

מערכות הפעלה - תרגול 5

Signals

מתרגל-יורם סגל
yoramse@colman.ac.il

Contents

One

- לינוקס – בקרת תהליכים
- Introduction to signals

Two

- Process Vs Jobs
- Background (bg) vs Foreground (fg)

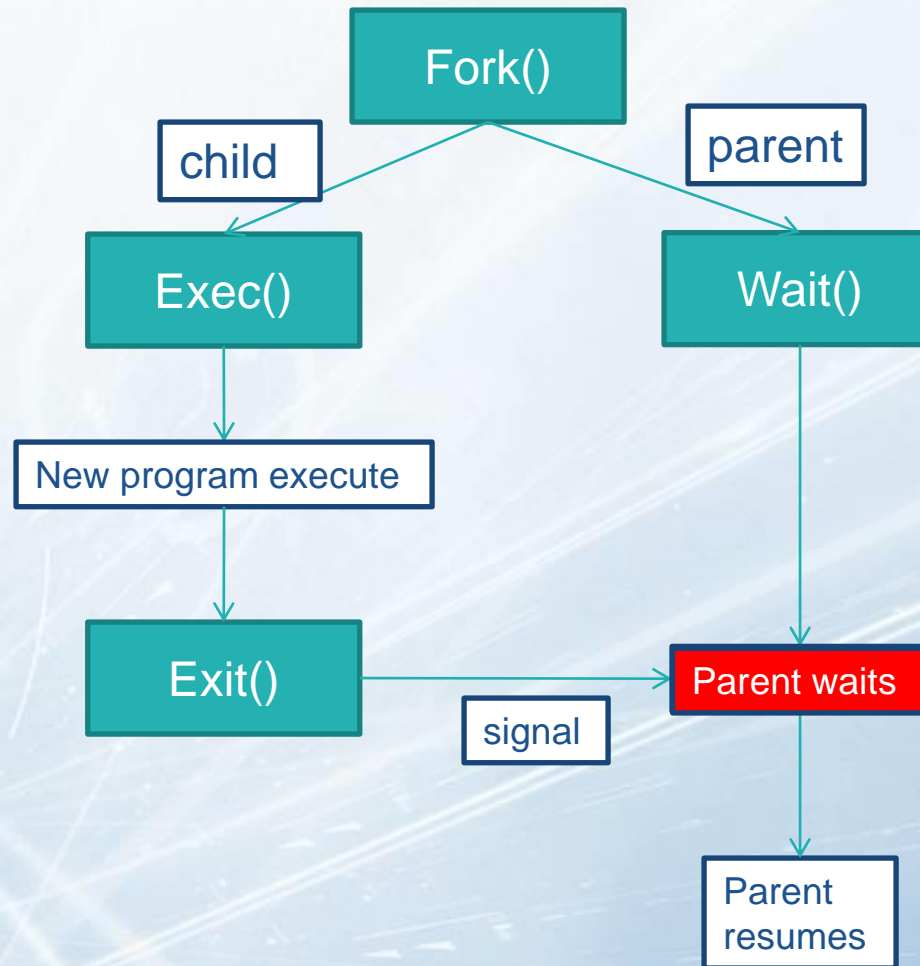
Three

- `signal()` → define Callback/Interrupt function
- `Kill()` → send signal to process
- `raise()` → like kill but can call to itself as well

Four

- `pause()` → wait for any signal
- `wait()` → wait for child process to exit
- `alarm()` timer

Wait flow



בקרת תהליכים

❖ כדי להציג את הסטטוס של התהליכים הרצים
אפשר להשתמש בפקודה: **ps [options]**

❖ ps:

```
shani@shani-VirtualBox:~/Desktop$ ps
  PID TTY          TIME CMD
 1995 pts/0        00:00:00 bash
 4600 pts/0        00:00:00 ps
```

The difference between a "job" and a "process"

- ❖ A **process** is any running program with its own address space.
- ❖ A **job** is a concept used by the shell - any program you interactively start that doesn't detach (במנותק מהמשתמש)
(i.e., under the direct control of a user) is a job.
 - The shell has an identifier number to each job (which is **different** from the PID)

job : קבוצה של **process-ים** הרצים ביחד כקבוצה תחת הshell ולא תחת המשתמש

בקרת תהליכים (הרצת תהליך ברקע)

❖ מספר תהליכים רצים על המערכת, בטרמינל השולט רק תהליך אחד מתקשר עם המקלדת והמסך – foreground job.

