רשתות תקשורת מחשבים

תרגילים 1 + 2:

תכנות TCP/IP באמצעות Sockets

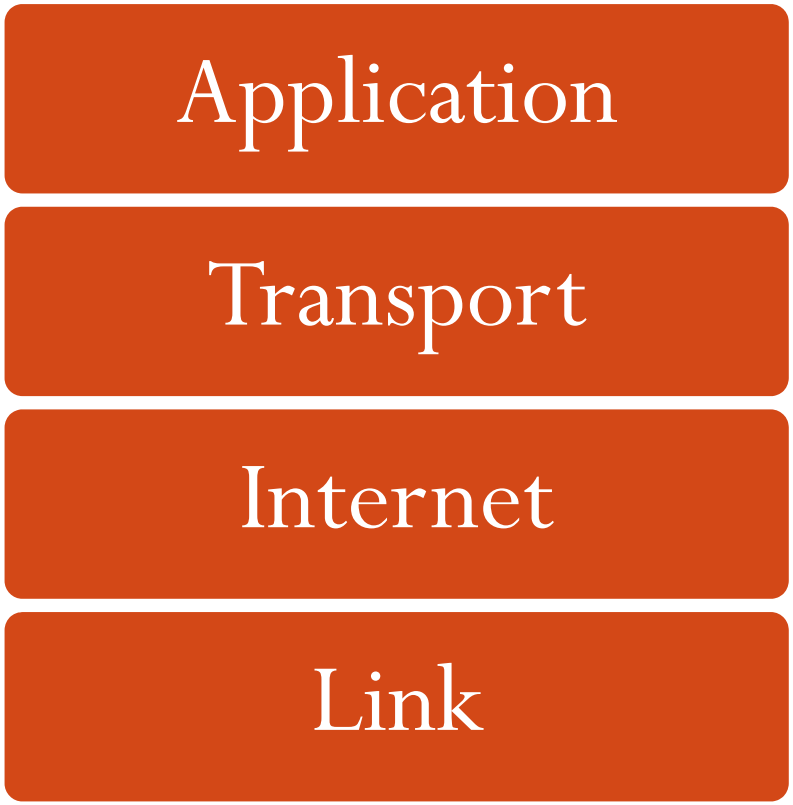
מתרגל: יהב נוסבאום

המטרה

כתיבת אפליקציית רשת

Internet Sockets

- ממשק של שכבת התובלה, לשימוש האפליקציה.
- מאפשרים לשלוח ולקבל מידע.
- מידע נכתב בצד אחד ונקרא בצד השני - כמו צינור.
- גישה דרך API.
  - קיימים מספר מימושים: Berkeley Sockets, Winsock.
- נקרא להם בקיצור Sockets.



כתובות ברשת IP

- לכל תחנה (host) ברשת יש כתובת.
  - נקראת IP Address.
  - כתובת IPv4 מורכבת מארבעה בתים.
  - למשל 132.67.252.100.
  - לרוב הכתובת נסתרת מהמשתמש האנושי שמעדיף שם קריא (hostname), כגון www.cs.tau.ac.il.
  - הקשר בין כתובת לתחנה הוא לא אחד לאחד.
- לכל שירות בתחנה יש כתובת.
  - נקראת port.
  - בין 0 ל- 65535 (כלומר 16 סיביות).
  - למשל 132.67.252.100:80
  - לרוב משויכת לשירות, כגון HTTP.

Port (פתחה)

- מזהה שירות או תוכנה מסוימים בתחנה.
  - Port בתוך IP Address, בדומה לדירה בתוך בניין.
- השרת מאזין על port מסוים, ומחכה לחיבור.
- לשירותים ידועים יש port תקני (well-known ports, registered ports).
  - למשל 80 עבור HTTP, 25 עבור SMTP, 22 עבור SSH.
- הלקוח שולח בקשה מ-port מסוים.
  - לשם תחזור התשובה.

Socket מחובר

צינור בין שתי כתובות.

132.67.252.100:80

132.67.104.229:48228

שני סוגי Sockets

SOCK_STREAM

- בחיבור (connection oriented).
- העברה אמינה.
- העברה לפי הסדר.
- מצריך הקמת קשר.
- TCP.

SOCK_DGRAM

- ללא חיבור (connectionless).
- ללא הבטחת אמינות.
- ללא הבטחת סדר.
- ללא מצב (stateless).
- UDP.

