תרגיל ריצה מספר 1:
fork, wait, execv, dup, signal

הוראות כלליות

- ההגשה, הן של קבצי הקוד והן של מסמך התיעוד החיצוני, תיעשה בצורה אלקטרונית ע"י הפקודה submitex. יש להקיש את הפקודה מחשבון ה-Unix שלכם במחשבי ה-Sunshine שבמחלקה (שימו לב: אין להקיש את הפקודה מחשבון של משתמש אחר!)
- בנוסף יש להגיש מסמך "תיעוד חיצוני". מסמך זה **לא מחליף** את התיעוד הפנימי בתוכנית (הערות).
- ההגשה תיעשה **ביחידים** בלבד. אין להגיש בזוגות או בקבוצות גדולות יותר.
- יש להקפיד על ההוראות המצויות במסמכים: guidelines.pdf ו-codingStyle.doc.
- יש לקמפל את התוכנית באמצעות gcc במחשבי ה-Sunshine שבמחלקה.
- בעת חישוב הציון לתרגיל יילקחו בחשבון המרכיבים הבאים (סדר חשיבות יורד): נכונות התוכנית, יעילות התוכנית, קריאות התוכנית.
- **כדאי להתחיל את התרגיל מוקדם!**

שימו לב:
תרגיל זה כולל 3 שלבים. שלב 1 מהווה תוכנית נפרדת משלבים 2 ו-3. שלב 3 הוא המשך של שלב 2 (הוספת אלמנטים מתקדמים לתוכנית מתרגיל 2).
יש להגיש את **שלב 1** ואת **שלב 3**. הגשתם תיעשה במועד אחד, אולם מומלץ לסיים את שלב 1 מוקדם.

שימו לב: המערכת לא תאפשר הגשת שלב לאחר מועד ההגשה הרשמי.

הגשת התרגיל (כל שלב בנפרד) תיעשה ע"י הקשת הפקודה submitex:
שם התרגיל - שלב א': fork_1
שם התרגיל - שלבים ב' + ג': fork_2

תאריך הגשה: 5.5.02 עד 23:59.

שלב א': Multi-tasking באמצעות fork:

בשלב זה עליכם ליצור תוכנית שמטרתה לבצע משימה פשוטה באמצעות חלוקה לתתי-משימות ותהליכים. כל תהליך יבצע תת-משימה ויחזיר את התוצאה לתהליך הראשי שיחזיר את התוצאה הכללית.

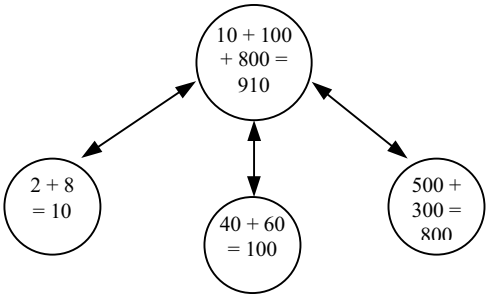
המשימה שיש לבצע מחולקת לפי מספרי תעודת זהות. יש לעשות רק את המשימה המתאימה למספר תעודת-הזהות שלכם. באם תהיה חוסר התאמה הציון יהיה אוטומטית 0 והתרגיל לא ייבדק.

הערה: ניתן להניח שאף תהליך בן לא מחזיר מספר שלילי או מספר גדול מ-255 (אין זה אומר שאין לטפל במספרים שליליים, זה רק אומר שהתוצאה המוחזרת תהיה בסדר).

א. סטודנטים שמספר תעודת-הזהות שלהם (ללא ספרת הביקורת) מסתיים ב-1, 3, 5:

ביצוע חיבור:
על התוכנית הראשית לקבל 2 מספרים ולהחזיר את תוצאת החיבור של שני המספרים. פעולת החיבור תעשה ספרה-ספרה ע"י תהליכים-בנים שיחזירו את התוצאה לתוכנית הראשית. התוכנית הראשית תאסוף את כל התוצאות ותחזיר את התוצאה הכללית, לדוגמה:

```
% a.out 542 368
```



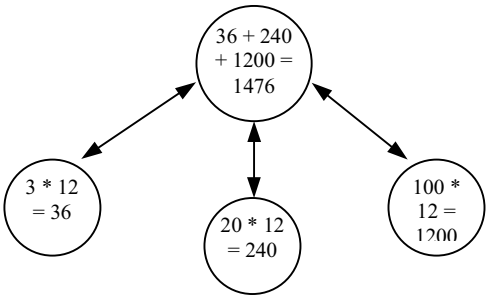
שימו לב: על התוכנית לטפל גם באפשרות שאחד או שני המספרים הוא שלילי.

