Operating Systems
371-1-1631

Administrative Details

Ben-Gurion University of the Negev
Communication Systems Engineering Department

Welcome to Operating Systems course!

- Teaching Assistant: **Yoram Segal**
  - yoramse@post.bgu.ac.il

- Reception hours:
  - Every Wednesday from 20:30 to 21:30 with advance registration.
    In the model website you can find a link to a registration file for the reception hours.
    At the top of that file, there is a link to the zoom session depending on the date you selected.

Attendance at practices

1. Mandatory attendance at practices.
2. Up to 2 subtractions per semester are allowed.
3. The obligation to complete attendance is a threshold condition for obtaining a grade in the course.
4. It is mandatory to open cameras. Participation in the practice without an open camera will not count as participation in the practice.
5. Anyone who is not present at the practice, can complete it from the recording as long as he submits a printed summary within 6 days from the practice date.
6. The summary should be typed or written with clear handwriting in a PDF document containing 1250 words (about 2 pages) + Screenshots of the exercises.
7. The summary must receive a passing grade.
8. According to the University Laws (Reserves, etc.), Deductions for any justifiable reason will allow for an extension and submit the missing amount of practice until the end of the semester.

Operating Systems
371-1-1631

Tutorial 1 – System Calls

Ben-Gurion University of the Negev
Communication Systems Engineering Department

מה היא מערכת ההפעלה?

ואת הנשמה המתוקה שלי
היחידה שמדליקה אותי
ואיתך אני כל העולם
ואיתך אני כל היקום
בלעדייך אני חצי בן אדם
בלעדייך אני בעצם כלום

מילים: עוזי חיטמן
מתוך השיר "**לכל אחד יש**"

Outline

- **Motivation & basics**
- Process control
- File management
- Concluding example

Motivation

- A process is ***not supposed*** to access the kernel.
  - It can't access the kernel memory or functions.

- This is strictly enforced ("protected mode") for good reasons:
  - Can jeopardize other processes running.
  - Cause physical damage to devices.
  - Alter system behavior.

- The system call mechanism provides a safe mechanism to ***request*** specific kernel operations.

