**Rutine, subrutine si proceduri in cadrul sistemului de operare**

Lamba Andrei

Grupa 443A

**Introducere**

În programare, o subrutină (numită si procedura, metoda, funcţie, sau rutină) este o porţiune de cod din cadrul unui program mai mare, care îndeplineşte o sarcină specifica (task) şi este relativ independent de restul codului.  
 Dupa cum numele de "subprogram" sugerează, o subrutină se comporta aproximativ în acelaşi fel ca si un program, care este de fapt un pas într-un program mai mare sau intr-un alt subprogram. O subrutină este adesea codificata astfel încât să poată fi apelata de mai multe ori şi / sau din mai multe locuri in timpul unei singure executii a programului, inclusiv din alte subrutine , şi apoi sa fie trimisa inapoi (return) la instrucţiunea urmatoare după ce  apelul sarcinii subrutinei este incheiat.  
 Subrutinele reprezinta un instrument puternic de programare şi sintaxa multor limbaje de programare include suport pentru scrierea şi utilizarea lor. Folosirea rationala a subrutinelor (de exemplu, prin abordarea programarii structurate) deseori va reduce, în mod substanţial ,costurile dezvoltarii şi menţinerii unui program mare, crescând în acelaşi timp calitatea şi fiabilitatea. Subrutinele, adesea colectate în biblioteci, sunt un mecanism important pentru schimbul de informaţii şi pentru software-ul comercial. Disciplina programarii orientate pe obiect este bazata pe obiecte şi metode (subrutine care sunt anexate la aceste obiecte sau la clasele obiect).  
 În tehnica de compilare numita “cod filetat”(threaded code), programul executabil este ,de fapt, o secvenţă a apelurilor subrutinei.

**Concepte de baza**

Conţinutul unei subrutine este de fapt corpul său (body), partea de cod a programului care este executata atunci când subrutina este apelata.  
 O subrutină poate fi scrisa astfel incat sa se aştepte obţinerea a una sau mai multe valori de date dupa apelarea programului ( parametrii sai). Aceasta de asemenea poate întoarce o valoare calculata apelantului sau (valoare returnata), sau poate furniza diferite valori rezultate sau parametri de iesire(“output parameters”). Într-adevăr, o utilizare des intalnita a subrutinelor este de a implementa funcţii matematice, în care scopul lor principal este pur si simplu de a calcula unul sau mai multe rezultate ale căror valori sunt în întregime determinate de parametrii gasiti in subrutina.(Exemple: calculul logaritmului unui număr sau determinantul unei matrice).  
 Cu toate acestea, un apel de subrutină poate avea şi efecte adverse, cum ar fi modificarea structurilor de date în memoria calculatorului, citirea din sau scrierea intr-un dispozitiv periferic, crearea unui fişier, oprirea programului sau a masinii de calcul, sau chiar întârzierea executiei programului pentru un anumit timp. Un subprogram cu efecte adverse poate returna rezultate diferite de fiecare dată când este apelat, chiar dacă este apelat cu aceiaşi parametri. Utilizarea pe scară largă a subrutinelor cu efecte secundare este o caracteristică a limbajelor de programare obligatorii.  
 O subrutină pot fi codificata astfel încât să se poata apela recursiv, în vederea îndeplinirii sarcinilor sale. Aceasta tehnica permite punerea în aplicare directă a funcţiilor definite prin inducţie matematică şi prin algoritmi recursivi de tip “divide et impera”.  
 O subrutină al cărei scop este de a calcula o singură funcţie de valoare booleana(adevarat sau fals) este numita afirmatie(“predicate”).

**Avantaje**  
 Avantajele impartirii unui program în subrutine includ:

* reducerea copiilor de cod în cadrul unui program
* permiterea reutilizarii codului în programe multiple
* descompunerea unor sarcini complexe de programare în paşi mai simpli
* împărţirea unei sarcini mari de programare intre programatori diferiti, sau diferite etape ale unui proiect
* ascunderea detaliilor de implementare de utilizatorii subrutinei.

