Problema Clássico: Deadlock

- **Escreva um programa com dois processos MPI no qual cada processo envia seu número (identidade) no comunicador para o outro processo.**

Deadlock Versão 1

```
#include "mpi.h"
#include <stdio.h>

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

  int nProc, esteProc, outroProc;
  int ierr;
  MPI_Status status[MPI_STATUS_SIZE];

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nProc);
  MPI_Comm_rank(MPI_COMM_WORLD, &esteProc);

  ierr = MPI_Recv(&outroProc, 1, MPI_INTEGER, ((esteProc+1)%2),
       10, MPI_COMM_WORLD, status);
  ierr = MPI_Send(&esteProc, 1, MPI_INTEGER, ((esteProc+1)%2),
       10, MPI_COMM_WORLD);
  printf("este=%d, outro=%d\n", esteProc, outroProc);
  ierr = MPI_Finalize();
  return(0);
}
```

Erro em Deadlock Versão 1

- **Os dois processos esperam eternamente em MPI_Recv, pois os MPI_Send correspondentes não são executados.**

- **O programa não garante que os pares SEND – RECV correspondentes sejam executados**
  - **pois a execução do MPI_Send pressupõe o término da execução do MPI_Recv anterior.**

```
MPI_Recv(&outroProc, 1, MPI_INTEGER, 1, 10, ...)
MPI_Send(&esteProc,  1, MPI_INTEGER, 1, 10, ...)
```

PROC 0

```
MPI_Recv(&outroProc, 1, MPI_INTEGER, 0, 10, ...)
MPI_Send(&esteProc,  1, MPI_INTEGER, 0, 10, ...)
```

PROC 1

Deadlock Versão 2

```
#include "mpi.h"
#include <stdio.h>

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

  int nProc, esteProc, outroProc;
  int ierr;
  MPI_Status status[MPI_STATUS_SIZE];

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nProc);
  MPI_Comm_rank(MPI_COMM_WORLD, &esteProc);

  ierr = MPI_Send(&esteProc, 1, MPI_INTEGER, ((esteProc+1)%2),
                  10, MPI_COMM_WORLD);
  ierr = MPI_Recv(&outroProc, 1, MPI_INTEGER, ((esteProc+1)%2),
                  10, MPI_COMM_WORLD, status);
  printf("este=%d, outro=%d\n", esteProc, outroProc);
  ierr = MPI_Finalize();
  return(0);
}
```

Erro em Deadlock Versão 2

- **A semântica do par MPI_Send e MPI_Recv não garante que MPI_Send seja executado sem que o MPI_Recv correspondente seja emitido.**

- **Os dois processos podem esperar eternamente em MPI_Send, pois os MPI_Recv correspondentes podem não são executados, caso a implementação MPI escolha utilizar o modo síncrono de comunicação.**

- **O programa não garante que os pares SEND – RECV correspondentes (tag 10) sejam executados**
  - **pois a execução do MPI_Send pode requerer o início da execução do MPI_Recv posterior.**

Versão sem Deadlock

```
if (esteProc==0) {
    ierr = MPI_Recv(&outroProc, 1, MPI_INTEGER,
                    ((esteProc+1)%2),
                    10, MPI_COMM_WORLD, status);
    ierr = MPI_Send(&esteProc, 1, MPI_INTEGER,
                    ((esteProc+1)%2),
                    20, MPI_COMM_WORLD); }
  else {
    ierr = MPI_Send(&esteProc, 1, MPI_INTEGER,
                    ((esteProc+1)%2),
                    10, MPI_COMM_WORLD);
    ierr = MPI_Recv(&outroProc, 1, MPI_INTEGER,
                    ((esteProc+1)%2),
                    20, MPI_COMM_WORLD, status);  }
```

Porque versão correta?