❖ שאר התהליכים נקראים background jobs

❖ כדי להריץ תהליך ברקע (למה?) נשתמש ב "&" לאחר הרצת התהליך:

❖ ./a.out &

בקרת תהליכים

❖ שליחת תהליך לרקע:

- שלב א': עצירת התהליך CTRL+Z
- שלב ב': המשך התהליך ברקע בעזרת הפקודה: bg

```
# ./a.out
```

```
# [CTRL-Z]
```

```
# bg
```

Play with ./demo_bg_fg.out

בקרת תהליכים

הצגת כל התהליכים שרצים ברקע תעשה
על ידי שימוש בפקודה **jobs**

jobs

[1] Running bash download-file.sh &

[2] - Running evolution &

[3] + Done nautilus .

תהליך אחד לפני אחרון
שהועבר לרקע

התהליך האחרון
שהועבר לרקע.

בקרת תהליכים – סדר חזרה לקידמת הבמה

הבאת תהליך מהרקע:

שלב א': הצגת כל התהליכים הרצים ברקע

שלב ב': בחירת התהליך בעזרת הפקודה: `fg %[num]`

```
# jobs
```

```
[1] Running bash download-file.sh &
```

```
[2]- Running evolution &
```

```
[3]+ Done nautilus .
```

```
# fg %1
```

התהליך האחרון שהועבר
לרקע (מסומן על ידי "+"),
יהיה הראשון שיחזור
במידה ונשתמש בפקודה
`fg` בלי ארגומנטים

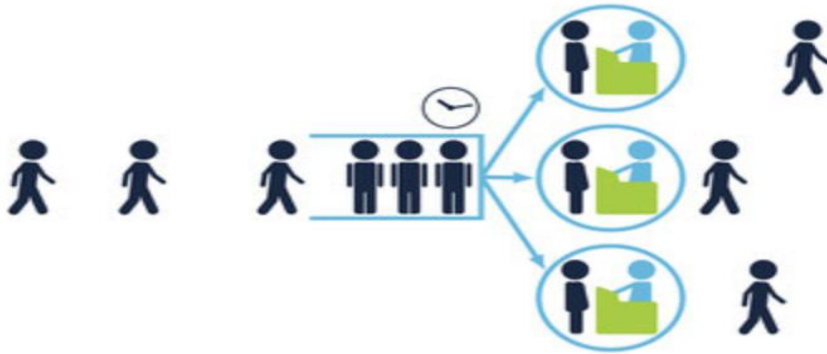
תרגיל c.1_4t

- ❖ לקמפל את התרגיל
- ❖ לתקן את הערה `#include <unistd.h>`
- ❖ שם קובץ ההרצה יהיה `t4_1.out`
- ❖ להריץ את התוכנית. (מה קורה?)
- ❖ להפסיק את התוכנית
- ❖ להריץ את התוכנית פעם 2.
- ❖ להעביר את התוכנית לרקע תוך כדי הרצה [CTRL-Z]
- ❖ לנסות להפסיק את התוכנית
- ❖ להעביר את התוכנית קדימה (רשימת JOB-ים, מספר הJOB וfg על מספר הג'וב

Signals

תהליכים מקביליים

הודעות בין תהליכים



הודעה

רמה מסינגר
מילים: רמה מסינגר
לחן: אריק אביגדור

היום השליח בא עם הודעה
איזה מזל שקראתי, החזרתי, אמרתי:
תודה, עוד לא.
באמצע החיים, באמצע המרוץ
עוצרים אותך פתאום, אומרים עכשיו
לנשום.
לנשום, להסתכל, להתחיל להתרגל...



מבוא ל signals (אינטרפט/Callback)

❖ Signals הינו מכניזם שבעזרתו תהליכים מודעים על מאורעות המתרחשים במערכת.

❖ מאפיין חשוב: איתות הינו מאורע אסינכרוני

- תהליך צריך להיות מוכן לקבל איתות בכל שלב של ריצתו (אפילו בזמן שהוא מבצע sysCall).