ב. סטודנטים שמספר תעודת-הזהות שלהם (ללא ספרת הביקורת) מסתיים ב-0, 4, 7:

ביצוע כפל:

על התוכנית הראשית לקבל 2 מספרים ולהחזיר את תוצאת הכפל של שני המספרים. פעולת הכפל תעשה ע"י חלוקת המספר השני לספרות וכפילתו במספר הראשון (כמו בכפל ארוך). התוכנית הראשית תאסוף את כל התוצאות ותחזיר את התוצאה הכללית, לדוגמה:

```
% a.out 12 123
```



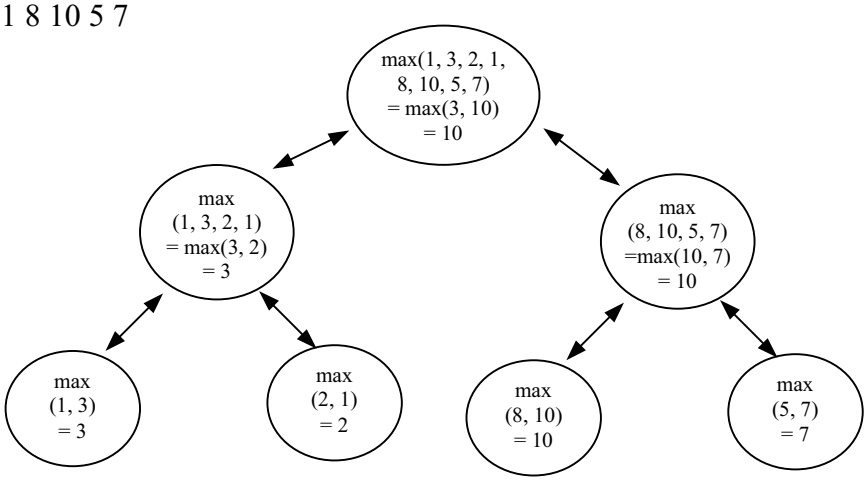
שימו לב: על התוכנית לטפל גם באפשרות שאחד או שני המספרים הוא שלילי.

ג. סטודנטים שמספר תעודת-הזהות שלהם (ללא ספרת הביקורת) מסתיים ב-2, 6, 8, 9:

חישוב מקסימום:

על התוכנית הראשית לקבל רשימת מספרים שלמים (חיוביים או שליליים; אין צורך לבדוק את תקינות המספרים) ולהחזיר את המספר המקסימלי מביניהם. חישוב המקסימום יעשה ע"י תהליכי-בנים באופן רקורסיבי, באופן הבא: תהליך האב שולח חצי מהרשימה לתהליך-בן אחד וחצי מהרשימה לתהליך בן שני. כל תהליך בן כזה שולח חצי מהמספרים שקיבל לתהליך-בן נוסף וחצי אחר לתהליך בן אחר וכן הלאה, עד שתהליך מקבל 2 מספרים ומחזיר את המקסימום ביניהם ומשרשר כלפי מעלה, וכל תהליך משרשר כלפי מעלה את המקסימום מבין 2 מספרים, עד התהליך הראשי. לדוגמה:

```
% a.out 1 3 2 1 8 10 5 7
```



שימו לב: על התוכנית לטפל גם באפשרות שאורך רשימת המספרים הוא אי-זוגי.

מועד הגשת שלב א' (קוד + תיעוד חיצוני): 5.5.02

שלב ב'+ג': Mini-Shell:

רקע:

1) ה-shell הנה תכנית פשוטה הכתובה ב-C. התוכנית מציגה על המסך סמן (=prompt) ומאפשרת למשתמש להקליד פקודות ב-Unix. לאחר לחיצה על Enter, תבוצע הפקודה שהוקלדה ויוצג prompt חדש, עבור פקודה חדשה.

