תרגיל בית 1

תיאור התרגיל

לתרגיל זה שני חלקים, חלק מעשי וחלק תיאורטי.

חלק תיאורטי (30% ניקוד)

בחלק זה יהיה עליכם לענות על מספר שאלות תיאורטיות.

חלק מעשי (70% ניקוד)

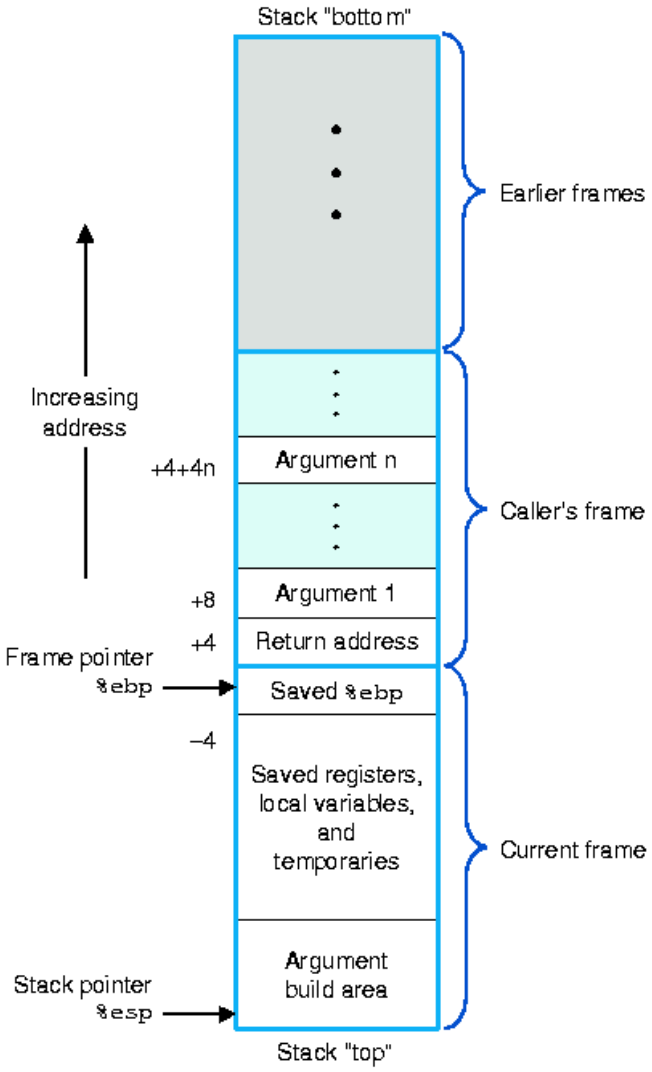
בחלק זה יהיה עליכם לממש שתי תוכניות.

1. בתוכנית הראשונה תממשו shell פשוט (תוכנית שורת פקודה, בדומה ל- shell הקיים ב- Linux).
2. בתוכנית השניה תממשו תקשורת client/server באמצעות Sockets API.

