7.12.2013

EGE ÜNİVERSİTESİ BİLGİSAYAR MÜHENDİSLİĞİ

PARALEL PROGRAMLAMA

ÖDEV 1

ABDULLAH PEKMEZ 05100000079

# 1.Line Topoloji

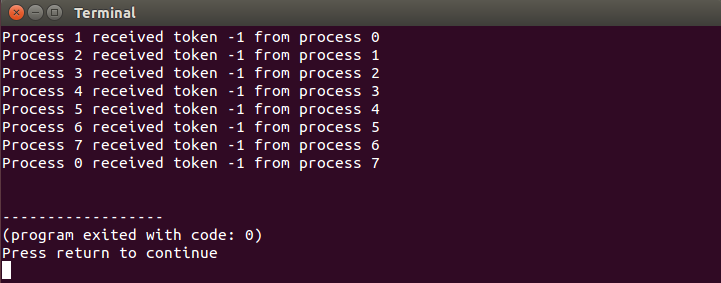
|  |
| --- |
| #include <stdio.h>  #include "mpi.h"  //Line topolojisini MPI\_Send and MPI\_Recv fonksiyonlarını kullanarak  // her proses bir sonraki prosese veri gönderiyor  // Veri gönderimi roottan(0) başlıyor.  int main(int argc, char \*argv[]){  int MyRank, Numprocs;  char\* messages = { "\"Line topolojisi\"" };  int Source, Destination, Root = 0;  MPI\_Status status;  MPI\_Init(&argc, &argv);//MPI\_Init size bir değer dönmektedir  // bu “MPI\_SUCCESS” olursa kod geri kalan MPI\_X fonksiyonlarını kullanabilecektir,  // eğer bu değer dönmezse MPI\_X fonksiyonlarını kullanamazsınız.  MPI\_Comm\_size(MPI\_COMM\_WORLD, &Numprocs);//Bu fonksiyonumuz proses  //sayısını(Numprocs) bize vermektedir  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &MyRank);//Bu MPI fonksiyonu çağıran prosesi  //MyRank değişkenine atıyoruz  if (MyRank == Root) //Rank root mu(0)kotrol ediyor  {Destination = MyRank + 1;  //Hangi prosesse veri gönderilecek belirleniyor(0->1)  // MPI\_COMM\_WORLD değişkeni ise çalıştığımız environmenti (çevre) vermekte  //messages gönderilecek veri adresini gösteriyoruz 1 yollanan veri sayisi  //Veri hangi prosese gönderileceği  //MPI\_INT Datatype parametresi gidecek verinin türünü belirtmektedir (int,float,char,...)  //Tag parametresi ise 0-32767 arası sayısal bir değer alır ve kullanıcı tarafından yanlış makinelerden mesajlar gelmesini önlemek için  //verilebilecek bir tür güvenlik kodudur send ve recive eşit olmalı  //Status mesajın geldiği kaynağı bize bildirmektedir  MPI\_Send(&messages, 1, MPI\_INT, Destination, 0, MPI\_COMM\_WORLD);  //Mesaj Kökten Destinationa gönderiliyor  }else//rank kökten farklı ise  {if (MyRank<Numprocs - 1)//Rank Proses sayısından 1 eksik ise  {Source = MyRank - 1;//Hangi prosesten veri gelmiş belirleniyor.Rankın bir öncesinden  MPI\_Recv(&messages, 1, MPI\_INT, Source, 0, MPI\_COMM\_WORLD, &status);  //Source veri hangi prosesten gelmiş  printf("Proces %d received messages %s from process %d\n", MyRank, messages, Source);  // sonuçları print ediyoruz  Destination = MyRank + 1;//Hangi prosese veri gönderilecek .Rankın bir sonraki prosese.  MPI\_Send(&messages, 1, MPI\_INT, Destination, 0, MPI\_COMM\_WORLD);  }else  //En son rankın bir önceki proses veri gönderimi(8 proses için (0-7) 6 prosesten veri alınmış  {Source = MyRank - 1;  MPI\_Recv(&messages, 1, MPI\_INT, Source, 0, MPI\_COMM\_WORLD, &status);  printf("Proces %d received messages %s from process %d\n", MyRank, messages, Source);// sonuçları print ediyoruz  }  }MPI\_Finalize();//8 prosess işlemiş oldu.işlem bitti.  return 0;  } |

