תרגיל רטוב #1

smash – small shell

הקדמה

שימו לב – מקוריות הקוד תבדק. העתקת שיעורי בית היא עבירת משמעת בטכניון על כל המשתמע מכך.

בתרגיל זה נממש shell פשוט שיקבל פקודות מהמשתמש ויבצע אותן, בדומה לתרגיל בסוף תרגול 1. המשתמש יזין פקודות CLI (command line interface), והתוכנית תעבד אותם ותבצע אותם. התוכנית גם תלכוד סיגנלים מסוימים הנשלחים מהמקלדת ותטפל בהם ע"י הגדרת שגרות טיפול.

הערה: לאורך כל המסמך מסומן בסוגריים משולשים מקומות בהם יש להחליף מחרוזת מסוימת בערך שלה אצלכם בתוכנית. למשל, כפי שיפורט בהמשך, כל הודעות השגיאה בתוכנית יהיו מהצורה:

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

כאשר יש להחליף בתוכנית שלכם את <cmd> (כולל הסוגריים המשולשים) בפקודה עצמה. פירוט לגבי הודעות השגיאה של הפקודות השונות יפורטו בהמשך.

יש להקפיד על הדפסות לפי הפורמט הנתון בתרגיל – הדפסות לא נכונות/מדויקות עלולות להוביל להורדת נקודות. במידה ויש ספק – שאלו בפורום, לא נוכל לטפל בבעיות כאלו בדיעבד.

כמה טיפים:

1. לפני ששואלים שאלות בפורום – עשו מאמץ לוודא שהתשובה לשאלה שלכם לא כתובה בהנחיות התרגיל/נשאלה כבר בפורום/בקבוצה של הקורס.
2. בחלק הבא של המסמך תמצאו תיאור כללי של התרגיל, כאשר לאחריו יופיעו ההנחיות בצורה מדויקת. קראו קודם כל את התיאור הכללי על מנת להבין את המבנה של התוכנית, ואחר כך את ההנחיות המדויקות.
3. **תכננו קודם!** שרטטו תרשים זרימה כללי של התוכנית. מה הם מבני הנתונים הנדרשים? איך נממש אותם? הגדירו הצהרות של classes/structs, תכננו פונקציות שתרצו לממש ורק אח"כ תתחילו לכתוב. ברגע שמתחייבים למימוש מסוים שלא לוקח בחשבון את אחת הדרישות, קשה הרבה יותר לשנות.
4. בנינו לתרגיל שלד אופציונלי המחולק לקבצים הבאים:
   a. smash.c
   b. commands.c/.h
   c. signals.c/.h

   זוהי הצורה הכדאית לחלוקת הקוד, אבל אין חובה להשתמש בו.
5. למדו להשתמש ב-gdb – דיבוג של תוכניות עם קריאות מערכת רבות יכול להיות מורכב מאוד, והדפסות למסך בדר"כ לא מספקות פתרון מספק לכך. בקורס ממ"ת יש סדנא עם כל מה שצריך לדעת על gdb, ועם קצת חיפוש באינטרנט תוכלו להציג בדיוק איזה תהליך מבצע איזה קטע קוד, מה מצב המשתנים שלו וכו'.

בהצלחה!

תיאור התוכנית, ממשק משתמש ותהליך הרצה

עליכם לכתוב תכונית אשר תשמש כ-shell בסיסי למערכת ההפעלה Linux. התוכנית תבצע פקודות שונות אשר יוקלדו ע"י המשתמש. שם ה-shell יהיה – smash, או small shell. התוכנית תפעל באופן הבא:

