Dipesh Kafle

1. a). Create three programs, two of which are clients to a single server. Client1 will send a character to the server process. The server will decrement the letter to the next letter in the alphabet and send the result to client2. Client2 prints the letter it receives and then all the processes terminate.
b). Next write a socket program to enable client1 to send a float value to the server. The server process should increase the value of the number it receives by a power of 1.5. The server should print both the value it receives and the value that it sends. Client2 should print the value it receives from the server.
c). Send a C structure that includes data of type character, integer and float from client1 to the server. The server should change the values so that client2 receives a structure with entirely different data. It is not permitted that the data should be converted to any other data type before transmission.

Question 1 a)

Code

server.cpp

```
#include <arpa/inet.h>
#include <iostream>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

using namespace std;

const int BUF_SIZE = 512;

#define HANDLE_SEND_RECV_ERRORS(st)                                    \
  if ((st) == -1) {                                                    \
    perror("Error in send()/recv()");                                  \
```

```
    return 254;                                                                \
  } else if ((st) == 0) {                                                      \
    perror("Connection is closed because send/recv returned 0");               \
    return 255;                                                                \
  }

int get_port(const sockaddr_storage *addr) {
  if (addr->ss_family == AF_INET) {
    return ((struct sockaddr_in *)addr)->sin_port;
  } else {
    return ((struct sockaddr_in6 *)addr)->sin6_port;
  }
}

void *get_in_addr(const sockaddr_storage *addr) {
  if (addr->ss_family == AF_INET) {
    return &((struct sockaddr_in *)addr)->sin_addr;
  } else {
    return &((struct sockaddr_in6 *)addr)->sin6_addr;
  }
}

void print_conn_name(int fd, const sockaddr_storage *addr) {
  char buf[1000];
  inet_ntop(addr->ss_family, get_in_addr(addr), buf, sizeof(sockaddr_storage));
  int port = get_port(addr);
  printf("Connected to %s at port %d\n", buf, port);
}

int main(int argc, char **argv) {
  struct addrinfo hints, *result;
  const char *port_no = "3000";
  char buf[BUF_SIZE];
  memset(buf, 0, sizeof(buf));
  // stores status of various syscalls
  int status;

  if (argc >= 2)
    port_no = argv[1];

  memset(&hints, 0, sizeof(hints));
  hints.ai_family = AF_UNSPEC;
  hints.ai_socktype = SOCK_STREAM;
  hints.ai_flags = AI_PASSIVE;

  if ((status = getaddrinfo(NULL, port_no, &hints, &result)) != 0) {
```

```
  printf("error in getaddrinfo: %s\n", gai_strerror(status));
  return 1;
}

/*
 * i'm assuming the first node in ll that we get from getaddrinfo is
 * the right one
 */
int sock_fd =
    socket(result->ai_family, result->ai_socktype, result->ai_protocol);
if (sock_fd == -1) {
  printf("unable to open socket\n");
  return 2;
}
status = bind(sock_fd, result->ai_addr, result->ai_addrlen);
if (status == -1) {
  printf("unable to bind to the fd\n");
  return 3;
}

status = listen(sock_fd, 2);
if (status == -1) {
  printf("error in listening\n");
  return 4;
}

// first client
struct sockaddr_storage addr1;
socklen_t len = sizeof(addr1);
int conn1 = accept(sock_fd, (struct sockaddr *)&addr1, &len);
if (conn1 == -1) {
  printf("error in accepting connection 1\n");
  return 5;
}
print_conn_name(conn1, &addr1);
// I only want one character so the buffer size is 1 here
status = recv(conn1, buf, 1, 0);
HANDLE_SEND_RECV_ERRORS(status);
buf[1] = '\0';
printf("Received : %s\n", buf);
close(conn1);

// second client
struct sockaddr_storage addr2;
int conn2 = accept(sock_fd, (struct sockaddr *)&addr2, &len);
if (conn2 == -1) {
```

```
    printf("error in accepting connection 2\n");
    return 6;
  }
  // changing the buffer , decrementing the letter received
  print_conn_name(conn2, &addr2);
  buf[0]--;
  status = send(conn2, buf, strlen(buf), 0);
  HANDLE_SEND_RECV_ERRORS(status);
  printf("Sent: %s\n", buf);
  close(conn2);

  freeaddrinfo(result);
}
```

client1.cpp

```
#include <arpa/inet.h>
#include <iostream>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define HANDLE_SEND_RECV_ERRORS(st)                                          \
  if ((st) == -1) {                                                          \
    perror("Error in send()/recv()");                                        \
    exit(254);                                                               \
  } else if ((st) == 0) {                                                    \
    perror("Connection is closed because send/recv returned 0");             \
    exit(255);                                                               \
  }

int main(int argc, char **argv) {
  const char *buf = "c";
  int yes = 1;
  struct addrinfo hints, *res;
  int status;
  const char *port_no = "3000";
  int fd;
  struct sockaddr_storage *addr;
  socklen_t len = sizeof(
```

```
    std::remove_reference_t<decltype(*std::declval<decltype(addr)>())>);

printf("HI\n");
// 2nd argument is port number
if (argc >= 2)
  port_no = argv[1];

// 3rd argument is the letter to send
if (argc >= 3)
  buf = argv[2];

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;

if ((status = getaddrinfo(NULL, port_no, &hints, &res)) != 0) {
  printf("error in getaddrinfo: %s\n", gai_strerror(status));
  return 1;
}

struct addrinfo *result = res;

for (; result != nullptr; result = result->ai_next) {
  fd = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
  if (fd == -1) {
    perror("socket()");
    continue;
  }
  if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes)) == -1) {
    perror("setsockopt() failed: ");
    exit(10);
  }
  status = connect(fd, res->ai_addr, res->ai_addrlen);
  if (status == -1) {
    perror("connect()");
    continue;
  }
  break;
}

freeaddrinfo(res);
if (result == nullptr) {
  perror("connect() failed");
  exit(20);
}
```

```
  // send
  status = send(fd, buf, strlen(buf), 0);
  HANDLE_SEND_RECV_ERRORS(status);
  printf("Sent %s successfully\n", buf);
  fflush(stdout);
  close(fd);
  return 0;
}
```

