תרגיל בית #1

מטרות התרגיל:

1. להעמיק את הבנתכם בנושא תהליכים וזימון תהליכים ב-Linux.
2. הכרות בסיסית עם קריאות מערכת אלמנטריות.
3. הבנה של נושא האיתותים ב-Linux.

תרגיל רטוב: כתיבת smash

שימו לב: מקוריות הקוד תיבדק, להזכירכם העתקת שיעורי בית הינה עבירת משמעת בטכניון לכל המשתמע מכך.

עליכם לכתוב תוכנית אשר תשמש כ-shell חדש למערכת ההפעלה Linux. התוכנית תבצע פקודות שונות אשר יוקלדו ע"י משתמש.

השם smash נגזר מצרוף המילים **Sma**ll **Sh**ell.

<u>התוכנית תפעל בצורה הבאה:</u>

- התוכנית ממתינה לפקודות אשר יוקלדו ע"י המשתמש ומבצעת אותן (וחוזר חלילה).
- התוכנית תוכל לבצע מספר קטן של פקודות built-in, הפקודות יפורטו בהמשך.
- כאשר התוכנית תקבל פקודה שהיא לא אחת מפקודות ה-built-in היא תנסה להפעיל אותה כמו shell רגיל. אופן הפעלת פקודה חיצונית יפורט בהמשך.
- במידה והוכנסה פקודת built-in עם פרמטרים לא חוקיים או כמות פרמטרים לא נכונה, תופיע הודעת השגיאה הבאה:

```
smash error: "line"
```

כאשר:

- הגרשיים יופיעו בהודעת השגיאה.
- *line* היא שורת הפקודה כפי שהוקשה על ידי המשתמש.

* על השגיאות שפורטו ויפורטו בהמשך, התוכנית מגיבה בהדפסת הודעת שגיאה מתאימה ועוברת לפענוח וביצוע שורת הפקודה הבאה.
* כאשר התוכנית ממתינה לקלט מהמשתמש מודפסת בתחילת שורה חדשה ההודעה[1]

```
smash >
```

<u>הנחות:</u>

* כל פקודה מופיעה בשורה נפרדת ואורכה לא יעלה על 80 תווים.
* ניתן להניח שמספר הארגומנטים המקסימאלי הינו 20.
* ניתן להשתמש בכל מספר רווחים בין מילים המופיעות באותה שורת פקודה, ובתחילת השורה.
* כל פקודה נגמרת בתו '\n'.
* ניתן להשאיר שורות ריקות.
* על התוכנית לתמוך בעד 100 תהליכים הרצים בו זמנית.

אופן פענוח שורת פקודה ב smash

כפי שיפורט בהמשך התוכנית תבדוק אם הפקודה היא פקודת built-in או פקודה חיצונית, ותטפל בפקודה בהתאם.

סוגי הפקודות הנתמכות

על smash לתמוך בשני סוגים של פקודות: פקודות built-in ופקודות חיצוניות. פקודות built-in הן פיצ'רים שהטרמינל מממש עבור המשתמש. פקודות built-in בדרך כלל רצות כחלק מהקוד של הטרמינל (אותו התהליך) ולא מייצרות תהליך חדש (fork+exec) לשם הרצתן. באופן כללי, פקודות built-in הן פשוטות – בדר"כ מריצות מספר קריאות מערכת לשם מימושן ומעדכנות מבני נתונים פנימיים.

פקודות חיצוניות (external) דורשות הרצה של executable חיצוני. לשם כך, על הטרמינל לייצר תהליך בן שיריץ את קובץ ההרצה (באמצעות fork ו-execv כפי שנלמד בתרגול).

---

[1] ההודעה "smash >" ידועה גם בשמה הטכני - prompt.