1. התוכנית תמתין לפקודות אשר יוקלדו ע"י משתמש ואז מבצעת אותן, וחוזר חלילה עד לקבלת פקודה שתסגור אותה.
2. התוכנית תוכל לבצע רשימת פקודות שתנתן בהמשך בצורה מלאה, ואת הטיפול בהן יש לממש בקוד בצורה מלאה – פקודות אלו יקראו **פקודות מובנות** (built-in commands). הקוד עבור פקודות אלו נמצא באותו executable כמו ה-smash עצמו. פקודות כאלו יוכלו לרוץ באותו תהליך כמו ה-smash או בתהליך נפרד – יפורט בהמשך.
3. בקבלת פקודה שאינה ברשימת הפקודות המובנות, ה-smash ינסה לבצע אותה בתור תוכנית חיצונית ע"י fork ואז exec. פקודות אלו יקראו **פקודות חיצוניות** (external commands). פקודות חיצונית ידרשו בהכרח יצירת תהליך נפרד ע"י fork ולאחר מכן exec (אין אפשרות להריץ אותן בתהליך של ה-smash מכיוון שנדרש להביא קוד מקובץ הרצה אחר).
4. אם התו "%" מופיע בסוף הפקודה (עם/בלי רווחים), יש להריץ את הפקודה ברקע – פרטים בהמשך.
5. כאשר התוכנית ממתינה לקלט מהמשתמש מודפסת בתחילת שורה חדשה ה-prompt:

```
smash >
```

עם רווח לפני ואחרי ה-<. אם נסמן בקן תחתון את הרווחים נקבל: smash_>_.

הערה: ב-bash סטנדרטי התו & משומש להרצה ברקע, לצורך התרגיל החלפנו אותו ב-% על מנת להמנע מבלבול עם שרשור פקודות עם && (מפורט בהמשך).

ניתן להניח כי:

1. כל פקודה מופיעה בשורה נפרדת ואורכה לא יעלה על 80 תווים.
2. ניתן להניח שמספר הארגומנטים המקסימלי לכל פקודה הוא 20.
3. ניתן להשתמש בכל מספר רווחים בין מילים המופיעות באותה שורת פקודה ובתחילת השורה. למשל, כל הפקודות האלו שקולות ותקינות:

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

4. כל פקודה נגמרת בתו "\n".
5. הפקודה הריקה "\n" היא פקודה חוקית ואין לבצע כלום בקבלה שלה (פרט להמשיך לטיפול בפקודה שאחריה).
6. על התוכנית לתמוך בלכל היותר 100 תהליכים הרצים בו זמנית.

הערה: הנחה מס' 1 לא נכונה באופן כללי והיא מהווה אתגר משמעותי בבניית shell אמיתי. בדרך כלל יש לקרוא מספר מסוים של תווים מהמשתמש לתוך חוצץ בגודל שרירותי, לברר כמה תווים נקראו ובמידת הצורך להקצות עוד כדי להתאים לגודל קלט המשתמש.

עבודות (Jobs)

עבודה היא תהליך בן שה-smash מנהל ונוצר על ידו. לכל עבודה יש לתת מזהה יחודי (job id) שה-smash נותן לעבודה כשהיא נכנסת לרשימת העבודות. לאחר מכן, המזהה לא משתנה לכל אורך חיי התהליך המבצע אותה. כיוון שכל העבודות הן גם תהליכי בנים של תהליך ה-smash, היא מזוהה גם ע"י ה-PID (process ID) של אותו הבן. עם זאת, נדגיש כי ה-PID ניתן ע"י מערכת ההפעלה באופן בלתי תלוי ב-smash בעוד ה- job id ינתן לתהליך ע"י ה-smash לפי אלגוריתם שיפורט בהמשך, ולכן הם באופן כללי שונים.

כל עבודה יכולה להיות באחד משלושה מצבים אפשריים:

1. **חזית (foreground)** – פקודה רגילה הרצה באותו תהליך של ה- smash או בתהליך נפרד שתהליך ה-smash ממתין לסיומו. אם יש עבודה במצב זה ה-prompt אינו מוצג וה-smash ממתין לסיום העבודה.
2. **רקע (background)** – כאשר משורשר לפקודה התו "%", יש להריץ את הפקודה ברקע. כאשר פקודה רצה ברקע, ה-smash לא ממתין לסיומה ומיד מציג את ה-prompt לקבלת פקודה נוספת. במקביל תהליך בן חדש מבצע את הפקודה שהתקבלה עם "%".
3. **עצור (stopped)** – במידה ורצה עבודה בחזית והתקבל סיגנל מהמשתמש ע"י CTRL+Z, העבודה תעצר ותפסיק לרוץ. העבודה תמשיך לרוץ רק שישלח אליה הסיגנל SIGCONT.