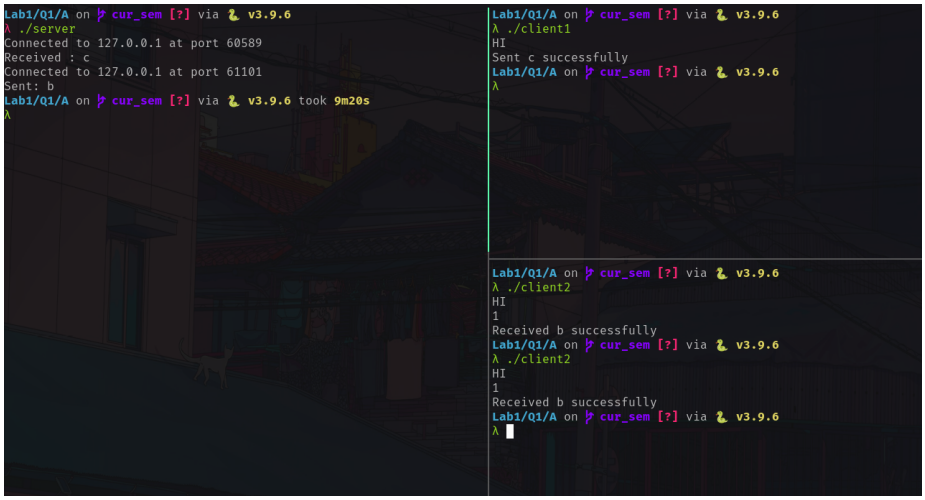
client2.cpp

```
#include <arpa/inet.h>
#include <iostream>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define HANDLE_SEND_RECV_ERRORS(st)                                          \
  if ((st) == -1) {                                                          \
    perror("Error in send()/recv()");                                        \
    exit(254);                                                               \
  } else if ((st) == 0) {                                                    \
    perror("Connection is closed because send/recv returned 0");             \
    exit(255);                                                               \
  }

int main(int argc, char **argv) {
  char buf[1000];
  int yes = 1;
  struct addrinfo hints, *res;
  int status;
  const char *port_no = "3000";
  int fd;
  struct sockaddr_storage *addr;
  socklen_t len = sizeof(
      std::remove_reference_t<decltype(*std::declval<decltype(addr)>())>);

  printf("HI\n");
  // 2nd argument is port number
```

```
if (argc >= 2)
  port_no = argv[1];

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;

if ((status = getaddrinfo(NULL, port_no, &hints, &res)) != 0) {
  printf("error in getaddrinfo: %s\n", gai_strerror(status));
  return 1;
}

struct addrinfo *result = res;

for (; result != nullptr; result = result->ai_next) {
  fd = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
  if (fd == -1) {
    perror("socket()");
    continue;
  }
  if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes)) == -1) {
    perror("setsockopt() failed: ");
    exit(10);
  }
  status = connect(fd, res->ai_addr, res->ai_addrlen);
  if (status == -1) {
    perror("connect()");
    continue;
  }
  break;
}

freeaddrinfo(res);
if (result == nullptr) {
  perror("connect() failed");
  exit(20);
}

status = recv(fd, buf, sizeof(buf), 0);
HANDLE_SEND_RECV_ERRORS(status);
printf("%d\n", status);
buf[status] = 0;
printf("Received %s successfully\n", buf);
fflush(stdout);
close(fd);
```

```
  return 0;
}
```

Output



Question 1 b)

Codes

server.cpp

```
#include <arpa/inet.h>
#include <iostream>
#include <math.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

using namespace std;

const int BUF_SIZE = 512;

#define HANDLE_SEND_RECV_ERRORS(st)                                    \
  if ((st) == -1) {                                                    \
    perror("Error in send()/recv()");                                  \
```

```
    return 254;                                                                \
  } else if ((st) == 0) {                                                      \
    perror("Connection is closed because send/recv returned 0");               \
    return 255;                                                                \
  }

int get_port(const sockaddr_storage *addr) {
  if (addr->ss_family == AF_INET) {
    return ((struct sockaddr_in *)addr)->sin_port;
  } else {
    return ((struct sockaddr_in6 *)addr)->sin6_port;
  }
}

void *get_in_addr(const sockaddr_storage *addr) {
  if (addr->ss_family == AF_INET) {
    return &((struct sockaddr_in *)addr)->sin_addr;
  } else {
    return &((struct sockaddr_in6 *)addr)->sin6_addr;
  }
}

void print_conn_name(int fd, const sockaddr_storage *addr) {
  char buf[1000];
  inet_ntop(addr->ss_family, get_in_addr(addr), buf, sizeof(sockaddr_storage));
  int port = get_port(addr);
  printf("Connected to %s at port %d\n", buf, port);
}

int main(int argc, char **argv) {
  struct addrinfo hints, *result;
  const char *port_no = "3000";
  char buf[BUF_SIZE];
  memset(buf, 0, sizeof(buf));
  // stores status of various syscalls
  int status;

  if (argc >= 2)
    port_no = argv[1];

  memset(&hints, 0, sizeof(hints));
  hints.ai_family = AF_UNSPEC;
  hints.ai_socktype = SOCK_STREAM;
  hints.ai_flags = AI_PASSIVE;

  if ((status = getaddrinfo(NULL, port_no, &hints, &result)) != 0) {
```

```
  printf("error in getaddrinfo: %s\n", gai_strerror(status));
  return 1;
}

/*
 * i'm assuming the first node in ll that we get from getaddrinfo is
 * the right one
 */
int sock_fd =
    socket(result->ai_family, result->ai_socktype, result->ai_protocol);
if (sock_fd == -1) {
  printf("unable to open socket\n");
  return 2;
}
status = bind(sock_fd, result->ai_addr, result->ai_addrlen);
if (status == -1) {
  printf("unable to bind to the fd\n");
  return 3;
}

status = listen(sock_fd, 2);
if (status == -1) {
  printf("error in listening\n");
  return 4;
}

// first client
struct sockaddr_storage addr1;
socklen_t len = sizeof(addr1);
int conn1 = accept(sock_fd, (struct sockaddr *)&addr1, &len);
if (conn1 == -1) {
  printf("error in accepting connection 1\n");
  return 5;
}
print_conn_name(conn1, &addr1);
status = recv(conn1, buf, sizeof(double), 0);
HANDLE_SEND_RECV_ERRORS(status);
printf("Received : %f\n", *(double *)buf);
close(conn1);

// second client
struct sockaddr_storage addr2;
double received = *(double *)buf;
received = pow(received, 1.5);
*(double *)buf = received;
int conn2 = accept(sock_fd, (struct sockaddr *)&addr2, &len);
```

```
  if (conn2 == -1) {
    printf("error in accepting connection 2\n");
    return 6;
  }
  print_conn_name(conn2, &addr2);
  buf[0]--;
  status = send(conn2, buf, sizeof(double), 0);
  HANDLE_SEND_RECV_ERRORS(status);
  printf("Sent: %f\n", *(double *)buf);
  close(conn2);

  freeaddrinfo(result);
}
```