חלק תיאורטי

שאלה 1

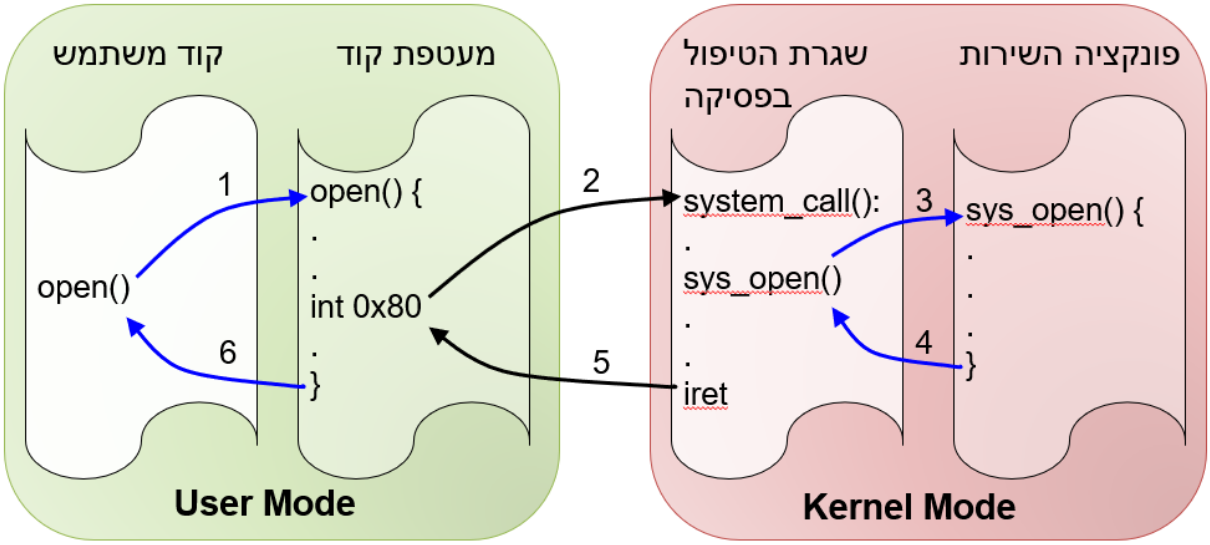
הסבירו **<u>במשפט אחד בלבד</u>** מדוע בקונבנציית GCC לקריאה לפונקציה, שומרים בסדר הפוך (במחסנית) את הפרמטרים המועברים לפונקצייה הנקראת:



כלומר, הסבירו מדוע תחילה מבצעים push לפרמטר ה- n (אחרון), ורק לבסוף מבצעים push לפרמטר ה- 1 (ראשון).

שאלה 2

הסבירו **<u>במשפט אחד בלבד</u>** מדוע לא ניתן להעביר פרמטרים דרך המחסנית, בזמן ביצוע קריאת מערכת (System call), כאשר עוברים מה- User mode ל- Kernel mode. כלומר, הסבירו מדוע "פונקציית המעטפת" (באיזור הירוק) לא יכולה להעביר פרמטרים דרך המחסנית לשגרת הטיפול בפסיקה (באיזור האדום), כפי שמתואר בתרשים הבא:



שאלה 3

ציינו את כל הפלטים האפשריים (למסך) של קטע הקוד הבא: **(נמקו!)**

```
pid_t pid = fork();
if (pid < 0)
{
        exit(1);
}
else if (pid > 0)
{
        printf("%d", getpid());
        exit(0);
}
else
{
        char *const argv[] = {"sleep", "1", NULL};
        execv("/bin/sleep", argv);
        printf("%d", getpid());
}
```

<u>ניתן להניח כי:</u>

```
pid(father) = 8
pid(son) = 13
```

שאלה 4

ציינו את כל הפלטים האפשריים (למסך) של קטע הקוד הבא: **(נמקו!)**

```
int value = 0;

if (fork() != 0)

{

       wait(&value);

       value = WEXITSTATUS(value);

       value += 3;

}

printf("%d\n", value);

value += 4;

exit(value);
```

שאלה 5

הסבירו **<u>במשפט אחד בלבד</u>** מה מבצע קטע הקוד הבא:

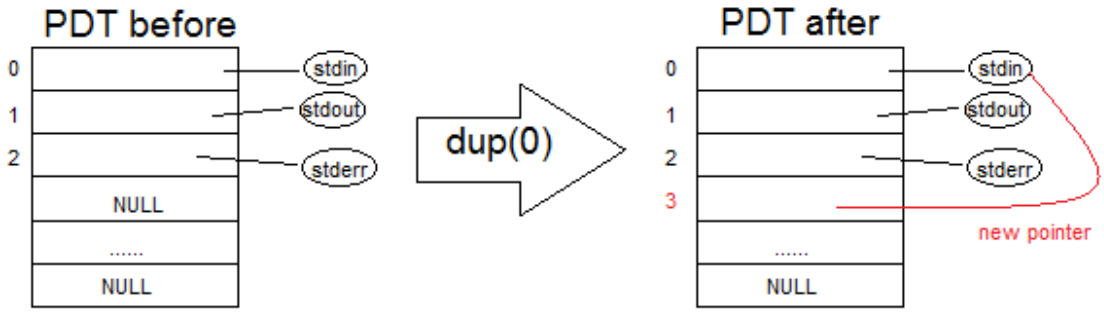
```
int fd[2];
pipe(fd);

int status = fork();
if (status == 0)
{ /* first child */
  close(1);
  dup(fd[1]);
  close(fd[0]);
  close(fd[1]);
  char *const parmList[] = {"/bin/ls", "-l", "./", NULL};
  execv("/bin/ls",parmList);
}

wait(&status);

status = fork();
if (status == 0)
{  /* second child */
  close(0);
  dup(fd[0]);
  close(fd[0]);
  close(fd[1]);
  char *const parmList[] = {"/bin/cat", NULL};
  execv("/bin/cat",parmList);
}

close(fd[0]);
close(fd[1]);
```

מספר הערות:

1. ניתן להניח כי כל קריאות המערכת מצליחות.
2. הסבר על קריאת המערכת dup: (תיעוד רשמי)

```
int dup(int oldfd);
```

The **dup**() system call creates a copy of the file descriptor *oldfd*, using the lowest-numbered unused file descriptor for the new descriptor.

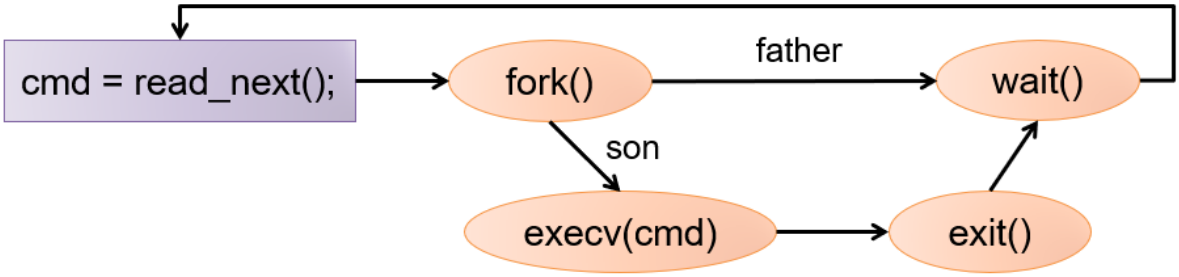


חלק מעשי

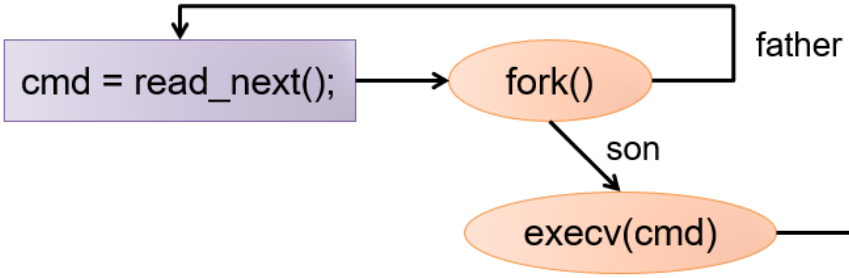
מימוש Shell

עליכם לממש תוכנית Shell פשוטה, אשר תאפשר למשתמש להריץ פקודות בחזית וברקע.

- הרצה בחזית:



- הרצה ברקע:



התוכנית תדפיס למסך את המחרוזת " <my-shell" לפני קבלת הפקודה הבאה מהמשתמש. לאחר שהמשתמש הקליד את פקודתו (למשל "mkdir newfolder"), התוכנית תריץ את הפקודה. אם הפרמטר האחרון שהמתמש הקליד (כחלק מהפקודה) היה התו "&" אז התוכנית תריץ את הפקודה ברקע, אחרת, התוכנית תריץ את הפקודה בחזית. התוכנית תסיים את ריצתה כאשר המשתמש יקליד "exit".

