```
            /* Finally, tell Node 2 what the id is for the new memory */
            send_to_node2(id);

            next_buf++;

      }

      /* Node 2 has finished, so Node 1 can demote the memory regions it
made available remotely,
            and then free the corresponding memory
      */

      for(int i=0; i<next_buf; i++)
      {
            /* Demote piece of remote memory to no longer be remotely
visible */
            mrapi_rmem_delete(buffers[i].handle, &status);

            // CHECK status FOR ERRORS - OMITTED

            /* Now actually free the local memory which corresponded to this
remote memory */
            free(buffers[i].pointer);

      }

      return 0;

};




/*-------------------------------------------------*/
/* Node 2 side of use case                         */
/*-------------------------------------------------*/


/* Helper functions for Node 2 - these are not part of MRAPI, and could be
   implemented using various appropriate mechanisms */

/* Function which uses some mechanism (e.g. MCAPI) to receive a remote
memory id from Node 1 */
mrapi_rmem_id_t receive_id_from_node1();

/* Function which uses some mechanism (e.g. MCAPI) to send an integer
message to Node 1 */
void send_message_to_node_1(int);

int node2_remote_memory_use_case_2()
{
```