**Dezavantaje**

* Apelarea unei subrutine (mai degrabă decât folosind linii de cod) impune cheltuieli suplimentare de calcul în cadrul mecanismului de apel în sine
* Subrutina necesită de obicei cod standard de menaj - atât la intrare ,cat şi la ieşire,funcţia, salvand ,de obicei,registre de uz general şi returnand adresa ca cerinţă minimă)

**Procedurile stocate**

Reprezinta acele programe care se executa direct pe server-ul de baze de date ,eliberand astfel o mare parte din traficul in retea si folosind resursele ceva mai puternice ale serverului.Una dintre modificarile aduse de SQL99 a fost introducerea unui capitol destinat procedurilor stocate,denumit SQL/Persistent Stored Modules (SQL/PSM).Procedurile stocate asigura standardului SQL posibilitatea realizarii unei prelucrari procedurale complete.Aceste procedure stocate pot fi scrise in limbaje ce respecta mai mult sau mai putin standardul SQL (PL/SQL-Oracle,SQL Procedures language-DB2 IBM,Microsoft SQL Server) sau in alte limbaje de programare generalizate (Ada,C,C++,Fortran,Pascal sau Java).In orice limbaj ar fi implementate,ele contin pe langa comenzi de interogare sau manipulare a datelor si comenzi destinate controlului executiei,in asa fel incat cu ele sa poata fi implementata cea mai mare parte a logicii economice a aplicatiilor.

SQL PSM asigura constructori sintactici si semantici pentru specificarea modulelor stocate,procedurilor si functiilor.Rutinele stocate pot fi apelate din orice trazactie SQL din interiorul sau din exteriorul sistemului.

**Avantajele procedurilor stocate**

Unul dintre principalele avantaje ale procedurilor stocate este folosirea resurselor performante ale serverului.De obicei intr-o organizatie server-ele au cele mai performante configuratii,mult peste resursele hardware ale calculatoarelor personale.Prin folosirea procedurilor stocate,operatiile de prelucrare a datelor se realizeaza pe sisteme mult mai rapide decat statiile de lucru,ceea ce poate duce la micsorarea timpului de asteptare a sistemului de catre utilizator.

Fiecare client,atuci cand apeleaza procedure stocate,apeleaza ultima variant a acestora.astfel ca modificarea aplicatiilor client se face mau rar,ceea ce presupune o intretinere mai usoara a prelucrarilor.De multe ori organizatiile care au apelat la firme de software pentru dezvoltarea sistemelor informatice ofera posibilitatea conectarii acestora la propriile servere de baze de date pentru o intretinere mai rapida si pentru intelegerea corecta a diverselor situatii aparute.Aceste posibilitati de conectare pot fi folosite si pentru a modifica mai usor modul de prelucrare a datelor.

Procedurile stocate se evidentiaza si prin reutilizarea codului de catre mai multe aplicatii,evitandu-se astfel duplicarea scrierii codului si toate dezavantajele ce decurg din acest lucru.Interfata se paote schimba usor daca o mare parte din logica aplicatiei este inclusa in procedurile stocate.Trecerea de la o arhotectura traditional client-server la o arhitectura pe n niveluri se realizeaza mult mai simplu daca in prima versiune a sistemului s-au folosit procedure stocate.

Prelucrarea clasica a datelor intr-o arhitectura client-server presupune transferul datelor de intrare in aplicatia client,prelucrarea acestora si transferul rezultatelor inapoi la server-ul de baze de date.Toata aceasta activitate implica folosirea din plin a resurselor retelei.Prin prelucrarea datelor acolo unde sunt si stocate se diminueaza considerabil gradul de folosire a resurselor retelei.

Procedurile stocate extind capacitatea de interogare si de manipulare a datelor unui SGBD.De exemplu.o functie stocata poate fi folosita in cadrul unei fraze SQL,in conditiile in care,in toate implementarile,apelul unei functii sau procedure stocate este foarte simplu.

Procedurile stocate ofera prin intermediul declansatorilor o posibilitate de implementare a unei politici de acces ,controlandu-se cine,cand,de unde si cu ce se efectueaza modificarile.

In plus,procedurile stocate ,scrise in SQL/PSM prezinta un avantaj in faptul ca sunt usor de inteles de programatorii neeexperimentati datorita simplitatii sintaxei.SQL/PSM a fost proiectat in asa fel incat implementarile sa poata suporta supraincarcarea procedurilor si a functiilor.In acest fel pot exista procedure si/sau functii cu acelasi nume,dar cu semnaturi diferite(care sa aiba parametric diferiti ca si numar si tip).

Toate tipurile de date din SQL/Foundation se regasesc si in SQL/PSM,ceea ce elimina o parte din problemele de trasformare intre diferite tipuri de date.In procedurile stocate scrise in SQL/PSM se pot folosi si tipuri de date definite de utilizator.

