărului de fire disponibile pentru executarea regiunilor paralele.

```
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpiCC -fopenmp -o Exemplu3.2.1.exe
    Exemplu3.2.1.cpp
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-
    0-1 Exemplu3.2.1.exe
Value of n = 5
```

```
Number of threads în parallel region: 1
Print statement executed by thread 0
Value of n = 10
Number of threads în parallel region: 10
Print statement executed by thread 0
Print statement executed by thread 1
Print statement executed by thread 2
Print statement executed by thread 3
Print statement executed by thread 4
Print statement executed by thread 5
Print statement executed by thread 6
Print statement executed by thread 7
Print statement executed by thread 8
Print statement executed by thread 9
[Hancu_B_S@hpc Open_MP]$
```

b) Cazul cand (void) omp_set_dynamic(1)- abilitează ajustarea dinamică (de sistemul de execuție) a numărului de fire disponibile pentru executarea regiunilor paralele.

```
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpiCC -fopenmp -o Exemplu3.2.1.exe
    Exemplu3.2.1.cpp
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-
    0-1 Exemplu3.2.1.exe
Warning: dynamic adjustment of threads has been set
Value of n = 5
Number of threads în parallel region: 1
Print statement executed by thread 0
Value of n = 10
Number of threads în parallel region: 4
Print statement executed by thread 0
Print statement executed by thread 1
Print statement executed by thread 2
Print statement executed by thread 3
[Hancu_B_S@hpc Open_MP]$
```

15.2.3 OMP_GET_MAX_THREADS

Scop: returnează valoarea maximă care poate fi returnată de un apel la functia **omp_get_num_threads**.

Format în C/C++:

```
#include <omp.h>
int omp_get_max_threads(void)
```

Note şi restrictii: Reflectă în general numărul de fire setat de variabila de mediu OMP_NUM_THREADS sau de rutina omp_set_num_threads() din bibliotecă. Poate fi apelată atât din regiunile seriale ale codului cât şi din cele paralele.

15.2.4 OMP_GET_NUM_PROCS

Scop: returnează numărul de procesoare care sunt la dispozitia programului.

Format în C/C++:

```
#include <omp.h>
int omp_get_num_procs(void)
```

15.3 Rutina utilizata pentru returnarea identificatorul firului

15.3.1 OMP_GET_THREAD_NUM

Scop: returnează numărul de fir al firului, în interiorul fasciculului, prin acest apel. Numărul acesta va fi între 0 şi **omp_get_num_threads-1**. Firul master din fascicul este firul 0.

Format în C/C++:

```
#include <omp.h>
int omp_get_thread_num(void)
```

Note şi restrictii: dacă este apelată dintr-o regiune paralel una-în-alta, functia aceasta returnează 0.

Exemple de determinare a numărului de fire dintr-o regiune paralela:

Exemplul 1 este modul corect de a determina identificatorul de fir într-o regiune paralela.

Exemplul 2 este incorect – variabila **TID** trebuie să fie **private**.

Exemplul 3 este incorect – apelul **omp_get_thread_num** este în afara unei regiuni paralele.

Exemplul 1:

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
int TID;
#pragma omp parallel private(TID)
{
TID=omp_get_thread_num();
printf("Hello from thread number %d\n", TID);
}
}
```

Exemplul 2

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
int TID;
#pragma omp parallel
{
TID=omp_get_thread_num();
printf("Hello from thread number %d\n", TID);
}
}
```

Exemplul 3:

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
int TID;
TID=omp_get_thread_num();
printf("Hello from thread number %d\n", TID);
#pragma omp parallel
    {
    }
}
```

15.4 Rutinele utilizate pentru generarea dinamica a firelor

15.4.1 OMP_IN_PARALLEL

Scop: poate fi apelată pentru a determina dacă secțiunea de cod în execuţie este paralelă sau nu.

Formatul în C/C++:

```
#include <omp.h>
int omp_în_parallel(void)
```

Note şi restrictii: În C/C++ ea returnează un întreg nenul dacă apelul este dintr-o zonă paralela, zero în caz contrar .

Exemplu 15.4.1. *În acest exemlu se ilustrează modul de utilizare a rutinei* ***omp_în_parallel***

```
#include <stdio.h>
#include <omp.h>
void mode(void)
{
if(omp_în_parallel()) printf("Se executa instructiuni din regiunea paralela\n");
else printf("Se executa instructiuni din regiunea segventiala\n");
}
int main(int argc, char *argv[])
{
mode();
  #pragma omp parallel
  {
#pragma omp master
{
mode();
}
  }
}
```

Rezultatele vor fi urmatoarele:

```
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpiCC -fopenmp -o Exemplu3.4.1.exe
    Exemplu3.4.1.cpp
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-
    0-1 Exemplu3.4.1.exe
Se executa instructiuni din regiunea segventiala
Se executa instructiuni din regiunea paralela
[Hancu_B_S@hpc Open_MP]$
```

15.4.2 OMP_SET_DYNAMIC

Scop: abilitează sau desabilitează ajustarea dinamică (de sistemul de execuţie) a numărului de fire disponibile pentru executarea regiunilor paralele.

Formatul în C/C++:

```
#include <omp.h>
void omp_set_dynamic(int dynamic_threads)
```

Note şi restrictii: Pentru C/C++, dacă argumentul **dynamic_threads** este nenul, atunci mecanismul este abilitat, altminteri este desabilitat. Subrutina **omp_set_dymamic** are întâietate fată de variabila de mediu **OMP_DYNAMIC**. Setarea prin default depinde de implementare. Poate fi apelată dintr-o secțiune serială/secvențială a programului.

15.4.3 OMP_GET_DYNAMIC

Scop: determinarea stării abilitat/desabilitat a modificării dinamice a firelor.

Formatul în C/C++:

```
#include <omp.h>
int omp_get_dynamic(void)
```

Note şi restrictii: Pentru C/C++, rezultatul apelului este non-zero/zero pentru cele două situatii.

Exemplu 15.4.2. *Aceste exemplu ilustriaza modalitatea de utilizare a rutinelor* **`omp_set_dynamic()`** *şi* **`omp_get_dynamic()`**

```
#include <stdio.h>
 #include <omp.h>
 int main(int argc, char *argv[])
 {
   printf("Valoarea  initiala (prestabilita) a variabilei de mediu OMP_DYNAMIC: %d\n",
      omp_get_dynamic());
   omp_set_dynamic(0);
   printf("Valoarea variabilei de mediu dupa executarea rutinei omp_set_dynamic()
      OMP_DYNAMIC : %d\n", omp_get_dynamic());
 #pragma omp parallel num_threads(128)
   {
 #pragma omp master
     {
       printf("Regiunea paralela contine, %d fire\n",
            omp_get_num_threads());
     }
   }
 }
```

Rezultatele vor fi urmatoarele:

a) Cazul omp_set_dynamic(1):

```
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpiCC -fopenmp -o Exemplu3.4.2.exe
      Exemplu3.4.2.cpp
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-
      0-1 Exemplu3.4.2.exe
Valoarea  initiala (prestabilita) a variabilei de mediu OMP_DYNAMIC: 0
Valoarea variabilei de mediu dupa executarea rutinei omp_set_dynamic() OMP_DYNAMIC
      : 1
Regiunea paralela contine, 4 fire
```

a) Cazul omp_set_dynamic(0):