מה אנחנו צריכים מה-API?

- **יצירת** socket.
- **האזנה לרשת בהמתנה לחיבור.**
- **יצירת חיבור.**
- **שליחת מידע.**
- **קבלת מידע.**
- **סגירת ה-**socket.

קבצי header

- `#include <sys/types.h>` – **טיפוסי נתונים.**
  - **אינו נחוץ לנו.**
- `#include <sys/socket.h>` – **עבודה עם** sockets.
- `#include <netinet/in.h>` – **כתובות אינטרנט.**
- `#include <arpa/inet.h>` – **פעולות אינטרנט.**
  - **מספק את הקודם.**

יצירת socket – socket()

```
int socket(int domain, int type, int protocol);
```

- **פרמטרים:**
  - domain – PF_INET **עבור** IPv4 sockets.
  - type – **לצרכים שלנו** SOCK_STREAM **או** SOCK_DGRAM.
  - protocol – **עבור שני הסוגים הללו תמיד** 0.
- **ערך חזרה:**
  - File descriptor **של ה-**socket **שנוצר.**
  - **או** 1- **אם אירעה שגיאה.**
- **ה-**socket **מזוהה על ידי** file descriptor. **ניתן להשתמש בו בכל פונקציה של** fd.
  - **סוגרים את ה-**socket **בעזרת** `close()`.

כתובת

- ה-API משתמש בטיפוס הכללי struct sockaddr.
- לכתובות IP משתמשים בטיפוס:

```
struct sockaddr in {
  sa_family_t    sin_family; /* = AF_INET */
  uint16_t       sin_port;
  struct in_addr sin_addr;
};
struct in addr {
  uint32_t       s_addr;
}
```

- הרכיבים השונים של הכתובת מכילים מספרים המורכבים מכמה בתים.

סידור בתים

- בית מורכב מ-8 סיביות.
- מה קורה כשאנחנו רוצים לייצג מספר גדול יותר? משתמשים בכמה בתים.
- למשל 0x1990 = 6544 וקיבלנו שני בתים 0x19 ו-0x90.
- איך מסדרים אותם בזיכרון?
- Big-endian (מהצד הגדול)
  - קודם את הבית ה"גדול". למשל (שמאל לימין) 0x90 0x19 = 6544.
- Little-endian (מהצד הקטן)
  - קודם את הבית "הקטן". למשל (שמאל לימין) 0x19 0x90 = 6544.

באיזה סדר משתמשים?

- **שניהם!**
  - **כל תחנה ברשת משתמשת ב-**host order **משלה.**
  - **מעבדי** x86 **משתמשים ב-**little-endian.
- **כדי לקבוע אחידות יש להשתמש ב-**network order **שהוא** big-endian.
- **הכתובת ב-**`struct sockaddr_in` **היא** big-endian.
- **כדי להמיר בין משתמשים בפונקציות** `htonl()`, `htons()`, `ntohl()`, `ntohs()`.

שיוך כתובת מקומית – bind()

```
int bind(int sockfd, const struct sockaddr *addr,
  socklen_t addrlen);
```

- **פרמטרים:**
  - sockfd – **ה-**socket **שעליו מתבצעת הפעולה.**
  - addr, addrlen – **הכתובת עליה מאזינים, וממנה שולחים.**
    - INADDR_ANY (= 0.0.0.0) – **כל כתובת.**
- **ערך חזרה:**
  - 0 **עבור הצלחה,** 1- **עבור כישלון.**
- **רצוי עבור השרת, אפשרי עבור לקוח.**
- **ללא** bind(), **מוקצה** port **מקרי על** INADDR_ANY.

התחברות – connect()

```
int connect(int sockfd, const struct sockaddr
  *serv_addr, socklen_t addrlen);
```

- **פרמטרים:**
  - sockfd – **ה-**socket **שעליו מתבצעת הפעולה.**
  - sockaddr, addrlen – **הכתובת אליה מתחברים, ואליה שולחים.**
- **ערך חזרה:**
  - 0 **עבור הצלחה,** 1- **עבור כישלון.**
- **הכרחי עבור תקשורת בחיבור, רשות עבור תקשורת ללא חיבור.**
  - **לשם יצירת חיבור השרת חייב לצפות לו (על כך מייד).**

תהליך יצירת חיבור

- שני צדדים:
  - אקטיבי – יוזם החיבור, בדרך כלל הלקוח.
  - פסיבי – מקבל החיבור, בדרך כלל השרת.
- פרט ליצירת החיבור, שני הצדדים שווים.
  - שניהם יכולים לשלוח ולקבל מידע.
  - שניהם יכולים לנתק את החיבור.