Procedurile stocate in limbaje de programare generalizate au ca avantaje specific:portabilitatea,scalabilitatea si dezvoltarea rapida.O procedura stocata scrisa intr-un limbaj de programare stocata poate rula,cu mici modificari,in cadrul unei aplicatii client sau intr-un server de aplicatii.In cadrul organizatiilor in care se implementeaza un sistem informatics ce foloseste un SGBD ce suporta astfel de procedure stocate,pot exista déjà rutine scrise in limbaje de programare generalizate.O portabilitate deosebita se obtine si intre diferite SGBD-uri ce suporta aceleasi limbaje de programare pentru scrierea procedurilor stocate.

Datorita atentiei in implementarea masinilor virtuale s-a reusit sa se pastreze caracteristicile de scalabilitate ale SGBD-urilor.Astfel,experimentele au demonstrat faptul ca necesarul suplimentar de memorie per conexiune in cazul unor implementari Oracle este de numai 28kB,fiind deci posibila gestiunea multor utilizatori simultan.Procedurile stocate pot fi create pe baza unor componente deja verificate,totul realizandu-se in mod vizual.Testarea functionalitatii procedurii se poate face in cadrul unui program client,urmand ca apoi sa se faca punerea in exploatare in cadrul serverului de baze de date.

In plus,in unele implementari (cum ar fi Oracle),apelul unei procedure stocate scriise intr-un limbaj de programare generalizat (Java) nu presupune lansarea in executie a unor noi procese.

**Dezavatajele procedurilor stocate**

Ca orice functionalitate nou aparuta intr-un sistem,aduce cu ea si noi provocari de protectie si securitate a datelor.Pentru atacatori,procedurile stocate nu reprezinta doar noi tinte,ci si noi modalitati de atacare atelor din bazele de date.Pot exista procedure stocate care sa contina troieni.viermi sau chiar virusi.

Procedurile stocate nu pot contine chiar toate restrictiile economice si toata logica afacerii.Sunt anumite elemente care raman de implementat in interfata si care se declanseaza,de exemplu,pa masura ce utilizatorul parcurge un formular.

Centralizarea prelucrarilor pe serverul de baze de date poate adduce neplaceri atunci cand executia uneia sau mai multor procedure stocate consuma cea mai mare parte din resursele serverului,iar celelalte conexiuni la baza de date isi pierd din performanta.Nespecificarea importantei relative a prelucrarilor facute de procedurile stocate duce la regula “primul venit,primul servit”,care in anumite cazuri presupune o intarziere a unor prelucrari cu o importanta deosebita.

In cazul procedurilor stocate,scrise in SQL/PSM s-a observant in unele implementari invalidarea in cascada a procedurilor stocate ca urmare a unor modificari in structura datelor sau ca urmare a unor invalidari ale altor procedure stocate.Se pune astfel in discutie una dintre regulile fundamentale ale modelului relational si anume regula independentei logice a datelor fata de prelucrari.

Procedurilor stocate,scrise in SQL/PSM,le scade portabilitatea pe masura ce folosesc pachetele puse la dispozitie de SGBD-ul respectiv.Tot in cazul lor se constata o grija in plus in ceea ce priveste versiunile rutinelor din cadrul aceluiasi pachet din moment ce procedurile si functiile incluse nu pot fi modificate decat in acelasi timp cu tot pachetul.

Codul sursa al procedurilor stocate scrise in SQL/PSM poate sa ajunga in posesia unor persoane care nu au dreptul decat de executie asupra acestora.Desi in unele implementari exista posibilitatea criptarii acestor proceduri stocate,au aparut si programe care sa le decripteze.

In cazul procedurilor stocate scrise intr-un limbaj de programare generalizat pot aparea diferente intre tipurile de date suportate de nucleul SQL al SBGD-ului respective si tipurile de date suportate de limbajul de programare in care sunt implementate procedurile stocate.

Majoritatea avantajelor prezentate sugereaza o crestere a performantelor sistemelor ce folosesc procedure stocate.Se subintelege astfel ca beneficiile majore ale procedurilor stocate se observa in cadrul sistemelor informatice ce folosesc baze de date mari si care au rutine ce folosesc intens datele din bazele de date.Procedurile stocate nu exceleaza in schimb la algoritmi de calcul optimizati si nici nu au o portabilitate deosebita daca nu sunt scrise intr-un limbaj de programare generalizat.

