

תרגיל רטוב 1#



SmaSh – small shell

הקדמה

שיםו לב – מקורות הקוד תבדק, העתקת שיעורי בית היא עברית משמעת בטכניון על כל המשתמע מכך.
הקוד שמוגש שיר لكم גם אם נכתב בעזרת כלים של GenAi.

בתרגיל זה נמッシュ shell פשוט שיקבל פקודות מהמשתמש ויבצע אותן, בדומה לתרגיל בסוף תרגול 1. המשמש יזין פקודות CLI (command line interface), והתוכנית תעבד אותן ויבצע אותן. התוכנית גם תלמוד סיגנלים מסוימים הנשלחים מהמקלדת ותטפל בהם ע"י הגדרת שגרות טיפול.

הערה: לאור כל המספר מסומן בסוגרים מושלמים מקומות בהם יש להחליף מחרוזת מסוימת בערך שלו אצלם בתוכנית. למשל, כפי שיפורט בהמשך, כל הודעות השגיאה בתוכנית יהיו מהצורה:

```
smash error: <cmd>: <error message>
```

כאשר יש להחליף בתוכנית שלכם את <cmd> (כולל הסוגרים המושלמים) בפקודה עצמה. פירוט לגבי הודעות השגיאה של הפקודות השונות יפורט בהמשך.

יש להקפיד על הדפסות לפי הפורמט הנתון בתרגיל – **הדפסות לא נכונות/מודיקות יובילו להורדת נקודות.** לתרגיל יהיה מצורף קובץ `smashout`-ים השוו את התוצאות שלכם לתוצאות שם.

כמה טיפים:

- לפני ששאליהם שאלות בפורים – עשו מאמץ לוודא שההתשובה לשאלתכם לא כתובה בהנחות התרגיל/נשאה כבר בפורים/בקבוצה של הקורס.
- בחילק הבא של המספר תמצאו תיאור כללי של התרגיל, כאשר לאחריו יופיעו ההנחות בצורה מדויקת. קראו קודם כל את התיאור הכללי על מנת להבין את המבנה של התוכנית, ולאחר מכן את ההנחות המדויקות.
- תכנו קוד!** ככה עושים בחימ האמייתים! שרטטו תרשימים זרימה כללי של התוכנית. מה הם מבני הנתונים הנדרשים? איך נממש אותן? הגדרו הצהרות של `classes/structs`, תכנו פונקציות שתרצו למש וירק אח"כ תתחילו לכתוב. ברגע שמתchingים למימוש מסוים שלא לוקח בחשבון את אחת הדרישות, קשה הרבה יותר לשנות.
- לשימושכם בנינו לתרגיל בלבד אופציוני המחולק לקבצים הבאים:
a. `smash.c`
b. `commands.c/h`
c. `signals.c/h`
d. `o system_call.h/o` בו אסור לגעת!

זהי הצורך הכספי לחלוקת הקוד, אבל אין חובה להשתמש בו.

- למדו לשימוש ב-`gdb` – דיבוג של תוכניות עם קריאות מערכות ובות יכול להיות מורכב מאוד, והדפסות למסך בדר"כ לא מספקות פתרון מספק לכך. בקורס מ"ת יש סדנא עם כל מה שצריך לדעת על `gdb`, ועם קצת חיפוש באינטרנט תוכלו להציג בדיק איזה תהליך מבצע איזה קטע קוד, מה מצב המשתנים שלו וכו'.

בהצלחה!

תיאור התוכנית, ממשק משתמש ותהליך הרצה

עליכם לנתח תוכנית אשר תשתמש בסיסי למערכת הפעלה Linux. כפי שLEARNT בתרגול 1, shell – (command line interface CLI) הוא ממשק משתמש בממשק CLI (command line interface). התוכנית תבצע פקודות שונות אשר יוקלדו ע"י המשתמש. שם ה-shell יהיה smash, או shell.small. התוכנית תפעל באופן הבא:

1. התוכנית תמתן לפקודות אשר יוקלדו ע"י המשתמש ואז תבצע אותן, וחזר חלילה עד לקבלת פקודה שתסגור את התוכנית עצמה.
2. בהמשך התרגיל תינתן רשימה של פקודות אותן יש למש בקוד באופן מלא ע"י שימוש בקריאות מערכת ולוגיקה מובנת שלהם – פקודות אלו יקראו **פקודות מובנות** (built-in commands). פקודות אלו ימצאו באותו קובץ הרצה יחד עם ה-smash עצמו. פקודות אלו יכולו לזרוץ באותו תהליך של smash או בטהילך נפרד.
3. בקלט פקודה שאינה פקודה מובנת, smash ינסה לבצע אותה בתהליך נפרד ע"י fork ואז exec. פקודות אלו יקראו **פקודות חיצונית** (external commands). פקודות חיצונית לא ניתנות לביצוע ע"י תהליך smash (מכיוון שאין שדרשות להביא קוד מקובץ הרצה אחר מה-smash).
4. ה-smash יתמוך בביצוע פקודות ברקע (background) ע"י התווים & – פרטיהם בהמשך.
5. כאשר מתבצעת פקודה שאינה ברקע, ה-smash ימתן סימן הפוקדה (בין אם היא פקודה smash ויתמוך בשרשור פקודות מורכבות ע"י התווים "&&" ו-";" – פרטיהם בהמשך).
6. כאשר התוכנית ממתינה לקלט מהמשתמש מודפסת בתחילת שורה חדשה -prompt:

```
smash >
```

עם רווח לפני ואחרי ה->. אם נסמן בקו תחתון את הרוחחים נקבל: _>_smash.