## EKRAN ÇIKTISIC:\Users\abdullah\AppData\Local\Microsoft\Windows\INetCache\Content.Word\B2-A.Lineer.png

# 2.Ring Topoloji

|  |
| --- |
| // Ring topolojisini MPI\_Send and MPI\_Recv fonksiyonlarını kullanarak oluşturuyor.  //Roottan başlıyor roota geri dönüyor.Döngüsel gibi oluyor.  #include <mpi.h>  #include <stdio.h>  #include <stdlib.h>  int main(int argc, char\*\* argv) {  //MPI\_Init size bir değer dönmektedir bu “MPI\_SUCCESS” olursa kod geri kalan MPI\_X fonksiyonlarını kullanabilecektir,  // eğer bu değer dönmezse MPI\_X fonksiyonlarını kullanamazsınız.  MPI\_Init(NULL, NULL);//world\_rank 0  // Find out rank, size  int world\_rank;  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);//Hangi proses çalışıyorsa world\_rank değişkenine set ediyoruz  int world\_size;  MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);//process sayisini world\_size tutuyoruz  int token;// veri  if (world\_rank != 0) {//world\_rank default 0 veya -1 olarak set edilir.default 0 dır.  MPI\_Recv(&token, 1, MPI\_INT, world\_rank - 1, 0, MPI\_COMM\_WORLD,  MPI\_STATUS\_IGNORE);  printf("Process %d received token %d from process %d\n", world\_rank, token,//ekrana basma  world\_rank - 1);  }  else {// prosess 0 ise token -1 atıyoruz  token = -1;  }  MPI\_Send(&token, 1, MPI\_INT, (world\_rank + 1) % world\_size, 0,//Başa dönsün diye proses sayisina göre mod alıyoruz  MPI\_COMM\_WORLD);  if (world\_rank == 0) {//(0=>7) Son prosess ilk prosese veri gönderiyor.Dairesel gibi  MPI\_Recv(&token, 1, MPI\_INT, world\_size - 1, 0, MPI\_COMM\_WORLD,MPI\_STATUS\_IGNORE);  printf("Process %d received token %d from process %d\n", world\_rank, token,  world\_size - 1);  }  MPI\_Finalize();//İşlemi sonlandır.  return 0;  } |