**MPI(Message Passing Interface)-Interfata Transferului de Mesaje**

MPI este o specificaţie pentru dezvoltatorii şi utilizatorii de biblioteci de mesaje transferate. Singura, nu formeaza o bibliotecă , ci mai degrabă specificatia a ceea ce ar trebui sa fie o biblioteca.  
  
 Pur şi simplu , scopul MPI este de a oferi un standard utilizat pe scară largă pentru scrierea de programe de tip “message passing”(transfer de mesaje). Interfaţa tinde să fie practica,portabila,eficienta si flexibila. Specificaţiile interfetei au fost definite pentru programarea in C / C + + şi Fortran.

Cateva motive pentru a utiliza MPI:

|  |
| --- |
| * Standardizare - MPI este singura biblioteca de transfer de mesaje care poate fi considerata standard. Aceasta interfata este disponibila pe aproape toate platformele HPC. Practic, înlocuieste toate bibliotecile anterioare de transfer de mesaje. * Portabilitate - Nu este nevoie sa se modifice codul sursă, atunci când se muta aplicatia pe o platforma diferita care acceptă (şi este în conformitate cu) standardul MPI. * Oportunitati de performanţă - Implementarile furnizorului ar trebui să poată exploata caracteristicile hardware-ului nativ pentru a optimiza performanta. * Functionalitate - peste 115 de rutine sunt definite în MPI-1. * Disponibilitate - O varietate de implementări sunt disponibile atât furnizorului cat şi domeniului public. |
|  |  |  |

Fisier header:

* Necesar pentru toate programele/rutinele care formeaza biblioteca de apeluri MPI.

|  |  |
| --- | --- |
| **C include file** | **Fortran include file** |
| **#include "mpi.h"** | **include 'mpif.h'** |

Formatul apelurilor MPI:

|  |  |
| --- | --- |
| **C Binding** | |
| **Format:** | **rc = MPI\_Xxxxx(parameter, ... )** |
| **Examplu:** | **rc = MPI\_Bsend(&buf,count,type,dest,tag,comm)** |
| **Cod eroare:** | Returned as "rc". MPI\_SUCCESS if successful |

|  |  |
| --- | --- |
| **Fortran Binding** | |
| **Format:** | **CALL MPI\_XXXXX(parameter,..., ierr) call mpi\_xxxxx(parameter,..., ierr)** |
| **Examplu:** | **CALL MPI\_BSEND(buf,count,type,dest,tag,comm,ierr)** |
| **Cod eroare:** | Returned as "ierr" parameter. MPI\_SUCCESS if successful |

Structura generala a programelor MPI:

|  |
| --- |
| General MPI Program Structure |

|  |
| --- |
| **Rutine de administrare a mediului** |
|  |

Rutinele de administrare a mediului MPI sunt folosite intr-o multitudine de scopuri ,cum ar fi initializarea si incheierea mediilor MPI,chestionarea mediului si identitatii, etc. O mare parte dintre cele mai frecvent folosite sunt descrise in cele ce urmeaza.

[**MPI\_Init**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Init.txt)

Initializeaza executia mediului MPI. Aceasta functie trebuie sa fie apelata in fiecare program MPI,trebuie apelata inaintea oricarei alte functii MPI si numai o singura data intr-un program MPI. In programele in C, MPI\_Init poate fi folosita pentru a trece argumentele liniei de comanda pe la fiecare proces in parte,desi acest lucru nu e necesar pentru standard si este dependent de implementare.

MPI\_Init (&argc,&argv)   
MPI\_INIT (ierr)

[**MPI\_Comm\_size**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Comm_size.txt)

Determina numarul de procese in cadrul grupului, asociate unui mijloc de comunicatie. In general utilizat in interiorul unui mijloc de comunicatii,determina numarul de procese care sunt utilizate de aplicatia respectiva.

MPI\_Comm\_size (comm,&size)   
MPI\_COMM\_SIZE (comm,size,ierr)

[**MPI\_Comm\_rank**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Comm_rank.txt)

Determina pozitia procesului de apelare in interiorul mijlocului de comunicatie. Initial, fiecare proces va fi alocat unei pozitii unice notata cu un numar cuprins intre 0 si numarul de procesoare.Aceasta pozitie este adesea intalnita sub numele de ID task. Daca un proces devine asociat cu alte mijloace de comunicare, va avea o pozitie unica in fiecare dintre acestea deopotriva.