```
[Hancu_B_S@hpc Open_MP]$
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpiCC -fopenmp -o Exemplu3.4.2.exe
      Exemplu3.4.2.cpp
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-
      0-1 Exemplu3.4.2.exe
Valoarea  initiala (prestabilita) a variabilei de mediu OMP_DYNAMIC: 0
Valoarea variabilei de mediu dupa executarea rutinei omp_set_dynamic() OMP_DYNAMIC
      : 0
Regiunea paralela contine, 128 fire
[Hancu_B_S@hpc Open_MP]$
```

15.5 Rutinele utilizate pentru generarea paralelizmului unul-în-altul (nested parallelism)

15.5.1 OMP_SET_NESTED

Scop: abilitarea sau desabilitarea paralelismului unul-în-altul.

Formatul în C/C++:

```
#include <omp.h>
void omp_set_nested(int nested)
```

Note şi restrictii: Pentru C/C++, dacă variabila **`nested`** este nenulă, atunci paralelismul unul-înaltul este abilitat; dacă este nulă îl desabilitează. Defaultul este cu paralelismul unul-în-altul desabilitat. Apelul este mai tare ca precedentă fată de variabila de mediu **`OMP_NESTED`**.

15.5.2 OMP_GET_NESTED

Scop: determină dacă paralelismul unul-în-altul este abilitat sau nu.

Formatul în C/C++:

```
#include <omp.h>
int omp_get_nested (void)
```

Note şi restrictii: În C/C++, se returnează o valoare nenulă dacă paralelismul unul-în-altul este abilitat şi zero în caz contrar.

Exemplu 15.5.1. *Aceste exemplu ilustriaza modalitatea de utilizare a rutinelor* **`omp_get_nested()`** *şi* **`omp_set_nested()`**.

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#ifdef _OPENMP
 #include <omp.h>
 #define TRUE  1
 #define FALSE 0
#else
 #define omp_get_thread_num() 0
 #define omp_get_num_threads() 1
 #define omp_get_nested() 0
#endif
int main()
{
int id;
#ifdef _OPENMP
  (void) omp_set_dynamic(FALSE);
  if (omp_get_dynamic()) {printf("Warning: dynamic adjustment of threads has been
     set\n");}
  (void) omp_set_num_threads(3);
  (void) omp_set_nested(1);
  if (! omp_get_nested()) {printf("Warning: nested parallelism not set\n");}
#endif
  printf("Nested parallelism is %s\n",
         omp_get_nested() ? "supported" : "not supported");
#pragma omp parallel private(id)
  {
   id=omp_get_thread_num();
   #pragma omp parallel num_threads(2)
      {
        sleep(id);
       printf(" Thread %d from outer parallel region executes the thread %d from inner
     parallel region\n",id,omp_get_thread_num());
      } //End of inner parallel region
  }  // End of outer parallel region -
  return(0);
}
```

Rezultatele vor fi urmatoarele:

a) Cazul (void) omp_set_nested(1):

```
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpiCC -fopenmp -o Exemplu3.5.1.exe
    Exemplu3.5.1.cpp
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-
    0-1 Exemplu3.5.1.exe
Nested parallelism is supported
Thread 0 from outer parallel region executes the thread 0 from inner parallel region
 Thread 0 from outer parallel region executes the thread 1 from inner parallel region
 Thread 1 from outer parallel region executes the thread 1 from inner parallel region
 Thread 1 from outer parallel region executes the thread 0 from inner parallel region
 Thread 2 from outer parallel region executes the thread 0 from inner parallel region
 Thread 2 from outer parallel region executes the thread 1 from inner parallel region
 [Hancu_B_S@hpc Open_MP]$
```

2) Cazul (void) omp_set_nested(0):

```
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpiCC -fopenmp -o Exemplu3.5.1.exe
    Exemplu3.5.1.cpp
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-
    0-1 Exemplu3.5.1.exe
Warning: nested parallelism not set
Thread 0 from outer parallel region executes the thread 0 from inner parallel region
Thread 1 from outer parallel region executes the thread 0 from inner parallel region
Thread 2 from outer parallel region executes the thread 0 from inner parallel region
[Hancu_B_S@hpc Open_MP]$
```

Astfel, fiecare regiune paralela "exterioara" din numarul total de 3 fire "genereaza" la randul sau 2 fire care executa regiuni paralele "interioare".

