```
void shutdown_lb()
{
  mcapi_status_t status;
      mcapi_request_t request;
  int i;

  // Shutdown each worker in turn
  for(i = 0; i < NUM_WORKERS; i++)
  {
    // Send an “invalid” packet to trigger worker shutdown
    PacketInfo invalid_work;
    invalid_work.is_valid = false;
    mcapi_pktchan_send(work_requests_out_hndl[i], &invalid_work,
sizeof(invalid_work), &status);
    CHECK_STATUS(status);

    // Close our ports; don't worry about errors (what would we do?)
    mcapi_pktchan_send_close_i(work_requests_out_hndl[i], &request,
&status);
    mcapi_sclchan_recv_close_i(acks_in_hndl[i], &request, &status);
  }
}
```

7.5.1.3 worker.c

```
/*
 * MCAPI 2.000 Packet Processing Use Case
 * worker.c
 *
 */

#include "common.h"
#include "mcapi.h"
#include <stdbool.h>

void some_function_or_another(PacketInfo* packet_info);

// Local port for incoming work requests from the load balancer.  We
// use a packet channel because each work request is a structure with
// multiple fields specifying the work to be done.
mcapi_port_t work_request_in;

// Local port for outgoing acks to the load balancer.  We use a scalar
// channel because each ack is a single word indicating the number of
// work items that have been completed.
mcapi_port_t ack_out;

// a local buffer for use by the incoming packet channel
char work_channel_buffer[1024];


// function declarations
void bind_channels(void);
void do_work(void);
void shutdown(void);

// The function called within each new worker thread.  This function
// binds takes an endpoint in the load balancer as its parameter so
// that it can communicate with the load balancer and create channels
// between the worker and the load balancer.
```