client1.cpp

```
#include <arpa/inet.h>
#include <iostream>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define HANDLE_SEND_RECV_ERRORS(st)                                          \
  if ((st) == -1) {                                                          \
    perror("Error in send()/recv()");                                        \
    exit(254);                                                               \
  } else if ((st) == 0) {                                                    \
    perror("Connection is closed because send/recv returned 0");             \
    exit(255);                                                               \
  }

int main(int argc, char **argv) {
  double buf = 1.5;
  int yes = 1;
  struct addrinfo hints, *res;
  int status;
  const char *port_no = "3000";
  int fd;
  struct sockaddr_storage *addr;
  // 2nd argument is port number
  if (argc >= 2)
```

```
  port_no = argv[1];

// 3rd argument is the letter to send
if (argc >= 3)
  buf = atof(argv[2]);

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;

if ((status = getaddrinfo(NULL, port_no, &hints, &res)) != 0) {
  printf("error in getaddrinfo: %s\n", gai_strerror(status));
  return 1;
}

struct addrinfo *result = res;

for (; result != nullptr; result = result->ai_next) {
  fd = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
  if (fd == -1) {
    perror("socket()");
    continue;
  }
  if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes)) == -1) {
    perror("setsockopt() failed: ");
    exit(10);
  }
  status = connect(fd, res->ai_addr, res->ai_addrlen);
  if (status == -1) {
    perror("connect()");
    continue;
  }
  break;
}

freeaddrinfo(res);
if (result == nullptr) {
  perror("connect() failed");
  exit(20);
}

// send
status = send(fd, &buf, sizeof(buf), 0);
HANDLE_SEND_RECV_ERRORS(status);
printf("Sent %f successfully\n", buf);
```

```
  fflush(stdout);
  close(fd);
  return 0;
}
```

client2.cpp

```
#include <arpa/inet.h>
#include <iostream>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define HANDLE_SEND_RECV_ERRORS(st)                                                \
  if ((st) == -1) {                                                                \
    perror("Error in send()/recv()");                                              \
    exit(254);                                                                     \
  } else if ((st) == 0) {                                                          \
    perror("Connection is closed because send/recv returned 0");                   \
    exit(255);                                                                     \
  }

int main(int argc, char **argv) {
  double buf;
  int yes = 1;
  struct addrinfo hints, *res;
  int status;
  const char *port_no = "3000";
  int fd;
  struct sockaddr_storage *addr;

  // 2nd argument is port number
  if (argc >= 2)
    port_no = argv[1];

  memset(&hints, 0, sizeof(hints));
  hints.ai_family = AF_UNSPEC;
  hints.ai_socktype = SOCK_STREAM;
  hints.ai_flags = AI_PASSIVE;

  if ((status = getaddrinfo(NULL, port_no, &hints, &res)) != 0) {
```

```
    printf("error in getaddrinfo: %s\n", gai_strerror(status));
    return 1;
  }

  struct addrinfo *result = res;

  for (; result != nullptr; result = result->ai_next) {
    fd = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
    if (fd == -1) {
      perror("socket()");
      continue;
    }
    if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes)) == -1) {
      perror("setsockopt() failed: ");
      exit(10);
    }
    status = connect(fd, res->ai_addr, res->ai_addrlen);
    if (status == -1) {
      perror("connect()");
      continue;
    }
    break;
  }

  freeaddrinfo(res);
  if (result == nullptr) {
    perror("connect() failed");
    exit(20);
  }

  status = recv(fd, &buf, sizeof(buf), 0);
  HANDLE_SEND_RECV_ERRORS(status);
  printf("Received %f successfully\n", buf);
  fflush(stdout);
  close(fd);
  return 0;
}
```

Output

Question 1 c)

Codes

server.cpp

```
#include <arpa/inet.h>
#include <iostream>
#include <math.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

using namespace std;

const int BUF_SIZE = 512;

struct __attribute__((packed)) _data {
  char c;
  uint32_t x;
  double y;
  _data(char _c, uint32_t _x, double _y) : c(_c), x(_x), y(_y) {}
  string str() {
```

```
    return string("data( ") + c + " , " + to_string(x) + " , " + to_string(y) +
           " )";
  }
};

// handles byte ordering
// network to host=> ntoh
// host to network => hton
_data hton(_data other) { return _data(other.c, htonl(other.x), other.y); }
_data ntoh(_data other) { return _data(other.c, ntohl(other.x), other.y); }

#define HANDLE_SEND_RECV_ERRORS(st)                                          \
  if ((st) == -1) {                                                          \
    perror("Error in send()/recv()");                                        \
    return 254;                                                              \
  } else if ((st) == 0) {                                                    \
    perror("Connection is closed because send/recv returned 0");             \
    return 255;                                                              \
  }

int get_port(const sockaddr_storage *addr) {
  if (addr->ss_family == AF_INET) {
    return ((struct sockaddr_in *)addr)->sin_port;
  } else {
    return ((struct sockaddr_in6 *)addr)->sin6_port;
  }
}

void *get_in_addr(const sockaddr_storage *addr) {
  if (addr->ss_family == AF_INET) {
    return &((struct sockaddr_in *)addr)->sin_addr;
  } else {
    return &((struct sockaddr_in6 *)addr)->sin6_addr;
  }
}

void print_conn_name(int fd, const sockaddr_storage *addr) {
  char buf[1000];
  inet_ntop(addr->ss_family, get_in_addr(addr), buf, sizeof(sockaddr_storage));
  int port = get_port(addr);
  printf("Connected to %s at port %d\n", buf, port);
}

int main(int argc, char **argv) {
  struct addrinfo hints, *result;
  const char *port_no = "3000";
```

```
_data buf('c', 12345, 123.45);
// stores status of various syscalls
int status;

if (argc >= 2)
  port_no = argv[1];

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;

if ((status = getaddrinfo(NULL, port_no, &hints, &result)) != 0) {
  printf("error in getaddrinfo: %s\n", gai_strerror(status));
  return 1;
}

/*
 * i'm assuming the first node in ll that we get from getaddrinfo is
 * the right one
 */
int sock_fd =
    socket(result->ai_family, result->ai_socktype, result->ai_protocol);
if (sock_fd == -1) {
  printf("unable to open socket\n");
  return 2;
}
status = bind(sock_fd, result->ai_addr, result->ai_addrlen);
if (status == -1) {
  printf("unable to bind to the fd\n");
  return 3;
}

status = listen(sock_fd, 2);
if (status == -1) {
  printf("error in listening\n");
  return 4;
}

// first client
struct sockaddr_storage addr1;
socklen_t len = sizeof(addr1);
int conn1 = accept(sock_fd, (struct sockaddr *)&addr1, &len);
if (conn1 == -1) {
  printf("error in accepting connection 1\n");
  return 5;
```

```
  }
  print_conn_name(conn1, &addr1);
  // I only want one character so the buffer size is 1 here
  status = recv(conn1, &buf, sizeof(_data), 0);
  HANDLE_SEND_RECV_ERRORS(status);
  buf = ntoh(buf);
  printf("Received : %s\n", buf.str().c_str());
  close(conn1);

  // second client
  struct sockaddr_storage addr2;
  buf.c++;
  buf.x += 100;
  buf.y *= 10;
  int conn2 = accept(sock_fd, (struct sockaddr *)&addr2, &len);
  if (conn2 == -1) {
    printf("error in accepting connection 2\n");
    return 6;
  }
  print_conn_name(conn2, &addr2);
  buf = hton(buf);
  status = send(conn2, &buf, sizeof(_data), 0);
  HANDLE_SEND_RECV_ERRORS(status);
  buf = ntoh(buf);
  printf("Sent: %s\n", buf.str().c_str());
  close(conn2);

  freeaddrinfo(result);
}
```