Jobs

Job הוא תהליך שהטרמינל מנהל. כלומר, תהליך בן שהטרמינל יצר. לכל job קיים מזהה ייחודי (Job ID) שהטרמינל נותן ל-job כאשר הוא נכנס לרשימת ה-jobs. לאחר מכן, המזהה אינו משתנה. כיוון שה-job הוא גם תהליך בן, הוא גם מזוהה באמצעות מזהה תהליך (Process ID).
כל Job יכול להיות באחד מהמצבים הבאים:

1. חזית (Foreground) – כאשר מריצים פקודה בטרמינל, הפקודה מתבצעת מיד והטרמינל ממתין עד לסיום הפקודה. בזמן זה, ה-prompt אינו מוצג והטרמינל ממתין לסיום תהליך הבן שאותו יצר.
2. רקע (Background) – כאשר משורשרת לפקודה התו '&', יש להריץ את הפקודה ברקע. כאשר פקודה רצה ברקע, הטרמינל אינו ממתין עד לסיום הפקודה ומיד מציג את ה-prompt ומוכן להרצת פקודה נוספת. במקביל, תהליך הבן מריץ את הפקודה.
3. עצור (Stopped) – משתמשים יכולים לעצור תהליך הרץ בחזית ע"י הקשה על Ctrl+Z. בשלב זה, ה-job ישאר במצב עצור (כלומר, ריצתם נעצרת) עד שישלח אליהם הסיגנל SIGCONT שבעקבותו ימשיכו את ריצתם.

את כל ה-Jobs הטרמינל מנהל ברשימת ה-Jobs. כל job שנמצא במצב 2 או 3 יתווסף לרשימת ה-jobs. בצורה כזו, משתמשים יכולים לנהל את ה-jobs ע"י הפעלת פקודות המנהלות את הרשימה. למשל, משתמש יכול להדפיס את כל ה-jobs שאותם הטרמינל מנהל כרגע באמצעות פקודת ה-built-in: 'jobs'.
כפי שנתאר בהמשך, jobs יכולים להיות מוסרים מהרשימה באמצעות הפקודה fg אשר בוחרת job מרשימת ה-jobs ומעבירה אותו לרוץ בחזית (foreground). אם job הוסר מהרשימה באמצעות fg, ניתן להחזיר אותו לרשימה באמצעות Ctrl+Z. בנוסף, Job יוסר מהרשימה גם כאשר הוא מסתיים.
<u>מתי יש להסיר Job שהסתיים מהרשימה?</u> לפני הרצת כל פקודה, לפני הדפסת רשימת ה-jobs ולפני הוספה של job חדש לרשימה.
<u>איך נבחר מזהה ה-Job?</u> Job חדש הנכנס לרשימה מקבל מזהה בהכנסה הראשונה שלו. אותו המזהה לא משתנה מנקודה זו והלאה. המזהה נקבע להיות המזהה המקסימלי מבין כל ה-jobs שנמצאים כרגע ברשימה + 1. אם הרשימה ריקה, המזהה יהיה 1. אם קיימים job שהסתיימו ברשימה, יש קודם להסיר אותם מהרשימה ורק אז להקצות מזהה ל-job חדש.

פקודות built-in של smash :

יש לתמוך בפקודות הפנימיות הבאות. שימו לב: על הפקודות לרוץ בתהליך ה-smash. אין ליצור תהליך חדש לשם הרצת הפקודות או להשתמש בקריאת המערכת system. עליכם לממש את הפקודות באמצעות שימוש בקריאות מערכת ועדכון מבני נתונים פנימיים של ה-smash. פקודות פנימיות רצות רק בחזית ואינן יכולות לרוץ ברקע, ולכן יש להתעלם מהסימן '&' עבור פקודות פנימיות.

showpid

תיאור: הפקודה מדפיסה את ה-PID של ה-smash. לשם מימוש הפקודה יש להעזר בקריאת המערכת getpid().
דוגמה:

```
smash > showpid
smash pid is 12339
```

שגיאות: אם הועברו ארגומנטים, יש להתעלם מהם.

pwd

תיאור: הפקודה מדפיסה את הנתיב המלא של המדריך הנוכחי. באמצעות הפקודה הבאה (cd) ניתן להחליף את המדריך הנוכחי. לשם מימוש הפקודה יש להעזר בקריאת המערכת getcwd() או בוריאציה שלה.
דוגמה:

```
smash > pwd
/home/OS/046209/smash
```