❖ מי יכול לאותת לתהליך?

- מערכת ההפעלה

- תהליך אחר

- התהליך עצמו

© Original Artist
Reproduction rights obtainable from
www.CartoonStock.com



search ID: fabn16

מבוא ל signals

❖ סיבות לשליחת איתותים:

■ שינוי סטטוס של תהליך :

- לסיים תהליך (**SIGKILL**)
- לעצור תהליך (**SIGSTOP**)
- להמשיך תהליך שנעצר (**SIGCONT**).

■ לידע תהליך על בעיה:

- חריגה בזיכרון (**SIGSEGV**)
- שגיאת פעולה אריתמטית (למשל, חלוקה באפס) (**SIGFPE**)

■ תקשורת בין תהליכים

- **SIGUSR1, SIGUSR2**

מבוא ל signals

❖ דוגמא לסיגנל המודיע על חריגה בזיכרון (**SIGSEGV**)

❖ :

```
int main()
{
    char arr[100];
    arr[1000000000]=10;
}
```

```
$ gcc main.c
```

```
$ ./a.out
```

```
Segmentation fault (core dumped)
```

מבוא ל signals

❖ לכל איתות מוגדרת פעולת ברירת מחדל

- **TERM** - סיום התהליך
- **IGN** – להתעלם מהאיתות
- **CORE** – סיום התהליך וזריקת core
- **STOP** – עצירת התהליך.
- **CONT** – המשכת התהליך אם במצב עצור.

core dump

הוא קובץ המכיל את שטח הכתובות (זיכרון) של תהליך, כאשר התהליך מסתיים במפתיע.

מבוא ל signals - פעולת ברירת מחדל

Signal	Value	Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from abort (3)
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers
SIGALRM	14	Term	Timer signal from alarm (2)
SIGTERM	15	Term	Termination signal
SIGUSR1	30, 10, 16	Term	User-defined signal 1
SIGUSR2	31, 12, 17	Term	User-defined signal 2
SIGCHLD	20, 17, 18	Ign	Child stopped or terminated
SIGCONT	19, 18, 25	Cont	Continue if stopped
SIGSTOP	17, 19, 23	Stop	Stop process
SIGTSTP	18, 20, 24	Stop	Stop typed at tty
SIGTTIN	21, 21, 26	Stop	tty input for background process

CTRL+C

Alpha

Sparc

x86,
arm

Mips

שעון מעורר

Where three values are given, the first one is usually valid for alpha and sparc, the middle one for x86, arm, and most other architectures, and the last one for mips.

מבוא ל signals

❖ מה אפשר לעשות שמקבלים איתות?

- לבצע את פעולת ברירת המחדל: SIG_DFL

- להתעלם: SIG_IGN

(אי אפשר להתעלם משני איתותים: SIGSTOP, SIGKILL)

- לבצע פונקציה שהוגדרה מראש. כאשר הפונקציה שהוגדרה מראש חוזרת, השליטה חוזרת לקוד הראשי להמשך הרצה.



מבוא ל signals

❖ פוקציית טיפול באיתות:

```
#include <signal.h>
```

```
typedef void signal(*sighandler_t)(int);
```

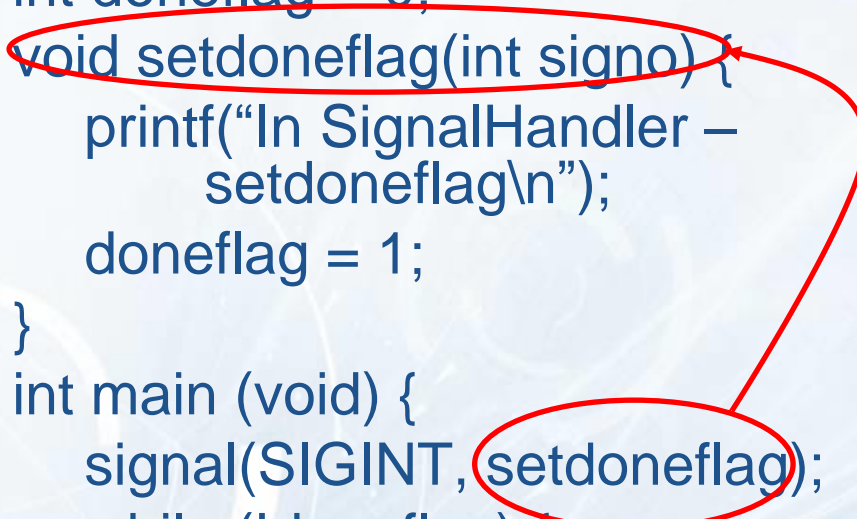
```
sighandler_t signal(int signum, sighandler_t handler);
```

❖ נראה לא משהו...