MPI\_Comm\_rank (comm,&rank)   
MPI\_COMM\_RANK (comm,rank,ierr)

[**MPI\_Abort**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Abort.txt)

Incheie toate procesele de tip MPI asociate cu mijlocul de comunicatie. In majoritatea implementarilor MPI,aceasta functie incheie toate procesele referitoare la mijloacele de comunicatie specificate.

MPI\_Abort (comm,errorcode)  
MPI\_ABORT (comm,errorcode,ierr)

[**MPI\_Get\_processor\_name**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Get_processor_name.txt)

Returneaza numele procesorului. De asemenea returneaza lungimea numelui. Buffer-ul pentru "nume" trebuie sa fie cel putin de dimensiunea MPI\_MAX\_PROCESSOR\_NAME. Ce este returnat in "nume"dependent de implementare – posibil sa nu fie asemanator cu iesirea de la invelisul de comenzi de tip "hostname" sau"host" .

MPI\_Get\_processor\_name (&name,&resultlength)  
MPI\_GET\_PROCESSOR\_NAME (name,resultlength,ierr)

[**MPI\_Initialized**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Initialized.txt)

Indica daca MPI\_Init a fost apelata – returneaza flag de valoare logica adevarat (1) sau fals (0). MPI necesita ca MPI\_Init sa fie apelata o singura data de catre fiecare proces.Aceasta poate cauza o problema pentru modulele care vor sa utilizeze MPI si sunt pregatite sa apeleze MPI-Init daca e necesar. MPI\_Initialized rezolva aceasta problema.

MPI\_Initialized (&flag)   
MPI\_INITIALIZED (flag,ierr)

[**MPI\_Wtime**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Wtime.txt)

Returneaza timpul ramas in secunde sub forma unui ceas (dubla precizie) in procesorul apelant.

MPI\_Wtime ()  
MPI\_WTIME ()

[**MPI\_Wtick**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Wtick.txt)

Returneaza rezolutia in secunde (dubla precizie) a MPI\_Wtime.

MPI\_Wtick ()  
MPI\_WTICK ()

[**MPI\_Finalize**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Finalize.txt)

Incheie mediul de executie de tip MPI. Aceasta functie ar trebui sa fie ultima rutina de tip MPI apelata in fiecare program MPI-nicio alta rutina nu poate fi apelata dupa ea.

MPI\_Finalize ()  
MPI\_FINALIZE (ierr)

**Exemple: Rutine de administrare a mediului**

|  |
| --- |
| https://computing.llnl.gov/tutorials/images/page01.gifLimbajul C – Exemplu de rutine de administrare a mediului  #include "mpi.h"  #include <stdio.h>  int main(argc,argv)  int argc;  char \*argv[]; {  int numtasks, rank, rc;  rc = MPI\_Init(&argc,&argv);  if (rc != MPI\_SUCCESS) {  printf ("Error starting MPI program. Terminating.\n");  MPI\_Abort(MPI\_COMM\_WORLD, rc);  }  MPI\_Comm\_size(MPI\_COMM\_WORLD,&numtasks);  MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);  printf ("Number of tasks= %d My rank= %d\n", numtasks,rank);  /\*\*\*\*\*\*\* do some work \*\*\*\*\*\*\*/  MPI\_Finalize();  } |

|  |
| --- |
| https://computing.llnl.gov/tutorials/images/page01.gifFortran - Exemplu de rutine de administrare a mediului  program simple  include 'mpif.h'  integer numtasks, rank, ierr, rc  call MPI\_INIT(ierr)  if (ierr .ne. MPI\_SUCCESS) then  print \*,'Error starting MPI program. Terminating.'  call MPI\_ABORT(MPI\_COMM\_WORLD, rc, ierr)  end if  call MPI\_COMM\_RANK(MPI\_COMM\_WORLD, rank, ierr)  call MPI\_COMM\_SIZE(MPI\_COMM\_WORLD, numtasks, ierr)  print \*, 'Number of tasks=',numtasks,' My rank=',rank  C \*\*\*\*\*\* do some work \*\*\*\*\*\*  call MPI\_FINALIZE(ierr)  end |

|  |
| --- |
| Rutine de comunicatii punct-la-punct |

**Concepte generale**

Tipuri de operatii punct-la-punct:

* Operatiile punct-la-punct implica de obicei transferul de mesaje numai prin doua task-uri MPI. Un task realizeaza trimiterea operatiei ,iar celalalt realizeaza primirea unei operatii asemanatoare.
* Exista diferite tipuri de rutine de transmisie si receptie utilizate in diferite scopuri.De exemplu:
  + Transmisia sincrona
  + Blocarea transmisiei /blocarea receptiei
  + Anti-blocarea transmisiei/ Anti-blocarea receptiei
  + Rutine cu buffer de transmisie
  + Rutine combinate de trimisie/receptie
  + Rutina cu transmisie efectuata(“ready”)
* Orice alt tip de rutina de transmisie poate fi grupata cu orice alt tip de rutina de receptie.
* MPI, de asemenea, furnizeaza cateva rutine asociate cu operatiile de transmisie-receptie,cum ar fi cele folosite pentru a astepta sosirea unui mesaj sau pentru a incerca sa afle daca un mesaj a ajuns.

**Parametrii rutinelor de tip MPI**

Rutinele de comunicatie de tip punct-la-punct, de obicei, au o lista de parametri ce au unul din urmatoarele formate:

|  |  |
| --- | --- |
| **Blocarea transmisiei** | **MPI\_Send(buffer,count,type,dest,tag,comm)** |
| **Anti-blocarea transmisiei** | **MPI\_Isend(buffer,count,type,dest,tag,comm,request)** |
| **Blocarea receptiei** | **MPI\_Recv(buffer,count,type,source,tag,comm,status)** |
| **Anti-blocarea receptiei** | **MPI\_Irecv(buffer,count,type,source,tag,comm,request)** |

**Rutine de blocare a transferului de mesaje**

Cel mai frecvent utilizate rutine de blocare a transferului de mesaje de tip MPI sunt descrise in cele ce urmeaza.

[**MPI\_Send**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Send.txt)

Este principala operatie de blocare a transmisiei.Rutina returneaza numai dupa ce buffer-ul aplicatiei din task-ul de transmisie este disponibil pentru reutilizare.De retinut este faptul ca aceasta rutina poate fi implementata diferit pe sisteme diferite.Standardul MPI permite utilizarea unui buffer de sistem, dar nu solicita unul.Unele implementari pot, de fapt, sa utilizeze o transmisie sincrona pentru a implementa operatia principala de blocare a transmisiei.

MPI\_Send (&buf,count,datatype,dest,tag,comm)   
MPI\_SEND (buf,count,datatype,dest,tag,comm,ierr)

[**MPI\_Recv**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Recv.txt)

Receptioneaza un mesaj si il blocheaza inainte ca datele solicitate sa fie disponibile in buffer-ul de aplicatii din operatia(task-ul) de receptie.

MPI\_Recv (&buf,count,datatype,source,tag,comm,&status)   
MPI\_RECV (buf,count,datatype,source,tag,comm,status,ierr)

[**MPI\_Ssend**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Ssend.txt)

Reprezinta blocarea sincrona a transmisiei: Transmite un mesaj si il blocheaza inainte ca buffer-ul de aplicatii al operatiei de transmisie sa fie disponibil pentru reutilizare si inainte ca procesul destinatie sa inceapa sa receptioneze mesaje.

MPI\_Ssend (&buf,count,datatype,dest,tag,comm)   
MPI\_SSEND (buf,count,datatype,dest,tag,comm,ierr)

[**MPI\_Bsend**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Bsend.txt)

Transmisia blocata cu buffer:permite programatorului sa aloce spatiul de buffer necesar in care datele pot fi copiate inainte de a fi livrate.Izoleaza de toate problemele legate de spatiul insuficient al buffer-ului de sistem. Rutina returneaza dupa ce datele au fost copiate din aplicatie,spatiu de buffer pentru buffer-ul de transmisie alocat. Trebuie utilizata cu rutina MPI\_Buffer\_attach.

MPI\_Bsend (&buf,count,datatype,dest,tag,comm)   
MPI\_BSEND (buf,count,datatype,dest,tag,comm,ierr)

[**MPI\_Buffer\_attach**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Buffer_attach.txt) [**MPI\_Buffer\_detach**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Buffer_detach.txt)

Utilizate de catre programator pentru a aloca/dealoca spatiu de buffer pentru mesaje pentru a fi folosit de rutina MPI\_Bsend. Parametrul “size”(dimensiune) este specificat in octetii de date actuali – nu un numar de elemente de date.Numai un singur buffer poate fi atasat unui proces intr-un anumit moment.

