רשתות תרגיל מעשי 2

מגישים:

דוד ליפשיץ, ת.ז. 206107740, אימייל: davidl2@mail.tau.ac.il

דניס גוליאנוב, ת.ז. 320882988, אימייל: denisg@mail.tau.ac.il

תוכן עניינים:

תיאור הפרוטוקול:

הודעת שהשרת שולח ללקוחות:

1. **כאשר לקוח חדש מתחבר לשרת, נשלחת ללקוח הודעת פתיחה.**

1א. **אם החיבור אושר**, כלומר, בשרת יש מקום ללקוח, תשלח ההודעה הבאה:

[byte 1] [byte 2] [byte 3] [byte 4] [byte 5]

בייט 1 – במקרה זה, החיבור אושר, יקבל את הערך 1.

בייט 2- מכיל ערך isMisere (0 אם false, 1 אם true)

בייט 3- ערך p (מספר המשחקים, תחת ההנחה שp לא עובר את הערך 9, מספיק בייט אחד)

בייט 4- מספר לקוח (תחת ההנחה שהמספר לא עובר את 9 (בונוס), מספיק בייט אחד)

בייט 5 – מקבל 1 אם הלקוח יכול לשחק, מקבל 0 אם הוא רק צופה

1ב. **אם החיבור לא אושר**, כלומר כבר יש 9 לקוחות מחוברים לשרת, אזי נשלח ללקוח שמנסה להתחבר בייט בודד:

[byte 1]

עם הערך 0.

הערה: על הלקוח לפרש את הערכים כ-unsigned char.

הודעות 2 והילך נפתחות בבייט שמסמל את סוג ההודעה

2. **הודעת עדכון מספרי איברים בכל heap:**

2א. הודעת עדכון של מספר האיברים בכל ערימה, שהשרת שולח ללקוחות:

[Message type byte] [short 1] [short 2] [short 3] [short 4] [game over byte]

(כל הגדלים מפורשים כ-unsigned)

game over byte –

מכיל 0 אם המשחק לא הסתיים, מכיל 1 אם המשחק הסתיים (כל הערימות הן 0)

Message type byte–

עבור הודעה מסוג זו, נשמור את הערך 1.

Short 1, 2, 3, 4 –

מכילים את גדלי הערימות, כאשר ה-short1 מייצג את המחסנית השמאלית ביותר (A). מאחר וגודל המחסניות חסום על ידי 1500, מספיק short כדי לשמור את מספר האיברים בכל ערימה.

2ב. אם game over byte == 1, **דהיינו המשחק נגמר**, השרת שולח בייט נוסף לכל הלקוחות המשחקים בלבד (לא שולח את הבייט הזה לצופים), שמכיל 0 אם השחקן אליו נשלח הבייט הפסיד, ומכיל 1 אם ניצח.

3. הודעת תור שחקן

כאשר השרת רוצה לאפשר לשחקן מסוים לבצע מהלך, הוא שולח לו בייט בודד, עם הערך 2. דהיינו, ההודעה היא [Message type byte] בודד, עם הערך 2.

4. הודעת אישור מהלך שחקן [Positive ACK]

כאשר השרת רוצה לאשר מהלך שביצע שחקן, הוא שולח לו בייט בודד, עם הערך 3. דהיינו, ההודעה היא

[Message type byte]

בודד, עם הערך 3.

5. הודעת אי אישור מהלך שחקן [Negative ACK]

כאשר השרת רוצה להודיע לשחקן כי המהלך שהוא שלח אינו חוקי (או כלל אינו התור שלו ללכת), הוא שולח לו בייט בודד, עם הערך 4. דהיינו, ההודעה היא

[Message type byte]

בייט בודד, עם הערך 4.

6. הודעת MSG משחקן אחר

[לפי המתרגל אפשר להניח שגודל כל הודעה חסום, נניח ע"י 255 בתים ולכן המספר נכנס לתוך בייט בודד]

6א. כאשר השרת רוצה לשלוח הודעה ללקוח x מלקוח y, הוא שולח ללקוח x את ההודעה הבאה:

[Message type byte] [sender id byte] [destination id byte][length byte][sequence of bytes (#bytes = specified in length byte)]

הערה: את כל הערכים יש לפרש כ- unsigned.

Message type byte-

יקבל את הערך 5

Sender id byte-

מקבל את המספר הסידורי של y (היעד)

Sender id byte-

מקבל את המספר הסידורי של x (השולח)

Length byte–

גודל ההודעה עצמה [ללא התחילית] בבתים.

Sequence of bytes –

רצף התווים המהווים את ההודעה. אורכו הוא כמספר הבתים שמצוין ב-length byte.

