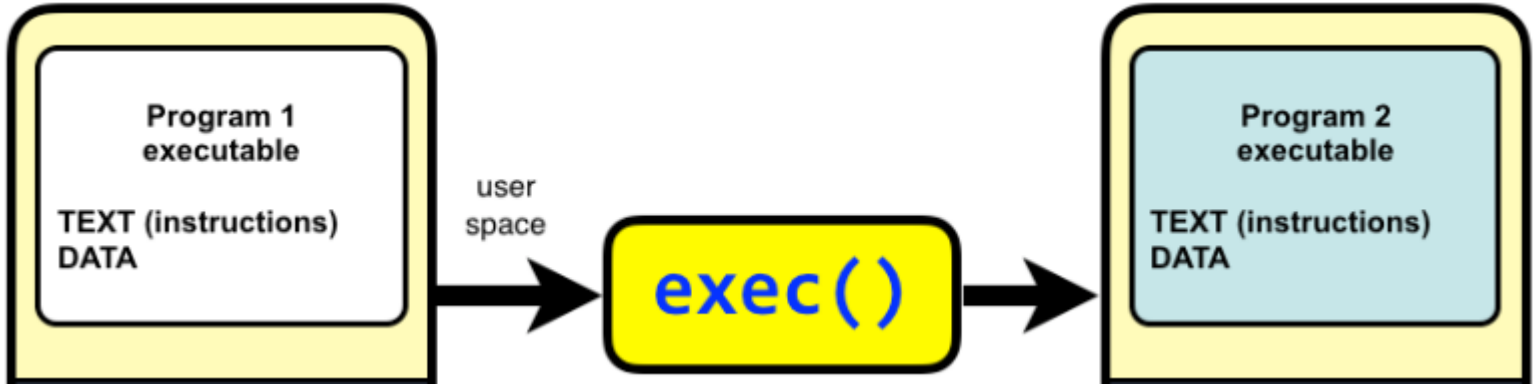
exec() functions + pipes

exec()

- **exec -** ב Unix ניתן להפעיל תוכנית אשר הקוד שלה הוא לא קוד של התוכנית שרצה כרגע אלא נמצא בקובץ אחר.



- **לדוגמא:** ראינו במעבדות קודמות את פקודת המערכת ls.

איך נוכל להפעיל אותה מהתוכנית שאנו כותבים?

exec() functions

int execl(const char ****path*, const char ****arg***, ...);**

int execlp(const char ****file*, const char ****arg***, ...);**

int execv(const char ****path*, char *const** *argv***[]);**

int execvp(const char ****file*, char *const** *argv***[]);**

- כל אחת מהפונקציות הנל מקבלת מסלול לקובץ המכיל תכנית לטעינה ורשימת פרמטרים עבור התכנית.

- לשימוש בפונקציות אלו יש להוסיף #include<unistd.h>

exec() functions

- **path** - נתיב מלא בו מאוחסנת התכנית החיצונית שנרצה להפעיל.

- **file** - שם התכנית החיצונית אותה נרצה להפעיל.

- **[]argv / *args** - מערך ארגומנטים/מצביע לרשימת ארגומנטים שהתכנית החיצונית שאותה נרצה להפעיל מקבלת.
  1. האיבר הראשון ברשימה/מערך ארגומנטים יכיל את שם התכנית.
  2. האיבר האחרון ברשימה/מערך ארגומנטים יכיל את הערך NULL .

- במקרה של הצלחה – התכנית החיצונית מופעלת , בכישלון , מוחזר ערך -1.

int execlp(const char **file*, const char **arg*, ...);

- דוגמא – מה הפלט?

```
#include<stdio.h>
#include<unistd.h>
int main(){
if(execlp("ls" , "ls" , "-l" ,NULL) == -1){
   printf("error");
}
return 0;
}
```

int execlp(const char **file*, const char **arg*, ...);

```
sysadmin@localhost:~$ ./a.out
total 48
drwxr-xr-x 2 sysadmin sysadmin 4096 Apr 24  2019 Desktop
drwxr-xr-x 4 sysadmin sysadmin 4096 Apr 24  2019 Documents
drwxr-xr-x 2 sysadmin sysadmin 4096 Apr 24  2019 Downloads
drwxr-xr-x 2 sysadmin sysadmin 4096 Apr 24  2019 Music
drwxr-xr-x 2 sysadmin sysadmin 4096 Apr 24  2019 Pictures
drwxr-xr-x 2 sysadmin sysadmin 4096 Apr 24  2019 Public
drwxr-xr-x 2 sysadmin sysadmin 4096 Apr 24  2019 Templates
drwxr-xr-x 2 sysadmin sysadmin 4096 Apr 24  2019 Videos
-rwxrwxr-x 1 sysadmin sysadmin 8344 Apr  2 07:40 a.out
-rw-rw-r-- 1 sysadmin sysadmin  127 Apr  2 07:40 lab1.c
```

- פלט – הרצנו תכנית חיצונית בשם "ls" והעברנו לתכנית דגל "l-"

הפונקציה הסתיימה בהצלחה לכן קיבלנו פלט – רשימת כל הקבצים בתיקייה נוכחית.