15.6 Rutinele utilizate pentru blocări de domeniu a firelor

15.6.1 OMP_INIT_LOCK

Scopul: subrutina initializează un lacăt asociat cu variabila de tip **`lock.`**

Formatul în C/C++:

```
#include <omp.h>
void omp_init_lock(omp_lock_t *lock)
void omp_init_nest_lock(omp_nest_lock_t *lock)
```

*Note şi restrictii:*Starea initială este unlocked.

15.6.2 OMP_DESTROY_LOCK

Scop: această subrutină disociază o variabilă lacăt de orice lacăt.

Formatul în C/C++:

```
#include <omp.h>
void omp_destroy_lock(omp_lock_t *lock)
void omp_destroy_nest_lock(omp_nest_lock_t *lock)
```

Note şi restrictii: Este nepermis a apela această subrutină cu o variabilă lock care nu este initializată.

15.6.3 OMP_SET_LOCK

Scop: această subrutină obligă firul executant să astepte până când un lacăt specificat este disponibil. Un fir este proprietar al unui lock când acesta devine disponibil.

Formatul în C/C++:

```
#include <omp.h>
void omp_set_lock(omp_lock_t *lock)
void omp_set_nest_lock(omp_nest_lock_t *lock)
```

Note şi restrictii: Este nepermis a apela această rutină cu o variabilă lock neinitializată.

15.6.4 OMP_UNSET_LOCK

Scop: această subrutină elibereaza(descuie) un **lock** dintr-o subrutină în execuţie.

Formatul în C/C++:

```
#include <omp.h>
void omp_unset_lock(omp_lock_t *lock)
void omp_unset_nest_lock(omp_nest_lock_t *lock)
```

Note şi restrictii: Nu este permis a se apela această subrutină cu o variabilă lock e initializată.

Exemplu 15.6.1. *Aceste exemplu ilustriaza modalitatea de utilizare a rutinelor* **omp_unset_lock(),omp_set_lock(),omp_destroy_lock(),omp_init_lock.**

```
#include <stdio.h>
#include <omp.h>
#include <iostream>
omp_lock_t lock,lock1;
int main(int argc, char *argv[])
{
int n;
omp_init_lock(&lock);
omp_init_lock(&lock1);
omp_set_num_threads(3);
#pragma omp parallel private (n)
    {
omp_set_lock(&lock1);
#pragma omp master
{
printf("Se genereaza %d procese OpenMP (fire)\n", omp_get_num_threads());
}
omp_unset_lock(&lock1);
    n=omp_get_thread_num();
    omp_set_lock(&lock);
printf("================\n");
    printf("Inceputul sectiei inchise, firul %d\n", n);
    printf("Sfarsitul sectiei inchise, firul %d\n", n);
printf("================\n");
       printf("\n");
    omp_unset_lock(&lock);
    }
omp_destroy_lock(&lock1);
omp_destroy_lock(&lock);
}
```

Rezultatele vor fi urmatoarele:

```
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpiCC -fopenmp -o Exemplu3.6.2.exe
    Exemplu3.6.2.cpp
[Hancu_B_S@hpc Open_MP]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-
    0-1 Exemplu3.6.2.exe
Se genereaza 3 procese OpenMP (fire)
================
Inceputul sectiei inchise, firul 0
Sfarsitul sectiei inchise, firul 0
================
```

```
================
Inceputul sectiei inchise, firul 2
Sfarsitul sectiei inchise, firul 2
================

================
Inceputul sectiei inchise, firul 1
Sfarsitul sectiei inchise, firul 1
================

[Hancu_B_S@hpc Open_MP]$
```

Astfel utilizand aceste rutine se poate "sincroniza,, procesul de scoatere la tipar: un fir începe să tipărească numai după ce un alt fir a terminat să tipărească.