ה-smash ינהל רשימת עבודות לפי הפירוט הבא:

1. לרשימה יכנסו עבודות הנמצאות ברקע או במצב עצור (מצבים 2 ו-3) בלבד.
2. המשתמש יכול לנהל את רשימת העבודות ע"י הפעלת פקודות שונות שיפורטו בהמשך – פקודה מובנת jobs שתדפיס את כל העבודות, או פקודה מובנת fg שתעביר עבודה כלשהי מרשימת העבודות לרוץ בחזית.
3. עבודה שרצה בחזית וקיבלה סיגנל CTRL+Z תעבור לרשימת העבודות.
4. עבודה שמסתיימת תוסר מהרשימה לפני שיכנסו לרשימה כל עבודות חדשות.
5. לפני הרצת כל פקודה, לפני הדפסת רשימת העבודות ולפני הוספה של עבודה חדשה לרשימה יש לוודא כי מצב העבודות עדכני ותקין (למשל – הסרת פקודות שהסתיימו).
6. כל עבודה שנכנסת לרשימה מקבלת מזהה – מספר חיובי גדול/שווה ל-1. המזהה יהיה המספר המינימלי הפנוי שניתן לתת לעבודה. למשל, אם העבודות 1, 2 ו-3 קיימות, העבודה הבאה שתכנס תקבל את המזהה 4 – ואם העבודות 1, 2 ו-4 קיימות, העבודה הבאה שתכנס תקבל את המזהה 3.

פקודות מובנות

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

הנחיות:

1. פקודות מובנות ירוצו בתהליך שמריץ את ה-smash ללא יצירת תהליך נפרד, אלא אם התקבל % כחלק מהפקודה הפנימית ואז יש להריץ את הפקודה בתהליך נפרד.
2. אין להשתמש בקריאת המערכת system עבור פקודות מובנות.
3. במידה והוכנסה פקודה מובנת עם פרמטרים לא חוקיים או כמות פרמטרים לא נכונה, יש להדפיס הודעת שגיאה בפורמט הבא:

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

כאשר שם הפקודה והודעת השגיאה יפורטו בהמשך לכל פקודה. לאחר ההדפסה יש לעבור לטיפול בשורת הפקודה הבאה. בכל מקום בו קיימת יותר משגיאה אחת, יש להדפיס הודעת שגיאה אחת בלבד, לפי הסדר שהם מופיעות במסמך הזה.

<u>showpid:</u>

פקודה זו תדפיס את ה-PID של ה-smash, גם אם היא נדרשה לרוץ ברקע בתהליך נפרד. ניתן להשתמש בקריאת המערכת ()getpid. ההדפסה תהיה בפורמט הבא:

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

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

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

<u>pwd:</u>

פקודה זו תדפיס את הנתיב הנוכחי בו נמצא ה-smash. ניתן להשתמש בקריאת המערכת ()getcwd.
דוגמא:

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

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

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

:cd

הפקודה תקבל כקלט ארגומנט יחיד שמתאר את הנתיב הרלטיבי/המלא אליו יש לעבור, ותעבור אליו. במידה והנתיב הניתן הוא התו "-", יש לחזור לנתיב הקודם בו היה ה-smash. במידה והנתיב הוא התווים "..", יש לחזור לתיקיית האם של התיקייה הנוכחית, אם קיימת. אין צורך לזכור יותר מנתיב אחד אחורה. ניתן להשתמש בקריאת המערכת chdir(). במידה וניתנה הנחיה לפקודה לרוץ ברקע (ע"י %), יש לוודא כי התהליך שמבצע שינוי pwd הוא התהליך החדש ולא התהליך של ה-smash.

עבור שגיאות:

1. במידה וסופק יותר מארגומנט אחד יש להדפיס שגיאה כפי שמתואר בדוגמא.
2. במידה ונתון הארגומנט "-" ולא קיים נתיב קודם, יש להדפיס שגיאה כפי שמתואר בדוגמא.
3. במידה ונתון הארגומנט ".." ותיקיית האם של התיקייה לא קיימת, אין לבצע כלום.
4. במידה והנתיב שסופק אינו נתיב קווים, יש להדפיס שגיאה כפי שמתואר בדוגמא.