<u>מספר הערות:</u>

1. **סרטון אחד שווה אלף מילים, ולכן מצורף לינק ל- YouTube בו תוכלו לראות הרצה לדוגמה.**
2. מסופק לכם קובץ myshell.c עם שלד התוכנית שעליכם לממש.
3. השתמשו ב- strtok כדי להפריד את הפקודה לחלקיה, ע"י שימוש בתו רווח (whitespace) כמפריד בין חלקי הפקודה (delimiter).
4. ניתן להניח שהמשתמש מקליד רצף פקודות כלשהו הבנוי אך ורק מהפקודות הבאות:
   a. ls
   b. pwd
   c. date
   d. cal
   e. cp

f. mkdir
g. rmdir
h. rm
i. cat
j. mv
k. ps
l. touch

5. **שימו לב:** במידה וזה לא היה ברור – אין עליכם לממש פקודות Linux Shell כגון ls או mkdir, כל שעליכם לעשות הוא להשתמש במתודה execvp כדי להריץ פקודות (כגון הנ"ל) שהמשתמש הקליד. דוגמה לשימוש ב-execvp ניתן למצוא כאן.

מימוש Client/Server באמצעות Sockets API

בסעיף זה עליכם לממש תוכנית Client/Server אשר תאשר ללקוח (client) לשלוח לשרת (server) שאילתות על תחרויות האירוויזיון (Eurovision) לדורותיהן, ועל השרת לענות ללקוח תשובה עם המידע המתאים. בסיס הנתונים עמו השרת יעבוד נלקח מתוך ויקיפדיה (קישור), והוא כבר משובץ בתוך הקוד של תוכנית השלד שתקבלו (מידע נוסף בהמשך).

נתונות לכם שתי תוכניות שלד – client.c ו- server.c. עליכם להוסיף את הפונקציונליות המתאימה, כדי שהשרת יוכל לענות עבור הלקוח על השאילתות הבאות:

1. **winner #year:**
שאילתה זו תגרום לשרת להחזיר ללקוח פרטים על המדינה שניצחה בתחרות האירוויזיון שהתקיימה בשנת #year. בשלדת התוכנית server.c מסופקת לכם פונקציה הפונקציה הבאה:

```
void print_winner(...)
```

השתמשו בפונקציה זו כדי לייצר את התשובה אשר השרת יחזיר ללקוח עבור השאילתה הנ"ל.

שימו לב: בין שני חלקי פקודה יכולים להיות אחד או יותר רווחים (whitespaces). גם לפני תחילת הפקודה ולאחר סוף הפקודה יכולים להופיע אפס או יותר רווחים.

2. **victorious #from_year #to_year #country:**
שאילתה זו תגרום לשרת להחזיר ללקוח פרטים על מספר הפעמים שמדינה #country ניצחה את תחרות האירוויזיון בין השנים #from_year ו- #to_year (כולל שני קצוות הקטע). בשלדת התוכנית server.c מסופקת לכם פונקציה הפונקציה הבאה:

```
void print_times_victorious(...)
```

השתמשו בפונקציה זו כדי לייצר את התשובה אשר השרת יחזיר ללקוח עבור השאילתה הנ"ל.

3. **runner-up #from_year #to_year #country:**
שאילתה זו תגרום לשרת להחזיר ללקוח פרטים על מספר הפעמים שמדינה #country סיימה במקום השני בתחרות האירוויזיון בין השנים #from_year ו- #to_year. בשלדת התוכנית server.c מסופקת לכם פונקציה הפונקציה הבאה:

```
void print_times_runner_up(...)
```

השתמשו בפונקציה זו כדי לייצר את התשובה אשר השרת יחזיר ללקוח עבור השאילתה הנ"ל.