שגיאות: אם הועברו ארגומנטים, יש להתעלם מהם.

cd <path>

תיאור: הפקודה מקבלת כקלט ארגומנט יחיד (path) שמתאר את הנתיב הרלטיבי או המלא שיהווה המדריך הנוכחי החדש.
במקרה בו path שווה ל "-", משנים את המדריך הנוכחי אל הקודם. אם אין מדריך קודם, יש להדפיס שגיאה. צריך לזכור רק מדריך אחד אחורה. לשם מימוש הפקודה יש להעזר בקריאת המערכת chdir().
לדוגמא:

```
smash > cd -
smash error: cd: OLDPWD not set
smash > cd a b
smash error: cd: too many arguments
smash > cd /home/OS
smash > pwd
/home/OS
smash > cd ..
smash > pwd
/home
smash > cd -
smash > pwd
/home/OS
smash > cd -
smash > pwd
/home
```

שגיאות: אם הועבר יותר מארגומנט אחד יש להדפיס את ההודעה:

```
smash error: cd: too many arguments
```

אם לא קיים מדריך קודם על "cd -" להדפיס:

```
smash error: cd: OLDPWD not set
```

jobs

תיאור: הפקודה jobs מדפיסה את רשימת ה-jobs אשר מכילה: jobs שלא הסתיימו ורצים ברקע ו-jobs במצב stopped. פורמט ההדפסה עבור jobs במצב stopped הינו:

```
[<job-id>] <command> : <process id> <seconds elapsed> (stopped)
```

פורמט ההדפסה עבור jobs שרצים ברקע:

```
[<job-id>] <command> : <process id> <seconds elapsed>
```

כאשר <seconds elapsed> הוא הזמן שעבר מאז שה-job הוכנס לרשימה. יש להעזר בקריאת המערכת time() ובפונקציה difftime().
יש להדפיס את הרשימה בצורה ממוינת בסדר עולה לפי המזהה ה-job. יש להסיר את כל ה-jobs שהסתיימו לפני הדפסת הרשימה.
שימו לב: אם job חוזר לרשימה לאחר שיצא ממנה, יש לעדכן את הזמן הכנסה שלו.
דוגמה:

```
smash > /bin/sleep 100&
```

```
smash > /bin/sleep 200
^Zsmash: caught ctrl-Z
smash: process 234 was stopped
smash > jobs
[1] /bin/sleep 100& : 12340 18 secs
[2] /bin/sleep 200 : 234 11 secs (stopped)
```

שגיאות: אם הועברו ארגומנטים, יש להתעלם מהם.

kill -<signum> <job-id>

תיאור: הפקודה שולחת את ה-Signal שמספרו signum אל ה-job עם המזהה job-id (מרשימת jobs) ומדפיסה למסך הודעה מתאימה.

דוגמה:

```
smash > kill -9 1
signal number 9 was sent to pid 30985
smash > kill -8 3
smash error:  kill: job-id 3 does not exist
```

שגיאות: אם הועבר מזהה job שלא קיים, יש להדפיס את ההודעה הבאה:

smash error: kill: job-id <job-id> does not exist

אם הסינטקס (מספר הארגומנטים או הפורמט) לא תקין, יש להדפיס את ההודעה הבאה:

smash error: kill: invalid arguments

fg <job-id>

תיאור: הפקודה תגרום להרצה ב- foreground של job עם המזהה job-id. הפקודה מדפיסה למסך את ה-command line של אותו ה-job ואת מזהה התהליך (ראו דוגמה) ואז שולחת את הסיגנל SIGCONT לתהליך וממתינה לסיומו (יש להעזר בקריאת המערכת waitpid()). הפעלת הפקודה ללא פרמטרים, תעביר ל–foreground את ה-job בעל המזהה המקסימלי מרשימת ה-jobs. לאחר העברת ה-job לחזית, יש להסיר את ה-job מהרשימה.