נניח עבור דוגמא זו כי הנתיב home/os/ קיים:

```
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 -
smash > pwd
/home/os
cd -
smash > pwd
/home
smash > cd xyz
smash error: cd: target directory does not exist
```

jobs:

הפקודה תדפיס את רשימת העבודות (אלו שרצות ברקע ואלו שבמצב stopped). פורמט ההדפסה יהיה:

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

כאשר:

1. <job id> הוא המזהה היחודי של העבודה כפי שניתן לה בכניסתה לרשימת העבודות.
2. <command> הוא המחרוזת אותה הכניס המשתמש בביצוע הפקודה
3. <process id> הוא ה-PID של התהליך המריץ אותה (כפי שניתן לה ע"י מערכת ההפעלה)
4. <seconds elapsed> הוא הזמן שעבר בשניות מאז שהוכנסה העבודה לרשימת העבודות. יש להשתמש בקריאות המערכת time() ו-difftime().
5. עבור עבודות שהן stopped במקום <stopped> יודפס "(stopped)", עבור עבודות שאינן stopped במקום <stopped> לא יודפס כלום.

שימו לב כי במידה ועבודה חוזרת לרשימה אחרי שנכנסת יש לעדכן את זמן ההכנסה שלה. ניתן להשתמש בקריאת המערכת time() ו-difftime(). יש להדפיס את הרשימה בצורה ממוינת בסדר עולה לפי מזהה העבודה.

אם הועברו ארגומנטים לפקודה, יש להדפיס הודעת 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)
```

הסבר לגבי הסיגנלים ינתן בהמשך.

kill -<signum> <job id>

הפקודה שולחת את הסיגנל שמספרו signum אל העבודה עם המזהה job id מרשימת העבודות ומדפיסה את ההודעה הבאה:

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

כאשר PID הוא המזהה של העבודה כפי שניתן לה ע"י מערכת ההפעלה ו-signum הוא מספר הסיגנל כפי שניתן על ידי המשתמש בפקודה.

לדוגמא:

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

אם הועבר מזהה של עבודה שאינה קיימת יש להדפיס:

```
job id <job id> does not exist
```

אם מספר הארגומנטים או הפורמט שלהם אינו תקין יש להדפיס הודעת שגיאה:

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

fg <job id>:

הפקודה תגרום להרצה בחזית של העבודה המזוהה עם המזהה job id. הפקודה מדפיסה למסך את הפקודה עצמה של אותה עבודה ואת מזהה התהליך, ואז שולחת לתהליך שלו סיגנל SIGCONT על מנת שימשיך לרוץ. לאחר קבלת הסיגנל ה-smash ימתין לסיום העבודה (מכיוון שהיא רצה בחזית). לאחר העברת העבודה לחזית יש להסירה מרשימת העבודות. הפעלת fg ללא ארגומנט <job id> שקולה להפעלת fg עבור העבודה עם ה-<job id> המקסימלי ברשימת העבודות (אם קיים).

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

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

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

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

אם מספר הארגומנטים או הפורמט שלהם אינו תקין יש להדפיס הודעת שגיאה:

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

bg <job id>:

הפקודה תחזיר לריצה ברקע של עבודה שבמצב stopped עם המזהה job id. הפקודה מדפיסה למסך את הפקודה עצמה של אותה עבודה ואת מזהה התהליך, ואז שולחת לתהליך שלו סיגנל SIGCONT. על העבודה להמשיך לרוץ ברקע. הפעלת bg ללא ארגומנט <job id> שקולה להפעלת bg עבור העבודה עם ה-<job id> המקסימלי ברשימת העבודות (אם קיים). לאחר שימוש בפקודה וביצוע פקודת jobs, הסימון stopped לא יהיה מוצג ברשימת העבודות.

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 הודעת שגיאה:

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

אם הועבר מזהה עבודה שקיים אבל אינו עצור, יש להדפיס הודעת שגיאה:

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

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

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

אם פורמט הארגומנטים או מספרם אינו תקין יש להדפיס הודעת שגיאה:

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

:quit [kill]

הפקודה מסיימת את תהליך ה-smash עם ערך החזרה 0. אם הועבר הפרמטר kill (אופציונלי) יש להרוג את כל העבודות הקיימות ברשימת העבודות לפני 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> - "
```

שימו לב לרווח לפני ואחרי המקף.