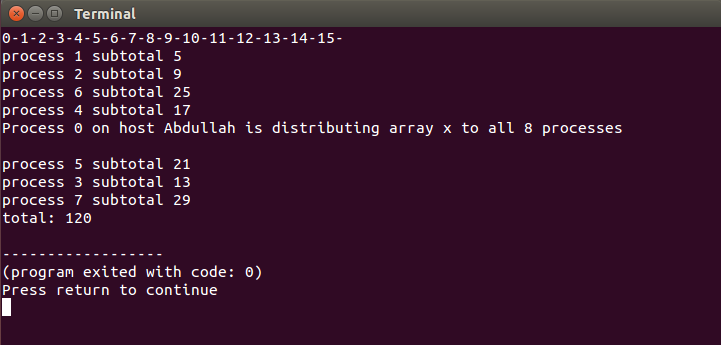
## EKRAN ÇIKTISI



# 3.BROADCAST VE SEND-RECEIVE

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <mpi.h>  #define LENGTH 16 //Dizi uzunluğu  /\*Broadcast ile bir diziyi bütün proseslere gönderilip sonra dizi proseslere eşit bölüp her prosesteki işlemleri  \* toplayıp aratplam yaratıp bütün aratoplamları masterda toplatmak.Dizinin toplamı bulunmuş oldu\*/  int main(int argc, char\* argv[])  {  int i, rank, size, pSize, x[LENGTH],toplam = 0, araToplam;  const int root = 0;  char\* myname = { "Abdullah" };  MPI\_Init(&argc, &argv); /\* Initialize MPI \*/  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size); //Bu fonksiyonumuz proses sayısını bize vermektedir  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);//Bu MPI fonksiyonu çağıran prosesin sırasını vermektedir  pSize = LENGTH / size;//Diziyi proseslere paylaştırıyoruz  if (rank == root)  {  /\* Diziye deger atıyoruz. \*/  for (i = 0; i<LENGTH; i++)  {  x[i] = i;  printf("%d-", x[i]);//Dizini elemanlarını ekrana basıyoruz  }  MPI\_Bcast(&x, LENGTH, MPI\_INT, root, MPI\_COMM\_WORLD);//Root(Master)olarak sayılan prosesten arabellekteki (buffer) x dizisini  // “MPI\_COMM\_WORLD ortamında ki bütün proseslere yollamaktadır.length veri sayisi  printf("\nProcess %d on host %s is distributing array x to all %d processes\n\n", rank, myname, size);  araToplam = 0;  for (i = pSize\*rank; i<pSize\*(rank + 1); i++)//master prosesin yaptığı işlemin aralığını toplatıyor.  araToplam += x[i];  toplam += araToplam; //master prosesin işlemini tek bir yerde topluyor.  for (i = 1; i<size; i++)  {  MPI\_Recv(&araToplam, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);//Masterda bütün proseslerdeki  toplam += araToplam; //yapılan işlemleri topluyor  }  /\* Dizinin toplamını ekrana basıyor \*/  printf("total: %d", toplam);  }  else  { //else Master dışındaki proseslerin sonuçlarını toplatıyor  //x dizisi bütün proseslere gönderiliyor rootan  MPI\_Bcast(&x, LENGTH, MPI\_INT, root, MPI\_COMM\_WORLD);  araToplam = 0;  for (i = pSize\*rank; i<pSize\*(rank + 1); i++)//rank=1 için i=2;i<2\*2 arası  {  araToplam += x[i];  }  printf("process %d subtotal %d\n", rank, araToplam);//Her prosesin hesapdığı değeri yazdırıyor  MPI\_Send(&araToplam, 1, MPI\_INT, root, 0, MPI\_COMM\_WORLD);//her prosesteki yapılan işi mastera gönderiyor.  }  MPI\_Finalize();//Bütün işlemlerimizden sonra MPI\_Finalize diyerek işlemlerimizi sonlandırıyoruz.  return 0;  } |