15.6.5 OMP_TEST_LOCK

Scop: această subrutină încearcă a seta un **lock** dar nu blochează dacă lacătul este indisponibil.

Formatul în C/C++:

```
#include <omp.h>
int omp_test_lock(omp_lock_t *lock)
int omp_test_nest_lock(omp_nest_lock_t *lock)
```

*Note şi restricţii :*Pentru C/C++ este restituită o valoare nenulă dacă lacătul a fost setat cu succes, altminteri este restituit zero. Este interzis a apela această subrutină cu o variabilă lock neinitializată.

15.7 Rutine portabile pentru măsurarea timpului universal (wall clock time

15.7.1 OMP_GET_WTIME

Scop: furnizează o rutină portabilă de temporizare în timp universal (wall clock). Returnează o valoare în dublă precizie egală cu numărul de secunde scurse de la un anumit punct în trecut. Uzual, este folosită în pereche cu valoarea de la primul apel scăzută din valoarea apelului al doilea pentru a obtine timpul scurs pentru un bloc de cod. Proiectată a da timpii "pe fir" şi de aceea nu poate fi consistentă global pe toate firele unui fascicul; depinde de ceea ce face un fir comparativ cu alte fire.

Formatul în C/C++:

```
#include <omp.h>
double omp_get_wtime(void)
```

Note şi restrictii: Necesită suportul versiunii OpenMP 2.0.

15.7.2 OMP_GET_WTICK

Scop: furnizează o rutină portabilă de temporizare în timp universal (wall clock). Returnează o valoare în dublă precizie egală cu numărul de secunde între două tic-uri succesive ale ceasului.

Formatul în C/C++:

```
#include <omp.h>
double omp_get_wtick(void)
```

Note şi restrictii: Necesită suportul versiunii OpenMP 2.0. Variable de mediu (de programare) OpenMP furnizează patru variabile de mediu pentru controlul executiei unui cod paralel. Toate numele variabilelor de mediu sunt scrise cu majuscule. Valorile atribuite lor nu sunt sensibile la upper/lower case.

15.8 Variable de mediu (de programare)

OpenMP furnizează patru variabile de mediu pentru controlul executiei unui cod paralel. Toate numele variabilelor de mediu sunt scrise cu majuscule. Valorile atribuite lor nu sunt sensibile la upper/lower case.

15.8.1 OMP_SCHEDULE

Se aplică numai la directivele for, parallel for (C/C++) care au clauzele lor schedule setată în timpul executiei (runtime). Valoarea acestei variabile determină modul cum sunt planificate iteratiile buclei pe procesoare. De exemplu:

```
setenv OMP_SCHEDULE "guided, 4"
setenv OMP_SCHEDULE "dynamic"
```

15.8.2 OMP_NUM_THREADS

Fixează numărul maxim de fire utilizate în timpul executiei. De exemplu:

```
setenv OMP_NUM_THREADS 8
```

15.8.3 OMP_DYNAMIC

Abilitează sau desabilitează ajustarea dinamică a numărului de fire disponibile pentru executarea unei regiuni paralel. Valorile valide sunt TRUE sau FALSE. De exemplu:

```
setenv OMP_DYNAMIC TRUE
```

15.8.4 OMP_NESTED

Abilitează sau desabilitează paralelismul unul-în-altul. Valorile valide sunt TRUE sau FALSE. De exemplu:

```
setenv OMP_NESTED TRUE
```

Note de implementare: O implementare sau alta poate să permită sau nu paralelismul unul-în-altul si/sau firele dinamice. Dacă paralelismul unul-în-altul este suportat, el este adesea numai nominal, în aceea că o regiune paralel una-în-alta poate avea numai un fir. Pentru detalii trebuie consultată documentatia implementării utilizate sau se poate experimenta pentru a afla ceea ce nu se poate găsi în documentatie.