2. שליחת סיגנל SIGTERM והדפסת ההודעה " ...sending SIGTERM" (שימו לב לרווח אחרי 3 הנקודות)
3. אם התהליך נהרג לאחר פחות מ-5 שניות, יודפס "done"
4. אם התהליך לא נהרג אחרי 5 שניות מקבלת סיגנל ה-SIGTERM, שליחת סיגנל SIGKILL והדפסה: "sending SIGKILL… done

כל ההדפסות עבור עבודה ספציפית יבוצעו ללא ירידת שורה, ולאחר כל עבודה תבוצע ירידת שורה. 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
```

בדוגמא התהליך שמריץ b.out לא הגיב לסיגנל SIGTERM ולכן נשלח לו גם SIGKILL.

במידה והועברו ארגומנטים שאינם kill יש להדפיס הודעת 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 קיים, יש להדפיס הודעת שגיאה:

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

אם שני הנתיבים קיימים אבל לפחות אחד מהם הוא תיקייה ולא קובץ, יש להדפיס הודעת שגיאה:

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

פקודות חיצוניות

כאשר ה-smash מקבל פקודה חיצונית (=אינה אחת מהפקודות המובנות המפורטות בפרק הקודם), היא תנסה להפעיל את הפקודה כתוכנית בתהליך חיצוני. למשל:

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

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

במידה והנתיב הניתן ע"י המילה הראשונה בפקודה אינו קיים (כלומר, התוכנית אותה מנסים 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 ls לא נדרשת למימוש ב-smash, היא מובאת בדוגמא על מנת להבהיר שהקובץ a.bin קיים וניתן להריץ אותו.

פקודות מורכבות

בצורה דומה ל-bash, ה-smash יתמוך בשרשור פקודות ע"י התווים ";" ו-"&&". להזכירכם, התו ";" בין שתי פקודות גורם להן להתבצע אחד אחרי השנייה, ללא קשר להצלחה/כשלון של הפקודה הראשונה. למשל:

```
cd x; pwd
```

הפקודה pwd תבוצע גם אם הנתיב x לא קיים למשל, מה שיגרום לפקודה הראשונה להכשל.

בצורה דומה, התווים "&&" בין שתי פקודות גורם להן להתבצע אחד אחרי השנייה, אבל רק אם הפקודה הראשונה הצליחה. למשל בדוגמא הקודמת, אם הנתיב x לא קיים, הפקודה השנייה לא תתבצע. פקודה שנכשלת תוגדר בתור:

1. כל פקודה פנימית שהסתיימה באחת השגיאות שציינו למעלה. כדאי לייצר "ערך סיום" לכל פקודה פנימית על מנת לבדוק זאת.
2. כל פקודה חיצונית שערך ההחזרה שלה אינו 0.

לטובת כך, יש לקרוא ולכתוב את ערך היציאה של תהליכים הנוצרים במערכת. דוגמת קוד לכך מסופקת בקובץ commands.c.

התווים ";" ו-"&&" יכולים להופיע עם כל מספר של רווחים ביחס לפקודות הקודמות. למשל, כל הוריאציות הבאות חוקיות:

```
cd x;cd y           && cd z         ; pwd
cd x ; cd y && cd z;;;;;pwd
```

טיפול בסיגנלים

על ה-smash לתמוך בצירופי המקשים CTRL+C ו-CTRL+Z:

1. הצירוף CTRL+C מפסיק את התהליך שרץ בחזית (שולח לו SIGKILL).
2. הצירוף CTRL+Z משהה את התהליך שרץ בחזית (שולח לו SIGSTOP) ומוסיף אותו לרשימת העבודות (עם סימון שהתהליך מושהה).

שימו שמבחינת bash התהליך שאליו ישלחו הסיגנלים הללו הוא התהליך שמריץ את ה-smash. לכן, על ה-smash ללכוד את הסיגנלים הנוצרים על ידי לחיצות המקלדת CTRL+Z/+C, ולשלוח בהתאמה סיגנל לתהליך שרץ בחזית של ה-shell. אם אין תהליך שרץ בחזית, צירופים אלו לא ישפיעו על ה-shell. כמובן שניתן להשהות/להרוג תהליך פנימית ע"י ה-smash ע"י שליחת סיגנל מתאים עם kill.

לסיכום:

1. כאשר מתקבל הצירוף CTRL+C יש לבצע את הפעולות הבאות:
    1. להדפיס למסך:

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

    2. אם קיים תהליך שרץ בחזית, יש לשלוח לו SIGKILL ולהדפיס:

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

2. כאשר מתקבל הצירוף CTRL+Z יש לבצע את הפעולות הבאות:
    1. להדפיס למסך:

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

    2. יש קיים תהליך שרץ בחזית, יש לשלוח לו SIGSTOP ולהדפיס:

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

יצירת תהליכים

ברוב המקרים גם התהליך המריץ את ה-smash וגם כל תהליכי הבן שלו מקבלים את הסיגנלים CTRL+C ו-CTRL+Z למרות שה-smash שלכם לא שולח אותו לתהליך הבן. בעיה זו מתרחשת בגלל שה-shell האמיתי בו ה-smash רץ שולח את הסיגנלים לכל התהליכים בעלי אותו ה-group id, מנגנון אליו אנחנו לא נכנסים בקורס. בשביל להמנע מבעיה זו צריך לשנות את ה-group id 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 **הבנים** שה-smash מייצר מריצים ()setpgrp לפני שהם מריצים כל קוד אחר.

טיפול בשגיאות

()perror היא פונקציה סטנדרטית בלינוקס המקבלת הודעת שגיאה שהמשתמש מייצר ומוסיפה אותה להודעה שהיא מייצרת ע"י בחינה של errno. ההודעה מודפסת ל-stderr ולא stdout. יש להשתמש בה בשגיאות בקריאת מערכת בלבד.

אם קריאת מערכת נכשלת יש להדפיס הודעת שגיאה באמצעות הפונקציה ()perror לפי הפורמט הבא:

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

למשל, דוגמא לשימוש אידיומטי ומתומצת ב-perror יחד עם קריאת מערכת:

```
FILE* fh;
if((fh = fopen("myfile", "r")) == NULL) {
    perror("could not open file");
    exit(1);
}
```

בשורה השנייה מבצעים את קריאת המערכת, לוכדים את הערך שלה ובודקים אותה לשגיאות בשורה אחת.

הנחיות נוספות

1. יש להקפיד על הדפסות התואמות ב-100% את הדרישות בתרגיל – שגיאה בהדפסות עלולה להוביל להורדת נקודות.
2. יש להשתמש ב-()fork ו-()exec למימוש פקודות חיצוניות (יש לבחור את הוריאנט המתאים של exec לדרישות התרגיל).
3. אין להשתמש בפונקציית הספרייה system בתרגיל.
4. על התוכנית לבדוק הצלחה של כל פקודה ולהדפיס הודעת שגיאה מתאימה במידה של כשלון.
5. יש לכתוב ב-C או ++C בלבד.
6. הפורום עומד לשירותכם בשאלות בכל נושא הקשור לתרגיל.

קומפילציה ולינקוג'

יש לוודא שהקוד מתקמפל ע"י הפקודות הבאות, בהתאמה ל-++C ול-C:

```
g++ -std=c++11 -Wall -Werror -pedantic-errors -DNDEBUG -pthread *.cpp -o smash
gcc -std=c99 -Wall -Werror -pedantic-errors -DNDEBUG -pthread *.c -o smash
```

הדגל Werror גורם ל-warnings רגילות בקומפיילר gcc/g++ להפוך ל-errors. יש לוודא שהקוד שלכם מתקמפל **על המכונה של הקורס** ללא warnings/errors.

קוד שלא יתקמפל/יתקמפל עם אזהרות לא יקבל ציון.

עליכם לספק Makefile – הכללים שחייבים להופיע בו הם:

1. כלל smash שיבנה את התוכנית smash
2. כלל עבור כל קובץ נפרד שקיים בפרוייקט
3. כלל clean המוחק את כל תוצרי הקומפילציה

וודאו שהתוכנית נבנת ע"י הפקודה make. יש לקמפל וללנקג' את התוכנית עם הדגלים המופיעים למעלה.

מומלץ להפריד את המימוש לקבצי c/cpp. ו-h. על מנת להקל על בניית התוכנית – יש להתייחס לכך בקובץ ה-Makefile.

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

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

בהצלחה!