T4_1.c

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
int doneflag = 0;
void setdoneflag(int signo) {
    printf("In SignalHandler -
        setdoneflag\n");
    doneflag = 1;
}
int main (void) {
    signal(SIGINT, setdoneflag);
    while (!doneflag) {
        printf("press CTRL+C to kill the Loop\n");
        sleep(1);
    }
    printf("Program terminating ...\n"); return 0;}

```



Output:

```
$ ./a.out
press CTRL+C to kill the Loop
press CTRL+C to kill the Loop
press CTRL+C to kill the Loop
^C
In SignalHandler - setdoneflag
Program terminating ...
$

```

סיגנל הגורם להפעלת פונק'

```
signal(SIGINT,signal_function);
```

Signal	Value	Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from <code>abort(3)</code>
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers
SIGALRM	14	Term	Timer signal from <code>alarm(2)</code>
SIGTERM	15	Term	Termination signal
SIGUSR1	30,10,16	Term	User-defined signal 1
SIGUSR2	31,12,17	Term	User-defined signal 2
SIGCHLD	20,17,18	Ign	Child stopped or terminated
SIGCONT	19,18,25	Cont	Continue if stopped
SIGSTOP	17,19,23	Stop	Stop process
SIGTSTP	18,20,24	Stop	Stop typed at tty
SIGTTIN	21,21,26	Stop	tty input for background process

שם/סוג
הסיגנל

מצביע/הנדלר
לפונקציה

❖ הפקודה `signal()` מייצרת מצביע (handler) לפונקציית שרות המיועדת לטפל ב `signal` המסוויים.

❖ ללא הוספת ה- `signal()` לקוד אזי כאשר מתפרץ הסיגנל תופעל פונקצית בררית מחדל.

❖ במקרה של בעיה היא תחזיר `SIG_ERR`

סיגנל הגורם להפעלת ברירת מחדל

מה קורה בהתרחש סיגנל?

❖ ביצוע ה signal handler
(הפעלה והרצת פונקציה הטיפול בסיגנל),

❖ בסיום ביצוע פונקציה הסיגנל...

❖ **הסיגנל מקבל בחזרה את handler הטיפולטיבי**
(מקבל הגדרה של פונקציה בררת-מחדל לביצוע),

❖ לכן, בקוד, **הגדרנו שוב**, פונקציה חלופית משלנו לביצוע,
(הגדרנו handler לפונקציה שלנו).

❖ כלומר קראנו מחדש (פעם נוספת) ל:
`signal(SIGINT,ctrl_c);`

נא להריץ את t4_2.c

kill

❖ שליחת איתות בין תהליכים מתבצעת ע"י הפקודה
:kill

