5CS022 Distribute and Cloud Systems Programming

Week 1 Workshop

Tasks

Download the sample MPI programs from the drive into your Linux system. Compile and run the program mpi01.c. To compile it, run the following command in the terminal:

```
mpicc mpi01.c -o mpi01
```

Running without any arguments

```
 ~ Output

[wizard@archlinux w1]$ mpirun ./a.out
I am 0 of 1
[wizard@archlinux w1]$
```

Running with number of processes 4

```
 ~

[wizard@archlinux w1]$ mpirun -np 4 ./a.out
I am 0 of 4
I am 1 of 4
I am 2 of 4
I am 3 of 4
[wizard@archlinux w1]$
```

Running with max number of processes (252)

```
 ~

[wizard@archlinux w1]$ mpirun -np 252 ./a.out
I am 0 of 252
I am 1 of 252
I am 2 of 252
I am 3 of 252
...
I am 252 of 252
[wizard@archlinux w1]$
```

Compile and run the program mpi02.c. Try running it with 2, 3 and 4 processes. Eg.:

Running With: `mpirun -n 2 ./mpi02`

```
 ~

[wizard@archlinux w1]$ mpirun -n 2 ./a.out
This program needs to run on exactly 3 processes
```

Running With: `mpirun -n 3 ./mpi02`

```
~

[wizard@archlinux w1]$ mpirun -n 3 ./a.out
Process 1 received 9
Process 2 received 17
```

Running With: `mpirun -n 4 ./mpi02`

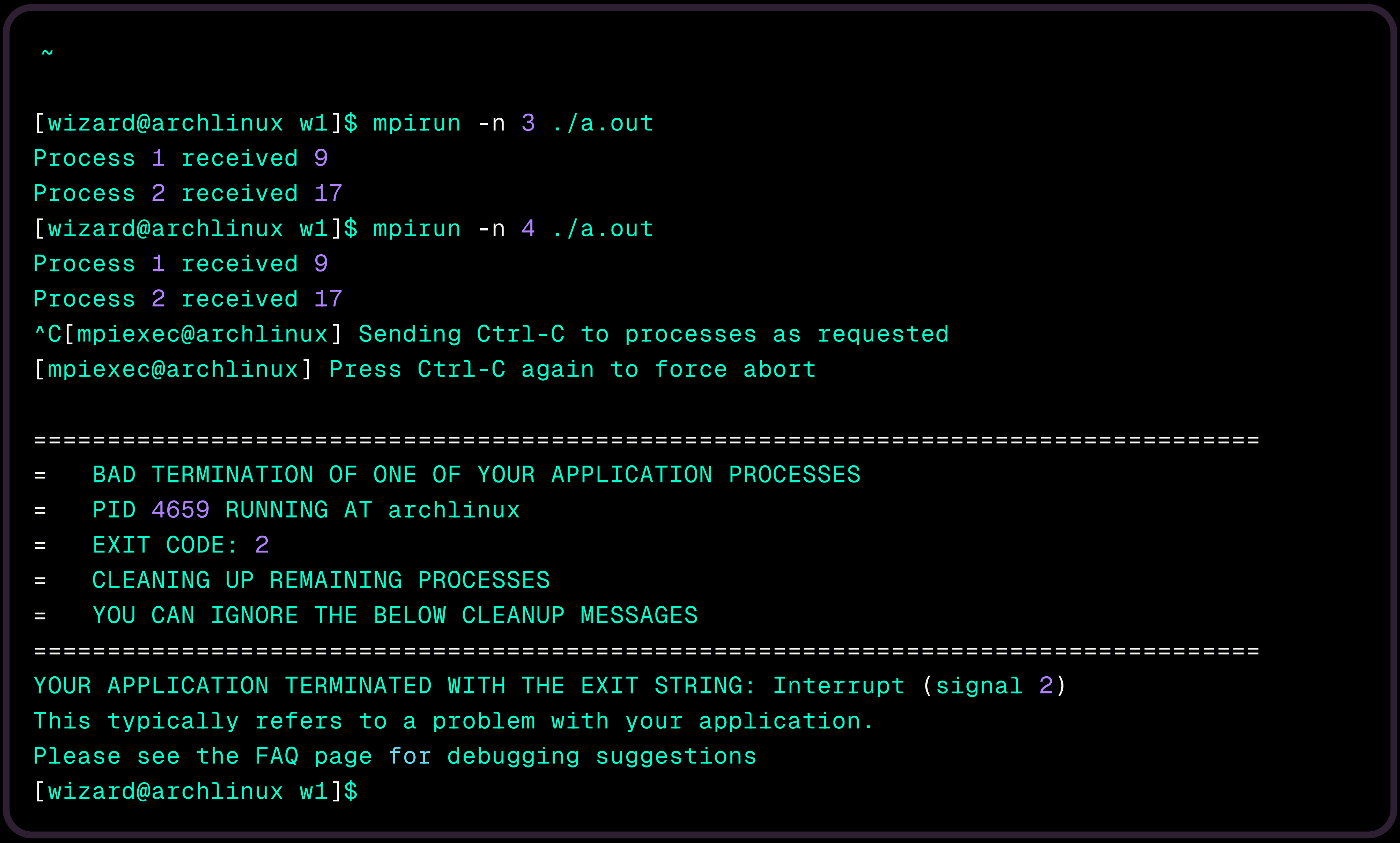
```
~

[wizard@archlinux w1]$ mpirun -n 4 ./a.out
This program needs to run on exactly 3 processes
```