client1.cpp

```
#include <arpa/inet.h>
#include <iostream>
#include <math.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
```

```cpp
using namespace std;

#define HANDLE_SEND_RECV_ERRORS(st)                                          \
  if ((st) == -1) {                                                          \
    perror("Error in send()/recv()");                                        \
    exit(254);                                                               \
  } else if ((st) == 0) {                                                    \
    perror("Connection is closed because send/recv returned 0");             \
    exit(255);                                                               \
  }

struct __attribute__((packed)) _data {
  char c;
  uint32_t x;
  double y;
  _data(char _c, uint32_t _x, double _y) : c(_c), x(_x), y(_y) {}
  string str() {
    return string("data( ") + c + " , " + to_string(x) + " , " + to_string(y) +
           " )";
  }
};

_data hton(_data other) { return _data(other.c, htonl(other.x), other.y); }
_data ntoh(_data other) { return _data(other.c, ntohl(other.x), other.y); }

int main(int argc, char **argv) {
  _data buf('c', 12345, 123.45);
  int yes = 1;
  struct addrinfo hints, *res;
  int status;
  const char *port_no = "3000";
  int fd;
  struct sockaddr_storage *addr;
  socklen_t len = sizeof(
      std::remove_reference_t<decltype(*std::declval<decltype(addr)>())>);

  printf("HI\n");
  // 2nd argument is port number
  if (argc >= 2)
    port_no = argv[1];

  memset(&hints, 0, sizeof(hints));
  hints.ai_family = AF_UNSPEC;
  hints.ai_socktype = SOCK_STREAM;
  hints.ai_flags = AI_PASSIVE;
```

```
  if ((status = getaddrinfo(NULL, port_no, &hints, &res)) != 0) {
    printf("error in getaddrinfo: %s\n", gai_strerror(status));
    return 1;
  }

  struct addrinfo *result = res;

  for (; result != nullptr; result = result->ai_next) {
    fd = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
    if (fd == -1) {
      perror("socket()");
      continue;
    }
    if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes)) == -1) {
      perror("setsockopt() failed: ");
      exit(10);
    }
    status = connect(fd, res->ai_addr, res->ai_addrlen);
    if (status == -1) {
      perror("connect()");
      continue;
    }
    break;
  }

  freeaddrinfo(res);
  if (result == nullptr) {
    perror("connect() failed");
    exit(20);
  }

  // send
  buf = hton(buf);
  status = send(fd, &buf, sizeof(buf), 0);
  HANDLE_SEND_RECV_ERRORS(status);
  buf = ntoh(buf);
  printf("Sent %s successfully\n", buf.str().c_str());
  fflush(stdout);
  close(fd);
  return 0;
}
```

client2.cpp

```
#include <arpa/inet.h>
#include <iostream>
```

```cpp
#include <math.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

using namespace std;

#define HANDLE_SEND_RECV_ERRORS(st)                                                \
  if ((st) == -1) {                                                                \
    perror("Error in send()/recv()");                                              \
    exit(254);                                                                     \
  } else if ((st) == 0) {                                                          \
    perror("Connection is closed because send/recv returned 0");                   \
    exit(255);                                                                     \
  }

struct __attribute__((packed)) _data {
  char c;
  uint32_t x;
  double y;
  _data(char _c, uint32_t _x, double _y) : c(_c), x(_x), y(_y) {}
  string str() {
    return string("data( ") + c + " , " + to_string(x) + " , " + to_string(y) +
           " )";
  }
};

_data hton(_data other) { return _data(other.c, htonl(other.x), other.y); }
_data ntoh(_data other) { return _data(other.c, ntohl(other.x), other.y); }

int main(int argc, char **argv) {
  _data buf('c', 12345, 123.45);
  int yes = 1;
  struct addrinfo hints, *res;
  int status;
  const char *port_no = "3000";
  int fd;
  struct sockaddr_storage *addr;
  socklen_t len = sizeof(
      std::remove_reference_t<decltype(*std::declval<decltype(addr)>())>);
```

```
printf("HI\n");
// 2nd argument is port number
if (argc >= 2)
  port_no = argv[1];

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;

if ((status = getaddrinfo(NULL, port_no, &hints, &res)) != 0) {
  printf("error in getaddrinfo: %s\n", gai_strerror(status));
  return 1;
}

struct addrinfo *result = res;

for (; result != nullptr; result = result->ai_next) {
  fd = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
  if (fd == -1) {
    perror("socket()");
    continue;
  }
  if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes)) == -1) {
    perror("setsockopt() failed: ");
    exit(10);
  }
  status = connect(fd, res->ai_addr, res->ai_addrlen);
  if (status == -1) {
    perror("connect()");
    continue;
  }
  break;
}

freeaddrinfo(res);
if (result == nullptr) {
  perror("connect() failed");
  exit(20);
}

status = recv(fd, &buf, sizeof(buf), 0);
HANDLE_SEND_RECV_ERRORS(status);
buf = ntoh(buf);
printf("Received %s successfully\n", buf.str().c_str());
```

```
  fflush(stdout);
  close(fd);
  return 0;
}
```

Output

2. Create a database of parts for an automobile that includes part number, part name, part price, part quantity, account number of user, and a part description. Write procedures on the server that search for a part, obtain a part name from a part number, give the quantity of parts available, order a part, and obtain a list of subpart numbers. Create a set of commands (atleast 5) on the client that the server can interpret in order answer client inquiries.