דוגמא:

```
smash > /bin/sleep 10 &
smash > /bin/sleep 20 &
smash > /bin/sleep 30 &
smash > /bin/sleep 40 &
smash > jobs
[1] /bin/sleep 10 & : 123 14 secs
[2] /bin/sleep 20 & : 124 11 secs
[3] /bin/sleep 30 & : 125 8 secs
[4] /bin/sleep 40 & : 126 1 secs
smash > fg 2
/bin/sleep 20 & : 124
```

שגיאות: אם הועבר מזהה job שלא קיים, יש להדפיס את ההודעה הבאה:

smash error: fg: job-id <job-id> does not exist

אם לא הועבר job-id והרשימה ריקה, יש להדפיס את ההודעה הבאה:

smash error: fg: jobs list is empty

אם הסינטקס (מספר הארגומנטים או הפורמט) לא תקין, יש להדפיס את ההודעה הבאה:

smash error: fg: invalid arguments

bg <job-id>

תיאור: הפקודה תחזיר לריצה ברקע job שבמצב stopped עם המזהה job-id. על הפקודה להדפיס תחילה את ה-command line של ה-job ולאחר מכן להחזיר אותו לריצה (באמצעות שליחת סיגנל SIGCONT). על ה-job להמשיך לרוץ ברקע. הפעלת הפקודה ללא פרמטרים, תעביר ל– background את התהליך התהליך עם job id מקסימלי שריצתו הושהתה. לאחר החזרת ה-job לריצה ברקע, הסימון stopped לא יהיה מוצג ברשימת ה-jobs.

דוגמא:

```
smash > jobs
[1] /bin/sleep 10 & : 123 14 secs
[2] /bin/sleep 20 & : 124 11 secs
[3] /bin/sleep 30 & : 125 8 secs (stopped)
[4] /bin/sleep 40 & : 126 1 secs
smash > bg 3
/bin/sleep 30 & : 125
```

שגיאות: אם הועבר מזהה job שלא קיים, יש להדפיס את ההודעה הבאה:

smash error: bg: job-id <job-id> does not exist

אם הועבר מזהה job קיים אבל ה-job לא במצב stopped, יש להדפיס את ההודעה הבאה:

smash error: bg: job-id <job-id> is already running in the background

אם לא הועבר job-id ואין job במצב stopped ברשימה, יש להדפיס את ההודעה הבאה:

smash error: bg: there are no stopped jobs to resume

אם הסינטקס (מספר הארגומנטים או הפורמט) לא תקין, יש להדפיס את ההודעה הבאה:

smash error: bg: invalid arguments

quit [kill]

תיאור: הפקודה מסיימת את ה-smash עם ערך חזרה 0. אם הועבר הפרמטר kill (אופציונאלי) אז יש להרוג את כל ה-jobs לפני סיום התוכנית.

הפקודה quit kill תהרוג את התהליכים לפי האלגוריתם הבא:

1. שליחת סיגנל SIGTERM.
2. (**רק**) אם התהליך לא נהרג אחרי 5 שניות לאחר קבלת סיגנל ה-SIGTERM , שליחת סיגנל SIGKILL.

לדוגמה:

```
smash > jobs
[1] a.out : 12340 56 secs
[2] /usr/bin/ls : 12341 23 secs
[3] b.out : 12342 10 secs
smash > quit kill
[1] a.out – Sending SIGTERM... Done.

[2] /usr/bin/ls – Sending SIGTERM… Done.

[3] b.out – Sending SIGTERM… (5 sec passed) Sending SIGKILL… Done.
```

הערה: תהליך b.out לא הגיב לסיגנל SIGTERM, ולכן נשלח לו גם SIGKILL.

שגיאות: אם הועברו ארגומנטים שאינם "kill", יש להתעלם מהם.

diff <f1> <f2>

תיאור: הפקודה משווה את תוכן הקבצים f1 ו-f2. אפשר להניח כי f1 ו-f2 הינם קבצים ולא תיקיות. הפונקציה תדפיס למסך "1" אם תוכן הקבצים שונה, ואחרת תדפיס "0".