System Call - Definition

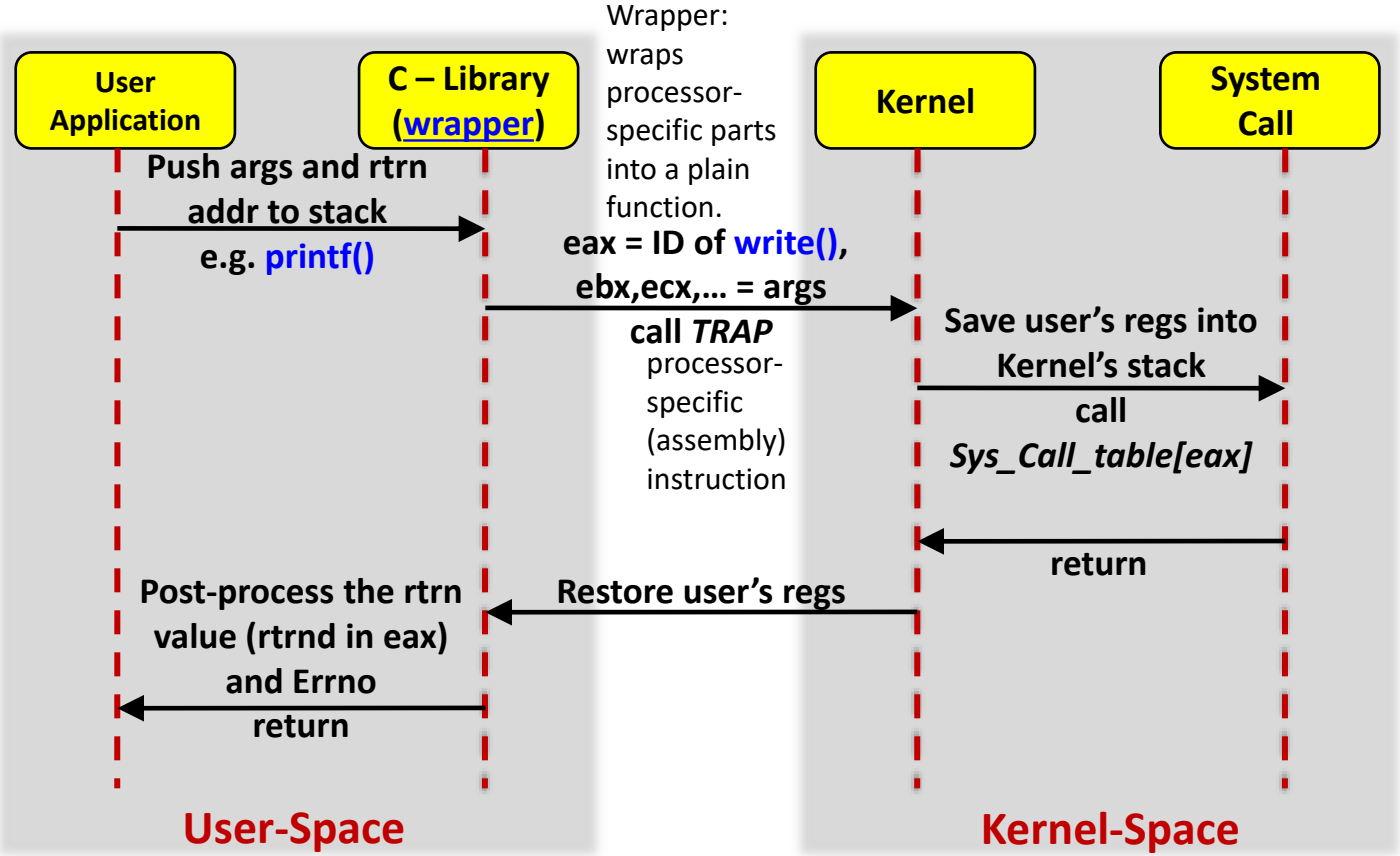
- What is a System Call?
  - An ***interface*** between a ***user application*** and a ***service*** provided by the operating system (or kernel).
- System call interface – see next slide
  - More info

System Calls - Categories

- System calls can be roughly grouped into ***five*** major categories:
  - Process control (e.g. create/terminate process)
  - File management (e.g. read, write)
  - Device management (e.g. logically attach a device)
  - Information maintenance (e.g. set time or date)
  - Communications (e.g. send messages)

System Calls - Interface

- Calls are usually made with C/C++ library functions



eax – a special register, which holds the # of the syscall.

System calls in Linux are done through int 0x80. The system function number is passed in eax, and arguments are passed through registers, not the stack. There can be up to six arguments in ebx, ecx, edx, esi, edi, ebp consequently.

Outline

- Motivation & basics
- **Process control**
  - **Creating a new process**
  - Waiting for a process
  - Running a script / command
  - Error report
- File management
- Concluding example

Contents

תהליכים

תהליך (process)
הוא אבסטרקציה
של תכנית, פעולות
שהתכנית מבצעת
לפי הסדר.

כמו תאומים
בן המלך והעני
מילים ולחן: ירון כפכפי

איזה פלא!
להיות דומים כאלה.
רק החלפנו את הבגדים
אתה עכשיו אני,
והנסיך עני,
כמו תאומים זהים, אחים בדם
זה מדהים איך הבגדים עושים את האדם.

אם נקביל קובץ תוכנית למדים של חיילים

אזי כל חייל הלובש מדים הוא תהליך בפעולה

מושגי יסוד בתהליכים

- **Multiprogramming**: ישנם מספר תהליכים שצריכים להתבצע.
- **Scheduling**: איזה תהליך ירוץ על המעבד ומתי.
- **Context**: המערכת שומרת לכל תהליך **הקשר** משלו כדי שתוכל להריץ ולהחליף בין מספר תהליכים במקביל
  - הסביבה של התהליך (הזכרון שלו, זהות והרשאות, מצב הרגיסטרים עבור התהליך) שבה הוא חי.