6ב. **כאשר הלקוח רוצה לשלוח הודעה כזו ללקוח אחר,** הוא שולח הודעה לשרת באותו מבנה בדיוק כמו ב6א. בניגוד לשרת, הוא ראשי לכתוב ב-destination id את הערך 255 (-1), כדי לבצע broadcast.

7. **הודעת שינוי מעמד מצופה לשחקן**

כאשר שרת רוצה לאפשר לצופה להתחיל לשחק (להפוך אותו לשחקן), הוא שולח לו בייט בודד, עם הערך 6. כלומר ההודעה היא רק [Message type byte] בודד עם הערך 6.

הודעות שהלקוח שולח לשרת

1. **הודעת מהלך** – כאשר הלקוח רוצה לבצע מהלך, הוא שולח לשרת את ההודעה הבאה:

[Message type byte] [heap index byte] [amount to remove short]

הבייט הראשון, מסמל את סוג ההודעה, יקבל את הערך 7.

הבייט השני יכיל את האינדקס של הערימה ממנה המשתמש רוצה להסיר איברים. כמו בתרגיל הקודם הערכים נעים בתחום 0-3

ה-short האחרון מכיל את הכמות שהמשתמש רוצה להסיר, הערך צריך להיות חיובי קטן שווה 1500 (גודל ערימה מכסימלי).

מהלך נחשב ללא חוקי ויגרור שליחת הודעה מסוג " הודעת אי אישור מהלך שחקן [Negative ACK]" ללקוח אם התקיים אחד התנאים:

א. לקוח שלח הודעת מהלך שלא בתורו

ב. התור הוא אכן של הלקוח, אך המהלך עצמו אינו תקין (amount to remove גדול ממספר האיברים בערימה)

הערה: את כל הערכים יש לקרוא כ-unsigned.

2. **הודעת MSG ללקוח אחר –**

כאשר הלקוח רוצה לשלוח ללקוח אחר הודעה (או לבצע broadcast), הוא ישלח לשרת הודעה כפי שהוגדר ב-6ב' (ב"הודעות ששרת שולח ללקוחות")

מימוש הפרוטוקול

על מנת לאפשר ללקוח ולשרת לייצר הודעות פרוטוקול ביעילות ונוחות, מימשנו מבני נתונים שמייצגים את ההודעות שמועברות בין הצדדים ומתודות לטיפול בהודעות אלה, בקובץ מרכזי nim_protocol_tools.

בקובץ h. ניתן למצוא struct עבור כל הודעה, מתודות יצירה עבור כל סוג הודעה ומתודה לבדיקת נכונות השדות בהודעה שהתקבלה.

כמו כן, הקובץ מגדיר את הקבועים שהופיעו בפרוטוקול, לדוגמה NUM_HEAPS, MAX_HEAP_SIZE, CONNECTION_ACCEPTED ועוד. קבועים המאפשרים שינוי קל ומהיר של הפרמטרים המוגדרים בפרוטוקול (כל עוד גודל השדות בהודעות עצמן מתאים לקבועים).

כאמור, לכל סוג הודעה מוגדר קבוע עם ה-message type (פרט להודעת הפתיחה) כפי שהוגדר בפרוטוקול וכן struct שמייצג את ההודעה, לדוגמה:

```
/*
    struct represents the first opening message that a server sends to a new client
*/


typedef struct openning_message
{
    char connection_accepted;         /* CONNECTION_ACCEPTED or CONNECTION_DENIED */
    char isMisere;                    /* MISERE or REGULAR */
    char p;                           /* number of players <= 9 */
    char client_id;                   /* client id <= 9 (bonus) */
    char client_type;                 /* PLAYER or SPECTATOR */

} openning_message;

#pragma pack(pop)
```

```
/*
     represents a message that a server sends to the client in order
     to notify him about the amount of items in each heap
     each such message also contains a flag that indicates whether the game
     has ended or continues
*/

typedef struct heap_update_message
{
     char message_type;               /* HEAP_UPDATE_MSG     */
     short heaps[4];                  /* heap representation */
     char game_over;                  /* GAME_OVER or GAME_CONTINUES */

} heap_update_message;
```

מתודות יצירה מתאימות (המשך דוגמה):

```
void create_openning_message(openning_message* message, char isMisere, char p, char client_id,
char client_type);

void create_openning_message_negative(openning_message* message);

void create_heap_update_message(heap_update_message* message, short* heaps, char game_over);
```

על מנת לטפל בקלות בכל סוגי ההודעות בו זמנית, הגדרנו struct נוסף שמסוגל להכיל את כל סוגי ההודעות:

```
#define MAX_FILLER_SIZE 10          /* the biggest message is the heap update message */

typedef struct message_container
{
    char message_type;              /* message type as defined above */
    char filler[MAX_FILLER_SIZE];   /* message container */

} message_container;
```

מבני נתונים

בנוסף למבני הנתונים שמייצגים את ההודעות שעוברות בפרוטוקול, הגדרנו מבני נתונים שיאפשרו לנהל את התקשורת בין הצדדים באמצעות select.