Active	Passive
socket()	socket()
…	bind()
…	listen()
connect()	accept()
Connected	
close()	close()
…	…
…	accept()

המתנה לחיבור – listen()

```
int listen(int sockfd, int backlog);
```

- **פרמטרים:**
  - sockfd – **ה-**socket **שעליו מתבצעת הפעולה.**
  - backlog – **אורך התור להמתנה לחיבור.**
- **ערך חזרה:**
  - 0 **עבור הצלחה,** 1- **עבור כישלון.**
- **הכרחי עבור יצירת חיבור.**
- **כדאי לקרוא קודם ל-**`bind()`.

קבלת חיבור – accept()

```
int accept(int sockfd, struct sockaddr *addr,
  socklen_t *addrlen);
```

- **פרמטרים:**
  - sockfd – **ה-**socket **שעליו מתבצעת הפעולה.**
  - addr, addrlen – **הכתובת ממנה מגיע החיבור**
- **ערך חזרה:**
  - File descriptor **של ה-**socket **שנוצר.**
  - **או** 1- **אם אירעה שגיאה.**
- **יש לקרוא קודם ל-**`bind()` **ול-**`listen()`.
- **נוצר** socket **חדש עבור החיבור החדש.**
- **ה-**socket **המקורי ממשיך להאזין.**

דוגמה לקוד שרת

```
sock = socket(PF_INET, SOCK_STREAM, 0);

myaddr.sin_family = AF_INET;
myaddr.sin_port = htons( 80 );
myaddr.sin_addr = htonl( INADDR_ANY );

bind(sock, &myaddr, sizeof(myaddr));

listen(sock, 5);

sin_size = sizeof(struct sockaddr_in);
new_sock = accept(sock, (struct
  sockaddr*) &their_addr, &sin_size);
```

- בקוד אמיתי חשוב לבדוק שגיאות.

דוגמה לקוד לקוח

```
sock = socket(PF_INET, SOCK_STREAM, 0);

dest_addr.sin_family = AF_INET;
dest_addr.sin_port = htons( 80 );
dest_addr.sin_addr = htonl( 0x8443FC64 );

connect(sock, (struct sockaddr*)
  &dest_addr, sizeof(struct sockaddr));
```

- בקוד אמיתי חשוב לבדוק שגיאות.
- השימוש בקבוע לכתובת יעד הוא רק לצורך הדוגמה.

העברת מידע

עכשיו יש לנו חיבור.

אפשר לכתוב אליו ולקרוא ממנו כמו קובץ,

אבל עדיף להשתמש בפונקציות יעודיות.

שליחה – send(), sendto()

```
ssize_t send(int s, const void *buf, size_t len,
  int flags);
ssize_t sendto(int s, const void *buf, size_t
  len, int flags, const struct sockaddr *to,
  socklen_t tolen);
```

- **פרמטרים:**
  - s – **ה-**socket **עליו שולחים.**
  - buf, len – **המידע הנשלח.**
  - flags – **אפשרויות שליחה,** 0 **לשליחה רגילה.**
  - to, tolen – **כתובת היעד.**
- **ערך חזרה:**
  - **מספר הבתים שנשלחו.**
  - **או** -1 **אם אירעה שגיאה.**
- send() **מיועדת ל-**socket **מחובר,** sendto() **ל-**socket **לא מחובר.**

שליחה חלקית

- **הפונקציות** `send()` **ו-** `sendto()` **עלולות לשלוח רק חלק מ-**buf.
- **דוגמה לקוד אפליקציה שלוקח את זה בחשבון:**

```
to send = sizeof(buf);
while (to_send) {
  sent = send(s, buf, to send, 0);
  buf += sent;
  to_send -= sent;
}
```

- **בקוד אמיתי חשוב לבדוק שגיאות.**

קבלה– recv(), recvfrom()

```
ssize_t recv(int s, void *buf, size_t len, int
  flags);
ssize_t recvfrom(int s, void *buf, size_t len,
  int flags, struct sockaddr *from, socklen_t
  *fromlen);
```