הקשר של תהליך

- **PID** – process identifier, מספר יחודי שמתקבל בזמן יצירת התהליך.
- **UID** – user identifier, מספר יחודי של המשתמש שמריץ את התהליך.
- **PPID** – parent PID, המזהה של התהליך שיצר את התהליך הנוכחי.

הקשר של תהליך

- **Mode** – מצב התהליך (מוכן, ממתין) בהמשך...
- **Priority** – מספר המסמל את עדיפות הרצת התהליך, הרחבה בהמשך.
- מערכת ההפעלה שומרת לכל תהליך אזור הנקרא **U-area** המכיל מידע הספציפי לתהליך. למשל: קבצים פתוחים,הקוד של התהליך, הזכרון של התהליך.

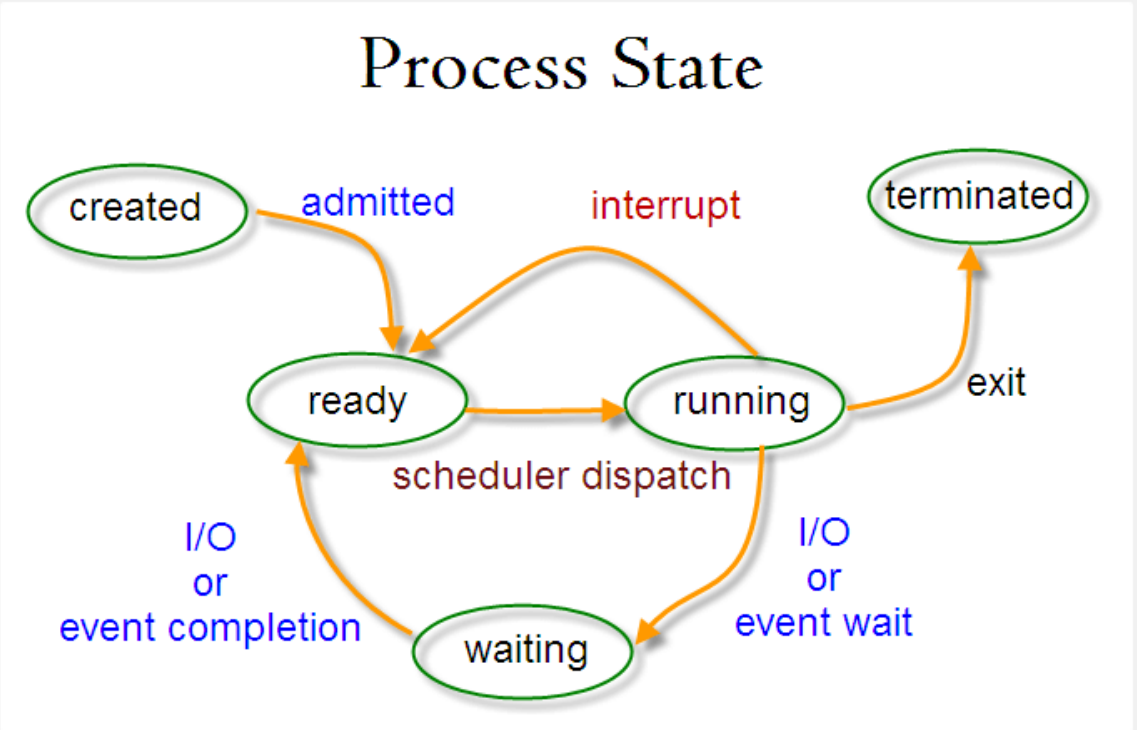
איך תהליך נוצר

- כאשר מערכת ההפעלה נטענת הגרעין יוצר 2 תהליכים:
  - Swapper: pid=0, אחראי על ניהול זכרון
  - Init: pid=1, האב של כל התהליכים, כל התהליכים צאצאים שלו.
- **בלינוקס כדי לייצר תהליך משתמשים בפקודה fork()**
- התהליך שנוצר הוא העתק של התהליך שיצר אותו (האב).

fork () and Copy On Write: motivation

- When fork is invoked, the parent's information should be copied to its child
- May be wasteful if the child will not need all this information
  - To avoid such situations, use ***Copy On Write*** (COW).

