```c
/* Template for creating a custom error handler for MPI and a simple program
   to demonstrate its' use.  How much additional information you can obtain
   is determined by the MPI binding in use at build/run time.

   To illustrate that the program works correctly use -np 2 through -np 4.

   To illustrate an MPI error set victim_mpi = 5 and use -np 6.

   To illustrate a system error set victim_os = 5 and use -np 6.

   2004-10-10   charliep    created
   2006-07-15   joshh       updated for the MPI2 standard
   2007-02-20   mccoyjo     adapted for folding@clusters
   2010-05-26   charliep    cleaned-up/annotated for the petascale workshop
*/
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "mpi.h"

void ccg_mpi_error_handler(MPI_Comm *, int *, ...);

int main(int argc, char *argv[]) {
    MPI_Status status;
    MPI_Errhandler errhandler;
    int number, rank, size, next, from;
    const int tag = 201;        /* Arbitrarily choose 201 as the tag. */
    const int server = 0;
    const int victim_mpi = 5;
    const int victim_os = 6;
    MPI_Comm bogus_communicator;

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

    /* Create the error handler.  This is done through a callback to the
       function whose address is passed to the create function.
    */
    MPI_Comm_create_errhandler(&ccg_mpi_error_handler, &errhandler);
    MPI_Comm_set_errhandler(MPI_COMM_WORLD, errhandler);

    /* Calculate the rank of the next process in the ring.  Use the modulus
       operator so that the last process "wraps around" to rank 0.
    */
    next = (rank + 1) % size;
    from = (rank + size - 1) % size;

    /* If we are the server process, get a integer from the user to specify
       how many times we want to go around the ring.
    */
    if (rank == server) {
        printf("Enter the number of times to go around the ring: ");
        fflush(stdout);
        scanf("%d", &number);
        --number;
        printf("Process %d sending %d to %d\n", rank, number, next);
        MPI_Send(&number, 1, MPI_INT, next, tag, MPI_COMM_WORLD);
    }
```

```
    /* Pass the message around the ring.  The exit mechanism works as follows:
       the message (a positive integer) is passed around the ring.  Each time
       is passes rank 0, it is decremented.  When each processes receives the
       0 message, it passes it on to the next process and then quits.
    */
    while (true) {
        MPI_Recv(&number, 1, MPI_INT, from, tag, MPI_COMM_WORLD, &status);
        printf("Process %d received %d\n", rank, number);

        if (rank == server) {
            number--;
            printf("Process 0 decremented number\n");
        }

        /* System error, not trapped by MPI.
        */
        if (rank == victim_os) {
            int a[10];

            printf("Process %d about to segfault\n", rank);
            a[15565656] = 56;
        }

        /* MPI error, trapped by MPI.
        */
        if (rank == victim_mpi) {
            printf("Process %d about to go south\n", rank);
            printf("Process %d sending %d to %d\n", rank, number, next);
            MPI_Send(&number, 1, MPI_INT, next, tag, bogus_communicator);
        } else {
            printf("Process %d sending %d to %d\n", rank, number, next);
            MPI_Send(&number, 1, MPI_INT, next, tag, MPI_COMM_WORLD);
        }

        if (number == 0) {
            printf("Process %d exiting\n", rank);
            break;
        }
    }

    /* The last process does one extra send to process 0, which needs to be
       received before the process/program can exit.
    */
    if (rank == server)
        MPI_Recv(&number, 1, MPI_INT, from, tag, MPI_COMM_WORLD, &status);

    MPI_Finalize();
    return 0;
}

void ccg_mpi_error_handler(MPI_Comm *communicator, int *error_code, ...) {
    char error_string[MPI_MAX_ERROR_STRING];
    int error_string_length;

    printf("ccg_mpi_error_handler: entry\n");
    printf("ccg_mpi_error_handler: error_code = %d\n", *error_code);

    MPI_Error_string(*error_code, error_string, &error_string_length);
    error_string[error_string_length] = '\0';
    printf("ccg_mpi_error_handler: error_string = %s\n", error_string);
```

```
    /* If the MPI binding you are using supports additional information
       process the varargs here to retrieve it.
    */

    /* If you want to unwind to a different place in the call stack use
       setjmp/longjmp here to manipulate the return.
    */

    printf("ccg_mpi_error_handler: exit\n");
    exit(1);
}
```