על מנת לפחות את המימוש ככל שאפשר, ניתן להניח את ההנחהות הבאות:

1. כל פקודה מופיעה בשורה נפרדת ואורכה לא עולה על 80 תווים.
2. ניתן להניח שמספר הארגומננטים המקסימלי לכל פקודה הוא 20.
3. ניתן להשתמש בכל מספר רווחים בין מילימ המופיעות באותה שורת פקודה ובתחילת השורה.

למשל, כל הפקודות האלו שקולות ותקינות:

```
smash > cd x  
smash > cd x  
smash > cd x
```

4. כל פקודה נגמרה בトー "ח".
5. הפוקודה הריקה "ח" היא פקודה חוקית וכן לבצע כלום בקבלה שלה (פרט להמשיך לטיפול בפקודה אחרתה).
6. על התוכנית לתמוך בכלל היוטר 100 תהליכי הרצים בו זמנית.

הערה: הנחה מס' 1 לא נכונה באופן כללי והיא מהווה אתגר ממשוני לבניית shell אמיתי. בדרך כלל יש לקרוא מספר מסויים של תווים מהמשתמש לתוכה חוץ גודל שרירותי, לברר כמה תווים נקראו ובמידת הצורך להזכיר עוד כדי להתאים לגודל קלט המשתמש.

עבודות (Jobs)

עבודה היא תהליך בן sha-smash מנהל ונוצר על ידו. כמו שראינו בתרגול 1, תהליכיים יכולים "להשגיח" על תהליכיים אחרים ולנהל עבודות ע"י קירiat ערכיו החזורה שלהם. לדוגמה, תהליך שמבצע עבודה מסוימת יכול להחזיר 0 בהצלחה, 1 אם נכשל עקב מתן ארגומנטים לא מתאימים ו-2 אם נכשל עקב כשל לוגי בתוכנית – תהליך האב שלו יוכל לקרוא את ערך ההחזירה זהה ולהציגו לכל סיטואציה בהתאם.

sha-smash ייחסיק רשותה של כל העבודות הקיימות בכל שלב של ריצת התוכנית. לכל עבודה יש לחת מצאה ייחודי (id job) sha-smash מייצר עם כניסה לרשותה העבודות – על המצאה לא להשתנות לכל אורך חי תהליכי המבצע את העבודה. כיוון שכל העבודות הן גם תהליכי בניים של תהליך smash, היא מזוהה גם ע"י ה-ID (ID process) של אותו הבן, כפי שניתן לו ע"י מ"ה. עם זאת, נציג כי ה-ID ניתן ע"י מערכת ההפעלה באופן בלתי תלויsha-smash בעוד id job ניתן לתהליך ע"י sha-smash לפי אלגוריתם שיפורט בהמשך, וכן הם באופן כללי שונים.

כל עבודה יכולה להיות באחד משלושה מצבים אפשריים:

1. **חזית (foreground)** – פקודה מובנת הרצה באוטו לתהליך sha-smash, או פקודה חיצונית הרצה בתהליך נפרד sha-smash ממתחם לסיומו. אם יש עבודה במצב זה sha-smash מוצג וה-sha-smash ממתחם לסיום העבודה לפני שיציג אותו שוב.
2. **רקע (background)** – כאשר משורשר לפקודה התו "&", יש להריץ את הפקודה ברקע. כאשר פקודה רצה ברקע, sha-smash לא ממתחם לסיומה ומיד מציג את ה-IDproc לקבלת פקודה נוספת. במקביל לתהליך בן חדש מבצע את הפקודה שהתקבלה עם "&". פקודות ברקע יכולות להיות מובנות או חיצונית.
3. **עצור (stopped)** – במידה ורצה עבודה בחזית והתקבל מהמשתמש סיגナル ע"י Z+CTRL, העבודה תעצור ותפסיק לרוֹץ. העבודה תמשיך לרוֹץ רק כשהישלח אליה הסיגナル SIGCONT.

sha-smash ינהל רשימת עבודות לפי הפירוט הבא:

1. לרשימה יכנסו עבודות הנמצאות ברקע או במצב עצור (מצבים 2 ו-3) בלבד.
2. המשמש יכול לניהל את רשימת העבודות ע"י הפעלת פקודות שונות שיפורטו בהמשך – פקודה מובנת jobs שתדפיס את כל העבודות, או פקודה מובנת fg שתעביר עבודה כלשהו מרשימה העבודות לרוץ בחזית.
3. עבודה שרצה בחזית וקיבלה סיגナル Z+CTRL תעבור לרשימה העבודות.
4. עבודה שמשמעותה תוסר מהרשימה לפני שיכנסו לרשימה כל עבודות חדשות.
5. לאחר הרצת כל פקודה, **לפני הדפסת רשימת העבודות ולפני הוספה של עבודה חדשה לרשימה** יש לוודא כי מצב העבודה עדכני ותקין (למשל – הסרת פקודות שהסתיעמו).
6. כל עבודה שנכנסה לרשימה מקבלת מצאה – מספר id בין גדול/שווה ל-0. המזאה יהיה המספר המינימלי הפניו שניתן למת לעובדה. למשל, אם העבודות [0, 1, 2, 3] קיימות, העבודה הבאה שתכנס תקבל את המזאה 4 – ואם העבודות [0, 1, 2, 4] קיימות, העבודה הבאה שתכנס תקבל את המזאה 3.