Now change the code so that you remove the check for only 3 processes. Now run it with 2, then 3 , then 4 and then more processes.

```
~

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

int main(int argc, char** argv) {
    int size, rank;

    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if(rank ==0){
        int x = 9;
        int y = 17;
        MPI_Send(&x, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        MPI_Send(&y, 1, MPI_INT, 2, 0, MPI_COMM_WORLD);
    } else {
        int number;
        MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Process %d received %d\n", rank, number);
    }
    MPI_Finalize();

    return 0;
}
```

```
MPI_Send(&y, 1, MPI_INT, 2, 0, MPI_COMM_WORLD); invalid

[wizard@archlinux w1]$ mpirun -n 2 ./a.out
Process 1 received 9
Abort(537553414) on node 0 (rank 0 in comm 0): Fatal error in internal_Send: Invalid rank,
error stack:
internal_Send(124): MPI_Send(buf=0x7ffd7768d3f4, count=1, MPI_INT, 2, 0, MPI_COMM_WORLD) failed
internal_Send(78).: Invalid rank has value 2 but must be nonnegative and less than 2
```

NOTE: This errors out because `mpirun -n 2 ./a.out` specifies the ranks to be `{0,1}`; thus making this line: `MPI_Send(&y, 1, MPI_INT, 2, 0, MPI_COMM_WORLD);` invalid.



```
 ~

[wizard@archlinux w1]$ mpirun -n 3 ./a.out
Process 1 received 9
Process 2 received 17
[wizard@archlinux w1]$ mpirun -n 4 ./a.out
Process 1 received 9
Process 2 received 17
^C[mpiexec@archlinux] Sending Ctrl-C to processes as requested
[mpiexec@archlinux] Press Ctrl-C again to force abort

===================================================================================
=   BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES
=   PID 4659 RUNNING AT archlinux
=   EXIT CODE: 2
=   CLEANING UP REMAINING PROCESSES
=   YOU CAN IGNORE THE BELOW CLEANUP MESSAGES
===================================================================================
YOUR APPLICATION TERMINATED WITH THE EXIT STRING: Interrupt (signal 2)
This typically refers to a problem with your application.
Please see the FAQ page for debugging suggestions
[wizard@archlinux w1]$
```