מטרה:

2) עליכם לכתוב shell פשוט המאפשר למשתמש להריץ shell scripts (**שלב ב'**).
3) ה-shell אותו תממשו יאפשר הרצת ברקע ובחזית (**שלב ב'**).
4) יש לטפל גם ב-pipe וב-redirection ובשליחת signals (**שלב ג'**).

שלב ב': יצירת shell להרצת shell scripts:

1. עליכם לכתוב shell שיציג בפני המשתמש prompt. המשתמש יוכל להזין שמות של shell scripts ועל ה-shell שלכם יהיה לבצע את ה-shell script .
2. אופן הרצת ה-shell sciprt יהיה על-ידי פענוח ה-shell sciprt **ולא** ע"י שימוש ב-execv עם שם ה-script.
3. **לא צריך לטפל** באפשרות הזנת ארגומנטים ל-shell sciprt.

4. **אופן הפענוח:**
יש לעבור על ה-shell sciprt שורה-שורה ולבצע את הפקודות בכל שורה. הגדרת מבנה של shell sciprt מוגדר בנספח. **שימו לב: מבנה ה-shell script מחולק לפי ת.ז. ואין צורך לממש פונקציונליות נוספות של קבוצות אחרות.** אין צורך לטפל ב-shell sciprt מסובכים יותר.
5. ה-shell שאותו תכתבו לא ינסה להבין את הפקודות *command* שב-shell, אלא יעביר את הפקודות והארגומנטים למערכת בעזרת קריאה ל-execv (או execve). כזכור, אין החזרת שליטה לתכנית לאחר קריאה ל-execv, ולכן יש ליצור תהליך חדש (fork) לפני הקריאה ל-execv. הקריאה ל-execv תבוצע בתהליך הבן.
6. אין צורך לבדוק את תקינות הפקודות ב-shell sciprt.
7. על ה-shell שלכם לאפשר הרצת ה-shell sciprt הן ברקע (ע"י הקשת & בסוף הפקודה) או בחזית (ברירת המחדל).
8. לאחר הרצת תהליך **ברקע** יש להציג למסך את ה-pid של התהליך החדש וכן את ה-jid שלו (מס"ד עבור הפקודה, כאשר רשימת המספרים תתחיל ב-1).
9. הקשת הפקודה ***jobs*** תציג את רשימת הפקודות הרצות ברקע ברגע הרצת הפקודה (**זוהי פקודה שעליכם לממש בעצמכם**).
10. ניתן להניח שכל הגדרות המשתנים (set var = int) נעשות בתחילת ה-shell-script.
11. כל הפעולות האריתמטיות הן פשוטות - כלומר 2 משתנים/ערכים ואופרטור אחד, למשל:

@ num = $num + 1 // **possible**
@ num = $num + 1 * 3 // **not possible**

12. גם התנאי עבור ה-if וה-while הוא תנאי פשוט.
13. אחרי if יש רק פקודה אחת. פקודה זו היא פקודת shell רגילה (ולא פקודת shell-script) (כלומר, צריך לשלוח אותה ל-execl לאחר הפרדת הארגומנטים ובדיקה אם היא צריכה להיעשות ברקע או לאו).
14. אחרי ה-while יכולות להיות כמה פקודות (עד ה-end). פקודות אלו הם כמו בסעיף 13 לעיל.

<u>הנחיות:</u>
1) עבור הפקודות שרצות ברקע, אין להמתין לסיום תהליך הבן, אלא יש להציג על המסך את ה-process id של התהליך המבוצע ברקע ולאפשר הזנת פקודות נוספות באופן מיידי.
2) יש לשמור במערך את רשימת הפקודות המבוצעות בכדי להציגן בעת הרצת 'jobs'.
3) הפקודה jobs תבוצע בחזית (foreground) ולא ברקע כשאר הפקודות. כמו כן, אין להקצות לה מספר job משל עצמה.