```
#include <sys/types.h>
```

```
#include <signal.h>
```

```
int kill(pid_t pid, int sig);
```

ערך מוחזר:

0 : הצלחה

-1 : כישלון

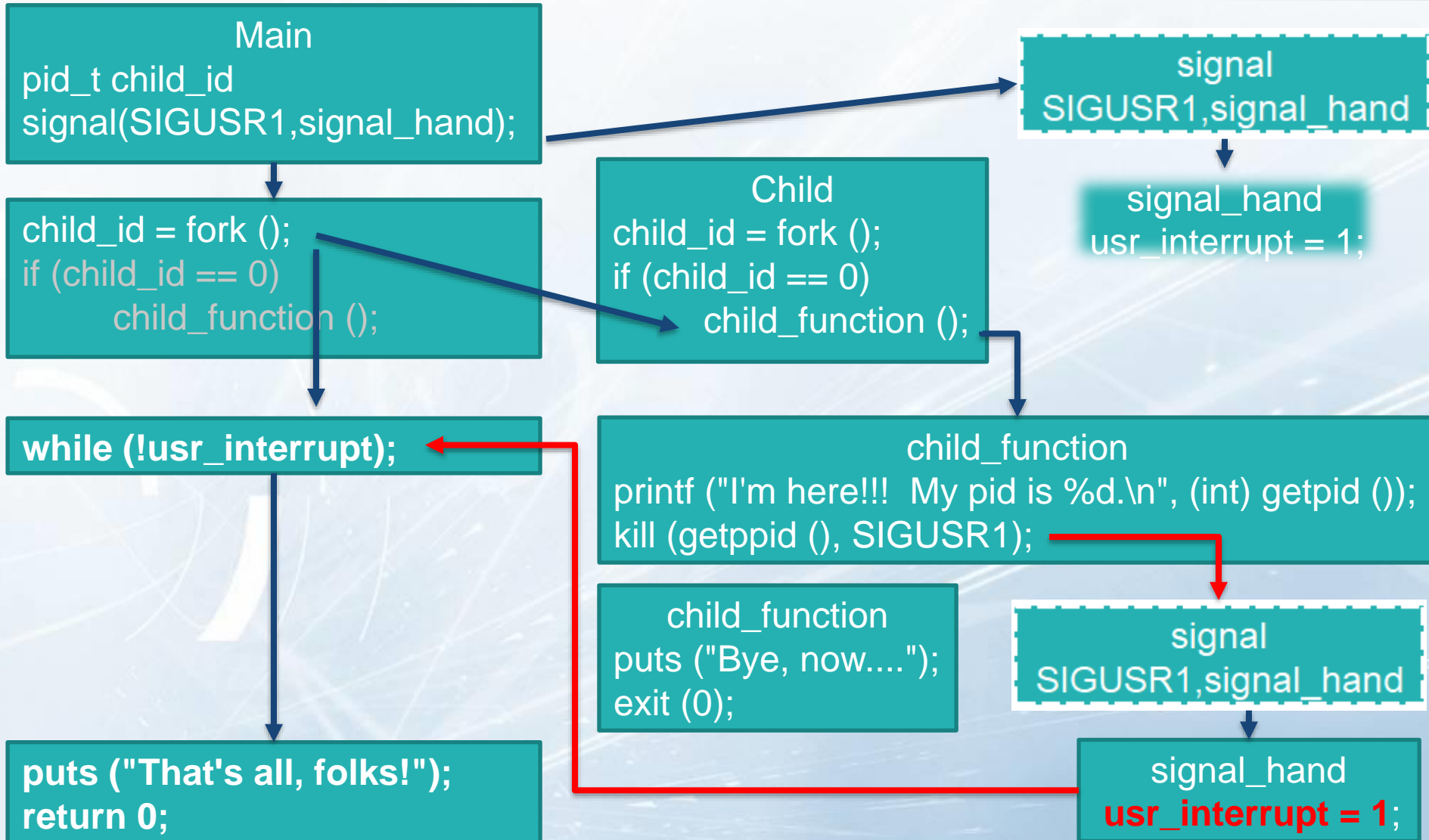
kill

❖ המשתנה sig המועבר לKILL נלקח מתוך רשימת האיתותים

❖ Pid:

- $Pid > 0$: ישלח איתות לתהליך שה process id שלו שווה ל pid
- $Pid = 0$: לכל התהליכים שה process group id שלהם שווה לזה של השולח
- $Pid = -1$: לכל התהליכים במערכת חוץ מתהליך init והתהליך השולח עצמו.
- $Pid < -1$: לכל התהליכים שה process group id שלהם שווה ל $|pid|$

הדמייה של הפקודה wait ע"י איתותים-T4_3.c



שליחת KILL מתוך טרמינל של לינוקס

X The `kill` command can also be used to send a signal to a process:

```
kill -l /* list all signals */  
kill -XXX pid1 pid ..... pid
```

X In the above `XXX` is the signal name without the initial letters `SIG`. e.g `SIGINT` will change into `INT`

X `kill -KILL 1357 2468` kills process `1357` and `2468`. Use `ps` command to get list of available process

X `kill -INT 6421` sends a `SIGINT` to process `6421`.

X A `kill` without a signal name is equivalent to `SIGTERM`.

X `-9` is equal to `-SIGKILL`.

Signal

SIGHUP

SIGINT
SIGQUIT
SIGILL
SIGABRT
SIGFPE
SIGKILL
SIGSEGV
SIGPIPE

SIGALRM
SIGTERM
SIGUSR1
SIGUSR2
SIGCHLD
SIGCONT
SIGSTOP
SIGTSTP
SIGTTIN

Kill via shell t4_4.c

Kill -signalNum/signalName pid