פקודות ברקע ובחזית

ניתן להגדיר לכל פקודה לזרץ "ברקע" ה-smash ע"י שרשור התו "&" לסוף הפקודה:

```
echo 1 &
```

כאשר פקודה רצאה ברקע, יש לפתח עבורה תהליך נפרד, להורות לו לבצע את הפקודה (בין אם היא פנימית או חיצונית), להוסיף אותו לרשימת העבודות ולהמשיך מידית לפקודה הבאה שיש לבצע, ללא המתנה. ניתן להניח כי במידה וניתן התו & להרצאה ברקע, הוא מופיע **בסוף** הפקודה בלבד עם לפחות רווח אחד לפניו (כלומר בפורמט "& echo 1&" ולא בפורמט "echo 1&& echo 1").

בניגוד לפקודה ברקע, פקודה בחזיות היא פקודה פנימית/חיצונית לה smash ממתין עד לסיומה, בין אם היא רצאה בתהליכי השם smash (ואז המתנה היא בעצם ביצוע הפקודה) או בתהליכי נפרד ואז יש לבצע מנגנון המתנה מחוכם יותר.

פקודות חיצונית

כאשר השם smash מקבל פקודה חיצונית (=אינה אחת מהפקודות המובנות המפורטות בפרק הבא), היא תנשה להפעיל את הפקודה כתוכנית בתהליכי רצון חיצוני. למשל:

```
smash > ls  
a.bin  
smash > a.out arg1 arg2
```

תגרום להפעלת התוכנית bin.a (הקיימת) עם הארגומנטים arg1 arg2.arg. יש להעזר בקריאת המערכת exec. במידה והווער & בסוף הפקודה, התוכנית תבוצע ברקע – היא תכנס לרשימת העבודות, תבוצע ברקע וה-smash יתקדם לביצוע הפקודה הבאה.

במקרה והנתיב הנתון ע"י המילה הראשונה בפקודה אינו קיים (כלומר, התוכנית אותה מניסים להריץ אינה קיימת), יש להדפיס את השגיאה הבאה:

```
smash error: external: cannot find program
```

בכל שגיאה אחרת יש להדפיס את השגיאה הבאה:

```
smash error: external: invalid command
```

הערה: הפקודה זו לא נדרשת למימוש ב-smash, היא מובאת בדוגמה על מנת להבהיר שהקובץ bin.a קיים ונitin להריץ אותו.

אייפה הפקודה שלי רצאה?

יש שני פרמטרים שקביעים את אופן הרצת הפקודה:

- אם הפקודה פנימית/חיצונית – הקוד עבור פקודה פנימית נמצא בקובץ הרצתה של השם smash (אתם תכתבו אותו), בעוד שהקוד עבור פקודה חיצונית נמצא בקובץ הרצתה אחר על המcona (ולכן יש להשתמש ב-exec על מנת להריץ אותו).
- האם הפקודה מתבצעת ברקע או לא (אם ניתן לה תו "&" בסופה או לא). פקודה שמתבצעת ברקע בהכרח מתבצעת בתהליכי נפרד מה-smash.

נקבל את הטבלה הבאה:

פרמטר	פנימית	חיצונית
לא ברקע	ריצה בתחילת ה-smash ומריצה קוד של ה-smash (אין צורך ב-fork/exec) ללא קריאות מערכת לניהול תהליכיים	ריצה בתחילת נפרד, מריצה קוד חיצוני, צריך ללחכות עד שתסתטיים: יש להשתמש ב-fork+exec+waitpid
ברקע	ריצה בתחילת נפרד, מריצה קוד של ה-smash, אין צורך לה: יש להשתמש ב-fork+exec	ריצה בתחילת נפרד, מריצה קוד חיצוני, צריך ללחכות עד שתסתטיים: יש להשתמש ב-fork+exec+waitpid

מיוחדיםsyscall

לצורך תרגול והבנה מעמיקה של השימוש בקריאות מערכות, מצורפים לתרגיל שני קבצים:
`my_system_call.o` ו-`my_system_call.h`

אין לשנות את הקבצים הללו, ואיןכם חשופים למימוש המלא הנמצא מאחורי הקלעים בקובץ ה-`libc`.

רשימת 10 קריאות המערכת בהן אסור להשתמש ישורות:

`fork()` , `execp()` , `waitpid()` , `signal()` , `kill()` , `pipe()` , `read()` , `write()` , `open()` , `close()`

משפחה exec מותר בתרגיל להשתמש רק ב-`execp`.