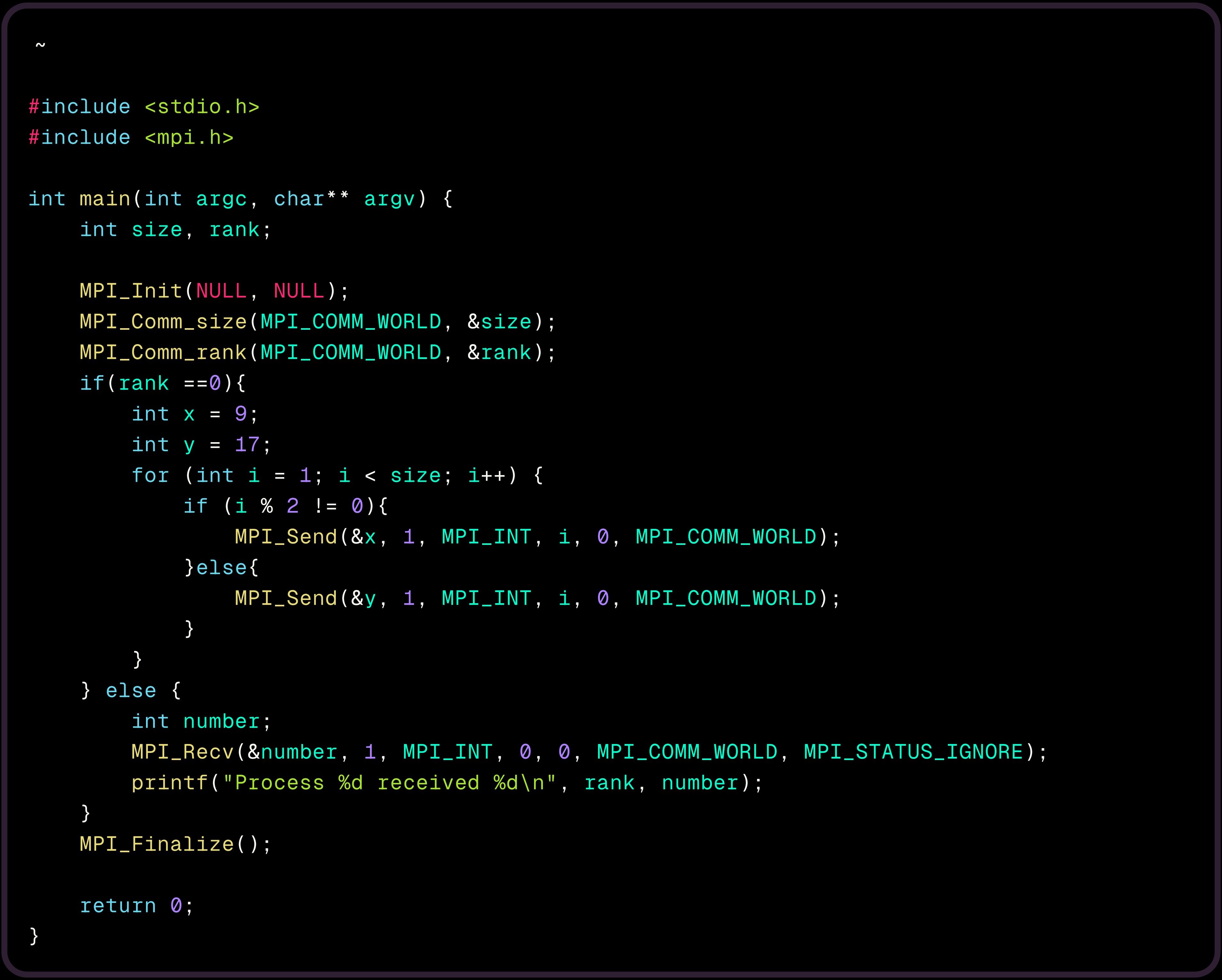
When you try to run it with 4 or more processes, it probably runs and appears to work, but never ends. You will have to end with "Ctrl-C". Why do you think it doesn't end when you run it with more than 3 processes? Change it so that it will work with any number of processes.

It never ends because the root thread `0` sends messages to `Ranks: {1,2}`, those 2 work fine; but the last process `Rank: 3` waits for a message From the root thread `0`:

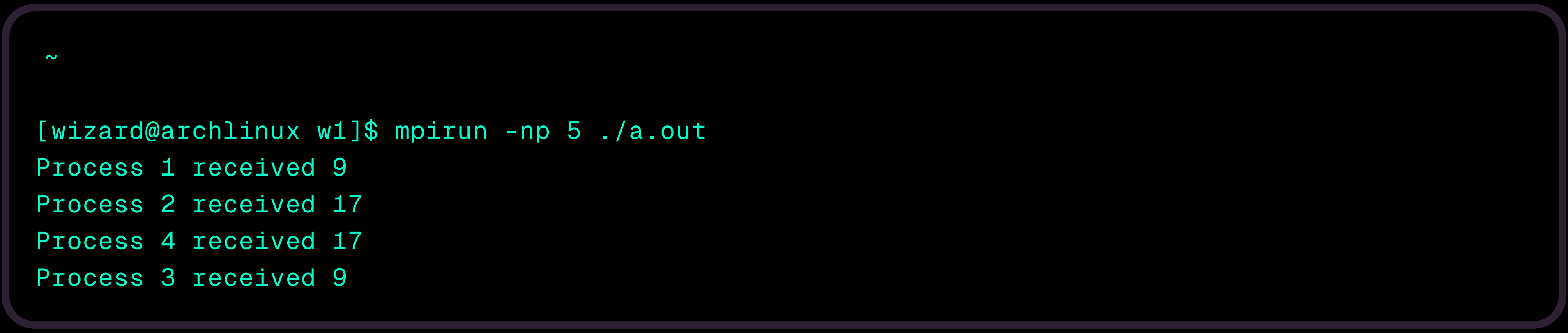
`MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);`

which never happens. `MPI_Recv` is a blocking operation so it just … blocks.