```
u2 89-231 23 :  
u2 89-231 23 : gcc t5_4.c  
u2 89-231 24 : ./a.out &  
[1] 25933  
u2 89-231 25 : kill -USR1 25933  
Hold me close, Red. It's gettin' dark.  
u2 89-231 26 : Tell aunty em to let old yeller out.  
Tell Tiny Tim I won't be coming home this Christmas.  
Tell Scarlett I do give a damn.  
Pardon me.  
Thank you!  
You love me.  
You really love me!  
[1] Exit 1 ./a.out  
u2 89-231 26 :  
u2 89-231 26 : █
```

pause

❖ כאשר תהליך רוצה להיות במצב "הפסקה" עד שיקבל איתות (מתהליך אחר) יש להשתמש ב
system call: pause()

❖ **int pause(void);**

❖ הפונקציה חוזרת רק כאשר פונקציית signal handler תפסה איתות וסיימה פעולתה.

❖ הערך שחוזר הינו 1- (שגיאה), אפשר להתעלם.

- pause: wait for signal
- wait: wait for child process to change state

T4_5.c

```
#include <signal.h>
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>

void signal_hand (int sig)
{
    signal(SIGUSR1,signal_hand);
    printf("got a signal\n");
}

int main (void)
{
    signal(SIGUSR1,signal_hand);
    printf("waiting for SIGUSR1\n");
    pause();
    printf("waiting for SIGUSR1 again\n");
    pause();
    printf("I'm done\n");
    return 0;
}
```


T4_5.c

```
bash-4.1$  
bash-4.1$ gcc t5_5.c  
bash-4.1$ ./a.out  
waiting for SIGUSR1  
^Z  
[1]+  Stopped                  ./a.out  
bash-4.1$ ps  
  PID TTY          TIME CMD  
11278 pts/6        00:00:00 tcsh  
11653 pts/6        00:00:00 bash  
11916 pts/6        00:00:00 a.out  
11931 pts/6        00:00:00 ps  
bash-4.1$ kill -CONT 11916  
bash-4.1$ kill -USR1 11916  
got a signal  
bash-4.1$ waiting for SIGUSR1 again  
kill -USR1 11916  
got a signal  
I'm done  
[1]+  Done                    ./a.out  
bash-4.1$
```

SIGCONT	19,18,25	Cont	Continue if stopped
---------	----------	------	---------------------

Pause לא מונע מסיגנל לסיים תהליך !

int pause(void);

מתי לא נוכל לראות את הערך החוזר?

כאשר הטיפול בסיגנל שנשלח לתהליך יוביל לסיום התהליך.
לדוגמא:

```
Int main{
    int i;
    pid_t pid;
    signal(SIGKILL,SIG_DFL);
    pid = fork();
    if (pid==0){
        sleep(5);
        kill(getppid(),SIGKILL);
        exit(0);
    }
    else{
        i = pause();
        if (i==-1){
            printf("You are the president of USA");
        }
        return 0;
    }
}
```

pause לא מונע מסיגנל לסיים תהליך !

pause - שאלה ממבחן

בהינתן קטע הקוד הבא:



SIGCHLD

