# Entrada/Saída em MPI

**Tópicos**:
- Operações de E/S
- Sincronização de Processadores

**Referência**: Pacheco,P.S. *Parallel Programming with MPI* Morgan Kaufmann, San Francisco, 1997.



# Operações de E/S

- **Problema**: Como associar *stdin, stdout, stderr*?
- **Possibilidades**:
  - Um único processador tem acesso (*stdin*=teclado, *stdout*/*stderr*=tela)
  - Todos os processadores têm acesso (como controlar?)
  - Nenhum dos processadores têm acesso! (I/O:arquivo)
- **Na prática**:
  - Forma de implementação depende de cada sistema
  - Primeira versão de MPI não assumia nenhum caso particular
  - Em geral, pelo menos *um* dos processadores tem acesso



# Operações de E/S (cont.)

- **Solução Típica**:
  - Designar um dos processadores responsável por I/O
  - Entrada de Dados:
    - Processador de I/O lê dados de entrada
    - Processador de I/O faz um *broadcast* dos dados lidos
    - Demais processadores *recebem* dados de entrada
  - Saída de Dados:
    - Cada processador *envia* dados a serem armazenados
    - Processador de I/O *recebe* dados de cada um dos demais
    - Processador de I/O escreve dados de saída



# Operações de E/S (cont.)

- Alternativas ao Uso de *stdin/stdout*:
  - Utilizar parâmetros de entrada na linha de comando

Exemplo:

Prog.Fonte:

```
printf("[%d] argumentos: %s %s %s \n",my_rank,argv[0],argv[1],argv[2]);
```

Execução:

```
{aldebaran}/home/usuarios/celso/% mpirun -np 4 teste arg1 arg2
[0] argumentos: /home/usuarios/celso/teste arg1 arg2
[2] argumentos: /home/usuarios/celso/teste arg1 arg2
[1] argumentos: /home/usuarios/celso/teste arg1 arg2
[3] argumentos: /home/usuarios/celso/teste arg1 arg2
```



# Operações de E/S (cont.)

- (Mais) Alternativas ao Uso de *stdin/stdout*:
  - Utilizar arquivos de E/S
  - Problemas:
    - Se há vários discos, um em cada processador : dados devem estar espalhados (pode ser necessário distribuir dados no início e reagrupar dados ao final do processamento)
    - Se há um único disco, acessado por todos os processadores: dois ou mais processadores não podem abrir arquivos com mesmo nome
  - Soluções Típicas:
    - Fazer E/S em arquivo por um único processador
    - Cada processador abre um arquivo diferente



# Operações de E/S (cont.)

Exemplo:

Prog.Fonte:

```
FILE* my_fp;
char  filename[100];    int my_rank;
. . .
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
sprintf(filename, “arq.%d”, my_rank) ;
my_fp = fopen( filename, “w” );
. . .
```

Durante a Execução:

Arquivos arq.0, arq.1, arq.2, ... , arq.P-1

OBS: Arquivos podem ser combinados posteriormente

## Sincronização de Processadores

- Objetivo:
  - Garantir que os processadores estão num ponto pré-determinado da execução, num certo instante
- Mecanismos de Sincronização em MPI:
  - Implícitos: funções *send/recv* síncronas ou c/ bloqueio
  - Explícitos: criação de *barreiras* (nenhum processador sai antes que todos tenham chegado)

  Implementação:

```
int MPI_Barrier (MPI_Comm comm)
```



## Sincronização de Processadores (cont.)

Exemplo do Uso de Barreiras:

```
/* Todos os processadores iniciam a execução juntos */
MPI_Init(...);
MPI_Barrier(MPI_COMM_WORLD);
... (Fase 1 do programa )
MPI_Barrier(MPI_COMM_WORLD);
... (Fase 2 do programa )
MPI_Barrier(MPI_COMM_WORLD);
... (Salva resultados)
MPI_Barrier(MPI_COMM_WORLD);
MPI_Finalize( );
/* Todos os processadores terminam a execução juntos */
```