במקום זאת עלייכם להשתמש ב-`wrapper` שספקנו לתרגיל:

```
long my_system_call(int syscall_number, ...);
```

השימוש שספק אינו מטפל בשגיאות הנובעות מהקריאה שלכם לפונקציית המעטפה. הטיפול בשגיאות של קריאת המערכת נשאר זהה, עלייכם לבדוק את ערך ההחזרה ולפעול בהתאם במקרה של תקללה.

הגשה שתכלול קריאה ישירה לאחת מעשר קריאות המערכת לעיל, ללא שימוש ב-`wrapper` שספק, לא לעמוד בדרישות התרגיל. אם הפתרון שלכם מצריך קריאות אחרות שאין ברשימה, ניתן להשתמש בהם כרגע.

פקודות מובנות

יש לתמוך בפקודות המובנות הבאות. הקוד עבור הפקודות נמצא באוטה תוכנית של ה-smash, ע"י שימוש בקריאות מערכת ועדיין מבני נתונים פנימיים שתממשו.

:הנחיות:

1. פקודות מובנות יריצו בתהיליך שמריץ את ה-smash ללא יצורת תהיליך נפרד, אלא אם התקבל & חלק מהפקודה הפנימית ואז יש להריץ את הפקודה בתהיליך נפרד.
2. אין להשתמש בקריאת המערכת system בתרגיל.
3. במידה והוכנסה פקודה מובנת עם פרמטרים לא חוקיים כמוות פרמטרים לא נcona או אחת השגיאות שתופיע בהמשך, יש להדפיס הודעה שגיאה בפורמט הבא:

```
smash error: <cmd>: <error message>
```

כאשר שם הפקודה והודעת השגיאה יופיעו בהמשך לכל פקודה. לאחר ההדפסה יש לעברו לטיפול בשורת הפקודה הבאה. בכל מקום בו קיימת יותר משגיאה אחת, יש להדפיס הודעה שגיאה אחת בלבד, לפי הסדר שהם מופיעות במסמך זהה.

:showpid

פקודה זו תדפיס את ה-PID של תהיליך smash, גם אם היא נדרשת לרווח תהיליך נפרד. דוגמא:

```
smash > showpid
smash pid is 123456789
```

במידה והועברו ארגומנטים כלשהם עם הפקודה, יש להדפיס הודעה שגיאה:

```
smash error: showpid: expected 0 arguments
```

:pwd

פקודה זו תדפיס את הנתיב הנוכחי בו נמצא התהיליך שמריץ אותה, גם אם היא נדרשת לרווח תהיליך נפרד. דוגמא:

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

במידה והועברו ארגומנטים כלשהם עם הפקודה, יש להדפיס בהודעת השגיאה:

```
smash error: pwd: expected 0 arguments
```

:cd

1. הפוקודה תקבל קקלט ארגומנט יחיד שמתאר את הנתיב הרלטי/המלא אליו יש לעבור, ותבצע את המעבר אליו.
2. במידה והנתיב הנתון הוא התו "-", יש לחזור לנטיב הקודם בו היה ה-h-smash ולהדפיס למסך את הנתיב המלא (לא רלטיבי) אליו התרבצע המעבר. יש לזכור נתיב אחד אחרת בלבד – ככלומר, אם נבצע "-cd" פעמים נחזיר לאוთה תיקייה (מכיוון שלآخر "-cd" הראשון, התיקייה הנוכחית הופכת לקודמת, ולאחר מכן "-cd" השני נחזיר לתיקייה בה התחלנו).
3. במידה והארגון הוא התווים "...". יש לחזור לתיקיית האם של התיקייה הנוכחית. אם עברו ארגומנט זה התיקייה בה אנו נמצאים היא "/" אין לבצע כלום.
4. במידה וניתנה הנחיה לפוקודה לזרץ בركע ("&"), יש לוודא כי התהיליך שמבצע שינוי pwd הוא התהיליך החדש ולא התהיליך של ה-h-smash.

עבור שגיאות:

1. במידה וسوفק יותר מארגומנט אחד או לא סופק ארגומנט יש להדפיס שגיאה כפי שמתואר בדוגמה.
2. במידה ונתון הארגומנט "-" ולא קיים נתיב קודם, יש להדפיס שגיאה כפי שמתואר בדוגמה.
3. במידה ונתון הארגומנט "...". ותיקית האם של התיקייה לא קיימת, אין לבצע כלום.
4. במידה וسوفק נתיב חוקי אבל לקובץ שאינו תקין, יש להדפיס שגיאה עם שם הקובץ כפי שמתואר בדוגמה.
5. במידה והנתיב שסופק אינו נתיב קיים, יש להדפיס שגיאה כפי שמתואר בדוגמה.