דוגמא:

```
smash > diff a.out b.out
```

1

שגיאות: אם הועבר מספר לא תקין של פרמטרים, יש להדפיס את ההודעה:
smash error: diff: invalid arguments

פקודות חיצוניות ב-smash:

<command> [arguments]

כאשר smash מקבלת פקודה חיצונית (כלומר אינה אחת מהפקודות built-in של smash) היא מנסה להפעיל את התוכנית command. וממתינה עד לסיום ביצוע התוכנית. לדוגמא, הפקודה:
`smash > a.out arg1 arg2`
תגרום להפעלת התוכנית `a.out` עם הארגומנטים `arg1 arg2`. יש להעזר בקריאת המערכת exec().

<command> [arguments] &

כמו בסעיף הקודם, אך ללא המתנה לסיום ביצוע התוכנית (הרצה ברקע). התהליך החדש יכנס לרשימת jobs.

טיפול בסיגנלים:

על ה-shell לתמוך בצירופי המקשים Ctrl+C ו-Ctrl +Z:

- הצירוף Ctrl +C מפסיק את ריצת התהליך שרץ ב-foreground (שולח SIGKILL).
- הצירוף Ctrl +Z משהה את התהליך שרץ ב-foreground (שולח לו SIGSTOP) ומוסיף אותו לרשימת ה-jobs (עם ציון שהתהליך מושהה).

שימו לב שמבחינת bash התהליך שאליו ישלחו הסיגנלים הללו הוא התהליך שמריץ את ה-smash. ולכן, על ה-smash לתפוס את הסיגנל ולשלוח סיגנל מתאים לתהליך שרץ ב-foreground של ה-shell. אם אין תהליך שרץ ב-foreground, צירופים אלו לא ישפיעו על ה-shell.
כמו כן, השהיית או הריגת תהליך יכולה להתבצע גם באמצעות kill עם סיגנל מתאים.
כאשר מתקבל הצירוף Ctrl+C יש לבצע את הפעולות הבאות:

- יש להדפיס למסך את ההודעה:
  smash: caught ctrl-C
- אם קיים תהליך שרץ בחזית, יש לשלוח לו את הסיגנל SIGKILL ולהדפיס את ההודעה הבאה:
  smash: process <process-id> was killed

כאשר מתקבל הצירוף Ctrl+Z יש לבצע את הפעולות הבאות:

- יש להדפיס למסך את ההודעה:
  smash: caught ctrl-Z
- אם קיים תהליך שרץ בחזית, יש להוסיף אותו לרשימת ה-jobs, לשלוח לו את הסיגנל SIGSTOP ולהדפיס את ההודעה הבאה:
  smash: process <process-id> was stopped

דוגמא:

```
smash > /bin/sleep 100
^Csmash: caught ctrl-C
smash: process 123 was killed
smash > /bin/sleep 100
^Zsmash: caught ctrl-Z
Smash: process 125 was stopped
```

```
smash > jobs
[1] /bin/sleep 100 : 125 3 secs (stopped)
```

יצירת תהליכים:

אתם עלולים לגלות כי גם תהליך ה-smash וגם כל תהליכי הבן שלו מקבלים את הסיגנלים ctrl+C ו-ctrl+Z למרות שה-smash שלכם לא שולח אותם לתהליך הבן. בעיה זו מתרחשת בגלל ה-shell האמיתי (tcsh, bash, …) שממנו רץ ה-shell שלכם, אשר שולח את הסיגנל לכל התהליכים בעלי אותו ה group-id. בכדי להימנע מבעיה זאת אתם פשוט צריכים לשנות את ה-group-id של כל תהליך בן אשר ה-shell שלכם מייצר באופן הבא:

```
// here your code runs the child process
pid = fork();
if( pid == 0 ) {
        setpgrp(); // THIS IS THE COMMAND THAT EACH CHILD SHOULD
                        // EXECUTE, IT CHANGES THE GROUP ID
        execv(...);// execute the needed process
        // Handle execv error if you got to this line
} else if( pid>0) {
        // Do parent - shell work here
} else {
        // handle the fork error here
}
```

טיפול בשגיאות:

אם קריאת מערכת נכשלת אז יש להדפיס הודעת שגיאה באמצעות perror() על מנת לדווח על מהות השגיאה. ההודעה שתועבר ל-perror תהיה בפורמט הבא:

"smash error: <syscall name> failed"

לדוגמה, אם קריאה ל-fork נכשלה אז יש לדווח על השגיאה באמצעות:

perror("smash error: fork failed");

שימו לב, את כל הודעות השגיאה שה-smash מדפיס יש להדפיס ל-STDERR ולא ל-STDOUT.