מאחר ואנו רוצים לקרוא מ-socket, כמה שניתן, רק כאשר היא read-ready ולכתוב ל-socket, כמו שניתן, רק כאשר היא write ready. אנו צריכים להתמודד עם קבלות וכתיבות חלקיות, דחיית שליחה או קריאה ולכן מימשנו buffer מעגלי:

1. **IO_buffer**[.c, .h]

מגדיר buffer מעגלי לצורך אגירת מידע על מנת שנוכל לשלוח ממנו מידע כאשר ה-socket היא write ready או לחילופין לשמור בו מידע כאשר ה-socket היא read ready. הגודל החוצץ קבוע, וניתן לשנות אותו על ידי שינוי הקבוע `MAX_IO_BUFFER_SIZE`.

ייצוג החוצץ:

```
/*
    this struct represents a FIFO circular buffer for buffered reading or writing to/from a socket
*/

typedef struct io_buffer
{
    int size;                       /* number of items currently in buffer */
    int head;                       /* index of the first byte */
    int tail;                       /* index of the last byte */
    char buffer[MAX_IO_BUFFER_SIZE];  /* the actual buffer */

} io_buffer;
```

כאמור משום שגודלו חסום, החלטנו לממש אותו כמעגלי על מנת לשפר ביצועים.

המתודות עיקריות שמסופקות:

א. הכנסה לתוך החוצץ (push)

ב. הוצאה מתוך החוצץ כמות בטים נתונה (pop, pop_no_return)

ג. הוצאת **הודעה שלמה** מתוך החוצץ – הודעה שימושית ביותר בה אנו משתמשים בצד שרת ולקוח, שולפת הודעה מתוך החוצץ רק כאשר יש הודעה שלמה כפי שהוגדר בפרוטוקול. כאשר יש רק הודעה חלקית מוחזר `MSG_NOT_COMPLETE`, וכאשר יש הודעה שגויה (אינה לפי כללי הפרוקוטול) מוחזר `INVALID_MESSAGE`. (תיעוד מלא ניתן למצוא בקובץ h.). כאמור מתודה זו גם מבצעת בדיקה של תקינות ההודעה, כלומר בודקת אם השדות בהודעה קיבלו ערכים שמוגדרים בפרוטוקול. חתימה:

```
int pop_message(io_buffer* buff, message_container* msg_container);
```

ד. הודעת **שליחה** מתוך החוצץ – השליחה היא חלקית בלבד (כמה שניתן, לא sendall). המתודה מטפלת בקשר שנסגר (כמו תמיד, תיעוד מלא בקבצים). חתימה:

```
int send_partially_from_buffer(io_buffer* buff, int sockfd, int num_bytes, int*
connection_closed);
```

2. buffered_socket[.h, .c]

מייצג חיבור שניתן לדחות אליו כתיבה וכן לדחות קריאה ממנו. במקרה של השרת, כל לקוח מיוצג על ידי buffered_socket. במקרה של לקוח, השרת מיוצג על ידי buffered_socket.

הייצוג:

```
/*
   this struct represents a buffered socket
   each socket has an input and output buffer
   to buffer protocol messages

   [represents a connection with a client or a server]
*/

typedef struct buffered_socket
{
    int sockfd;
    int client_stat;                        /* client status/type: SPECTATOR or PLAYER*/
    int client_id;                          /* client id                              */
    struct io_buffer* input_buffer;
    struct io_buffer* output_buffer;
    struct buffered_socket* next_client;    /* in case of a client, list, points to the next client */

} buffered_socket;
```

בשני המקרים, חיבור שכזה מכיל שני חוצצים (io_buffer כפי שהצגנו לפני כן) לכתיבה וקריאה.

מתודות מסופקות:

- יצירה (דינאמית, על ה-heap) יחד עם טיפול שגיאות של הקצאת זיכרון.
- פינוי buffered_socket (פינוי זיכרון)

הערה: במקרה של השרת, כל אובייקט כזה הוא איבר ברשימה של לקוחות (next_client מצביע ללקוח הבא)

3. client_list [.c, .h]

ייצוג של רשימה מעגלית של לקוחות, עבור השרת.