נניח עבור דוגמא זו כי הנתיב /home/os/exam.txt הוא תקין וחוקית וכי /home/os/exam/אין תקין:

```
smash > cd -  
smash error: cd: old pwd not set  
smash > cd a b  
smash error: cd: expected 1 arguments  
smash > cd /home/os  
smash > pwd  
/home/os  
smash > cd ..  
smash > pwd  
/home  
smash > cd -  
/home/os  
smash > pwd  
/home/os  
smash > cd -  
/home  
smash > pwd  
/home  
smash > cd xyz  
smash error: cd: target directory does not exist  
smash > cd /home/os/exam.txt  
smash error: cd: /home/os/exam.txt: not a directory
```

:jobs

הפקודה תדפיס את רשימת העבודות (אלו שרצות ברקע ואלו שבמצב stopped). פורמט הדפסה יהיה:
[<job id>: <process id> <seconds elapsed> <stopped>

כasher:

- .1. <id> הוא המזהה הייחודי של העבודה כפי שנitin לה בכניסתה לרשימת העבודות.
- .2. <command> הוא המחרוזת אותה הכנסיס המשמש ביצוע הפקודה.
- .3. <process id> הוא ה-PID של התהיליך המריץ אותה (כפי שנitin לה ע"י מערכת הפעלה).
- .4. <seconds elapsed> הוא הזמן ש עבר בשניות מאז שהוכנסה העבודה לרשימה של העבודות. יש להשתמש בקריאה המערכת (`time -l`).
- .5. עברו עבודות שהן stopped במקומות יודפס "stopped", עברו עבודות שאין stopped במקומות לא יודפס כלום.

שים לב כי במידה ועובדת חוזרת לרשימה אחריו שנכנסת יש לעדכן את זמן ההוכנסה שלה. יש להדפיס את הרשימה בצורה ממוינת בסדר עולה לפי מזהה העבודה.

אם העברו ארגומנטים לפקודה, יש להדפיס הודעה שגיאה:

```
smash error: jobs: expected 0 arguments
```

דוגמא:

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

הסביר לגבי הסיג널ים ניתן בהמשך.

kill <signum> <job id>

הפקודה שולחת את הסיגナル שמספרו signum אל העבודה עם המזהה id job עם הרשימה של העבודות ומדפיסה:

```
signal <signum> was sent to pid <PID>
```

כאשר PID הוא המזהה של העבודה כפי שנitin לה ע"י מערכת הפעלה -signum והוא מספר הסיגナル כפי שנitin על ידי המשתמש בפקודה. דוגמא:

```
smash > kill 9 1
signal 9 was sent to pid 123456789
```

אם העבר מזהה של עבודה שאינה קיימת יש להדפיס:

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

אם מספר הארגומנטים או הפורמט שלהם אינם תקין יש להדפיס הודעה שגיאה:

```
smash error: kill: invalid arguments
```

אין צורך למסח פונקציונליות נוספת שאינה כתובה כאן (למשל סימן עבודה ברשימה העבודות שנשלחו להם דרך kill בתור SIGSTOP.).

:fg <job id>

הפקודה תגרום להריצה בחזית של העבודה המזוהה עם המזהה id job. הפקודה מדיפסה למסח את הפקודה עצמה של אותה עבודה ואת מזהה התהיליך, ואז מעבירה את העבודה לחזית. אם העבודה עצורה ברקע, יש לשלוח לה SIGCONT על מנת שתתמשיך לרוץ. אם היא אינה עצורה, אין צורך לשלוח לה סיגナル. לאחר שליחת הסיגナル-hash smash ידפיס את הפקודה (כפי שניתנה בקונסיסטה לרשימה העבודות) וימתין לסיום העבודה (מיון שהוא רצה בחזית). לאחר העברת העבודה לחזית יש להסירה מרשימה העבודות. הפעלת fg ללא ארגומנט `>id job<` שකולה להפעלת fg עבור העבודה עם ->`id job` המקסימלי ברשימה העבודות (אם קיים).

ניתן להניחס שפקודת fg לא תתקבל להריצה ברקע (כלומר "& fg" אינה פקודה חוקית).

אם הועבר מזהה של עבודה שאינה קיימת יש להדפיס הודעה שגיאה:

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

אם לא הועבר מזהה עבודה ורשימת העבודות ריקה יש להדפיס הודעה שגיאה:

```
smash error: fg: jobs list is empty
```

אם מספר הארגומנטים או הפורמט שלהם אינם תקין יש להדפיס הודעה שגיאה:

```
smash error: fg: invalid arguments
```

לדוגמא:

```
smash > /bin/sleep 30 &
smash > jobs
[1] /bin/sleep 30 &: 28 secs
smash > fg
[1] /bin/sleep 30 &
```

:bg <job id>

הפקודה תחזיר ליריצה ברקע של עבודה שבמצב stopped עם המזהה id job. הפקודה מדיפסה למסח את הפקודה עצמה של אותה עבודה ואת מזהה התהיליך, ואז שולחת לתהיליך שלו סיגナル SIGCONT. על העבודה להמשיך לרוץ ברקע. הפעלת bg ללא ארגומנט `>id job<` שකולה להפעלת bg עבור העבודה עם ->`id job` המקסימלי ברשימה העבודות (אם קיים). לאחר שימוש בפקודה וביצוע פקודת jobs, הסימן stopped לא יהיה מוצג ברשימה העבודות.

ניתן להניחס שפקודת bg לא תתקבל להריצה ברקע (כלומר "& bg" אינה פקודה חוקית).