Process States



Linux Process State

פונק' fork (1)

- קריאת המערכת fork()
  - תחביר:

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork();
```

  - פעולה: מעתיקה את תהליך האב לתהליך הבן וחוזרת בשני התהליכים
    - קוד זהה (**ומיקום בקוד זהה בשני התהליכים**)
    - זיכרון זהה (משתנים וערכיהם, חוצצים)
    - סביבה זהה (קבצים פתוחים, file descriptors, ספרית עבודה נוכחית)
  - פרמטרים: אין
  - ערך מוחזר:
    - במקרה של כישלון: (-1) לאב (מינוס אחד)
    - במקרה של הצלחה:
      - לבן מוחזר 0
      - ולאב מוחזר ה-pid של הבן

פונק' fork (2)

- לאחר פעולת fork() מוצלחת, אמנם יש לאב ולבן את אותם משתנים בזיכרון, אך <u>בעותקים נפרדים (פרט לקבצים)</u>
  - כלומר, שינוי ערכי המשתנים אצל האב לא ייראה אצל הבן, וההיפך
- כמו כן, תהליך הבן הוא תהליך נפרד מתהליך האב לכל דבר. בפרט, יש לו pid משלו
- מה יכול להדפיס הקוד הבא?

```
main() {
fork();
printf("hello");
}
```

פונק' fork (3)

- תשובות אפשריות (בהנחה ש-()fork הצליחה):

אפשרות ראשונה hellohello

אפשרות שניה hheellollo

אפשרויות נוספת .....

- הסיבה: שני תהליכים כותבים פלט בצורה לא מתואמת.
- מסיבה זו, נוסף תקן הקובע כי פונקציות מסוימות, וביניהן printf, "ינעלו" את הגישה לפלט בזמן שהן מדפיסות כדי למנוע ערבוב. לאחר שprintf מסיימת להדפיס, היא משחררת את הנעילה. פעולה כזו נחשבת "אטומית".
- לא ניתן לדעת בוודאות האם מערכת מסויימת עומדת בתקן או לא, לכן יש להניח כי אם לא נדאג לכך, פלטים יכולים "להתערבב"

Question

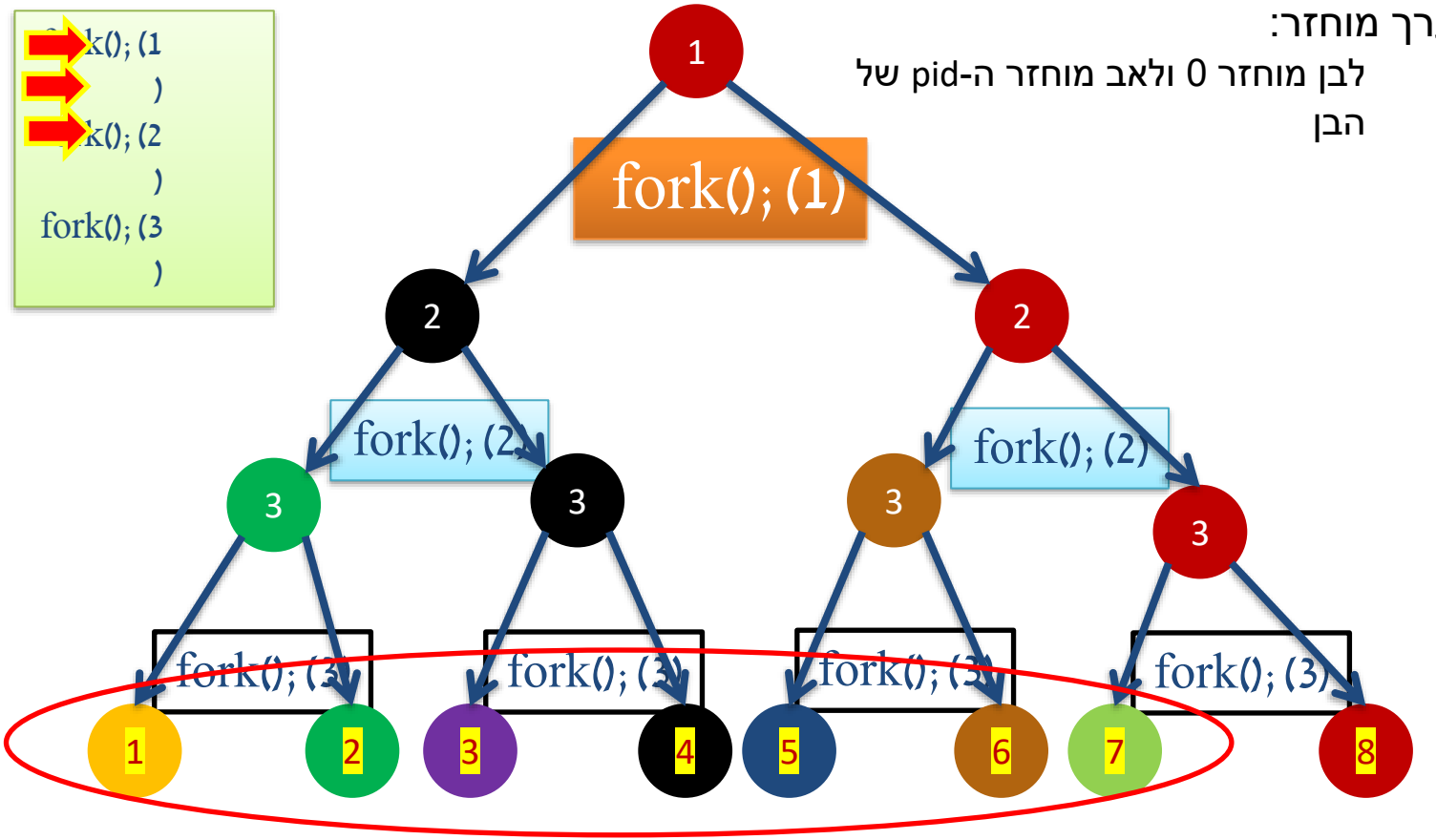
- A process execute the code :