מדובר ברשימה מקושרת שכל אבריה הם מסוג buffered_socket ומייצגים לקוחות. בתחילת הרשימה יופיעו כל השחקנים, ולאחר מכן, לפי סדר התחברותם, יופיעו הצופים.

למעשה מדובר ברשימה מקושרת, שהרישא שלה היא רשימה מעגלית של שחקנים, והסיפא שלה היא רשימה מקושרת רגילה של צופים.

היתרון של ייצוג כזה הוא ניהול הצופים ושחקנים יחד באותה הרשימה.

המתודות שמסופקות:

- הוספת צופה לסוף הרשימה (ככל שצופה התחבר יותר מוקדם, הוא מופיע יותר מוקדם ברשימה)
- מחיקה של לקוח (צופה\שחקן) מתוך הרשימה ופינוי הזיכרון שהוקצה לו
- הוספת שחקן לרשימה. השחקן מתווסף לסוף הרשימה המעגלית של השחקנים, דהיינו, רק לאחר שכל שאר השחקנים הקודמים ביצעו את מהלכם, השחקן החדש יבצע את המהלך שלו. בפועל המימוש הוא הוספת השחקן החדש בדיוק לפני השחקן הנוכחי.
- שחרור הרשימה והקצאת רשימה חדשה
- מציאת איבר ברשימה לפי המספר המזהה

- להפוך את הצופה הראשון ברשימה (דהיינו את הצופה הישן ביותר, שהתחבר הכי מוקדם מבין כל הצופים) לשחקן ולהוסיף אותו למקום המתאים ברשימה.

בונוס

בתרגיל זה מימשנו את הבונוס.

התור המעגלי שאנחנו משתמשים בו הוא מטיפוס client_list (ראה מבני נתונים סעיף 3, עמוד קודם)

כל הלקוחות שמחוברים לשרת, מופיעים בתור זה. הרישא של התור תמיד יכיל את השחקנים קודם (p לכל היותר) ולאחר מכן את הצופים (אם יש, לפי סדר התחברותם לשרת).

כאמור ניתן לראות רשימה זו כשתי רשימות :

א. החלק הראשון של הרשימה הוא רשימה מעגלית של שחקנים, כאשר נרצה להוסיף שחקן לרשימה, נוסיף אותו לפני השחקן הנוכחי (ולכן התור שלו יהיה אחרון בקרב השחקנים הקיימים)

ב. החלק השני הוא רשימה מקושרת של צופים, כאשר נרצה לקדם צופה להיות שחקן, ניקח את האיבר הראשון ברשימה זו. משום שהצופים נכנסים לרשימה בדיוק לפי סדר התחברותם לשרת, הצופה בתחילת הרשימה הוא הצופה הראשון שהתחבר לשרת.

פרטים נוספים לגבי המימוש:

- כאשר לקוח חדש מתחבר לשרת, הוא מקבל את המסדר הסידורי הקטן ביותר שלא הוקצה (בדיוק כמו שהוגדר בדרישה של הבונוס) ומתווסף לתור (הוספה שונה לצופים ושחקנים, כפי שמתואר מעלה)

- כאשר שחקן מתנתק ומתפנה מקום לשחקן נוסף, מוצאים את הלקוח הראשון בתור שהוא צופה והופכים אותו לשחקן. דהיינו, **הצופה שנבחר הוא הצופה הראשון שהתחבר מבין כל הצופים** (צופה שחיכה הכי הרבה זמן) <u>ולא (בהכרח)</u> הצופה בעל המספר המזהה הקטן ביותר.

- כאשר שחקן מבצע את תורו, התור עובר <u>לשחקן</u> הבא בתור (לשחקן בלבד, ולא צופה) בצורה מעגלית.

- בין אם שחקן נוסף בגלל שכאשר התחבר מספר השחקנים היה קטן מ-p, או בגלל שהפך לשחקן בגלל ששחקן אחר התנתק, הוא תמיד יהיה השחקן האחרון בתור המעגלי של השחקנים.

אופן הרצת התוכנית

מקמפלים את הקבצים על VM בעזרת ה-make file (פקודת make all). קובץ ההרצה nim מתאים ללקוח וקובץ ההרצה nim-server מתאים לצד שרת.