- **Versão correta apenas para 2 processos**
  - **Mecanismo conhecido como *red-black ordering***
  - **Pode ser estendido para pares de processos**

- **O par SEND-RECV com *tag* 10 é executado "simultaneamente"; não pressupõe nada sobre a forma de comunicação.**

- **O par SEND-RECV com *tag* 20 também é executado "simultaneamente"; só pressupõe que o par SEND-RECV anterior termina.**

  – **Obs: *tags* 10 e 20 apenas por razões didáticas; os dois tags poderiam ser os mesmos.**

```
MPI_Recv(&outroProc, 1, MPI_INTEGER, 1, 10, ...)
MPI_Send(&esteProc,  1, MPI_INTEGER, 1, 20, ...)
```

PROC 0

```
MPI_Send(&esteProc,  1, MPI_INTEGER, 0, 10, ...)
MPI_Recv(&outroProc, 1, MPI_INTEGER, 0, 20, ...)
```

PROC 1

Há soluções mais simples?

- **Utilize comunicação assíncrona**
  - **Não veremos; talvez não tão simples.**
- **Utilize comunicação não bloqueante**
  - **A cobrir.**
- **Utilize SENDRECV**
  - **Longa lista de argumentos, mas própria para essa situação.**

Envio e Recepção Bloqueante de Mensagem

```
ierr = MPI_Sendrecv (
    &sendbuf, sendcnt, sendtype, dest, sendtag,
    &recvbuf, recvcnt, recvtype, src, recvtag,
    comm, status);
```

- **Envia e recebe mensagem na mesma chamada**
  - **Evita dependências cíclicas entre SEND e RECV, que dão origem a *deadlocks*.**
  - **A implementação MPI, sabendo da possível dependência cíclica, evita-a.**

- **Distinções:**
  - **Os *buffers* tem que ser disjuntos**
  - **As mensagens podem ser distintas (ou não)**
  - **Tags podem ser idênticos (ou não)**

- **A comunicação correspondente à SENDRECV no outro processo pode ser SENDRECV, SEND ou RECV**

Solução SENDRECV ao Deadlock

```
ierr = MPI_Sendrecv (
    &esteProc, 1, MPI_INTEGER, ((esteProc+1)%nProc), 10,
    &outroProc, 1,  MPI_INTEGER, ((esteProc+1)%nProc), 10,
    MPI_COMM_WORLD, status);
```

```
MPI_Sendrecv(
    &esteProc,   1, MPI_INTEGER, 1, 10, ...,
    &outroProc, 1, MPI_INTEGER, 1, 10, ...)
```

PROC 0

```
MPI_Send(&esteProc,   1, MPI_INTEGER, 0, 10, ...)
MPI_Recv(&outroProc, 1, MPI_INTEGER, 0, 10, ...)
```

PROC 1

No Mesmo Buffer:

```
ierr = MPI_Sendrecv_replace (
    &buf, cnt, datatype, dest, sendtag,
    src, recvtag, comm, status);
```

- **Envia e recebe mensagem na mesma chamada e no mesmo *buffer***

- **Sendtag e recvtag podem ser idênticos**

Solução ao Deadlock via MPI_Sendrecv_replace

```
outroProc = esteProc;


ierr = MPI_Sendrecv_replace(
       &outroProc, 1, MPI_INTEGER,
       ((esteProc+1)%nProc), 10,
       ((esteProc+1)%nProc), 10,
       MPI_COMM_WORLD, status);
```

Exemplo: Troca de Mensagens em Anel

```
posterior = ((esteProc+1)%nProc);
anterior  = ((esteProc+nProc-1)%nProc);

ierr = MPI_Sendrecv(
       &esteProc,        1, MPI_INTEGER, posterior, 10,
       &numeroAnterior, 1, MPI_INTEGER, anterior,  10,
       MPI_COMM_WORLD, status);

ierr = MPI_Sendrecv(
       &esteProc,         1, MPI_INTEGER, anterior,  20,
       &numeroPosterior, 1, MPI_INTEGER, posterior, 20,
       MPI_COMM_WORLD, status);

printf("numeroAnterior=%d,esteProc=%d,numeroPosterior=
%d", numeroAnterior, esteProc, numeroPosterior);
```