```
fork(); (1)
fork(); (2)
fork(); (3)
```

What will be the total number of **child** processes created ?

Answer

ערך מוחזר:
לבן מוחזר 0 ולאב מוחזר ה-pid של הבן

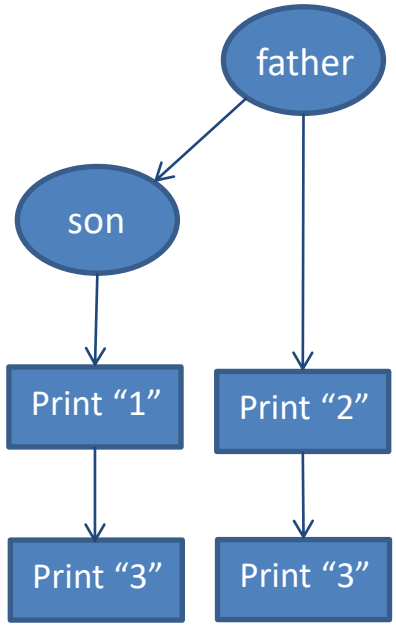


קוד ראשי

דוגמאות (1)

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

void main()
{
        pid_t pid;
        if (  (pid = fork()) == 0)
                printf("1");
        else
                printf("2");
        printf("3");
}
```



מה סדר ההדפסה?

- Output: 2133 or 1233 or 2313 or 1323 or 23 (3312 is not possible)
- גם אם printf אטומית, לא ניתן לשלוט בסדר בו תהליכים יתבצעו.

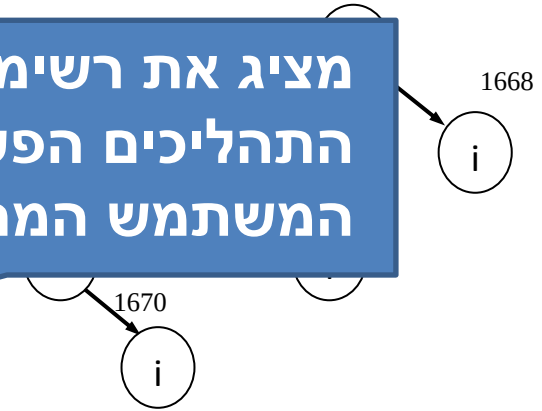
יתומים ואימוץ

- כאשר אב מסיים את פעולתו והבן עדיין חי אז הבן נחשב orphan.
- התהליך init מאמץ את כל היתומים.