מבנה התוכנית והקבצים

קבצי מימוש של **מבני הנתונים** שהצגנו ב"מבני נתונים" הם (ראה "מבני נתונים" להסבר יותר מפורט):

- [.c, .h] **IO_buffer** – מימוש של חוצץ מעגלי. מממש מתודות של הכנסה והוצאה, שליחת מידע מתוך חוצץ והוצאת הודעה מוכנה מתוך החוצץ
- [.c, .h] **buffered_socket** – מימוש של חיבור שניתן לדחות אליו שליחה של מידע, או לדחות קריאה של מידע ממנו. עבור לקוח משתמשים במבנה זה כדי לנהל שליחת וקבלת מידע מול השרת (מנהל חוצצי כתיבה וקריאה). השרת משתמש במבנה זה כדי לתחזק רשימה של לקוחות, לכל לקוח נשמר טיפוס כזה לצורך ניהול קריאה וכתיבה מול הלקוח.
- [.c, .h] **client_list** – מימוש של רשימה מעגלית של לקוחות. בחלקה הראשונה ברשימה מתוחזקים לקוחות שהם שחקנים (רשימה מעגלית) ובחלקה השני רשימה מקשורת של צופים, כאשר צופה (לא שחקן) שמתחבר מתווסף לסוף הרשימה.

מבני נתונים של ה**פרוטוקול** ומתודות משותפות לשרת ולקוח מופיעות ב:

- [.c, .h] **nim_protocol_tools** – [מוסבר באופן מפורט ב "מימוש הפרוטוקול] קבצים שמממשים מבני נתונים שמייצגים את כל ההודעות שעוברות בין לקוח ושרת. לכל סוג הודעה מוגדר מבנה נתונים משלה. כמו כן, מסופקות מתודות ליצירת הודעות ובדיקת נכונות של הודעות.
- [.c, .h] **socket_IO_tools** – מספקים מתודות נוחות לשליחה וקבלה של מידע דרך socket. לרבות כל סוגי השליחה (send all, send partially) עם טיפול בסגירת קשר בצד השני וכן כל סוגי השליחה (recv all, recv partially) עם טיפול בסגירת קשר בצד השני. מתודה נוספת היא סגירת SOCKET עם טיפול שגיאות.

מימוש של **לקוח** מופיע בקבצים:

- [.c, .h] **client** – הקובץ המרכזי שמממש את צד הלקוח. הקובץ אחראי על
  1. פתיחת הקשר עם צד השרת (+ טיפול שגיאות התחברות)
  2. תחזוק חוצץ שליחה וכתיבה מול השרת
  3. ריבוב (select) בין ה-STDIN וה- server socket, מילוי חוצץ השליחה כאשר הלקוח מבקש לשלוח הודעה מסוג כלשהו, הוצאת הודעות מתוך חוצץ הקבלה של השרת וטיפול בסוגי ההודעות השונים.
  4. טיפול בהודעות השונות (התמודדות עם האירועים השונים בפרוטוקול)

- [.h, .c] **client_game_tools** - אחראי על הדפסת הודעות מתאימות ללקוח לפי סוגי ההודעות השונים. כמו כן, מספק מתודה לניתוח הודעת הפתיחה מהשרת ובדיקת תקינותה.

מימוש של **השרת** מופיע בקבצים:

- [.c, .h] **advanced_server** – אחראי על אתחול השרת, יצירת ה-listening socket (+ טיפול שגיאות), בדיקת הארגומנטים שניתנו בשורת הפקודה, אחראי על ריבוב (select) בין הלקוחות השונים וה-listening socket. אחראי על קבלת חיבורים חדשים, אתחול buffered_socket לכל לקוח חדש והוספתו לרשימת הלקוחות. הלולאה המרכזית שמריצה את השרת, שמפעילה את ה-select, ממומשת במתודה play_nim().
- [.c, .h] **server_utils** – מספקת מתודות עבור השרת לטיפול באירועים שונים בפרוטוקול: טיפול בהודעות מוכנות, סיווג ההודעות לפי סוגן, הוצאה והכנסה של הודעות לתוך החוצצים של הלקוחות, טיפול בהודעת צ'אט, טיפול בהודעת מהלך, טיפול בהתנתקות לקוח וניהול התורות (+בונוס). כמו כן, מספקת מתודות לאתחול ושליחת ההודעות השונות של השרת אל הלקוח וטיפול בשגיאות.
- [.c, .h] **nim_game**- מספקים מימוש של ניהול המשחק עבור צד השרת. תחזוק של מערך הערימות, ביצוע מהלכים, בדיקת תקינות מהלך, בדיקת סוף משחק.

הערה: תיעוד לכל המתודות ניתן למצוא בקבצי המקור עצמם.

אופן פעולת הלקוח ומקרי קצה לקוח

אופן הפעולה

1. הלקוח בודק את תקינות הארגומנטים ויוצר קשר עם השרת.