Lista e não anel

- **Como utilizar SENDRECV em uma lista de processos, no lugar de um anel de processos?**
- **Utilize MPI_PROC_NULL nos extremos**
- **Qualquer comunicação de/para MPI_PROC_NULL não tem efeito:**
  - **SEND (RECV) para (de) MPI_PROC_NULL retorna imediatamente (sem alterar bfr)**

MPI:
Comunicação Ponto a Ponto
Não Bloqueante

Para que Comunicação Não Bloqueante?

- **Para evitar "deadlocks"**
  - **SENDs e RECVs bloqueantes geram *deadlock***
- **Para permitir simultaneidade entre computação e comunicação**
  - **Impedido por comunicações bloqueantes**

Modelo de Comunicação Não Bloqueante

1. **Processo requisita o início da comunicação invocando Isend ou Irecv**

3. **MPI inicia a comunicação e retorna controle ao processo**

5. **Processo continua sua computação enquanto MPI executa a comunicação**

7. **Quando conveniente, o processo:**
   - **Invoca Wait para aguardar o fim da comunicação, ou**
   - **Invoca Test para investigar o fim da comunicação**

Requisições

- **Como relacionar as duas partes da comunicação (requisição, investigação) ?**
  - **Ou seja, o par (ISEND/IRECV, WAIT/TEST)?**
  - **Podem haver múltiplas comunicações pendentes**
  - **Logo, múltiplos pares (ISEND/IRECV, WAIT/TEST)**

- **A operação que requer a comunicação retorna um identificador da requisição ("handle"), denominado "request"**

- **A operação que aguarda/investiga o fim da comunicação utiliza, como argumento de entrada, o "request"**

- **Ao término da comunicação, o "request" recebe o valor MPI_REQUEST_NULL**

Requisita Envio Não Bloqueante

ierr = MPI_Isend (&buf, cnt, datatype, dest, tag, comm, &request);

<type> &buf	Primeira posição dos dados a enviar
int cnt	Quantos dados a enviar (≥0); Size(buf) ≥ cnt
datatype	Tipo MPI dos dados a enviar
int dest	Número do processo a receber a mensagem (no comunicador)
int tag	Identificador da mensagem (≥0 e ≤MPI_TAG_UB)
comm	Comunicador da mensagem
MPI_Request request	Identificador da comunicação
Int ierr	Código de retorno

Requisita Recepção Não Bloqueante

int = MPI_Irecv (&buf, cnt, datatype, src, tag, comm, &request);

<type> &buf	Primeira posição dos dados a receber
Int cnt	Tamanho de buf (≥0); recebe ≤ cnt elementos ou reporta “overflow”
datatype	Tipo MPI dos dados a receber
int src	Número do processo que enviou a mensagem (no comunicador)
int tag	Identificador da mensagem (≥0 e ≤MPI_TAG_UB)
comm	Comunicador da mensagem
MPI_Request request	Identificador da comunicação
int ierr	Código de retorno

Espera Bloqueante do Término da Mensagem

ierr = MPI_Wait (&request, status);

MPI_Request request	Identificador da comunicação
MPI_Status status(mpi_status_size)	Informações sobre a comunicação
Int ierr	Código de retorno

Espera até que a mensagem termine

No retorno, request = MPI_REQUEST_NULL

Investigação do Término da Mensagem

Ierr = MPI_Test(&request, &flag, status);

MPI_Request request	Identificador da comunicação
boolean flag	Operação completa ou incompleta
MPI_Status status(mpi_status_size)	Informações sobre a comunicação
Int ierr	Código de retorno

Retorna, em flag, se comunicação terminou ou não
Não bloqueante
Se operação completa, *request* retorna MPI_REQUEST_NULL

