# <u>סיכום הרצאה 4 – מערכות הפעלה</u>

## נושא ההרצאה – תקשורת

1. **<u>מושגים:</u>**

**תקשורת** – סדרת API בה יוצרים דרך להעברת מידע בין 2 תהליכים באמצעות sockets.

**Socket** – חצי תקשורת, כל פעם שיש לי תקשורת עם מישהו, ישנו **socket**.

**Design patterns** (תבניות עיצוב) - מילון מקצועי שאנחנו מצפים מאנשים להבין, מעין שפה משותפת למתכנתים.

**Factory pattern** – תבנית עיצוב נפוצה מאוד המשמשת **מיפוי** (HASH) בין מפתח לבנאי רלוונטי. בתחום הנדסת התוכנה ,תבנית Factory Method היא תבנית עיצוב שתכליתה יצירת אובייקטים החולקים ממשק אחיד מבלי להכיר את המחלקות שלהם. מהות תבנית המפעל היא להגדיר ממשק ליצירת עצם תוך מתן האפשרות לתת-המחלקות הממשות את הממשק להחליט לאיזה מחלקה ליצור מופע.
דוגמא נפוצה לשימוש ב **Factory Method** היא **Codec** לסרטים. כשאנחנו מורידים סרטים, כל סרט מקודד במקודד כלשהו. אנו מקרינים את הסרט בתוכנות כמו VLC, Media Player ועוד, כאשר תוכנות אלו אינן קשורות למקודד בהם קודד הסרט. כאשר תוכנת ההקרנה פותחת את קובץ הסרט היא מקבלת מידע, לפי מידע זה יש לתוכנה **Factory** שמייצר לו את ה- **Codec** לפי המידע ומקבל **Codec** שמנהג כמו שהוא רוצה ובצורה הזאת הוא בונה את ה-**Factory .**
שימוש נוסף ב**Factory Method** : נניח אתם משחקים במשחק מחשב, במשחקים הללו יש "מנוע" ליצירת דמויות וחפצים. אותו מנוע עובד בשיטת **Factory**, בו מכניסים דרישות ויוצרים אובייקט לפי הקלט הנדרש. ישנן חברות שמתמחות רק בבניית מנועים שכאלו כיום (למשל UNITY).
בתחום התקשורת, פונקציה שעובדת עפ"י **Factory method** היא ה-**Socket .** אנחנו נותנים לפונקציה מפתח (3 int ים) שלפיו היא בונה לנו **socket** מהסוג הנדרש.

**IPv4 (Internet Protocol Version 4)** –היא הגרסה הנפוצה של פרוטוקול IP ומהווה את הפרוטוקול הבסיסי של רשת האינטרנט. כתובת IPv4 מורכבת מ-32 סיביות ומיוצגת באמצעות 32 ביטים. דוגמא לייצוג עשרוני מקובל 192.0.2.111. בשלב מסוים הגענו למצב שבו כתובות הIP בגרסה IPv4 צפויות להיגמר על אף שיש מיליארדי כתובות IP מהצורה IPv4. ואכן, כשעוד היו רק כמה מחשבים וכולם חשבו שמיליארדים הוא מספר גדול באופן בלתי אפשרי (ביחס לכמות האנשים בעולם), לכמה ארגונים גדולים הוקצו בנדיבות מיליוני כתובות IP עבור שימוש עצמי (כגון זירוקס, MIT , HP, יבמ,AT&T , ואיזו חברה קטנה בשם אפל). כלומר חלוקת הIP הייתה לא מוצדקת. בנוסף היום אנחנו חיים בעידן שבו לכל אדם יש כתובת IP לכל מחשב (בבית, בעבודה ועוד), כל טלפון, כל טאבלט, רכב, מוצרי הבית החכם ועוד (הרבה).
לאור המחסור המדובר יצאה גרסה חדשה, **גרסה 6 או בשמה המוכר IPv6**. כתובות בגרסה זו מורכבות מ-128 ביטים שזה $2^{128}$ אפשרויות לכתובות לעומת $2^{32}$ אפשרויות

בגרסה IPv4. 64 הסיביות הראשונות משמשות לזיהוי תת-הרשת(משתמש), ו- 64 הסיביות האחרונות משמשות כמזהה ממשק (כלומר ה"נאט" מובנה בתוך הפרוטקול). כמוכן הכתובת באמצעות סדרה של 8 מספרים בני 4 ספרות בבסיס הקסדצימלי שמופרדים בנקודתיים.

**דוגמא:** 2001:0db8:c9d2:aee5:73e3:934a:a5ae:9551

הכתובת 1:: משמשת lookback address כלומר הכתובת של המכונה שאני עובד בה.

חישוב check-sum ב-IPv6 לא מתבצע פעמיים. (מקביל ל127.0.0.1)

**חידושים שהגיעו עם IPv6:**

**Check sum** - אין כפילות של ביצוע **Check sum** לאור כך שגם ב-IP וגם ב TSP\UDP מבוצע **Check sum**.

**Jumbo packets - Headrs** תופסים הרבה מקום, ולכן IPv6 מאפשר **Jumbo packets** בהם אין משמעות לגודל ה-**headrs**.

**Multi cast** - נוצר ב-IPv4 אך מעולם לא מומש. מאפשר מצבים בהם נרצה לפצת את השידור להרבה קצוות.

ועוד (לדוג' אבטחת מידע ועוד פיצרים. הנושא מחוץ לסקופ שלנו)

2. **<u>פונקציות:</u>**

socket(2) - הפקורי מטוד היוצרת סוקט חדש

**`bind(2)`**- כאשר אנו פותחים **Socket** נדרש לקשר אותו לפורט במכונה שלנו, "לתפוס פורט". מספר הפורט משומש ע"י הקרנל על מנת לקשר את החבילות שעומדות להגיע אל ה-**Socket** הרלוונטי. מספרי פורט קבוע ומוכר למשל – פורט 80 הנועד לעבודה עם Web.