- **פרמטרים:**
  - s – **ה-**socket **ממנו מקבלים.**
  - buf, len – **מקום למידע המתקבל.**
  - flags – **אפשרויות קבלה,** 0 **לקבלה רגילה.**
  - to, tolen – **כתובת המקור.**
- **ערך חזרה:**
  - **מספר הבתים שהתקבלו.**
  - 0 **אם הצד השני ניתק את החיבור.**
  - **או** -1 **אם אירעה שגיאה.**
- recv() **מיועדת ל-**socket **מחובר,** recvfrom() **ל-**socket **לא מחובר.**

קבלה חלקית או מורכבת

- **הפונקציות** `recv()` **ו-** `recvform()` **קוראת את המידע הזמין ולכל היותר** len **בתים.**
- **המידע שנקרא יכול להיות רק חלק מהמידע שנשלח, או מורכב ממספר שליחות נפרדות.**

סגירת ה-socket

- **את ה-**socket **סוגרים בעזרת הפונקציה** `int close(int fd);`, **כמו כל** fd **אחר.**
  - **הפונקציה מנתקת את החיבור (אם קיים), וסוגרת את ה-**fd.
  - **אחריה ה-**socket **למעשה לא קיים, ואי-אפשר להשתמש בו.**
- **סגירת** socket **שנוצר ע"י** `accept()` **אינה סוגרת את ה-**socket **המאזין.**
- **לניתוק בלבד, ללא סגירה, קיימת הפונקציה** `int shutdown(int s, int how);`.
  - how – **איזה כיוון לסגור,** `SHUT_RD`, `SHUT_WR`, `SHUT_RDWR`.
  - **השליחה או הקבלה ב-**socket **נחסמת, אבל ה-**fd **נשאר.**
- **אין צורך ב-**`shutdown()` **לפני** `close()`.

שרת ולקוח TCP פשוטים

Client	TCP	Server
		socket()
		bind()
socket()		listen()
connect()	←יצירת חיבור→	accept()
send()	העברת מידע→	recv()
recv()	←העברת מידע	send()
close()	←ניתוק→	close()

שרת ולקוח UDP פשוטים

Client	UDP	Server
		socket()
socket()		bind()
sendto()	העברת מידע→	recvfrom()
recvfrom()	←העברת מידע	sendto()
close()		close()

פונקציות חוסמות

רוב הפונקציות יחזרו רק לאחר שסיימו את פעולתן.

מתי הן מסיימות?

- `connect()` – **כשנוצר חיבור.**
- `accept()` – **כשקיים חיבור.**
- `send()`, `sentto()` – **כשהמידע נכתב ל-**socket.
- `recv()`, `recvfrom()` – **כנקרא מידע מה-**socket **או כאשר הוא נסגר בצד השני.**

זה טוב או רע?

- לאפליקציה פשוטה זה טוב.
  - שומר על האופי הסדרתי.
- לאפליקציה מורכבת זה רע.
  - מונע ריבוי לקוחות.
  - מעקב פעולות אחרות.
- בעייתי במיוחד בכיוון הקריאה וההמתנה לחיבור.
  - `accept()`, `recv()`, `recvfrom()`.
  - מכיוון שבמקרים האלה התגובה לא מיידית.

מה עושים?

- משלימים עם המצב.
- עבודה במספר תהליכונים (threads).
- קוראים לפעולה רק אם היא לא תחסום –
  - `select()`, `pselect()`, `poll()`, `ppoll()`.
- לא מרשים לפעולה לחסום –
  - `O_NONBLOCK`, `MSG_DONTWAIT`.
- שילוב פתרונות.

ריבוב סינכרוני

- יש לנו מספר fd-ים.
- אנחנו רוצים לבצע עליהם פעולות במקביל.
- האפליקציה שלנו היא סדרתית.
- לכן, צריך לבצע פעולה רק אם היא לא תחסום.
- פונקציה כמו `select()` מתאימה בדיוק לתרחיש הזה.
- הפונקציה חוזרת כאשר לפחות אחד מה-fd-ים מוכן לפעולה, ומסמנת לנו מי מוכן.