```
int main (void)
{
    printf("waiting for SIGCHLD\n");
    pause();
    printf("waiting for SIGCHLD again\n");
    pause();
    printf("I'm done\n");
    return 0;
}
```

הניחו שקיים תהליך כלשהו השולח ברצף סיגנלים מסוג SIGCHLD (שפעולת ברירת המחדל שלו היא ignore) לתהליך המריץ את התוכנית הנתונה, מה יודפס על המסך כתוצאה מריצת התוכנית? הסבירו.

SIGCHLD	20,17,18	Ign	Child stopped or terminated
----------------	----------	-----	-----------------------------

pause - שאלה ממבחן

ההדפסה על המסך:

waiting for SIGCHLD

הסבר: לאחר ההדפסה הראשונה, התהליך נכנס למצב "הפסקה" עד שיקבל איתות כלשהו שהוגדר עבורו טיפול על ידי התהליך- רק במקרה זה יתעורר ממצב ה"הפסקה" וימשיך לרוץ.

אמנם ברקע רץ תהליך אחר שמייצר סיגנל SIGCHLD אך, כיוון שהתהליך שנמצא ב"הפסקה" לא הגדיר טיפול באיתות ה-SIGCHLD הוא לא יתעורר עקב הגעת איתות זה.

alarm (timer)

❖ תהליך יכול לבקש שישלח אליו איתות מסוג SIGALRM בעוד מספר שניות.

❖ מאוד שימושי אם רוצים להגביל פעולה בזמן.

❖ **unsigned alarm(unsigned seconds);**

e.g.: `ret = alarm(10);`

הערך החוזר: כמה זמן נשאר ל `alarm` הקודם אם לא הייתי מבצע שוב קריאה ל `alarm`.

:Stop Alarm Timer -

אם הערך שניתן ל `alarm` הוא 0, כל בקשת `alarm` קיימת מבוטלת.

Stop alarm example

```
#include <signal.h>
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
```

```
shani@shani-VirtualBox:~/Dropbox/Operating Systems/Tirgul/T7$ ./a.out
starting the search, ret val = 0
finished search, ret val = 8
```

```
void alarm_hand (int sig){
    signal(SIGALRM,alarm_hand);
    printf("got an alarm wakeup signal\n");}
```

```
int main (void){
    unsigned int ret;
    signal(SIGALRM,alarm_hand);
    ret = alarm(10);
    printf("starting the search, ret val = %u\n", ret);
    sleep(2);
    ret = alarm(0);
    printf("finished search, ret val = %u\n", ret);
    return 0;
}
```

stop_alarm_ex.c

T4_6.c

```
#include <signal.h>
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>

int val=0;
void alarm_hand (int sig) {
    signal(SIGALRM,alarm_hand);
    printf("got an alarm wakeup signal\n");
    val=5;
}
void DFS() {
    while(val==0);
    printf("val equals %d\n",val);
}

int main (void) {
    signal(SIGALRM,alarm_hand);
    alarm(10);
    printf("starting the search\n");
    DFS();
    printf("finished search\n");
    return 0;
}
```

```
starting the search
got an alarm wakeup signal
val equals 5
finished search
shani@shani-VirtualBox:~/Desktop$
```

T4_7.c

❖ דוגמא לשילוב בין alarm לבין pause.

- יש להריץ ברקע את התוכנית ולהמתין לתוצאת הטיימר

- יש להריץ פעם שניה ברקע

- יש להקליד

kill -USR1 2911

```
shani@shani-VirtualBox:~/Desktop$ gcc t4_7.c
shani@shani-VirtualBox:~/Desktop$ ./a.out
waiting for SIGUSR1 or 10 seconds, my pid is: 2827
got an alarm wakeup signal
woke up after 10 seconds,waiting for SIGUSR1 or 5 seconds
got an alarm wakeup signal
I'm done
shani@shani-VirtualBox:~/Desktop$
```

raise

int raise(int sig)

- ❖ C library function causes signal **sig** to be generated.
- ❖ The **sig** argument is compatible with the SIG macros.
- ❖ **sig** – This is the signal number to be sent.
- ❖ This function returns zero if successful, and non-zero otherwise.

The **kill** function sends a signal to a process or a group of processes. The **raise** function allows a process to send a signal to itself.

T4_8.c

```
#include <signal.h>
#include <stdio.h>

void signal_catchfunc(int);

int main()
{
    int ret;

    ret = signal(SIGINT, signal_catchfunc);

    if( ret == SIG_ERR)
    {
        printf("Error: unable to set signal handler.\n");
        exit(0);
    }
    printf("Going to raise a signal\n");
    ret = raise(SIGINT);

    if( ret !=0 )
    {
        printf("Error: unable to raise SIGINT signal.\n");
        exit(0);
    }

    printf("Exiting...\n");
    return(0);
}

void signal_catchfunc(int signal)
{
    printf("!! signal caught !!\n");
}
```

**Going to raise a signal
!! signal caught !!
Exiting...**