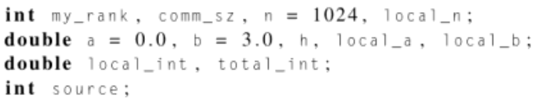
Dealing with I/O

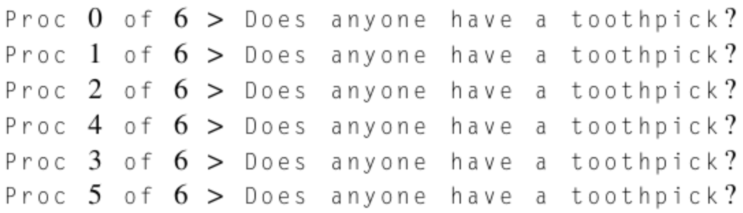


```
int my_rank, comm_sz, n = 1024, local_n;
double a = 0.0, b = 3.0, h, local_a, local_b;
double local_int, total_int;
int source;
```

This is part of the code from the trapezoidal program we saw earlier, notice that we have hard coded the length of the function to be a = 0 and b =3

Instead of having to edit the code and recompile every time we wanted to find the area of a new length we can implement some input and output

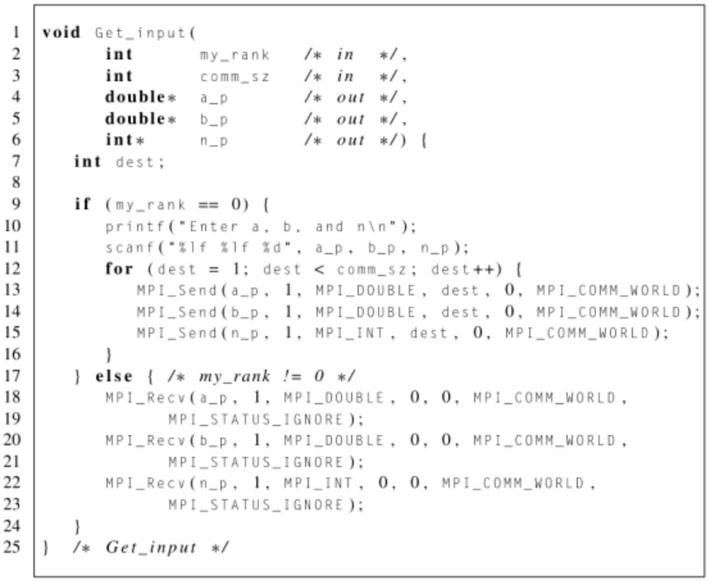
- OUTPUT:
- So far we have assumed that process 0 can write to stdout (by using printf.
- Now is a good time to say that all MPI implementations allow all the processes in MPI_COMM_WORLD access to sdout and stderr.
    - All processes can execute printf and fprintf
    - However, most MPI implementations don't provide any automatic scheduling of access to these devices. That is, if multiple processes are attempting to write to, say, stdout, the order in which the processes' output appears will be unpredictable.
        - Because of nondeterminism if we have some prg that has every process execute some message "Does anyone have a toothpick?" the order in which they print can vary run to run.



```
Proc 0 of 6 > Does anyone have a toothpick?
Proc 1 of 6 > Does anyone have a toothpick?
Proc 2 of 6 > Does anyone have a toothpick?
Proc 4 of 6 > Does anyone have a toothpick?
Proc 3 of 6 > Does anyone have a toothpick?
Proc 5 of 6 > Does anyone have a toothpick?
```

This happens because each process is competing for access to stdout

- if we do not want this to happen and we want there to be some sort of order we can modify the program to do so.
    - For example we could have each process that isnt 0 send its message to process 0 and it will handle the output in rank order.
- INPUT:
- Unlike output most MPI implementations only allow process 0 in MPI_COMM_WORLD access to stdin which handles input.
    - This makes sense because if it didn't then what process would get what line of input?
        - Would process 0 get the first line and process 1 get the second line? It wouldn't make much sense
- So, to write MPI programs that can use scanf, we need to branch on process rank, with process 0 reading in the data, and then sending it to the other processes.
    - We can create a function that can handle this called get_input(...)