2. מיד לאחר יצירת הקשר הוא מצפה לקבל את ההודעה הראשונה בפרוטוקול. לפי ההודעה הראשונה הוא מסיק אם הקשר אושר או לא, מדפיס את הפרטים המתאימים וממשיך (או נסגר אם השרת דחה את הלקוח)

3. אם הקשר אושר, הלקוח מאתחל buffered_socket (בגדול, חוצצים לקבלה ושליחה מידע מול השרת).

4. כעת הלקוח נכנס ללולאה שמבצעת select על ה-server socket ו-stdin.

5. בכל איטרציה בלולאה ב-(4), תחילה קוראים את ההודעה של הלקוח מ-stdin אם ה-stdin הוא read ready ומכניסים אותה לחוצץ שליחה.

לאחר מכן בודקים אם השרת הוא read-ready. אם כן, קוראים כמה שניתן (recv_partially) מה-socket ומכניסים את המידע שהתקבל לחוצץ קבלה.

כעת, בודקים האם השרת הוא write-ready ושולחים אליו כמה שניתן מתוך חוצץ השליחה.

לבסוף, מנסים לטפל בהודעה הראשונה השלמה בחוצץ הקבלה. אם אין הודעה שלמה בחוצץ, ממשיכים לאיטרציה הבאה (4). אם התקבלה הודעה שלמה, לפי סוג ההודעה מבצעים את הלוגיקה של הלקוח.

מקרי קצה

א. טיפול בקלט מהמשתמש:

א1. קלט מהמשתמש לצורך ביצוע מהלך צריך להיות מהצורה:

[A-D] [SPACE] [unsigned short <= 1500]

(כאשר המספר האחרון הוא חיובי ממש, קטן שווה 1500), למשל A 500

או מהצורה:

[A-D] [unsigned short <= 1500]

(כלומר כמו מקודם אך ללא הרווח), למשל D29

קלט שהוא בקשה לסגירת התוכנית צריך להיות התו Q בלבד, או כל מחרוזת אחרת שמתחילה ב-Q (לא נקרא את יתר התווים אחרי שקראו Q) למשל: Q או QAFD.

חריגה מהפורמט הנ"ל תגרור הדפסת הודעת שגיאה, שחרור המשאבים ויציאה מהתוכנית (בין אם זה התור של הלקוח או לא).

אם לקוח מנסה להכניס הודעה שכזו, כאשר זהו לא התור שלו, אך ההודעה תקינה לפי הפורמט הנ"ל, יודפס "Move rejected: this is not your turn" כמוגדר בתרגיל והתוכנית תמשיך לרוץ כרגיל.

א2. קלט מהמשתמש לצורך שליחת הודעה ללקוח אחר (או broadcast), חייב להיות מהצורה

MSG [destinationID] [SPACE] [actual message]

- הרווח בין MSG ל-destinationID הוא אופציאונלי
- destinationID חייב להיות **unsigned char (קטן מ-255)** במקרה של שליחה ללקוח אחר יחיד, או 1- במקרה של broadcast. אם הוא לא אחד מן השניים תודפס שגיאה מתאימה והלקוח ייסגר.
- הרווח בין ההודעה לבין ה-destinationID הוא חובה, אם הרווח לא יופיע תודפס שגיאה והלקוח ייסגר.
- actual message חייבת להיות הודעה שאורכה הוא לכל היותר 255, אם תוכנס הודעה ארוכה יותר, תודפס הודעת שגיאה והלקוח ייסגר.
- Actual message יכולה להיות הודעה באורך 0 (הודעה ריקה)

כל הודעה אחרת שהמשתמש ינסה להכניס לתוכנית, שלא מתאימה לפורמט אחת ההודעות הנ"ל, תגרור הדפסת שגיאת קלט, שחרור משאבים, סגירת הקשר עם השרת ויציאה מהתוכנית.

נשים לב שגם אם ההודעה היא חוקית, ייתכן כי **חוצץ השליחה מלא** (הגודל שלו חסום על ידי 2000 בכל רגע נתון), במקרה זה, תודפס שגיאה מתאימה ותוכנית תיסגר.

ב. טיפול בהודעות מהשרת

ב1. בכל מקרה בו מתקבלת הודעה מהשרת בחוצץ הקבלה שאינה חוקית לפי הפרוטוקול (השדות הם לא כפי שהוגדרו בפרוטוקול, לרבות שונים מהקבועים שהוגדרו ב-nim_protocol_tools.h) תודפס הודעת שגיאה והלקוח ייסגר. לדוגמה: אם התקבלה הודעת עדכון ערימות מהשרת, ואחד הערכים גדול מ-1500, היא תחשב כלא חוקית. כמו כן, אם התקבלה הודעה שכלל אינה נכונה לפי הפרוטוקול (למשל, message type אינו מוגדר בפרוטוקול) אזי היא תחשב כלא חוקית.