The Fix?

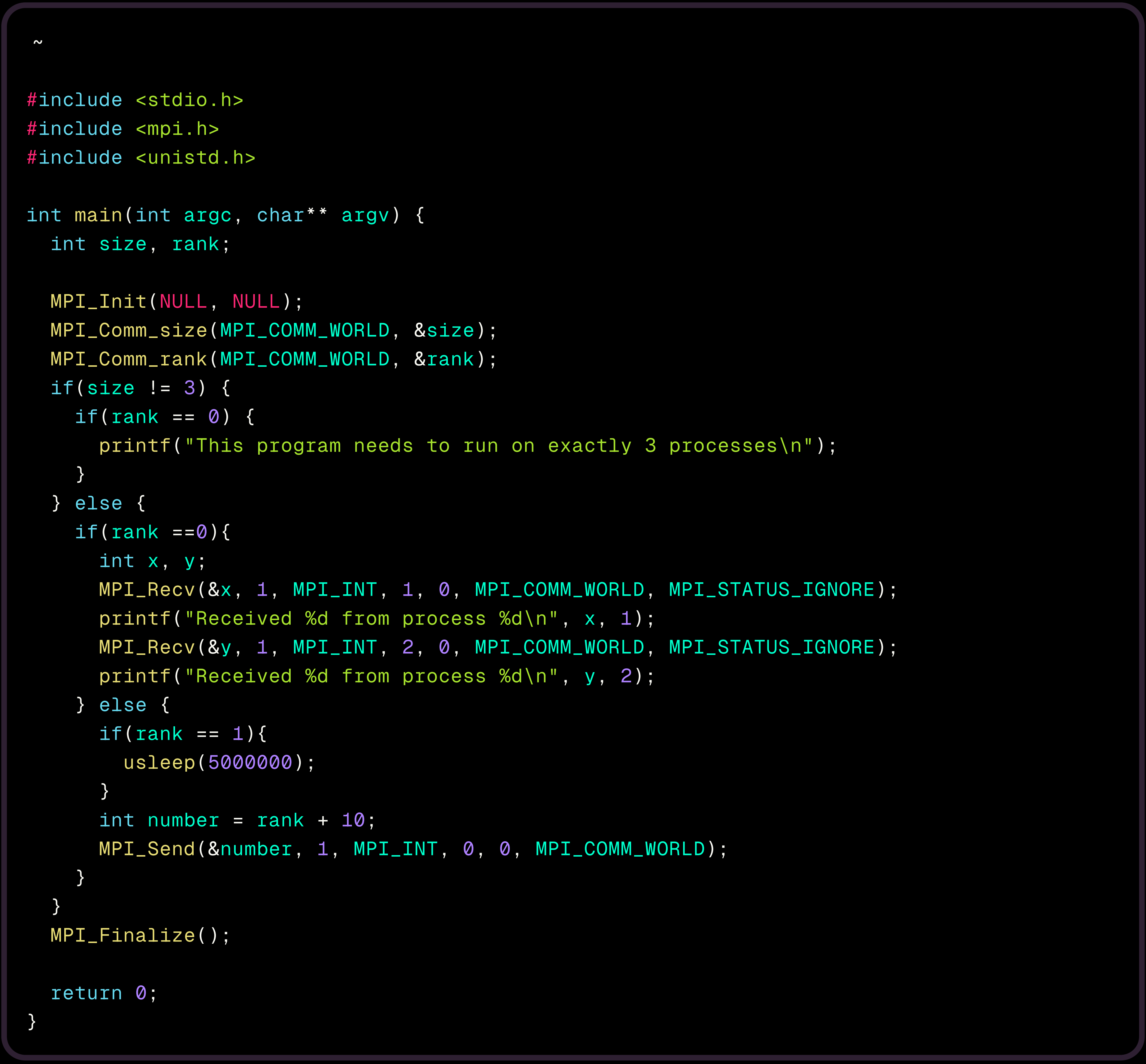


```c
#include <stdio.h>
#include <mpi.h>

int main(int argc, char** argv) {
    int size, rank;

    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if(rank ==0){
        int x = 9;
        int y = 17;
        for (int i = 1; i < size; i++) {
            if (i % 2 != 0){
                MPI_Send(&x, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
            }else{
                MPI_Send(&y, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
            }
        }
    } else {
        int number;
        MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Process %d received %d\n", rank, number);
    }
    MPI_Finalize();

    return 0;
}
```