<u>הערות כלליות</u>:
4) הפקודה execv מקבלת מערך של מחרוזות. התא הראשון יכיל את הפקודה לביצוע ושאר התאים יכילו את הארגומנטים. כדי לפרק את שורת הקלט למילים ניתן להשתמש ב-strtok.
5) שימו לב: יש לדאוג כי לא יישארו תהליכים במצב ***zombie*** לאורך זמן. דהיינו, יש לבצע wait בשלב מסוים עבור הפקודות שהורצו ברקע והסתיימו. יש לדאוג שה-wait לא יתקע עד שיסתיימו כל התהליכים שרצים ברקע, אלא רק ישחרר תהליכים שכבר הסתיימו.
6) לאחר סיום פקודה, יש להוציאה ממערך הפקודות (=רשימת ה-jobs).
7) תהליך ניקוי ה-zombies וניקוי המערך, יכולים להתבצע במרוכז אחת לכמה פקודות ואין צורך לבצעם לאחר כל פקודה. כמובן שהנחה זו עלולה לפגום בנכונות רשימת ה-jobs שלכם, אך לצורך תרגיל זה, זה יספק.

<u>שלב ג': טיפול ב-pipe וב-redirection ובשליחת signals:</u>

1. על ה-shell שלכם לתמוך ב-*command* המשלב גם pipes ו-redirections, למשל:

```
ls -l | sort
ls -l > file
```

יש לתמוך ב-redirections הבאים: <<, <, >, >>, וב-pipe: |
ניתן להניח שעבור < הקובץ אינו קיים לפני ה-redirection.

2. על ה-shell שלכם לתמוך בהריגת תהליכים שרצים ברקע. הקשת הפקודה **kill pid** (כאשר pid הוא מספר תהליך כלשהו) תגרום לשליחת signal שיהרוג את התהליך.
עליכם לממש את זה ולא לקרוא לביצוע הפקודה kill ע"י excev.

<u>תאריך הגשת שלב ג':</u> 5.5.02

נספח: מבנה של Shell Script

definition: set variable_name = int_value

command: any existing shell command, including arithmetic commands, containing the operators: +, -, *. For example:
date
@ num = $num + 3

simple-condition: Boolean condition, containing the operators: >, <. ==, !=.
No need to implement: -d, -e
For example:
if (-d $file_name)
while ($num < 10)

Note: A *simple-condition* doesn't contain any commands. It only contains the Boolean operators as above.

ID's that end with 2, 6, 8, 9 should also handle "if":
if: if (*simple-condition*) *command*

ID's that end with 0, 1, 3, 4, 5, 7 should also handle "while":
while: while (*simple-condition)*
command
command
…
end

Note: For phase A) the shell script will not include pipes and redirections in it.

Reminder: The first line in the script is #!/bin/csh
To access a value of a variable the script uses $.
To do arithmetic operation the script uses @.

דוגמת ריצה:

הערות:

1. בדוגמה שלהלן, ניקוי ה-zombies ומערך הפקודות מבוצע לאחר כל 4 פקודות.
2. ה-script להלן מכיל מקרים עבור 2 הקבוצות (if ו-while).

```
% cat myScript.csh
#!/bin/csh
set num = 5
if ($num > 4) ls -l /home
set i = 0
while ($i < 2)
        sleep 100 &
        @i = $i + 1
end
cat hello.doc &
if (-d dir1) jobs                              /* assuming dir1 is a valid directory */
sleep 100 &
jobs
cat hello.doc &

% my_shell
my_prompt> some_script.csh
Error: Can't execute file some_script.csh
my_prompt> my_script.csh
                ls –1  /home
mike/
amir/
oren/
[1] 29545       sleep 100 &
[2] 29547       sleep 100 &
[3] 29549       cat hello.doc &
hello there.
This is the contents of the file hello.doc
                jobs
1       29545           sleep 100
2       29547           sleep 100
3       29549           cat hello.doc
[4] 29551       sleep 100 &
                jobs
1       29545           sleep 100
2       29547           sleep 100
4       29551           sleep 100
```

מערכות הפעלה 88-288

תרגיל בית תכנותי

מספר 1

שלב א'

מגיש/ה:

שם מלא: _____

ת.ז./דרכון:

	-								

מספר קבוצה:

		-	8	8	2	-	8	8	

ציון:

תיעוד חיצוני	/15
קוד	/85
סה"כ	/100

מערכות הפעלה 88-288

תרגיל בית תכנותי

מספר 1

שלבים ב' + ג'

מגיש/ה:

שם מלא: _____

ת.ז./דרכון:

	-								

מספר קבוצה:

		-	8	8	2	-	8	8	

ציון:

תיעוד חיצוני	/15
קוד	/85
סה"כ	/100

1. הנחיות כלליות בנוגע לתרגילים מעשיים:

- מדיניות איחורים:
תרדנה 0.5 נקודות עבור כל שעת איחור עד 24 שעות.
החל מהשעה ה-25 ועד 48 שעות איחור: תרדנה 2 נקודות עבור כל שעת איחור.
לאחר 48 שעות לא תתאפשר ההגשה והציון יהיה 0.
לדוגמה, הציון המקסימלי עבור סטודנט/ית שהגיש/ה תרגיל 29 שעות לאחר המועד יהיה 78.

- מדיניות העתקות:
תרגיל שייחשד כמועתק יקבל ציון 0. עבור תרגיל עבורו יש חובת הגשה על-מנת לעבור את הקורס, הדבר יגרור כשלון בקורס.

נושאי האיחורים וההעתקות יאכפו כמה שניתן ולא תהיינה התפשרויות בנושאים אלו.

אין לפנות למתרגלים בעקבות הורדת ניקוד עקב איחורים. אם ידועה סיבה מוצדקת לאיחור יש לפנות **לפני** הגשת התרגיל ולקבל אישור מהמתרגל.

- הגשה:

א. קבצי קוד:
הגשת הקוד (ואך ורק הקוד - .c, .h) תתבצע ע"י הפקודה submitex אותה יש להקיש במחשבי ה-Sunshine שבמחלקה.

% submitex 88-288-## exercise-name file ...

כאשר ## מציין את מספר קבוצת התרגול.
שם התוכנית (exercise-name) יצוין בכל תרגיל.
יש להקיש news submitex בתחנות ה-Sunshine לקבלת פרטים נוספים.

את שמות הקבצים בתרגיל יש ליצור לפי תעודת הזהות, כאשר אם יש כמה קבצים יש ליצור אינדקסים בסדר רץ.
כמו כן יש לעיין בקובץ ***codingStyle.doc*** ולבצע את ההנחיות הכתובות בו.

דוגמה לאופן הגשת התרגיל:

```
% submitex 88-288-05 fork_1 012345678_1.c 012345678.h 012345678_2.doc ...
```

יש לשים לב שהפקודה מחזירה לכם הודעה שהקובץ שלכם התקבל בהצלחה.

שימו לב: הפקודה אינה מודיעה על שגיאה עבור שמות קבצים לא חוקיים (למשל, הקבצים אינם נשמרו לפי ת.ז.). במקרה כזה האחריות מוטלת עליכם וקבצים שלא יעמדו בהנחיות **לא יבדקו**.

אין לשלוח תרגילים בדוא"ל.

ב. תיעוד חיצוני:
בנוסף לקבצי הקוד יש להגיש מסמך שיכיל תיעוד חיצוני של התוכנית. מסמך זה יכיל תכנון מפורט של עיצוב התוכנית (design plan). כלומר:

- הסבר האלגוריתמים ומבני הנתונים בהם השתמשתם בתוכנית;
- המוטיבציה מאחורי בחירת כל אלגוריתם ומבנה נתונים;
- עץ פרוצדורות.

- דוגמאות ריצה והסבר מדוע נבחרו דווקא דוגמאות ריצה זו ומדוע דוגמאות הריצה מכסות את כל המקרים שהתוכנית צריכה לרוץ עליהם.

כיוון שאין להגיש את קוד התוכנית מודפס, ניתן לכלול קטעים נבחרים של הקוד בתוך המסמך, היכן שהדבר מתבקש.
את התיעוד החיצוני יש להגיש ביחד עם קוד התרגיל ע"י הפקודה submitex (ר' הסבר לעיל).

משקל התיעוד החיצוני יהווה 15% מהציון בכל תרגיל ולכן יש להשקיע מחשבה בכתיבתו.

העמוד הראשון של המסמך יהיה "עמוד שער". לכל תרגיל יצורף עמוד שער אותו יש למלא בכתב ברור. **חובה** שהעמוד הראשון בכל תרגיל יהיה עמוד שער כמצורף לתרגיל. תרגילים שלא יכללו עמוד כנ"ל **לא יבדקו**.