**`listen(2)`** - (אנאלוגי ל-**malloc**) לא עושה שום דבר שרלוונטי ברמת התקשורת בחומרה ובמחשב אלא רק מבקש לקבל מקום בזיכרון (מאלקץ) על מנת שיהיה מקום בזיכרון שמיועד לתקשורת העומדת לבוא. במידה ולא נאלקץ מקום, פעולות התחברות שיבואו ייכשלו.

**`connect(2)`** – מתחילה את הקישור בין צד הלקוח לכתובת IP ופורט המסופקים. הקישור יושלם רק אחרי שהצד השני יקרא לaccept(2)

**`accept(2)`** - מאשר לקבל תשדורת בפורט שסופק על ידי וציינתי שבו אני "מקשיב". מוציא את הלקוח האחרון מהתור ומתחיל לעבוד מולו. הפונקציה מחזירה את ה **file descriptor** ל- **socket** חדש להתחברות הבאה. פונקציה זו משלימה את ה"טרי וואי הנדשייק" זו הפונקציה הראשונה שמתרחש בה "תקשורת ממש" מכל הפונקציות של השרת.

recv(2)/send(2) - אנאלוגי ל-**read(2)/write(2)**, כאשר אנו קוראים ל- Send\Receive אנחנו בעצם קוראים ל-**read/write** עם Flags = 0 (**Flags** בפונקציית **send-recv** הדגלים יהיו 0 כברירת מחדל ואנחנו נשאיר זאת כך ברמת הקורס). פונקציות אלא (בהתאם לשמן) נועדו להתקשרות בין שני הצדדים של ה-**socket**. פונקציית **()send** מחזירה את מספר הבתים שהועברו בפועל ולא תמיד זה יהיה תואם את מה שבאמת נשלח, מה שעוזר לנו לוודא שכל ההודעה עברה ואם לא להעביר את השאר מאוחר יותר.

במידה ופונקציית ה-**recv(2)** מחזירה 0 ניתן להסיק כי הצד השני סגרת את התקשורת מולנו.

**sendto(2)\recvfrom(2)** - פונקציות דומות במהות ל-Send\Receive רק מיועדות ל**Datagram (UDP)**. במקרה של **UDP** מבחינת תקשורת אין חיבור, לא חובה לקרוא לפונקציות **listen(2)** ולא **connect(2)**.
כלומר יצרתי **socket(2)** כללי שאינו מנותב למישהו ספציפי ולאור כך, יחד עם ההודעה מועברים גם הפרטים של הלקוח אליו נרצה לבצע שליחה\קבלה מכיוון שאיננו מחבורים ל- **socket(2)** קבוע.
ישנה אפשרות ל פעולת connect(2) ב-UDP שכן מערכת ההפעלה "שומרת" את הכתובת של sockets בהם השתמשתי ואז נוכל להשתמש ב-send(2)/recv(2)
**Close\Shutdown** - סגירה של ה-**Socket**. **close(2)** ימנע על תקשורת אל ה-**socket**.
**shutdown(2)** מאפשר לנתק תקשורת בכיוון מסוים, מה שמאפשר שליטה על אופן סגירת ה-socket.

**Little/BigEndian** - סדר לשידור ה-**bytes**. בעבר, כאשר רשתות התקשורת היו איטיות מאוד, הייתה חשיבות לביצוע פעולה תוך כדי שליחה וללא המתנה עד למעבר הפאקטה כולה, כך שהמידע הנחוץ היה ב-MSB ולכן היינו משתמשים ב-**BigEndian**. כיום אין לכך חשיבות כלל, לנוכח העובדה שרשתות תקשורת מהירות מאוד.
**אבל ארכיטקטורות שונות הם כבר BIG או LITTLE ואנחנו תואמים אחורה.**
**המשך החומר בקורס תקשורת.**

3. **קוד:**

בסוף ההרצאה בוצע מעבר על המדריך:
https://beej.us/guide/bgnet/pdf/bgnet_usl_c_1.pdf עמודים 13-42
באופן כללי המדריך עד פרק 7.3 **הוא למבחן.**
הקוד המלא מפרק 6 – server, client, listener, talker

**הערכת צד:**

אם נרצה לתכנת ב-windows עם sockets נדרש לheaders הקוד הבאים:

```
#define WIN32_LEAN_AND_MEAN  // Say this...

#include <windows.h>         // And now we can include that.
#include <winsock2.h>        // And this.
```

```c
#include <winsock2.h>

{
    WSADATA wsaData;

    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        fprintf(stderr, "WSAStartup failed.\n");
        exit(1);
    }

    if (LOBYTE(wsaData.wVersion) != 2 ||
        HIBYTE(wsaData.wVersion) != 2)
    {
        fprintf(stderr,"Versiion 2.2 of Winsock is not available.\n");
        WSACleanup();
        exit(2);
    }
```