ב2. הודעה ממשתמש אחר, מודפסת למסך רק כאשר הגיעה במלואה לחוצץ הקבלה. באופן כללי, מטפלים בהודעה רק כאשר התקבלה באופן מלא בחוצץ הקבלה.

ג. הערות נוספות

ג1. במקרים הבאים:

- קלט בשורת הפקודה אינו תקין (לא כפי שהוגדר בתרגיל)
- שגיאה במציאת כתובת IP של השרת (שירות DNS) או שגיאה בהתחברות אל השרת
- שגיאה ביצירה או סגירת שקע
- שגיאה בקבלת מידע מהשרת, שגיאה בשליחת מידע לשרת

תודפס שגיאה מתאימה (לרבות יחד עם strerror(errno)), המשאבים יפונו והתוכנית תסתיים.

ג2. משום שבלקוח אנו משתמשים במתודה getaddrinfo, ניתן להכניס בפרמטר של ה-port גם של השירות המתאים לפורט, למשל "http" (המתאים לפורט 80).

ג3. חוצץ הקבלה מהשרת גם הוא חסום על ידי 2000, במקרה שבו הגיע מידע מהשרת שלא ניתן להכניס לחוצץ, תודפס שגיאה והתוכנית תסתיים.

ג4. בכל איטרציה מטפלים בהודעה אחת לכל היותר שנמצאת בחוצץ הקבלה. אבל**, אם גילינו כי השרת סגר עמנו את הקשר** (כאשר אנחנו כותבים או קוראים מננו), נוציא את כל ההודעות התקינות מחוצץ הקבלה ונטפל בהן אחת אחר השנייה (המטרה היא למצוא הודעת סיום, אם התקבלה כזו לפני עוד לפני סגירת הקשר. אם מצאנו הודעת סיום, התוכנית תסתיים בצורה תקינה עם הודעת סוף משחק הנכונה, אם לא מצאנו הודעת סיום, יודפס כי השרת סגר עמנו את הקשר והתוכנית תסתיים). כמובן שאם מצאנו הודעה לא תקינה נדפיס שגיאה מתאימה ואם יש רק חלק הודעה תקינה בחוצץ, מאחר והקשר כבר נסגר, נדפיס הודעה שהשרת סגר עמנו את הקשר והתוכנית תסתיים.

ג5. במידה והלקוח אינו צופה אלא שחקן פעיל, אם הוא קיבל הודעת עדכון מהשרת כי המשחק הסתיים, מעתה ואילך הוא מצפה לקבל רק את ההודעה האחרונה (בייט יחיד) שאומרת האם השחקן ניצח או הפסיד (בפרט, הוא מפסיק לטפל ב-stdin). הוא ינסה למצוא אותה על חוצץ הקבלה, ואם אינה נמצאת שם, יחכה לקבל אותה ישירות מהשרת.

אם הלקוח הוא צופה, כאשר מתקבלת הודעה כי המשחק הסתיים, מודפסת הודעת סוף משחק והתוכנית נסגרת.

אופן פעולת השרת ומקרי קצה שרת

אופן פעולה

1. השרת בודק את תקינות הארגומנטים שניתנים בשורת הפקודה (בפרט בודק כי אכן 2<= p <= 9), יוצר listening socket. מאתחל משחק חדש ואת רשימת הלקוחות שהצגנו בתיעוד ונכנס ללולאה המרכזית של השרת

2. בלולאה המרכזית מבצעים select עבור כל החיבורים (עם לקוחות) הקיימים לקריאה וכתיבה וכן מכניסים את ה- listening socket לקבוצה עליה מבצעים בדיקת read ready. כעת מבצעים את הפעולות הבאות לפי הסדר:

2א. בודקים אם יש חיבורים חדשים שממתינים, ומטפלים בחיבור חדש (יצירת החוצצים, בדיקה האם ניתן לאשר את החיבור, איזה סוג לקוח הוא צריך להיות וכו'). אם מספר הלקוחות כבר היה 9, יש לדחות את הלקוח החדש. השרת מניח (בהנחייתו של המתרגל), במקרה זה, כי הלקוח החדש הוא מיידית write ready ובפרט מסוגל לקבל בייט אחד. לכן, השרת שולח את הודעת הדחייה שהוגדרה בפרוטוקול, ואורכה הוא בדיוק בייט אחד ולאחר-מכן סוגר את הקשר עם לקוח זה.

2ב. קורא מכל הלקוחות שהם read ready, כמה שניתן, לתוך חוצצי הקבלה שלהם.

