

תרגיל בית #1

מועד ההגשה: יום שלישי 17/04/2019, 23:55

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

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

השם smash נגזר מצרף המילים Small Shell.

אופן ההפעלה של התוכנית: `> smash`

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

על מנת להקל על המימוש אפשר להניח כי עד 100 תהליכים יכולים לרוץ בו זמנית והשם של כל תהליך יכול להכיל עד 50 תווים. לתרגיל מצורף הקובץ `smash.zip`, בקובץ זה תמצאו קוד בסיס לטיפול בפקודות (`command.c`) וקוד מעטפת `smash.c` כמו גם `Makefile`, עליכם להשלים את הקוד המצוי `command.c` ולהוסיף טיפול בסיגנלים, הפונקציות אשר מטפלות ומגדירות את שגרת הטיפול בסיגנלים יהיו בקובץ `signals.c`.
לסיכום: עליכם לממש את הפקודות ב-`command.c` ולהוסיף טיפול בסיגנלים ב-`signals.c`.
`smash.c` עליכם רק להוסיף קריאות לפונקציות המגדירות את אופן הטיפול בסיגנלים.

- אם ברצונכם לשנות את חתימות הפונקציות או את הקבצים שניתנו לכם אין עם כך שום בעיה, יש לכם את החופש המוחלט לכך. כל שעליכם לוודא הוא שהקבצים `signals.c` ו-`command.c` יכילו את התוכן המבוקש.

התוכנית תפעל בצורה הבאה:

- 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, 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 ההודעה¹
`smash >`
- 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.
- ניתן להשאיר שורות ריקות.

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

אופן פענוח שורת פקודה ב smash :

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

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.

ניתן להניח שמספר הארגומנטים ל-command קטן מ-19. אם פעולת הפעלת התוכנית החיצונית לא הצליחה, תודפס הערת שגיאה המתארת את סיבת כישלון הפעולה (תזכורת – perror). אם הפקודה מסובכת מדי (ראה הסבר בהמשך). לדוגמא `ls | wc -l` התוכנית תפעיל את csh אשר תבצע את הפקודה בשבילה.

טיפ: ניתן להפעיל את csh באופן הבא: `csh -f -c "ls | wc -l"`

<command> [arguments] &

כמו בסעיף הקודם, אך ללא המתנה לסיום ביצוע התוכנית (הרצה ברקע). התהליך החדש יכנס לרשימת jobs (ראה פקודת jobs ברשימת פקודות ה-built-in).

פקודה מסובכת: מה היא?

פקודה היא מסובכת אמ"ם :

- מופיעה בפקודה אחת (או יותר) מהמילים הבאות:
- "&" או "&|".
- "<" או ">" או ">>".
- מופיע התו "*" או "?" בפקודה.

פקודות של built-in של smash :

pwd

הדפס את מיקומו של המדריך הנוכחי.

cd <path>

שנה את המדריך הנוכחי ל- path. אם ה- path אינו נכון הדפס הודעת שגיאה.

smash error: > "path" - path not found

במקרה בו path שווה ל "-", משנים את המדריך הנוכחי אל הקודם ומדפיס אותו (אם קיים).
צריך לזכור רק מדריך אחד אחורה. לדוגמא:

```
smash > pwd
/foo
smash > cd /bar
smash > pwd
/bar
smash > cd -
/foo
smash > cd -
/bar
smash > cd -
/foo
.
```

history

מדפיס למסך את היסטוריית הפקודות של smash, כל פקודה בשורה חדשה, יחד עם הפרמטרים, כאשר הפקודה שהורצה אחרונה הינה מודפסת למסך אחרונה. לדוגמא:

```
smash > history
pwd
pwd
cd -
cd -
cd -
```

אם לא הורצו פעולות, הפקודה לא תדפיס כלום. הפקודה תשמור עד 50 פקודות אחורה ותמחק את הרשומות הכי ישנות כאשר אין מקום ע"מ לפנות מקום עבור פקודה חדשה.