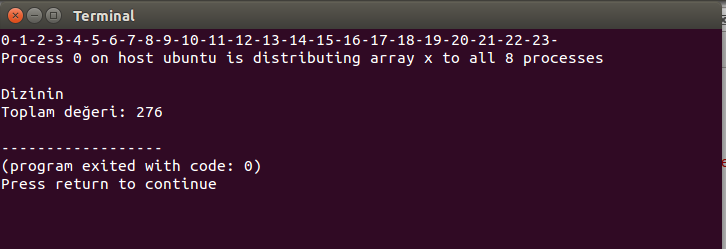
## EKRAN ÇIKTISI



# 4.SCATTER-GATHER

|  |
| --- |
| #include <stdio.h>  #include "mpi.h"  #define MAXPROC 8 /\* Max number of procsses \*/  #define NAMELEN 80 /\* Max length of machine name \*/  #define LENGTH 24 /\* Lengt of send buffer is divisible by 2, 4, 6 and 8 \*/  //Scatter ve Gather dizi bütün proseslere dagıtılıp tekrar Masterda toplatılıyor  int toplam(int\* d, int s)  {  int i;  int toplam = 0;  for (i = 0; i < s; i++)  toplam += d[i];  return toplam;  }  int main(int argc, char\* argv[])  {  int i, np, rank;  const int root = 0; /\* Root process in scatter \*/  int anaToplam;  int araToplam[MAXPROC];  char myname[NAMELEN]; /\* Local host name string \*/  int x[LENGTH]; /\* Send buffer \*/  int y[LENGTH]; /\* Receive buffer \*/  MPI\_Init(&argc, &argv); /\* Initialize MPI \*/  MPI\_Comm\_size(MPI\_COMM\_WORLD, &np); /\* Proses sayisi\*/  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank); /\*Hangi proses çalışıyorsa world\_rank değişkenine set ediyoruz \*/  gethostname(&myname, NAMELEN); /\* Get host name (Ubuntu) \*/  if (rank == 0)//Master Proses (0)için işlemler  {  /\* x dizisine deger atanıyor 0 .. LENGTH-1 \*/  for (i = 0; i < LENGTH; i++)  {  x[i] = i;  printf("%d-", i);  }  /\* Check that we have an even number of processes and at most MAXPROC \*/  printf("\nProcess %d on host %s is distributing array x to all %d processes\n\n", rank, myname, np);  /\* Scatter ile Rootan diziyi bölerek(LENGTH / np) y dizisine gönderiyor \*/  MPI\_Scatter(&x, LENGTH / np, MPI\_INT, &y, LENGTH / np, MPI\_INT, root, MPI\_COMM\_WORLD);  anaToplam = toplam(y, LENGTH / np);//Master prosesteki diziyi toplatıyor  //Master prosesteki dizinin toplamını Roota araToplam dizisinde topluyor  MPI\_Gather(&anaToplam, 1, MPI\_INT, araToplam, 1, MPI\_INT, root, MPI\_COMM\_WORLD);  /\* Print out own portion of the scattered array \*/  anaToplam = toplam(araToplam, np);//Ara toplam dizisinin elemenlarını topluyor  printf("Dizinin\n");  printf("Toplam değeri: %d", anaToplam);//Masterda bölünen dizini toplamı  }  else  { /\* Master dışındaki prosesler için işlemler\*/  //Scatter ile Masterda dizi bölünerek (LENGTH / np)tüm proseslere dagıtılıyor  MPI\_Scatter(&x, LENGTH / np, MPI\_INT, &y, LENGTH / np, MPI\_INT, root, MPI\_COMM\_WORLD);  anaToplam = toplam(y, LENGTH / np);//Her prosesteki diziyi toplatıyor  //Scatter ile Masterda proseslere gönderilen her dizinin toplamını (LENGTH / np) araToplam dizisinde masterda topluyor  MPI\_Gather(&anaToplam, 1, MPI\_INT, araToplam, 1, MPI\_INT, root, MPI\_COMM\_WORLD);  /\* Print out own portion of the scattered array \*/  anaToplam = toplam(araToplam, np);//Her Prsesin araToplam dizisini topluyor değişkene atıyor  }  MPI\_Finalize();//İşlemi sonlandır  return 0;  } |

## EKRAN ÇIKTISI



# 5. BROADCAST-REDUCE

|  |
| --- |
| #include "mpi.h"  #include <stdio.h>  #include <math.h>  #define MAXSIZE 100  int main(int argc, char \*\*argv)  {  int rank, numprocs;  int data[MAXSIZE], i, x, low, high, myresult = 0, result;  MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  if (0 == rank) {//Root 0 ise diziye deger atanıyor  for (i = 0; i < MAXSIZE; i++) {  data[i] = i + 1;  }  }  /\* Datayı roottan proseslere gönderiliyor\*/  MPI\_Bcast(data, MAXSIZE, MPI\_INT, 0, MPI\_COMM\_WORLD);  /\* add portion of data \*/  x = MAXSIZE / numprocs; /\* Dizi proceslere bölüştürülüyor\*/  low = rank \* x;//Proceslerin dizide alt ve üst sınırları bulunuyor  high = low + x;  for (i = low; i < high; i++) {//Her processin kendi işlemlerini toplatıp değişkene atıyor  myresult += data[i];  }  printf("I got %d from %d\n", myresult, rank);  /\* N tane makinede oluşmuş sonuçları derleyerek ana proseste result saklamaktadır. \*/  /\* N tane makinede oluşmuş sonuçları derleyerek ana proseste result saklamaktadır. \*/  MPI\_Reduce(&myresult, &result, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);  if (0 == rank) {  printf("The sum is %d.\n", result);  }  MPI\_Finalize();  return 0;  } |

## EKRAN ÇIKTISI