2ג. מטפל בהודעות מוכנות, הודעה אחת לכל היותר מכל חוצץ קבלה של לקוח. (טיפול בהודעה – ביצוע לוגיקה של משחק במקרה של מהלך חוקי, יצירת הודעות עדכון והכנסתן לחוצצי השליחה, העברת הודעה מלקוח ללקוח אחר באמצעות הכנסתה לחוצץ שליחה של הלקוח המתאים).

2ד. אם המשחק הסתיים בעקבות הטיפול בהודעות (התבצעה הודעת מהלך שסיימה את המשחק ב-2ג), אזי השרת עובר למצב מיוחד שבו הוא שולח את הודעות הסיום המתאימות לכל הלקוחות וסוגר את כל המשאבים (בפרט בשלב זה, הוא מפסיק לטפל בהודעות שמגיעות מהלקוחות השונים וביתר ההודעות שלא הספקנו לטפל בהן).

2ה. אם המשחק לא הסתיים, השרת שולח ל-sockets שהם write-ready, כמה שניתן, מתוך חוצצי השליחה של הלקוחות. בסוף שלב זה, חוזרים לתחילת 2.

מקרי קצה

1. במקרה שבו הוזנו פרמטרים שלא לפי הפורמט שהוגדר בתרגיל לשורת הפקודה (בפרט בודק כי אכן 2<= p <= 9), תודפס שגיאה והתוכנית תסתיים.

2. במקרה של כשלון ביצירת listening socket (socket, listen, bind), קבלת חיבור חדש (accept), כשלון בהקצאת זיכרון (לצורך רשימה למשל), כשלון בשליחת הודעת דחייה (בייט בודד), כשלון ב-select תודפס הודעת שגיאה מתאימה וכל הזיכרון המוקצה משוחרר והשרת מסיים את ריצתו.

3. בכל מקרה שבו יש לנו מידע לדחוף לחוצץ כלשהו, אך החוצץ מלא (גודל חסום על ידי 2000), תודפס שגיאה מתאימה, משאבים יפונו והתוכנית תסתיים. (בין אם השרת מבקש לשלוח הודעה ובין אם השרת מבקש לקרוא הודעה)

4. מבחינת טיפול בהודעות שהגיעו מלקוח כלשהוא: אם התקבלה הודעה בחוצץ קבלה שאינה לפי הפורמט שהוגדר בפרוטוקול, מודפסת שגיאה מתאימה והשרת נסגר.

5. אם לקוח x מבקש לשלוח הודעה ללקוח y (או broadcast), אך שלח הודעה שבה ה-sender id שונה מ-x, תודפס שגיאה מתאימה והשרת נסגר.

6. אם לקוח כלשהו שולח הודעת מהלך לא בתור שלו, השרת מכניס לחוצץ השליחה שלו הודעה על מהלך לא חוקי. [למרות שבמימוש שלנו הדבר נמנע בלקוח עצמו]

7. כאשר שולחים (ממש מנסים לבצע שליחה, כמה שניתן, לא sendall) ללקוח כלשהו ונתקלים בשגיאת שליחה, נחשיב זאת כהתנתקות רגילה של הלקוח.

8. כאמור, כאשר ביצענו מהלך של לקוח וגילינו שהמשחק הסתיים, השרת עובר למצב מיוחד שבו הוא דוחף את הודעות הסיום המתאימות עבור כל לקוח לחוצצי השליחה, מבצע select על כל הלקוחות (מבחינת write) ושולח למוכנים מבניהם כמה שניתן מתוך חוצצי השליחה. הריצה תסתיים כאשר עבור כל לקוח, או שהוא התנתק, או שכל המידע שהיה לו בחוצץ השליחה נשלח אליו. בשלב זה לא מבצעים select על קריאה, לא מקבלים הודעות חדשות ולא מטפלים ביתר ההודעות של הלקוחות שהיו בחוצצי הקבלה שלהם. ראה מתודה `send_final_data`

9. מימשנו את ה**בונוס** בתרגיל. במקרה שבו שחקן מתנתק ויש כמה צופים, **הצופה שהתחבר הכי מוקדם יהפוך להיות שחקן** (לא בהכרח בעל המזהה הקטן ביותר)

10. במקרה של הודעות משתמש למשתמש אחר שאינו קיים (אפילו אם היעד, נניח, גדול מ-9) השרת מתעלם מההודעה.

11. במקרה שבו שחקן עוזב את המשחק, כאשר זהו התור שלו, לא תשלח הודעת עדכון ערימות לכל הלקוחות הקיימים. ההודעה היחידה שתישלח היא הודעת עדכון לשחקן שזהו כעת התור שלו (אם יש כזה)