הנחיות לביצוע

* **יש** להשתמש בקריאות המערכת fork ו-exec עבור מימוש פקודות חיצוניות (יש לבחור את הצורה המתאימה של exec לדרישות התרגיל).
* **אין** להשתמש בפונקצית הספרייה system בתרגיל.
* על התוכנית לבדוק הצלחת ביצוע כל פקודה, בכל מקרה של כישלון יש להדפיס הודעת שגיאה מתאימה (תזכורת – perror).
* יש לממש את התרגיל ב-C או ++C בלבד.
* ניתן להשתמש בספריות STL של ++C בחופשיות, ולכן **מומלץ** לכתוב את התרגיל ב-++C על מנת להמנע מכתיבת מבני נתונים ב-C.
* לתרגיל זה מצורף שלד שיעזור לכם בפתרון התרגיל. לא חייב להשתמש בו וניתן לשנות אותו כרצונכם. בשלד קיימות פקודות נוספות שלא מופיעות בתרגיל. **אין צורך לממש אותן**. יש לממש רק מה שמפורט בתרגיל.
* שאלות על התרגיל יש לפרסם בפורום תרגילי בית רטובים. יש לעקוב אחרי הדיון "עדכונים תרגיל רטוב #1" במידה ונעדכן את דרישות התרגיל.

הידור קישור ובדיקה

יש לוודא שהקוד שלכם מתקמפל ע"י הפקודה הבאה:

אם כתבתם ב-C++:

```
> g++ -std=c++11 -Wall -Werror -pedantic-errors -DNDEBUG *.cpp -o smash
```

אם כתבתם ב-C:

```
> gcc -std=c99 -Wall -Werror -pedantic-errors -DNDEBUG *.c -o smash
```

יש לוודא שנוצר קובץ הרצה ללא שגיאות או warnings.

עליכם לספק Makefile עבור בניית הקוד. הכללים המינימליים שצריכים להופיע ב-Makefile הינם:

- כלל smash שיבנה את התוכנית smash.
- כלל עבור כל קובץ נפרד שקיים בפרויקט.
- כלל clean אשר מוחק את כל תוצרי הקימפול.
- יש לוודא שהתוכנית נבנית ע"י הפקודה make.
- יש לקמפל ע"י הדגלים המופיעים בחלק "הידור קישור ובדיקה" לעיל.

לתרגיל זה מצורף סקריפט check_submission.py המוודא (בצורה חלקית) את תקינות ההגשה. הסקריפט מצורף לנוחיותכם, ובנוסף לבדיקה באמצעות הסקריפט, **עליכם לוודא את תקינות ההגשה.**

הסקריפט מצפה ל-2 פרמטרים: נתיב ל-zip, ושם קובץ ההרצה. לדוגמא:

```
> ./check_submission.py 123456789_987654321.zip smash
```

הגשה

הנחיות כלליות על אופן הגשת תרגילי הבית הרטובים ניתן למצוא באתר הקורס תחת הכותרת "עבודות בית – מידע ונהלים".

- אנא עקבו אחר ההנחיות המופיעות בדף הנהלים. יש להגיש קובץ zip (ולא אף פורמט אחר) בלבד.
- אין להגיש קבצי הרצה.
- ניתן להגיש את התרגיל מספר פעמים, רק ההגשה האחרונה נחשבת.
- על ה-Makefile המצורף לייצר קובץ הרצה בשם "smash".

בבקשה, בדקו שהתוכניות שלכם עוברות קומפילציה
וההגשה נעשית על פי הנהלים.
תוכנית שלא תעבור קומפילציה לא תבדק!
הגשה שלא על פי הנהלים תגרור הורדת ציון.

בהצלחה!!!