MPI\_Buffer\_attach (&buffer,size)   
MPI\_Buffer\_detach (&buffer,size)   
MPI\_BUFFER\_ATTACH (buffer,size,ierr)   
MPI\_BUFFER\_DETACH (buffer,size,ierr)

[**MPI\_Rsend**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Rsend.txt)

Blocare cu transmisie efectuata (Blocking ready send). Ar trebui sa fie utilizata numai daca programatorul este sigur ca receptionarea compatibila este deja postata.

MPI\_Rsend (&buf,count,datatype,dest,tag,comm)   
MPI\_RSEND (buf,count,datatype,dest,tag,comm,ierr)

[**MPI\_Sendrecv**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Sendrecv.txt)

Transmite un mesaj si posteaza o receptie inainte de blocare. Va bloca pana cand buffer-ul aplicatiei de transmisie va fi liber pentru reutilizare si pana cand buffer-ul aplicatiei de receptie contine mesajele primite.

MPI\_Sendrecv (&sendbuf,sendcount,sendtype,dest,sendtag,   
...... &recvbuf,recvcount,recvtype,source,recvtag,   
...... comm,&status)   
MPI\_SENDRECV (sendbuf,sendcount,sendtype,dest,sendtag,   
...... recvbuf,recvcount,recvtype,source,recvtag,   
...... comm,status,ierr)

[**MPI\_Wait**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Wait.txt) [**MPI\_Waitany**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Waitany.txt) [**MPI\_Waitall**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Waitall.txt) [**MPI\_Waitsome**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Waitsome.txt)

MPI\_Wait blocheaza cat timp o operatie de transmisie sau receptie specificata anti-blocare este incheiata. Pentru operatii multiple anti-blocare,programatorul poate specifica oricare,toate sau doar o parte dintre executii.

MPI\_Wait (&request,&status)   
MPI\_Waitany (count,&array\_of\_requests,&index,&status)   
MPI\_Waitall (count,&array\_of\_requests,&array\_of\_statuses)   
MPI\_Waitsome (incount,&array\_of\_requests,&outcount,   
...... &array\_of\_offsets, &array\_of\_statuses)   
MPI\_WAIT (request,status,ierr)   
MPI\_WAITANY (count,array\_of\_requests,index,status,ierr)   
MPI\_WAITALL (count,array\_of\_requests,array\_of\_statuses,   
...... ierr)   
MPI\_WAITSOME (incount,array\_of\_requests,outcount,   
...... array\_of\_offsets, array\_of\_statuses,ierr)

[**MPI\_Probe**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Probe.txt)

Realizeaza un test de blocare pentru un mesaj.MPI\_ANY\_SOURCE si MPI\_ANY\_TAG pot fi folosite pentru a testa un mesaj din orice sursa sau cu orice eticheta.Pentru rutina in C,actuala sursa si eticheta vor fi returnate in statutul structurii ca “status”. In rutina de tip Fortran vor fi returnate in statutul matricei de numere intregi (MPI\_SOURCE) si “status”(MPI\_TAG).

MPI\_Probe (source,tag,comm,&status)   
MPI\_PROBE (source,tag,comm,status,ierr)

**Rutine de anti-blocare a transferului de mesaje**

Cel mai frecvent utilizate rutine de anti-blocare a transferului de mesaje MPI sunt descrise in paragrafele urmatoare.

[**MPI\_Isend**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Isend.txt)

Identifica o zona din memorie pentru a o deservi ca buffer de transmitere.Procesarea continua imediat fara a astepta mesajul de a fi copiata inafara buffer-ului de aplicatie.O cerere de comunicatie este returnata pentru a manevra starea mesajului in curs.Programul n-ar trebui sa modifice buffer-ul aplicatiei inainte de apelurile urmatoare catre MPI\_Wait or MPI\_Test sa indice faptul ca transmisia anti-blocare s-a incheiat.