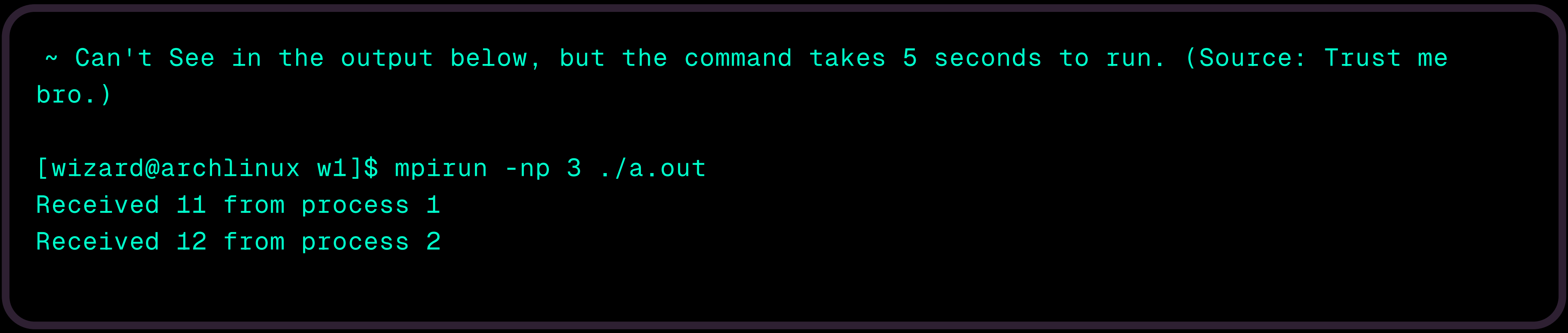
```
[wizard@archlinux w1]$ mpirun -np 5 ./a.out
Process 1 received 9
Process 2 received 17
Process 4 received 17
Process 3 received 9
```

Build and run the program mpi03.c. In this program Process 0 will wait for messages from Process 1 and Process 2. However, Process 1 ends up blocking Process 2 because it sleeps for 5 seconds.



~

```c
#include <stdio.h>
#include <mpi.h>
#include <unistd.h>

int main(int argc, char** argv) {
  int size, rank;

  MPI_Init(NULL, NULL);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  if(size != 3) {
    if(rank == 0) {
      printf("This program needs to run on exactly 3 processes\n");
    }
  } else {
    if(rank ==0){
      int x, y;
      MPI_Recv(&x, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      printf("Received %d from process %d\n", x, 1);
      MPI_Recv(&y, 1, MPI_INT, 2, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      printf("Received %d from process %d\n", y, 2);
    } else {
      if(rank == 1){
        usleep(5000000);
      }
      int number = rank + 10;
      MPI_Send(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }
  }
  MPI_Finalize();

  return 0;
}
```



```
~ Can't See in the output below, but the command takes 5 seconds to run. (Source: Trust me
bro.)

[wizard@archlinux w1]$ mpirun -np 3 ./a.out
Received 11 from process 1
Received 12 from process 2
```

This happens because again `MPI_Recv` is a blocking operation, So when thread 2 sleeps for 5 seconds, the root thread `0` waits for its response