jobs

ה- smash יחזיק רשימת jobs, את pid ואת הזמן בשניות מהרגע שהתהליך נכנס לרשימת ה- jobs של כל אחד מהם. ברשימה יופיעו כל התהליכים שהופעלו ברקע (ע"י &) אך טרם הסתיימו. הפקודה תציג רשימת jobs וpids, מספר סידורי לפני לכל תהליך. לדוגמא:

```
smash > jobs
[1] a.out : 12340 214 secs
[2] /usr/bin/ls: 12341 57 secs
[3] b.out : 12342 10 secs
```

kill -<signal> <job>

שליחת Signal שמספרו signal אל התהליך המזדהה עם **job** (מרשימת jobs). במידה וjob אינו קיים יש להדפיס :

```
smash error: > kill job – job does not exist
```

במידה ויש כישלון אחר בשליחת ה-Signal יש להדפיס :

```
smash error: > kill job – cannot send signal
```

הערה : job הינו מס' הjob ברשימת jobs ולא pid. יש "-" לפני מספר ה Signal.

showpid

ה-smash ידפיס את pid שלו (יודפס ה-pid של smash). לדוגמא :

```
smash > showpid
```

```
smash pid is 12339
```

fg [command number]

הפקודה תגרום להרצה ב- foreground של התהליך (job) המזוהה עם *command number*. לפני העברת ה-job ל- foreground יודפס שמו. הפעלת הפקודה ללא פרמטרים, תעביר ל- foreground את התהליך האחרון שהופעל ברקע. **כשהתהליך יסתיים הוא יוצא מרשימת jobs**. לדוגמא : (המשך לדוגמא מסעיף הקודם)

```
smash > fg
```

```
b.out
```

```
smash > jobs
```

```
[1] a.out : 12341 218 secs
```

```
[2] /usr/bin/ls : 12342 61 secs
```

```
smash > fg 1
```

```
a.out
```

bg [command number]

הפקודה תגרום להרצה ב- background של התהליך (job) המזוהה עם *command number*. לפני העברת ה-job ל- background יודפס שמו. הפעלת הפקודה ללא פרמטרים, תעביר ל- background את התהליך האחרון שריצתו הושהתה (ע"י CTRL-Z). **כשהתהליך יסתיים הוא יוצא מרשימת jobs**.

שימו לב: התהליך ירוץ ברקע, כלומר על smash לא לחכות לסיום התהליך אלא להחזיר את prompt מיידית. לדוגמא :

```
smash > jobs
```

```
[1] a.out : 12340 56 secs
```

```
[2] /usr/bin/less : 12341 23 secs
```

```
[3] c.out : 12342 10 secs
```

```
smash > bg
```

```
c.out
```

```
smash > bg 2
```

```
/usr/bin/less
```

quit[kill]

א. quit

יציאה מתוכנית ה-smash.

ב. quit kill

הרחבה לפקודת ה-quit היא לאפשר למשתמש להרוג את כל התהליכים בעת היציאה.

הפקודה quit 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.

הערה: אם ברצונכם לבדוק אופן זה של quit, ניתן לייצר תוכנית דמה אשר מתעלמת מסיגנל ה-

SIGTERM.

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.

cp <old name> <new name>

מעתיק את הקובץ old_name לnew_name. אפשר להניח כי old_name הינו קובץ ולא תיקייה.

לאחר העתקה מוצלחת יודפס למסך "<old_name> has been copied to <new_name>". אם

הייתה תקלה יש להדפיס אותה באמצעות perror

לדוגמא :

smash > cp a.out b.out

a.out has been copied to b.out

עליכם לממש את הפונק' ה built in הנ"ל ולא להשתמש בfork+exec

עבורם.