MPI\_Isend (&buf,count,datatype,dest,tag,comm,&request)   
MPI\_ISEND (buf,count,datatype,dest,tag,comm,request,ierr)

[**MPI\_Irecv**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Irecv.txt)

Identifica o zona din memorie pentru a o folosi ca buffer de receptie.Procesarea continua imediat fara a astepta,de fapt, ca mesajul sa fie receptionat si copiat in buffer-ul aplicatiei. O cerere de comunicatie este returnata pentru a manevra starea mesajului in curs.Programul trebuie sa utilizeze apeluri catre MPI\_Wait sau MPI\_Test pentru a determina cand operatia de receptie anti-blocare se incheie si mesajul solicitat este disponibil in buffer-ul aplicatiei.

MPI\_Irecv (&buf,count,datatype,source,tag,comm,&request)   
MPI\_IRECV (buf,count,datatype,source,tag,comm,request,ierr)

[**MPI\_Issend**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Issend.txt)

Transmisia sincrona anti-blocare.Este similara cu MPI\_Isend(), exceptand MPI\_Wait() sau MPI\_Test() care indica momentul in care procesul destinatie a receptionat mesajul.

MPI\_Issend (&buf,count,datatype,dest,tag,comm,&request)   
MPI\_ISSEND (buf,count,datatype,dest,tag,comm,request,ierr)

[**MPI\_Ibsend**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Ibsend.txt)

Transmisia cu buffer anti-blocare. Este asemanatoare cu MPI\_Bsend() exceptand MPI\_Wait() sau MPI\_Test()care indica momentul in care procesul destinatie a receptionat mesajul.Trebuie utilizata cu rutina MPI\_Buffer\_attach.

MPI\_Ibsend (&buf,count,datatype,dest,tag,comm,&request)   
MPI\_IBSEND (buf,count,datatype,dest,tag,comm,request,ierr)

[**MPI\_Irsend**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Irsend.txt)

Transmisia cu anti-blocare efectuata (Non-blocking ready send). Similara cu MPI\_Rsend(),exceptand MPI\_Wait() sau MPI\_Test() care indica momentul in care procesul destinatie a receptionat mesajul. Trebuie sa fie utilizata numai daca programatorul este sigur de faptul ca o receptie potrivita a fost deja postata.

MPI\_Irsend (&buf,count,datatype,dest,tag,comm,&request)   
MPI\_IRSEND (buf,count,datatype,dest,tag,comm,request,ierr)

[**MPI\_Test**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Test.txt) [**MPI\_Testany**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Testany.txt) [**MPI\_Testall**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Testall.txt) [**MPI\_Testsome**](https://computing.llnl.gov/tutorials/mpi/man/MPI_Testsome.txt)

MPI\_Test verifica starea unei operatii specificate de transmisia sau receptia anti-blocare. Parametrul "flag"este returnat cu valoarea logica adevarat (1),daca operatia este incheiata si cu valoarea logica fals (0), daca nu e incheiata.Pentru operatii multiple de anti-blocare,programatorul poate specifica oricare,toate sau doar o parte dintre executii.

MPI\_Test (&request,&flag,&status)   
MPI\_Testany (count,&array\_of\_requests,&index,&flag,&status)MPI\_Testall (count,&array\_of\_requests,&flag,&array\_of\_statuses)  
MPI\_Testsome (incount,&array\_of\_requests,&outcount,  
...... &array\_of\_offsets, &array\_of\_statuses)  
MPI\_TEST (request,flag,status,ierr)  
MPI\_TESTANY (count,array\_of\_requests,index,flag,status,ierr)  
MPI\_TESTALL (count,array\_of\_requests,flag,array\_of\_statuses,ierr)  
MPI\_TESTSOME (incount,array\_of\_requests,outcount,  
...... array\_of\_offsets, array\_of\_statuses,ierr)

**MPI\_Iprobe**

Realizeaza un test de anti-blocare a mesajului. MPI\_ANY\_SOURCE si MPI\_ANY\_TAG pot fi utilizate pentru a testa un mesaj din orice sursa sau cu orice eticheta.Parametrul intreg, "flag",este returnat de valoare logica adevarat (1) ca mesajul s-a transmis si fals (0) daca nu s-a transmis.Pentru rutinele in C,actuala sursa si eticheta vor fi returnate in starea structurii ca “status”:MPI\_SOURCE and status.MPI\_TAG. Pentru rutinele in Fortran, vor fi returnate sub forma unei matrice de numere intregi (MPI\_SOURCE) si “status”(MPI\_TAG).

MPI\_Iprobe (source,tag,comm,&flag,&status)  
MPI\_IPROBE (source,tag,comm,flag,status,ierr)