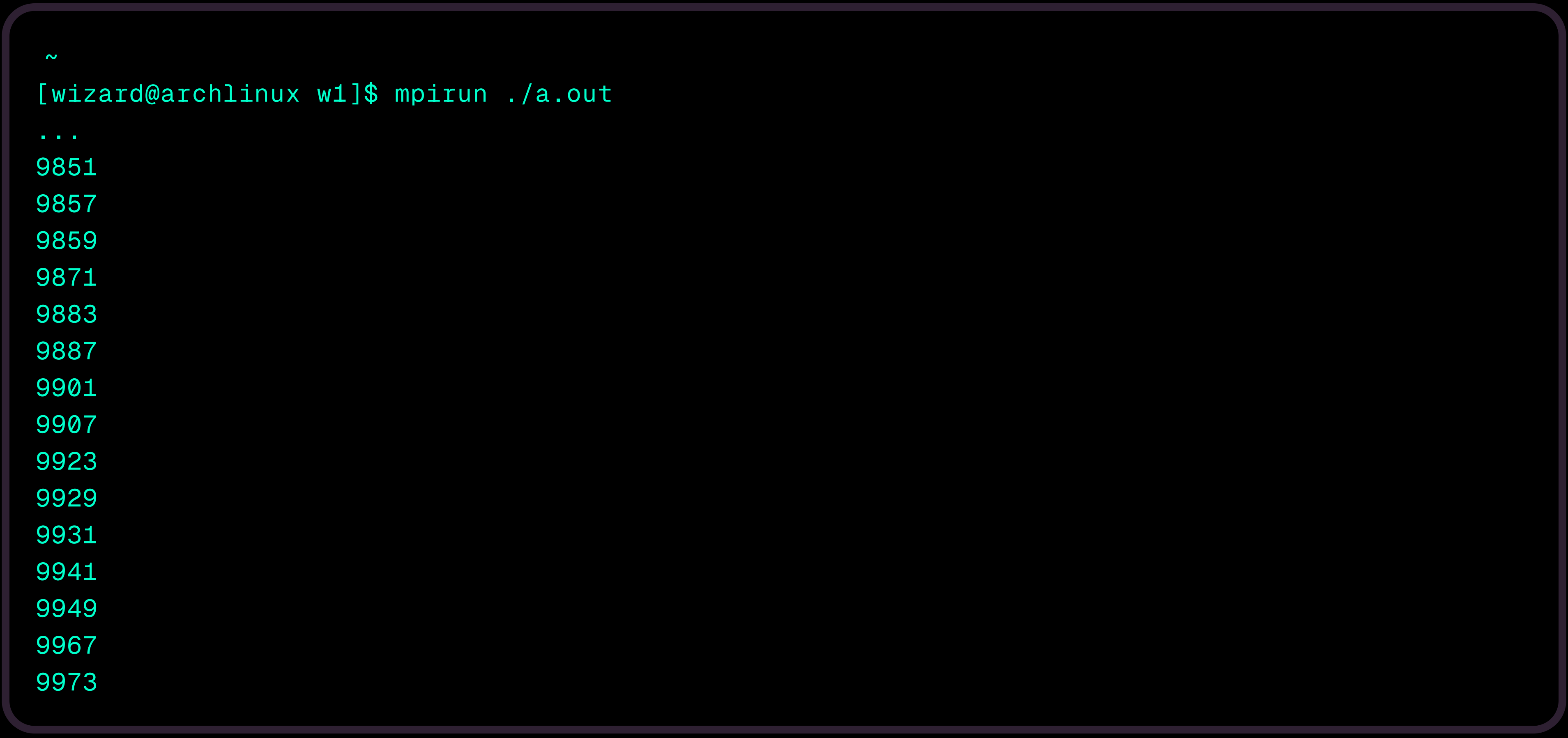
---

The following is a simple program that looks for prime numbers between 1 to 10000:

```
#include <stdio.h>;
int main(int argc, char **argv)
{
    int i, c;
    int nstart=1, nfinish=10000;
    printf("%s : Prime numbers between %d and %d are :\n", argv[0], nstart, nfinish);
    for(i=nstart; i<=nfinish; i++)
    {
        for(c=2; c<=i-1; c++)
        {
            if ( i%c==0 )
                break;
        }
        if ( c==i )
            printf(&quot;%s : %d\n&quot;,argv[0], i);
    }
    return 0;
}
```

MPI Version

```
 ~

#include <stdio.h>
#include <mpi.h>
#include <math.h>

int is_prime(int num) {
    if (num < 2) return 0;
    for (int i = 2; i * i <= num; i++) {
        if (num % i == 0) return 0;
    }
    return 1;
}

int main(int argc, char **argv) {
    int rank, size;
    int nstart = 1, nfinish = 10000;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int chunk_size = nfinish / size;
    int start = nstart + rank * chunk_size;
    int end = (rank == size - 1) ? nfinish : start + chunk_size - 1;

    printf("Rank %d: Primes between %d and %d:\n", rank, start, end);
    for (int i = start; i <= end; i++) {
        if (is_prime(i)) {
            printf("%d \n", i);
        }
    }

    MPI_Finalize();
    return 0;
}
```



```
 ~
[wizard@archlinux w1]$ mpirun ./a.out
...
9851
9857
9859
9871
9883
9887
9901
9907
9923
9929
9931
9941
9949
9967
9973
```