הצגת *signals* ב-*smash*:

כל פעם ששולחים signal לpid כלשהו, ה-*smash* יציג את הpid וסוג הסיגנל שנשלח. לדוגמא, אם fg צריך להעיר תהליך מושהה:

```
smash > fg 1
```

```
smash > signal SIGCONT was sent to pid 12340
```

השהיית/הריגת התהליך:

- על ה-shell לתמוך בצירופי המקשים CTRL+C ו-CTRL+Z:
 - הצירוף CTRL+Z משהה את התהליך שרץ ב-foreground (שולח לו SIGTSTP) ומוסיף אותו לרשימת ה-jobs (עם ציון שהתהליך מושהה) לדוגמא:

```
smash > jobs
```

```
[1] a.out 12340 23 secs (Stopped)
```

```
[2] /usr/bin/ls 12341 10 secs
```

- . לאחר השהיית התהליך, הקשת הפקודה fg תגרום לשחזור הריצה של התהליך המושהה ב-foreground (ע"י שליחת SIGCONT). בנוסף, יש לתמוך בפקודה bg אשר תגרום לשחזור הריצה של התהליך המושהה ב-background.
- הצירוף CTRL+C מפסיק את ריצת התהליך שרץ ב-foreground (שולח SIGINT).

שימו לב 1: אם אין פקודה ב-foreground, צירופים אלו לא ישפיעו על ה-shell.

שימו לב 2: ה-shell שלכם נדרש רק לנתב את הסיגנל לתהליך שרץ ב-foreground של ה-shell.

שימו לב 3: כאשר את מריצים תוכנית באמצעות exec כל ה-signal handlers חוזרים לdefault.

שימו לב 4: 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 ) {
```

```
    setpgid(); // 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
```

```
}
```

שימו לב 5: כעת כאשר נלחצים הצרופים CTRL+C ו-CTRL+Z הסיגנלים מנותבים ע"י ה-shell המקורי smash (ולא לתהליך הרץ בחזית).

שימו לב 6: עליכם לתפוס את הסיגנלים ב-smash ולנתב אותם לתהליך שרץ בחזית.

הנחיות לביצוע

- יש** להשתמש בקריאות המערכת fork ו-exec (יש לבחור את הצורה המתאימה של exec לדרישות התרגיל).
- אין** להשתמש בפונקציית הספרייה system.
- ניתן להשתמש בפונקציית הספרייה fgets בכדי לקרוא את שורת הפקודה מהמקלדת, וניתן להשתמש בפונקציות הספרייה strtok ו-strcmp לפענוח הקלט.
- ניתן להשתמש בקריאות מערכת ההפעלה wait, waitpid, chdir, getpid, time ובפונקציית הספרייה getcwd.

- על התוכנית לבדוק הצלחת ביצוע כל פקודה, בכל מקרה של כישלון יש להדפיס הודעת שגיאה מתאימה (תזכורת – perror).

הנחיות לתיעוד והגשה

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

http://moodle.technion.ac.il/pluginfile.php/383709/mod_resource/content/1/HW_info.pdf

הקפידו על הנחיות התיעוד כפי שניתנו בדפי המידע הכללי אשר חולקו בתרגול הראשון. תנו את הדעת על הנקודות הבאות:

1. אפיינו את התוכנית שברצונכם לכתוב - ציירו דיאגרמת בלוקים שתסביר את המבנה הכללי של התוכנית ותתאר את החלקים העיקריים בה. כל בלוק בדיאגרמה ימומש כמודול נפרד בקובץ נפרד. סמנו בחצים את מעבר הנתונים מבלוק לבלוק. הסבירו באופן כללי מהם הנתונים המועברים.
2. תארו את מבנה הנתונים בו תשתמשו - מנשק הפעולות, צורת ארגון הנתונים והשדות (מלבד האינפורמציה) אותם אתם כוללים בנתון לצורך שילובו במבנה הנתונים. נמקו מדוע בחרתם להשתמש במבנה נתונים זה.

**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.**

Useful Man Pages:

`exec(3),fork(2),wait(2),waitpid(2),pause(2),signal(2) or sigaction(2),
stat(2),open(2),read(2),write(2),close(2)`

בהצלחה!!!