Question 2

Codes

server.cpp

```
#include <arpa/inet.h>
#include <iostream>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
```

```
using namespace std;

#define MAXPENDING 5 /* Maximum outstanding connection requests */
#define NAMESIZE 10
#define DESCSIZE 10
#define BUFSIZE 30
#define PARTSSIZE 5
#define CUSTSIZE 5
#define SUBPARTSSIZE 2

#define HANDLE_SEND_RECV_ERRORS(st)                                          \
  if ((st) == -1) {                                                          \
    perror("Error in send()/recv()");                                        \
    exit(254);                                                               \
  } else if ((st) == 0) {                                                    \
    perror("Connection is closed because send/recv returned 0");             \
    exit(255);                                                               \
  }

typedef struct {
  int number;
  char name[NAMESIZE];
  int price;
  char desc[DESCSIZE];
  int qty;
  int subparts[SUBPARTSSIZE];
} part_t;

typedef struct {
  int partno;
  int qty;
} order_t;

typedef struct {
  int acno;
  order_t ord;
} customer_t;

part_t parts[PARTSSIZE];
customer_t customers[CUSTSIZE];

void populate() {
  // 0 - wheel
  // 1 - tyre
  // 2 - door
  // 3 - engine
```

```
// 4 - shaft
strcpy(parts[0].name, "Wheel");
strcpy(parts[0].desc, "Car Wheel");
parts[0].number = 0;
parts[0].price = 5000;
parts[0].qty = 100;
parts[0].subparts[0] = 1; // tyre
parts[0].subparts[1] = -1;

strcpy(parts[1].name, "Tyre");
strcpy(parts[1].desc, "Rubber Tyre");
parts[1].number = 1;
parts[1].price = 2000;
parts[1].qty = 200;
parts[1].subparts[0] = -1;
parts[1].subparts[1] = -1;

strcpy(parts[2].name, "Door");
strcpy(parts[2].desc, "Metal Door");
parts[2].number = 2;
parts[2].price = 10000;
parts[2].qty = 150;
parts[2].subparts[0] = -1;
parts[2].subparts[1] = -1;

strcpy(parts[3].name, "Engine");
strcpy(parts[3].desc, "Petrol Engine");
parts[3].number = 3;
parts[3].price = 50000;
parts[3].qty = 20;
parts[3].subparts[0] = 4; // shaft
parts[3].subparts[1] = -1;

strcpy(parts[4].name, "Shaft");
strcpy(parts[4].desc, "Crankshaft");
parts[4].number = 4;
parts[4].price = 7500;
parts[4].qty = 50;
parts[4].subparts[0] = -1;
parts[4].subparts[1] = -1;

customers[0].acno = 0;
customers[1].acno = 1;
customers[2].acno = 2;
customers[3].acno = 3;
customers[4].acno = 4;
```

```
}

void die_with_error(const char *errorMessage) {
  perror(errorMessage);
  exit(1);
}

int recv_int(int sock) {
  int res = -1;
  int recv_msg_size;

  if ((recv_msg_size = recv(sock, &res, sizeof(res), 0)) < 0)
    die_with_error("recv() failed");
  return res;
}

void send_str(int sock, char *msg_buf) {
  int status = send(sock, msg_buf, strlen(msg_buf), 0);
  HANDLE_SEND_RECV_ERRORS(status);
  if (status != strlen(msg_buf))
    die_with_error("Couldn't send() fully");
}

void send_int(int sock, int payload) {
  int status = send(sock, &payload, sizeof(payload), 0);
  HANDLE_SEND_RECV_ERRORS(status);
  if (status != sizeof(payload))
    die_with_error("Couldn't send() fully");
}

void recv_str(int sock, char buf[BUFSIZE]) {
  int recv_msg_size;

  recv_msg_size = recv(sock, buf, BUFSIZE, 0);
  HANDLE_SEND_RECV_ERRORS(recv_msg_size);
}

void handle_client(int client_socket, int queryno) {
  if (queryno < 0 || queryno > 5)
    return;
  // 1: obtain part name from part number
  // 2: give quantity of parts avaliable
  // 3: order a part
  // 4: obtain list of subpart numbers
  // 5: search for part
```

```
printf("Got query %d\n", queryno);

if (queryno == 1) {
  char name[NAMESIZE];
  strcpy(name, "Invalid");
  int partno = recv_int(client_socket);

  if (partno < PARTSSIZE)
    strcpy(name, parts[partno].name);

  send_str(client_socket, name);
} else if (queryno == 2) {
  int partno = recv_int(client_socket);
  int result = 0;

  if (partno < PARTSSIZE)
    result = parts[partno].qty;

  send_int(client_socket, result);
} else if (queryno == 3) {
  int custid = recv_int(client_socket);
  int partno = recv_int(client_socket);
  int qty = recv_int(client_socket);

  char response[BUFSIZE];
  strcpy(response, "Failed");

  if (partno < PARTSSIZE && parts[partno].qty >= qty) {
    if (custid < CUSTSIZE) {
      customers[custid].ord.partno = partno;
      customers[custid].ord.qty = qty;

      parts[partno].qty -= qty;

      strcpy(response, "Success");
    }
  }

  send_str(client_socket, response);
} else if (queryno == 4) {
  int partno = recv_int(client_socket);
  char response[BUFSIZE];
  strcpy(response, "Empty");

  if (partno < PARTSSIZE) {
    int num = 0;
```

```
      for (int i = 0; i < PARTSSIZE && parts[partno].subparts[i] != -1; ++i)
        ++num;

      send_int(client_socket, num);

      for (int i = 0; i < num; ++i) {
        strcpy(response, parts[parts[partno].subparts[i]].name);
        send_str(client_socket, response);
      }
    }
  } else if (queryno == 5) {
    char partname[BUFSIZE];
    recv_str(client_socket, partname);

    int partno = -1;
    for (int i = 0; i < PARTSSIZE; ++i)
      if (strcmp(partname, parts[i].name) == 0)
        partno = i;

    send_int(client_socket, partno);
  }
}

int main(int argc, char *argv[]) {
  int server_socket;
  int client_socket;
  struct sockaddr_storage client_addr;
  unsigned short port;
  unsigned int client_len;

  if (argc != 2) {
    fprintf(stderr, "Usage:  %s <Port>\n", argv[0]);
    exit(1);
  }

  port = atoi(argv[1]);

  populate();

  struct addrinfo hints, *res;
  memset(&hints, 0, sizeof(hints));
  hints.ai_family = AF_UNSPEC;
  hints.ai_socktype = SOCK_STREAM;
  hints.ai_flags = AI_PASSIVE;

  if (getaddrinfo(NULL, to_string(port).c_str(), &hints, &res) == -1) {
```

```
    die_with_error("getaddrinfo()");
  }

  if ((server_socket =
          socket(res->ai_family, res->ai_socktype, res->ai_protocol)) < 0)
    die_with_error("socket() failed");

  /* Bind to the local address */
  if (bind(server_socket, res->ai_addr, res->ai_addrlen) < 0)
    die_with_error("bind() failed");

  // frees heap memory
  freeaddrinfo(res);

  /* Mark the socket so it will listen for incoming connections */
  if (listen(server_socket, MAXPENDING) < 0)
    die_with_error("listen() failed");

  int queryno = 0;
  while (queryno != -1) {
    client_len = sizeof(client_addr);

    if ((client_socket = accept(server_socket, (struct sockaddr *)&client_addr,
                                &client_len)) < 0)
      die_with_error("accept() failed");

    while (queryno != -1) {
      queryno = recv_int(client_socket);
      handle_client(client_socket, queryno);
    }
  }

  close(client_socket);

  return 0;
}
```