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
```

אם הועבר מזהה עבודה שלא קיים, יש להדפיס הודעה שגיאה:

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

אם הועבר מזהה עבודה שאינו עוזר, יש להדפיס הודעה שגיאה:

```
smash error: bg: job id <job id> is already in background
```

אם לא הועבר מזהה עבודה ואין עבודה במצב עצור ברשימה, יש להדפיס הודעה שגיאה:

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

אם פורמט הארגומנטים או מספרם אינם תקין יש להדפיס הודעה שגיאה:

```
smash error: bg: invalid arguments
```

quit [kill]

הפקודה מסיימת את תהליך smash עם ערך החזרה 0. אם הועבר ה.Parameter kill (אופציונלי) יש להרוג את כל העבודות הקיימות ברשימה העבודות לפניה סיום התוכנית. במידה וניתן הארגומנט kill, הפקודה תהרוג את העבודות הקיימות לפי האלגוריתם הבא, באופן טורי לאורך רשימת העבודות מהמזהה הנמוך לגבוה:

1. הדפסת מזהה העבודה ופקודת העבודה הקיימת לפי הפורמט הבא:

```
"[<job id>] <command> - "
```

שימוש לב לרווח לפני ואחרי המקף.

2. שליחת סיגナル SIGTERM והדפסת הודעה "sending SIGTERM..." (שימוש לב לרווח אחרי 3 נקודות)

3. אם התהליך נהרג לאחר פחות מ-5 שניות, יודפס "done"

4. אם התהליך לא נהרג לאחר 5 שניות מקבל סיגナル SIGTERM, שליחת סיגナル SIGKILL והדפסה: "sending SIGKILL... done"

כל הדפסות עברו עבודה ספציפית יבוצעו ללא ירידת שורה, ולאחר כל עבודה תבוצע ירידת שורה. ניתן להניח כי הפקודה לא תתקבל להרצה ברקע (כלומר "& quit" אינה פוקודה חוקית). דוגמא:

```
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... sending SIGKILL... done
```

בדוגמה התהיליך שמרץ.out.b לא הגיב לSIGNEL SIGTERM ולכן נשלח לו גם SIGKILL.

במידה והועבר יותר מארגומנט אחד יש להדפיס הודעה שגיאה:

```
smash error: quit: expected 0 or 1 arguments
```

במידה והועברו ארגומנטים שאינם kill יש להדפיס הודעה שגיאה:

```
smash error: quit: unexpected arguments
```

:diff <f1> <f2>

התוכנית תשווה את תוכן הנתיבים f1 ו-f2. הפונקציה תדפיס למסך 1 אם הקבצים שונים ו-0 אם הם זהים.

אם אחד/שני הנתיבים לא קיימים יש להדפיס הודעה שגיאה:

```
smash error: diff: expected valid paths for files
```

אם שני הנתיבים קיימים אבל לפחות אחד מהם הוא תיקיה ולאקובץ יש להדפיס הודעה שגיאה:

```
smash error: diff: paths are not files
```

אם מספר הארגומנטים אינם 2 יש להדפיס הודעה שגיאה:

```
smash error: diff: expected 2 arguments
```

טיפול בסיגנלים

על ה-smash לתרmor בצירופי המקלשים C-CTRL+Z ו-C-CTRL+C:

1. הצירוף C-CTRL+C מפסיק את התהיליך שרצ בחזית ומציג מחדש את ה-prompt. במידה וההתהיליך שרצ בחזית הוא smash, הצירוף עוצר את ביצוע הפוקדה הנוכחית מיידית ומציג את ה-prompt. אחרת, התהיליך שלוח SIGKILL להתהיליך שרצ בחזית.
2. הצירוף Z-CTRL+C משאה את התהיליך שרצ בחזית (שולח לו SIGSTOP) ומוסיף אותו לרשימת העבודות (עם סימון שהטהיליך מושהה). אם התהיליך שרצ בחזית הוא smash, הצירוף לא עושה כלום.

שימוש שבוחינת bash התהיליך שאליו ישלחו הסיגנלים הללו הוא התהיליך שMRIIZ את smash. לכן, על smash למכוד את הסיגנלים הנוצרים על ידי לחיצות המקלדת +C/CTRL+Z, ולשלוח בהתאם סיגナル לתהיליך שרצ בחזית של ה-shell. אם אין תהיליך שרצ בחזית, צירופים אלו לא ישפיעו על ה-shell. כמובן שנייתן להשווות/להרוג תהיליך פנימית ע"י smash ע"י שליחת סיגナル מתאים עם kill.

לעומת פקודות פורכבות, נגידר כי פקודה שקובלה סיגナル היא פקודה שנכשלה ואין צורך לבצע פקודה שישורשה אליה ע"ז &. למשל, אם ניתנה הפקודה הפורכבת cmd2 && cmd1 והפקודה cmd1 בוצעה בחזית וקיבלה סיגナル, אין צורך לבצע את cmd2. במקרה בו הפקודה הפורכבת היא cmd2; cmd1 כיש לבצע את cmd2 cmd1 אם cmd1 קיבל סיגナル.

לסיכום:

1. כאשר מתתקבל הצירוף C-CTRL+C יש לבצע את הפעולות הבאות:

1. להדפיס למסך:

```
smash: caught CTRL+C
```

2. אם קיים תהיליך שרצ בחזית, יש לשלחו לו SIGKILL ולהדפיס:

```
smash: process <PID> was killed
```

2. כאשר מתתקבל הצירוף Z-CTRL+C יש לבצע את הפעולות הבאות:

1. להדפיס למסך:

```
smash: caught CTRL+Z
```

2. אם קיים תהיליך שרצ בחזית, יש לשלחו לו SIGSTOP ולהדפיס:

```
smash: process <PID> was stopped
```

פקודות מורכבות

תזכורת – ערך החזרה של תחיליך הוא:

1. במידה והתחליך סיים ע"י קריית המערכת exit – ערך החזרה שלו יהיה הערך הנוכחי ל-exit.
 2. במידה והתחליך חוזר מפונקציית ה-main שלו ע"י X return, ערך החזרה שלו יהיה X.
 3. במידה והתחליך סיים את פונקציית ה-main שלו אבל לא שם בה ערך החזרה, התנהגות אינה מוגדרת (ב-gcc וב-clang בדרך כלל ערך החזרה נקבע ל-0 במקרה זה)
- ניתן לקרוא ולפרש ערכו החזרה של תהליכיים ע"י קריאות המערכת pid וה-macros המתאימים להם, כפי שראינו בתרגול 1.

בצורה דומה ל-bash, ה-smash יתמוך ביצירת פקודות מורכבות ע"י האופרטורים "&&" ו"||":

1. התווים "&&" בין שתי פקודות גורם להן להתבצע אחת אחרי השניה, אבל רק אם הפקודה הראשונה הסתימה בהצלחה.
2. פקודה פנימית שנכשלת היא פקודה שהסתימה באחת השגיאות שציינו לעלה, בעוד שפקודה חייזנית שנכשלת היא כל פקודה שערכו החזרה שלה אינו 0.

נדגיש כי התווים "&&" מקשרים בין שתי פקודות סמוכות בלבד. ניתן מספר דוגמאות לתנהגות הרציה – בדוגמאות מופיעה הפקודה true שמצויה תמיד והפקודה false שנכשלת תמיד.

פקודה	פלט	הסביר
הפקודה הראשונה הצלילה ולכן הפקודה השנייה גם מתבצעת	1	echo 1 && echo 2
	2	
הפקודה השנייה נכשלה ולכן לא מודפס 2	1	echo 1 && false && echo 2

שיםו לב שבכל פקודה של ספק בנסיבות המימוש, ניתן לבדוק את הפלט הרצוי בכל בטרמינל bash – ההתנהגות להיות עקבית.

אין צורך לטפל בפקודות מורכבות שימוש בחלקן פקודות ברקע (למשל echo 1 && echo 2 (&) . נושא זה לא יבדק!

הערה – האופרטור "||" יכול להופיע עם כל מספר של רוחים ביחס לפקודות הקודמות. למשל, כל הפקודות הבאות חוקיות:

cd y && cd z

עם זאת, לא ניתן לשרש בין אופרטורים, בין אם מאותו סוג או שונים, כמו בדוגמה הבאה:

cd z && && cd w

Alias & Unalias

בגלל שמתכוונים הם עם עצמם, עליהם למשתמש תמייה בפקודת alias ו-unalias בתוך ה-h-smash שלהם. מטרת הפקודות היא לאפשר למשתמש להגדיר **כינויים לפקודות קיימות**, או ליצור **פקודות חדשות** בעלות התנהגות שונה מהרגיל. פורמט הפקודות שעליהם למשם הוא:

```
alias <new_cmd_name>=<the command>"
```

```
unalias <new_cmd_name>
```

דוגמאות נפוצות מהעולם האמיתי:

```
alias ll='ls -l'
```

```
alias gs='git status'
```

```
alias mkcd= alias mkcd='mkdir -p "$1" && cd "$1"'
```

הדוגמה الأخيرة מייצרת תיקה ונכנסת פנימה, דבר שהופך לתוכנת הממוצע 22 שניות שלמות 😊.

אין צורך למש פקודות alias אשר מקבלות פרמטר **בזמן הפעלה** (כמו mkdc). יש צורך למש שמקבלות פרמטר רק **בזמן הנדרתן**. דוגמה

```
alias cd2out="cd .. && cd .."
```

הפקודה cd2out תחזיר פעמים אחרה בעץ התיקיות.

פקודת unalias תסיר את ההגדרה של הכינוי מזכירן smash. לאחר ההסרה, לא ניתן יהיה עוד להשתמש בשם הכינוי, ויש להדפיס הודעה שגיאה מתאימה במקרה של ניסיון שימוש.

טייפ: זו אופציית smash ל-h-smash但她 רק לאחר שסיימתם את מרבית התרגול.

יצירת תהליכיים

ברוב המקרים גם התהיליך המrizץ את ה-smash וגם כל תהליך הבן שלו מקבלים את הסיג널ים C-CTRL+Z+-CTRL למרות שהוא-smash שלכם לא שולח אותו לתהיליך הבן. בעיה זו מתרחשת בגלל שה-shell האמתי בו ה-smash רץ שלוח את הסיג널ים לכל התהליכים בעלי אותו ה-id, group, מנגןנו אליו אנחנו לא נוכנים בקורס. בשביל להמנע מבעיה זו צריך לשנות את ה-pid של תהליך בן אותו ה-shell מייצר באופן הבא:

```
int pid = fork();
if(pid == 0) {
    setpgrp(); //run on child only! changes group ID
    execv(...); //child leaves to execute code
}
else if(pid > 0)
    //parent code
else
    /fork error code
```

זו כמובן דוגמת קוד לא שלמה – העיקר הוא לוודא כי **הבנייה** שה-smash מייצר מריםם ()setpgrp לפני שהם מריםם כל קוד אחר.

טיפול בשגיאות

(perror היא פונקציה סטנדרטית בLINNOKS המקבלת הודעה שגיאה שהמשתמש מייצר ומוסיפה אותה להודעת שהוא מיצרת ע"י בדינה של errno. ההודעה מודפסת ל-stderr ולא stdout. יש להשתמש בה בשגיאות בקריאה מערכת בלבד.

אם קריאת מערכת נכשلت (ולא עקב סיוטואציה שתוארה לעיל) יש להדפיס הודעה שגיאה באמצעות הפונקציה () perror לפי הפורמט הבא:

```
smash error: <syscall name> failed
```

בתרגיל יש לבצע ()exit רק במקרה של שגיאה קיונית שאינה מאפשרת המשך ריצה תקינה בrama הלוגית של התוכנית (למשל malloc או fork שיכשלו). במקרים אחרים, למשל chdir שהחזר ערך שאינו הצלחה, יש לזרות את השגיאה ע"י ערך החזרה ו/או errno ולטפל בה בהתאם (לדוגמה – לזרות chdir שנכשל מכיוון שהנתיב שנייתן אינו חוקי).

טיפים כלליים

1. הפקודה `>sleep <seconds` שימושית לטיולץ פקודה חיצונית שלוקחת זמן.
2. הפקודה `"read -t <seconds>"` או לולאה כמו `"for ((i=0; i < 500000; i++) ; do done;"` שימושת לטימלץ פקודה פנימית שלוקחת זמן.
3. הפקודות `"true"/bin/false` ו- `"false"/bin/true` תמיד מצילות ונכשלות בהתאם, ושימושות לדיבוג פקודות מורכבות. למשל:

```
smash > /bin/true && echo 1
1
smash > /bin/false && echo 1
smash >
```

הנחיות נוספות

1. יש להקפיד על הדפסות התואמות ב-100% את הדרישות בתרגיל – שגיאה בהדפסות עלולה להוביל להורדת נקודות (הטסחים הם אוטומטיים, בנמה אישית הימנעו מאיבוד נקודות על prints התרגיל מספיק קשה ואורך אל תאבדו נקודות על הדברים הכילים).
2. פרט לשימוש ב-`perror`, יש להדפיס ל-`stdout` בלבד.
3. יש להשתמש ב-`(fork -) exec` (בעזרת `execve`) לימוש פקודות חיצונית (יש לבחור את הוריאנט המתאים של exec לדרישות התרגיל).
4. אין להשתמש בפונקציית הספרייה `system` בתרגיל.
5. על התוכנית לבדוק הצלחה של כל פקודה ולהדפיס הודעה שגיאה מתאימה במידה של כשלון.
6. יש לכתוב ב-C או C++ בלבד.
7. הפוום עומד לשירותכם בשאלות בכל נושא הקשור לתרגיל.

קומPILEציה ולינקאג'

יש לוודא שהקוד מתكمפל ע"י הפקודות הבאות, בהתאם ל-C++ ו-L-C:

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

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

הדגל Werror גורם ל-warnings וריגלות בkompileר g++/gcc להפוך ל-errors והדגל -g שותף בקובץ ההרצה שלכם סימboleים לדיבוג -gdb יוכל לתפות. יש לוודא שהקוד שלכם מתكمפל **על המconaה של הקורס** ללא .warnings/errors

קוד שלא יתкомפל/יתקמפל עם זהירות לא יכול צוין.

עליכם לספק Makefile – הכללים שחייבים להופיע בו הם:

1. כל smash שבנה את התוכנית smash
2. כל עבור כל קובץ נפרד שקיים בפרויקט
3. כל clean המוחק את כל תוכרי הקומPILEציה

וודאו שהתוכנית נבנתה ע"י הפקודה make. יש لكمפל וללנקאג' את התוכנית עם הדגמים המופיעים לעילו.

מומלץ להפריד את המימוש לקבצי c/cpp.o.h. על מנת להקל על בנית התוכנית – יש להתיחס לכך בקובץ Makefile.

בנוסף לקבצים הנ"ל יש לספק קובץ readme לפי הפורמט בנהול תרגלי הבית.

לתרגיל מצורף סקריפט הבודק בצורה חלקית את תקינות ההגשה (מבחינה טכנית, לא מבחינת התרגיל עצמו). הסקריפט מצביע לשני פרמטרים – נתיב לקובץ zip ושם קובץ ההרצה. דוגמא:

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

בצלחה!