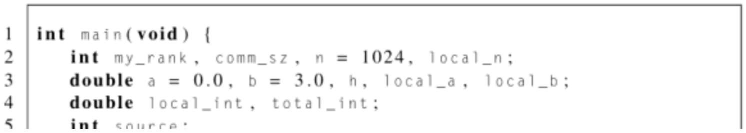
```
void Get_input(
      int      my_rank   /* in   */,
      int      comm_sz   /* in   */,
      double*  a_p       /* out  */,
      double*  b_p       /* out  */,
      int*     n_p       /* out */) {
   int dest;

   if (my_rank == 0) {
      printf("Enter a, b, and n\n");
      scanf("%lf %lf %d", a_p, b_p, n_p);
      for (dest = 1; dest < comm_sz; dest++) {
         MPI_Send(a_p, 1, MPI_DOUBLE, dest, 0, MPI_COMM_WORLD);
         MPI_Send(b_p, 1, MPI_DOUBLE, dest, 0, MPI_COMM_WORLD);
         MPI_Send(n_p, 1, MPI_INT, dest, 0, MPI_COMM_WORLD);
      }
   } else { /* my_rank != 0 */
      MPI_Recv(a_p, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
      MPI_Recv(b_p, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
      MPI_Recv(n_p, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
   }
}  /* Get_input */
```

Program 3.5: A function for reading user input.

Note:

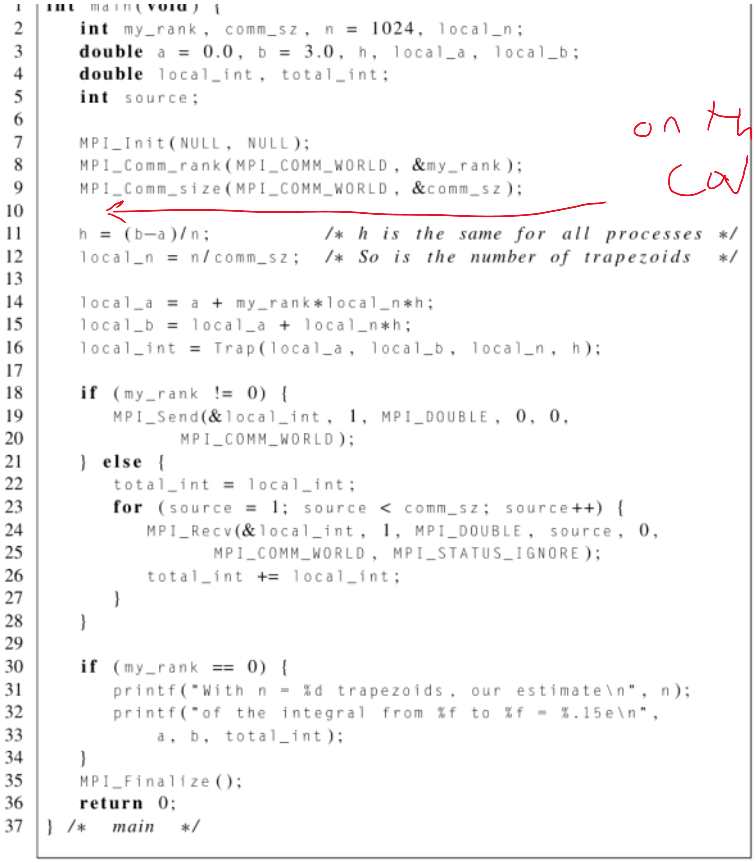
- What this function will do is that it will handle asking the user for data, storing it, and then handling having process 0 send all the data to all the other processes who will recv the data

- The new trapizoid rule prg would look like:



```
int main(void) {
   int my_rank, comm_sz, n = 1024, local_n;
   double a = 0.0, b = 3.0, h, local_a, local_b;
   double local_int, total_int;
   int source;
```

would make the



```
int main(void) {
   int my_rank, comm_sz, n = 1024, local_n;
   double a = 0.0, b = 3.0, h, local_a, local_b;
   double local_int, total_int;
   int source;

   MPI_Init(NULL, NULL);
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
   MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);

   h = (b−a)/n;          /* h is the same for all processes */
   local_n = n/comm_sz;  /* So is the number of trapezoids  */

   local_a = a + my_rank*local_n*h;
   local_b = local_a + local_n*h;
   local_int = Trap(local_a, local_b, local_n, h);

   if (my_rank != 0) {
      MPI_Send(&local_int, 1, MPI_DOUBLE, 0, 0,
            MPI_COMM_WORLD);
   } else {
      total_int = local_int;
      for (source = 1; source < comm_sz; source++) {
         MPI_Recv(&local_int, 1, MPI_DOUBLE, source, 0,
               MPI_COMM_WORLD, MPI_STATUS_IGNORE);
         total_int += local_int;
      }
   }

   if (my_rank == 0) {
      printf("With n = %d trapezoids, our estimate\n", n);
      printf("of the integral from %f to %f = %.15e\n",
          a, b, total_int);
   }
   MPI_Finalize();
   return 0;
} /* main */
```

Program 3.2: First version of MPI trapezoidal rule.

on this line we would make the call to get_input()