client.cpp

```
#include <arpa/inet.h>
#include <iostream>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

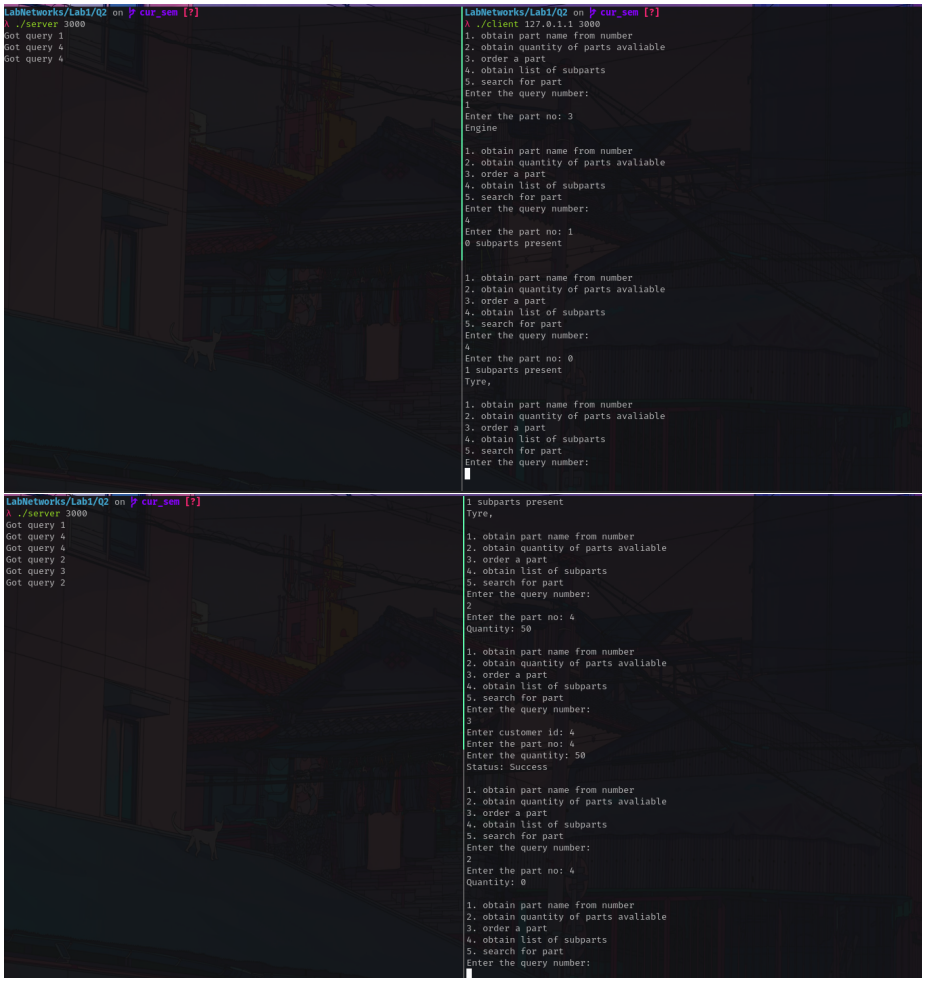
```cpp
#include <string>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
using namespace std;

#define HANDLE_SEND_RECV_ERRORS(st)                                          \
  if ((st) == -1) {                                                          \
    perror("Error in send()/recv()");                                        \
    exit(254);                                                               \
  } else if ((st) == 0) {                                                    \
    perror("Connection is closed because send/recv returned 0");             \
    exit(255);                                                               \
  }

#define BUFSIZE 30

void die_with_error(const char *errorMessage) {
  perror(errorMessage);
  exit(1);
}

int recv_int(int sock) {
  int res = -1;
  int recv_msg_size;

  recv_msg_size = recv(sock, &res, sizeof(res), 0);
  HANDLE_SEND_RECV_ERRORS(recv_msg_size);
  return res;
}

void send_str(int sock, char msg_buf[BUFSIZE]) {
  int len = send(sock, msg_buf, strlen(msg_buf), 0);
  HANDLE_SEND_RECV_ERRORS(len);
  if (len != strlen(msg_buf))
    die_with_error("Couldn't send() fully");
}

void send_int(int sock, int payload) {
  int len = send(sock, &payload, sizeof(payload), 0);
  HANDLE_SEND_RECV_ERRORS(len);
  if (len != sizeof(payload))
    die_with_error("Couldn't send() fully");
}

void recv_str(int sock, char buf[BUFSIZE]) {
```

```
  int recv_msg_size;

  recv_msg_size = recv(sock, buf, BUFSIZE, 0);
  HANDLE_SEND_RECV_ERRORS(recv_msg_size);
  buf[recv_msg_size] = '\0';
}

void handle(int sock, int queryno) {
  if (queryno < 0 || queryno > 5)
    return;

  if (queryno == 1) {
    int partno;
    printf("Enter the part no: ");
    scanf("%d", &partno);

    send_int(sock, partno);
    char partname[BUFSIZE];
    recv_str(sock, partname);

    printf("%s\n\n", partname);
  } else if (queryno == 2) {
    int partno;
    printf("Enter the part no: ");
    scanf("%d", &partno);

    send_int(sock, partno);
    int qty = recv_int(sock);

    printf("Quantity: %d\n\n", qty);
  } else if (queryno == 3) {
    int custid, partno, qty;

    printf("Enter customer id: ");
    scanf("%d", &custid);

    printf("Enter the part no: ");
    scanf("%d", &partno);

    printf("Enter the quantity: ");
    scanf("%d", &qty);

    send_int(sock, custid);
    send_int(sock, partno);
    send_int(sock, qty);
```

```
    char status[BUFSIZE];
    recv_str(sock, status);

    printf("Status: %s\n\n", status);
  } else if (queryno == 4) {
    int partno;
    printf("Enter the part no: ");
    scanf("%d", &partno);

    send_int(sock, partno);
    int num = recv_int(sock);

    printf("%d subparts present\n", num);

    char response[BUFSIZE];
    for (int i = 0; i < num; ++i) {
      recv_str(sock, response);
      printf("%s, ", response);
    }
    printf("\n\n");
  } else if (queryno == 5) {
    char partname[BUFSIZE];

    printf("Enter the part name: ");
    scanf("%s", partname);

    send_str(sock, partname);
    int partno = recv_int(sock);

    if (partno == -1)
      printf("Does not exist\n\n");
    else
      printf("Part no: %d\n\n", partno);
  }
}

int main(int argc, char *argv[]) {
  int sock;
  unsigned short server_port;
  char *server_ip;

  if (argc != 3) {
    fprintf(stderr, "Usage: %s <Server IP> <Server Port>\n", argv[0]);
    exit(1);
  }
```

```
  server_ip = argv[1];
  server_port = atoi(argv[2]);

  struct addrinfo hints, *res;
  memset(&hints, 0, sizeof(hints));
  hints.ai_family = AF_UNSPEC;
  hints.ai_socktype = SOCK_STREAM;

  int status =
      getaddrinfo(server_ip, to_string(server_port).c_str(), &hints, &res);
  if (status == -1)
    die_with_error("getaddrinfo() failed");

  if ((sock = socket(res->ai_family, res->ai_socktype, res->ai_protocol)) < 0)
    die_with_error("socket() failed");

  if (connect(sock, res->ai_addr, res->ai_addrlen) < 0)
    die_with_error("connect() failed");

  freeaddrinfo(res);

  int queryno = 0;
  while (queryno != -1) {
    printf("%s\n%s\n%s\n%s\n%s\n%s\n", "1. obtain part name from number",
           "2. obtain quantity of parts avaliable", "3. order a part",
           "4. obtain list of subparts", "5. search for part",
           "Enter the query number: ");

    scanf("%d", &queryno);
    send_int(sock, queryno);
    handle(sock, queryno);
  }

  close(sock);
  exit(0);
}
```

Output



Question 3

3. Send datagrams with two arrays of integers (only even numbers) to a server. The server should check the data, whether there are odd and/or fraction numbers. If it is not, the server sums the elements of each array and puts the sum in a third array that is returned to the client. If the server discovers that the arrays have erroneous then the server does not reply. A timeout period should be established by the client such that retransmission occurs after the period expires.

Code

server.cpp