הערות נוספות:

1. ניתן להניח כי השרת והלקוח יתקשרו דרך post מספר 8888.
2. ניתן להניח כי השרת נמצא ב- localhost, כלומר בכתובת 127.0.0.1.
3. ניתן להניח כי השרת והלקוח תמיד שולחים זה לזה הודעות בגודל 300 בתים.
4. התקשורת בין הלקוח לשרת תסתיים כאשר המשתמש יקליד את הפקודה "exit" בצד הלקוח.
5. השתמשו ב- strtok כדי להפריד את הפקודה לחלקיה, ע"י שימוש בתו רווח (whitespace) כמפריד בין חלקי הפקודה (delimiter).
6. אם המשתמש הקליד בצד הלקוח פקודה שלא תואמת את אחת משלושת התבניות הנ"ל, השרת יחזיר ללקוח את המחרוזת **error** (אפילו אם המשתמש הקליד פקודה עם רישא חוקית, אך הוסיף פרמטר נוסף שלא לצורך).
7. אם המשתמש הקליד פקודה חוקית לפי אחת משלושת התבניות לעיל, אך השתמש בשנה שאינה נמצאת בבסיס הנתונים, השרת יחזיר ללקוח את המחרוזת **error**.
8. אם #from_year גדול מ- #to_year השרת יחזיר ללקוח את המחרוזת **error**.
9. השרת צריך להיות case-insensitive לגבי המדינה שהמשתמש מקליד. כלומר, אם המשתמש הקליד:

```
victorious 1990 2005 Israel
```

אז השרת צריך להחזיר בדיוק את אותה התשובה שהוא היה מחזיר עבור:

```
victorious 1990 2005 israel
```

10. **ובמקום להרבות במילים, ניתן לצפות בלינק זה בדוגמת הרצה.**

קומפילציה והרצה

תוכנית Shell

בדומה למפורט בתרגיל בית 0, כדי להדר את התוכנית, הריצו את הפקודה הבאה:

```
gcc -Werror -std=c99 myshell.c -o myshell
```

בכדי להריץ התוכנית, הריצו את הפקודה:

```
./myshell
```

תוכנית Client/Server

בכדי להדר את התוכניות, הריצו את הפקודות הבאות:

```
gcc -Werror -std=c99 server.c -o server
```

```
gcc -Werror -std=c99 client.c -o client
```

כעת, פתחו שני חלונות Linux Shell (כפי שניתן לראות בוידאו לעיל), נווטו לסיפריה שבה הדרתם את התוכניות, והריצו בחלון אחד את:

```
./server
```

ובחלון השני הריצו:

```
./client
```

כעת, הקלידו את הפקודות לתוך החלון של ה- client. שימו לב, חובה להריץ תחילה את ה- server, ורק לאחר מכן להריץ את client.

הגשה

ההגשה הינה אלקטרונית דרך Moodle. עקבו אחר השלבים הבאים:

1. עליכם ליצור קובץ zip (השתמשו ב-zip או gzip בלבד) בשם hw1_id1_id2 כאשר id1,id2 מייצגים את מספרי תעודות הזהות של המגישים.
2. תכולת קובץ ה zip צריכה להיות התכולה הבאה (ללא תתי ספריות!):
   - myshell.c
   - client.c
   - server.c
   - dry.pdf (עם התשובות של החלק התיאורטי – **אך ורק קובץ PDF**).
   - קובץ בשם submitters.txt שמכיל את מספרי הזהות והשמות של מגישי התרגיל מופרדים על ידי פסיק במבנה הבא (לדוגמה):

```
Bill Gates,bill@microsoft.com,123456789

Linus Torvalds,linus@gmail.com,234567890
```

3. את קובץ ה- zip יש ליצור ע"י הרצת הפקודה הבאה:

```
zip hw1_id1_id2.zip myshell.c client.c server.c dry.pdf submitters.txt
```

4. הגישו את קובץ ה- zip דרך Moodle.