ריבוב סינכרוני – select()

```
int select(int nfds, fd set *readfds, fd set
  *writefds, fd_set *exceptfds, struct timeval
  *timeout);
```

- **פרמטרים:**
  - fds – **מספר ה-**fd**-ים למעקב, ה-**fd **הגבוה ביותר** + 1.
  - readfds, writefds, exceptfds – **קבוצות** fd**-ים למעקב כתיבה, קריאה, חריג.**
  - timeout – **הגבלת זמן.**
- **ערך חזרה:**
  - **מספר ה-**fd**-ים המוכנים.**
  - **או** 1- **אם אירעה שגיאה.**

מבנה fd_set

- **זהו למעשה** bitmap **של** fd**-ים.**
- **נגשים אליו באמצעות הפונקציות הבאות:**
  - `void FD_ZERO(fd_set *set);` – **מאפס את המבנה.**
  - `void FD_SET(int fd, fd_set *set);` – **מוסיף** fd.
  - `int FD_ISSET(int fd, fd_set *set);` – **בודק אם** fd **נמצא.**
- **הפונקציה** `select()` **מקבלת את המבנים, עוקבת אחרי ה-**fd**-ים המסומנים ומחזירה בהם את ה-**fd**-ים המוכנים.**
- **במקום קבוצה ריקה אפשר להשתמש ב-**NULL.
- **במקרה של שגיאה ב-**`select()`**, הערך של המבנים לא מוגדר.**

הגבלת זמן

- **הפרמטר** timeout **הוא מהמבנה הבא:**

```
struct timeval {
   long        tv sec;
   long        tv_usec;
}
```

- **מאפשר דיוק של מיקרו-שניות.**
- $\{0,0\}$ – **חוזר מייד.**
- NULL – **ללא הגבלה.**
- **כשהפונקציה חוזרת, היא מפחיתה את הזמן שעבר. או שלא.**
  - **ב-**Linux **כן.**

מוכן לקריאה

- **כש-**`select()` **חוזרת בהצלחה,** readfds **מכיל את ה-**fd**-ים שמוכנים לקריאה.**
- **קריאות לפונקציות כגון** `recv()` **ו-**`accept()` **על** socket **מ-**readfds **לא אמורות לחסום.**

שרת TCP מורכב (קצת יותר)

Server	
socket()	
bind()	
listen()	
FD_SET()	
select()	
accept() / recv()	לפי ה-socket המוכן לקריאה

במקרה של שגיאה

- **כל הפונקציות הנ"ל שמחזירות** 1- **במקרה של שגיאה, מכוונות את** errno **בהתאם.**

תרגום שם לכתובת אינטרנט

- פונקציה ישנה – `gethostbyname()`.
  - ממשק פשוט.
  - קיימת גם במערכות ישנות.
  - תומכת רק ב-IPv4,
- פונקציה חדשה – `getaddrinfo()`.
  - בעלת פרמטרים רבים.
  - מחזירה את הכתובת במבנה `struct sockaddr`.
  - תואמת תקנים חדשים.
  - פונקציות עזר יחודית `freeaddrinfo()`.
- שתיהן חוסמות.
- לשתיהן ערכי שגיאות יחודיים (במקום `errno`).

פונקציות נוספות לכתובות אינטרנט

- `inet_ntoa()` – **המרת כתובת** IPv4 **ממספר למחרוזת.**
- `inet_aton()` – **המרת כתובת** IPv4 **ממחרוזת למספר.**
- **קיימות פונקציות נוספות עם תפקיד דומה.**
  - `inet_ntop()` **ו-** `inet_pton()` **לא מוגבלות ל-**IPv4.
  - **פונקציות אחרות בדרך כלל ישנות יותר.**
- `getpeername()` – **כתובת הצד המרוחק של ה-**socket.
  - **שימושי אחרי** `accept()`.

כלים שימושיים

- Loopback device – **ממשק מדומה לתקשורת פנימית.**
  - **בכתובת** 127.0.0.1 **או בשם** localhost.
- Netcat – **"אולר שוויצרי" ל-**TCP/IP.
  - **כלי פשוט לקריאה וכתיבה מ- ואל הרשת.**
  - **כלקוח** `nc hostname port`.
  - **כשרת** `nc -l -p port`.
- Netstat – **מציד את חיבורי הרשת (ועוד).**
  - **כדי להציג באופן מספרי את כל חיבורי האינטרנט:**
    `netstat --protocol=inet -an`.