```cpp
#include <algorithm>
#include <arpa/inet.h>
#include <iostream>
#include <netdb.h>
#include <netinet/in.h>
#include <numeric>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

using namespace std;

#define BUF_SIZE 32

#define HANDLE_SEND_RECV_ERRORS(st)                                                \
  if ((st) == -1) {                                                                \
    perror("Error in send()/recv()");                                              \
    exit(254);                                                                     \
  } else if ((st) == 0) {                                                          \
    perror("Connection is closed because send/recv returned 0");                   \
    exit(255);                                                                     \
  }

int get_port(const sockaddr_storage *addr) {
  if (addr->ss_family == AF_INET) {
    return ((struct sockaddr_in *)addr)->sin_port;
  } else {
    return ((struct sockaddr_in6 *)addr)->sin6_port;
  }
}

void *get_in_addr(const sockaddr_storage *addr) {
  if (addr->ss_family == AF_INET) {
    return &((struct sockaddr_in *)addr)->sin_addr;
  } else {
    return &((struct sockaddr_in6 *)addr)->sin6_addr;
  }
}

void print_conn_name(const sockaddr_storage *addr) {
  char buf[1000];
  inet_ntop(addr->ss_family, get_in_addr(addr), buf, sizeof(sockaddr_storage));
  int port = get_port(addr);
```

```
  printf("Connected to %s at port %d\n", buf, port);
}

void die_with_error(const char *message, int err_code = 1) {
  perror(message);
  exit(err_code);
}

void ntoh_arr(int *arr, int n) {
  transform(arr, arr + n, arr, [](int x) { return (int)ntohl(x); });
}
void hton_arr(int *arr, int n) {
  transform(arr, arr + n, arr, [](int x) { return (int)htonl(x); });
}

bool check_integrity(int *arr, int n) {
  return all_of(arr, arr + n, [](int x) { return x % 2 == 0; });
}

int sum_all_elems(int *arr, int n) { return accumulate(arr, arr + n, 0); }

void print_nums(int *buf, int *buf3) {
  bool make_wrong = false;
  printf("Buf1: ");
  for (int i = 1; (i * sizeof(int)) <= BUF_SIZE; i++) {
    printf("%d ", buf[i - 1]);
  }
  printf("\n");
  printf("Buf2: ");
  for (int i = 1; i * sizeof(int) <= BUF_SIZE; i++) {
    printf("%d ", buf2[i - 1]);
  }
  printf("\n");
}

int main(int argc, char **argv) {
  char buf[BUF_SIZE];
  char buf2[BUF_SIZE];
  struct addrinfo hints, *res;
  struct sockaddr_storage client_addr, client_addr2;
  socklen_t len = sizeof(client_addr);
  int status;
  int yes = 1;
  const char *port = "3000";
  int fd, num_bytes;
```

```
if (argc >= 2)
  port = argv[1];

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_DGRAM;
hints.ai_flags = AI_PASSIVE;

status = getaddrinfo(NULL, port, &hints, &res);
if (status == -1)
  die_with_error("getaddrinfo() failed");

struct addrinfo *result = res;
for (; result != nullptr; result = result->ai_next) {
  fd = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
  if (fd == -1)
    continue;
  if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes)) == -1)
    die_with_error("setsockopt() failed");

  status = bind(fd, result->ai_addr, result->ai_addrlen);
  if (status == -1)
    continue;
  break;
}
freeaddrinfo(res);
if (result == NULL)
  die_with_error("binding failed");

while (true) {
  // arr1
  num_bytes = recvfrom(fd, (void *)buf, BUF_SIZE, 0,
                       (struct sockaddr *)&client_addr, &len);
  HANDLE_SEND_RECV_ERRORS(num_bytes);
  int recvd_buf_size1 = num_bytes / sizeof(int);
  // handle endianness
  ntoh_arr((int *)buf, recvd_buf_size1);

  // arr2
  num_bytes = recvfrom(fd, (void *)buf2, BUF_SIZE, 0,
                       (struct sockaddr *)&client_addr, &len);
  HANDLE_SEND_RECV_ERRORS(num_bytes);
  int recvd_buf_size2 = num_bytes / sizeof(int);
  // handle endianness
  ntoh_arr((int *)buf2, recvd_buf_size2);
  print_conn_name(&client_addr);
```

```
    print_nums((int *)buf, (int *)buf2);

    if (check_integrity((int *)buf, recvd_buf_size1) &&
        check_integrity((int *)buf2, recvd_buf_size2)) {
      int sm1 = sum_all_elems((int *)buf, recvd_buf_size1);
      int sm2 = sum_all_elems((int *)buf2, recvd_buf_size2);
      int sm = sm1 + sm2;
      printf("%d\n", sm);
      sm = htonl(sm);
      status = sendto(fd, &sm, sizeof(sm), 0,
                      (const struct sockaddr *)&client_addr, len);
      HANDLE_SEND_RECV_ERRORS(status);
      sm = ntohl(sm);
      printf("Sent %d\n", sm);
    }
  }
}
```

client.cpp

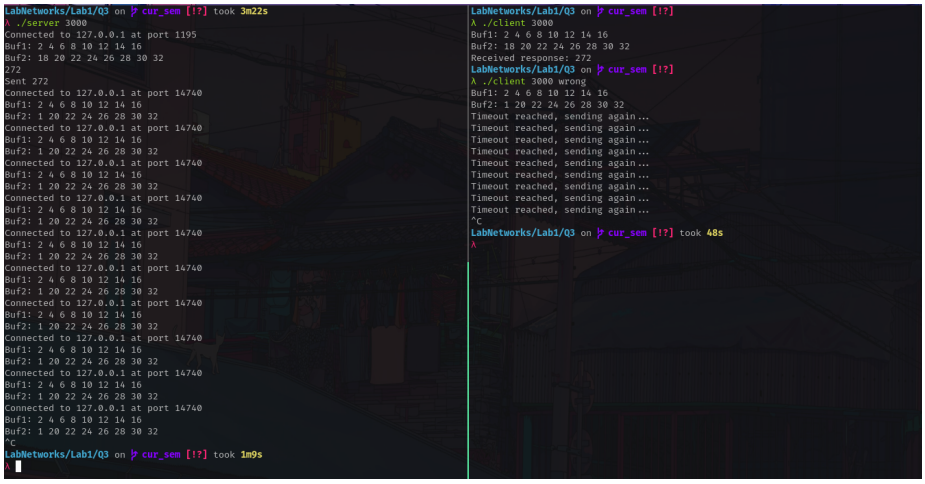
```
#include <algorithm>
#include <arpa/inet.h>
#include <iostream>
#include <netdb.h>
#include <netinet/in.h>
#include <numeric>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

using namespace std;