Semântica da Comunicação Não Bloqueante

- **O retorno de IRECV/ISEND indica:**
  - **O início da comunicação solicitada**
  - **Que *buf* não pode ser usado/modificado**
  - **Nada sobre a comunicação correspondente à solicitada**

- **O retorno de WAIT indica**
  - **Término da comunicação solicitada**
  - **Que buf já pode ser usado/modificado**
  - **Se a comunicação foi solicitada por IRECV, indica que o ISEND correspondente começou**
  - **Se a comunicação foi solicitada por ISEND, nada indica sobre o IRECV correspondente**

- **O retorno de TEST com flag=true tem a mesma semântica que o retorno de WAIT**

- **O retorno de TEST com flag=false Indica:**
  - **Que a comunicação não terminou**
  - **Que buf ainda não pode ser usado/modificado**
  - **Nada sobre a comunicação correspondente**

Exemplo

- **Escreva um programa com dois processos MPI no qual cada processo envia seu número (identidade) no comunicador para o outro processo.**

- **A identidade do processo é *esteProc***

- **A identidade do outro processo é *outroProc***

Exemplo (fração do programa)

```
int flag1, flag2;
MPI_Request request1, request2;
...
ierr = MPI_Irecv(&outroProc, 1, MPI_INTEGER,
        ((esteProc+1)%2), 10, MPI_COMM_WORLD, &request1);
ierr = MPI_Isend(&esteProc,  1, MPI_INTEGER,
        ((esteProc+1)%2), 10, MPI_COMM_WORLD, &request2);

flag1=0; flag2=0;
for(;;) {
    if (!flag1) ierr = MPI_Test(&request1, &flag1, status);
    if (!flag2) ierr = MPI_Test(&request2, &flag2, status);
    if (flag1 && flag2) break;
}
```

Livre de Deadlock!!!

Exemplo II (fração do programa)

```
MPI_Request request1, request2;
...
ierr = MPI_Irecv(&outroProc, 1, MPI_INTEGER,
      ((esteProc+1)%2), 10, MPI_COMM_WORLD, &request1);
ierr = MPI_Isend(&esteProc,  1, MPI_INTEGER,
      ((esteProc+1)%2), 10, MPI_COMM_WORLD, &request2);

ierr = MPI_Wait(&request1, status);
ierr = MPI_Wait(&request2, status);
```

Livre de Deadlock!!!

Exemplo

- **Implemente SENDRECV utilizando comunicação não bloqueante.**

Exemplo (fração do programa)

```
MPI_Sendrecv(
        &sendbuf, sendcnt, sendtype, dest, sendtag,
        &recvbuf, recvcnt, recvtype, src, recvtag, ...)
```

≡

```
MPI_Iecv(&recvbuf, recvcnt, recvtype, src, recvtag, ..., &request1, ...)
MPI_Isend(&sendbuf, sendcnt, sendtype, dest, sendtag, ..., &request2, ...)
flag1=0; flag2=0;
for(; ;) {
    if (!flag1) MPI_Test(&request1, &flag1, status);
    if (!flag2) MPI_Test(&request2, &flag2, status);
    if (flag1 && flag2) break;
}
```

Outras operações não bloqueantes

- **Há muitas outras operações**

- **MPI_WAITALL, MPI_WAITANY, MPI_WAITSOME**
  - **Aguarda todas, qualquer uma, algumas comunicações**

- **Idem para TEST**

- **Etc...**

Outras Comunicações Coletivas ainda não exploradas

- **GATHER (2 formas)**
  - **Recolhe em um processo dados esparramados pelos processos**
- **SCATTER (2 formas)**
  - **Esparrama os dados de um processo dentre os processos**

ALLGATHER (2 formas)

- **Recolhe em todos os processos ...**

ALLTOALL (2 formas)

- **Troca completa de dados entre os processos**

OP_CREATE, OP_FREE (para REDUCE)

- **Habilidade de criar e destruir novos operadores**

...