דוגמאות לזיהוי תהליכים בזמן ריצה(2)

```
int main(){
        int i;
        for (i = 0; i < 3; i++)
                if (fork() == 0)
                  while(1);
}
gcc -o example2 2_6.c
./example2
ps
```

מציג את רשימת התהליכים הפעילים ע"י המשתמש המחובר




```
UID   PID  PPID   NI STAT TT    TIME COMMAND
...
...
8385  1668   1     20  R  pts/0   0:12  example2
8385  1669   1     20  Z  pts/0   0:14  example2
8385  1670   1     20  S  pts/0   0:13  example2
```

State of a process:
O - Running.
S - Sleeping (process is waiting for an event to complete).
R - Runnable (process is on run queue).
Z - Zombie (process terminated and parent not waiting)
T - Stopped.

kill -KILL 1668 1669 1670

fork

- **pid_t getpid()**
  - **Returns:** process ID of calling process.
- **pid_t getppid()**
  - **Returns:** parent process ID of calling process.
- **uid_t getuid()**
  - **Returns:** user ID of calling process.
- **uid_t getgid()**
  - **Returns:** group ID of calling process.

דוגמא למספור תהליכים עקב fork

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

int main()
{
        pid_t val;
        printf("PID before fork: %d\n",(int)getpid());
           val = fork();
        if(val>0)
                printf("parent PID:
        else if(val ==0)
                printf("child PID: %d\n",(int)getpid());
           else {//if val==-1 print error to screen}
}
```

שיטה להבחנה בין תהליך האבא לתהליך הילד שרק נוצר

דוגמא

- כמה תהליכים ייווצרו לאחר הרצת קטע הקוד הבא? נמקו תשובתכם.

```
pid_t pid;
pid = getpid();
while (fork() == 0) {
    if (pid == getpid())
break;
}
```

Negative Value: creation of a child process was unsuccessful.
Zero: Returned to the newly created child process.
Positive value: Returned to parent or caller. The value contains process ID of newly created child process.

דוגמא

- <u>תשובה</u>
- התהליך המקורי קורא ל- fork(), יוצר בן ולאחר מכן מסיים (כי הערך שחוזר אצלו מה- fork() שונה מאפס).
- תהליך הבן שנוצר נכנס לתוך הלולאה ואצלו לא מתקיים התנאי pid = getpid(), כיוון שמזהה התהליך שלו שונה מהמזהה של תהליך האב. לכן הוא חוזר לראש הלולאה וקורא שוב ל- fork(). כתוצאה מכך הוא יווצר תהליך נכד חדש ותהליך הבן יסתיים (כי שוב הערך שחוזר אצלו מה- fork() שונה מאפס).
- תהליך הנכד נכנס ללולאה ושוב אצלו התנאי pid = getpid() לא מתקיים, לכן הוא יחזור לראש הלולאה, וכך הלאה. בסופו של דבר ייווצרו אינסוף תהליכים.

דוגמא (t3_1.c)

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

int    glob = 6;                    /* global variable */
char   buf[] = "a write to stdout";

int main(void)
{
    int    var;                     /* automatic variable on the stack */
    pid_t  pid;

    var = 88;
    if(puts(buf)==EOF)
        printf("error in writing to stdout");
    printf("before fork\n");

    if ( (pid = fork()) < 0)
        printf("fork error");
    else{
        if (pid == 0)
        {    /* child */
            glob++;      /* modify variables */
            var++;
        }else
        { /* parent */
            sleep(2);
```

```
shani@shani-VirtualBox:~/Dropbox/Operating Systems/Tirgul/T4$ ./a.out
a write to stdout
before fork
pid = 2802, glob = 7, var = 89
pid = 2801, glob = 6, var = 88
```