#define BUF_SIZE 32

#define HANDLE_SEND_RECV_ERRORS(st)                                          \
  if ((st) == -1) {                                                          \
    perror("Error in send()/recv()");                                        \
    exit(254);                                                               \
  } else if ((st) == 0) {                                                    \
    perror("Connection is closed because send/recv returned 0");             \
    exit(255);                                                               \
  }
```

```
void die_with_error(const char *message, int err_code = 1) {
  perror(message);
  exit(err_code);
}

void ntoh_arr(int *arr, int n) {
  transform(arr, arr + n, arr, [](int x) { return (int)ntohl(x); });
}
void hton_arr(int *arr, int n) {
  transform(arr, arr + n, arr, [](int x) { return (int)htonl(x); });
}

void populate(int *buf, int *buf2, bool make_wrong = false) {
  int num = 2;
  for (int i = 1; i * sizeof(int) <= BUF_SIZE; i++) {
    buf[i - 1] = num;
    num += 2;
  }
  for (int i = 1; i * sizeof(int) <= BUF_SIZE; i++) {
    buf2[i - 1] = num;
    num += 2;
  }
  if (make_wrong)
    buf2[0] = 1;
}

void print_nums(int *buf, int *buf2) {
  printf("Buf1: ");
  for (int i = 1; (i * sizeof(int)) <= BUF_SIZE; i++) {
    printf("%d ", buf[i - 1]);
  }
  printf("\n");
  printf("Buf2: ");
  for (int i = 1; i * sizeof(int) <= BUF_SIZE; i++) {
    printf("%d ", buf2[i - 1]);
  }
  printf("\n");
}

int main(int argc, char **argv) {
  char buf[BUF_SIZE];
  char buf2[BUF_SIZE];
  bool make_wrong = false;
  if (argc >= 3) {
    if (string(argv[2]) == string("wrong"))
      make_wrong = true;
```

```
}
populate((int *)buf, (int *)buf2, make_wrong);
print_nums((int *)buf, (int *)buf2);
int sz = BUF_SIZE / sizeof(int);
hton_arr((int *)buf, sz);
hton_arr((int *)buf2, sz);

struct addrinfo hints, *res;
struct sockaddr_storage server_addr;
socklen_t len = sizeof(server_addr);
int status;
const char *port = "3000";
int fd, num_bytes;

if (argc >= 2)
  port = argv[1];

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_DGRAM;
hints.ai_flags = AI_PASSIVE;

status = getaddrinfo(NULL, port, &hints, &res);
if (status == -1)
  die_with_error("getaddrinfo() failed");

struct addrinfo *result = res;
for (; result != nullptr; result = result->ai_next) {
  fd = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
  if (fd == -1)
    continue;

  // https://stackoverflow.com/questions/13547721/udp-socket-set-timeout
  struct timeval t;
  t.tv_sec = 5;
  t.tv_usec = 0;
  if (setsockopt(fd, SOL_SOCKET, SO_RCVTIMEO, &t, sizeof(t)) == -1)
    die_with_error("setsockopt() failed");
  break;
}
if (result == NULL)
  die_with_error("binding failed");

// first buffer sent
int sent_bytes =
    sendto(fd, buf, BUF_SIZE, 0, result->ai_addr, result->ai_addrlen);
```

```
  HANDLE_SEND_RECV_ERRORS(sent_bytes);

  // second buffer sent
  sent_bytes =
      sendto(fd, buf2, BUF_SIZE, 0, result->ai_addr, result->ai_addrlen);
  HANDLE_SEND_RECV_ERRORS(sent_bytes);

  int bytes_recvd;
  int ans = 0;
  while ((bytes_recvd = recvfrom(fd, &ans, sizeof(ans), 0, result->ai_addr,
                                 &result->ai_addrlen)) < 0) {
    if (bytes_recvd == 0) {
      perror("Connection closed");
      exit(20);
    }
    printf("Timeout reached, sending again...\n");
    sent_bytes =
        sendto(fd, buf, BUF_SIZE, 0, result->ai_addr, result->ai_addrlen);
    HANDLE_SEND_RECV_ERRORS(sent_bytes);

    // second buffer sent
    sent_bytes =
        sendto(fd, buf2, BUF_SIZE, 0, result->ai_addr, result->ai_addrlen);
    HANDLE_SEND_RECV_ERRORS(sent_bytes);
  }
  ans = ntohl(ans);
  printf("Received response: %d\n", ans);
  close(fd);
  freeaddrinfo(res);
}
```

Output


LabNetworks/Lab1/Q3 on cur_sem [!?] took 3m22s
λ ./server 3000
Connected to 127.0.0.1 at port 1195
Buf1: 2 4 6 8 10 12 14 16
Buf2: 18 20 22 24 26 28 30 32
272
Sent 272
Connected to 127.0.0.1 at port 14740
Buf1: 2 4 6 8 10 12 14 16
Buf2: 1 20 22 24 26 28 30 32
Connected to 127.0.0.1 at port 14740
Buf1: 2 4 6 8 10 12 14 16
Buf2: 1 20 22 24 26 28 30 32
Connected to 127.0.0.1 at port 14740
Buf1: 2 4 6 8 10 12 14 16
Buf2: 1 20 22 24 26 28 30 32
Connected to 127.0.0.1 at port 14740
Buf1: 2 4 6 8 10 12 14 16
Buf2: 1 20 22 24 26 28 30 32
Connected to 127.0.0.1 at port 14740
Buf1: 2 4 6 8 10 12 14 16
Buf2: 1 20 22 24 26 28 30 32
Connected to 127.0.0.1 at port 14740
Buf1: 2 4 6 8 10 12 14 16
Buf2: 1 20 22 24 26 28 30 32
Connected to 127.0.0.1 at port 14740
Buf1: 2 4 6 8 10 12 14 16
Buf2: 1 20 22 24 26 28 30 32
Connected to 127.0.0.1 at port 14740
Buf1: 2 4 6 8 10 12 14 16
Buf2: 1 20 22 24 26 28 30 32
^C
LabNetworks/Lab1/Q3 on cur_sem [!?] took 1m9s
λ
LabNetworks/Lab1/Q3 on cur_sem [!?]
λ ./client 3000
Buf1: 2 4 6 8 10 12 14 16
Buf2: 18 20 22 24 26 28 30 32
Received response: 272
LabNetworks/Lab1/Q3 on cur_sem [!?]
λ ./client 3000 wrong
Buf1: 2 4 6 8 10 12 14 16
Buf2: 1 20 22 24 26 28 30 32
Timeout reached, sending again...
Timeout reached, sending again...
Timeout reached, sending again...
Timeout reached, sending again...
Timeout reached, sending again...
Timeout reached, sending again...
Timeout reached, sending again...
Timeout reached, sending again...
Timeout reached, sending again...
^C
LabNetworks/Lab1/Q3 on cur_sem [!?] took 48s
λ