#תרגיל – יש להפעיל את התכנית "pwd" מתוך הקוד בעזרת פונקציה execlp.

int execv(const char **path*, char *const *argv*[]);

- דוגמא – מה הפלט ?

```
#include<stdio.h>
#include<unistd.h>
int main(){
char *argv[] = {"date" , NULL};
execv("/bin/date", argv);
printf("hello");
return 0;
}
```

תרגיל – יש להריץ תכנית חיצונית ls המדפיסה את כל התיקיות והתתי תיקיות בספרייה נוכחית בעזרת פונקציה execv.

נתיב מלא בו יושבת התכנית - /bin/ls

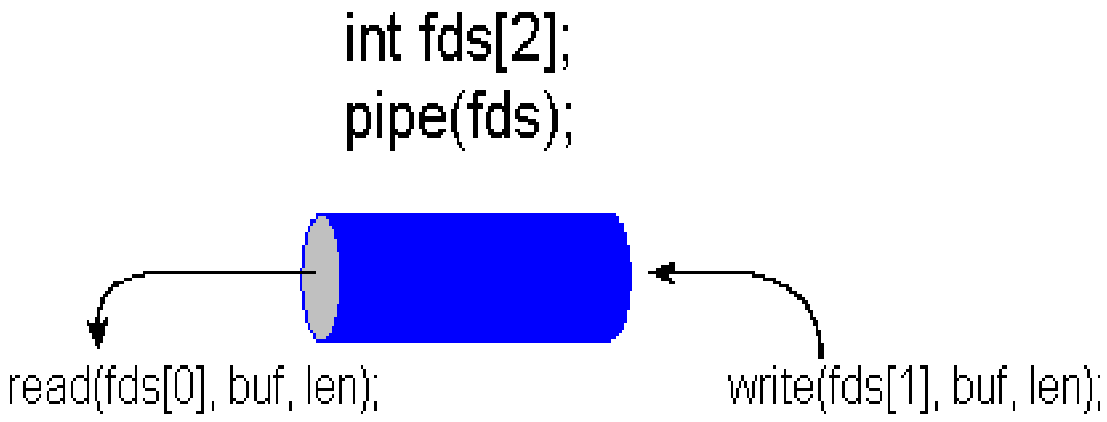
int execl(const char **path*, const char **arg*, ...);

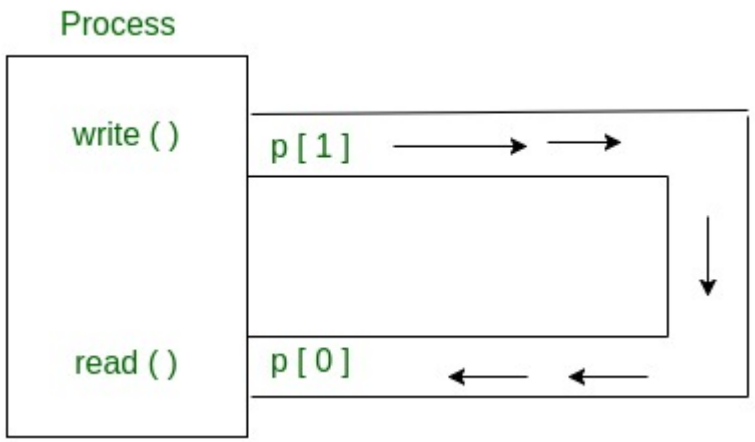
- שאלה ממבחן – מה הפלט ?

```
#include<stdio.h>
#include<unistd.h>
int main(){
int i;
for(i=0;i<3;i++)
{
execl("/bin/echo","echo","hello",NULL);
}
}
```

pipes - תקשורת בין תהליכים

- Pipe –זוג מצביעים אשר עובדים כ "צינור": אם נישלח מידע לקצה אחת אזי המידע יצא מצד השני.
- תקשורת חד כיוונית בלבד, כלומר אנו יכולים להשתמש בצינור כזה שתהליך אחד כותב לצינור, והתהליך השני קורא מהצינור.





Pipes – יצירה ומחיקה

- **int pipe(int fds[2])** - יוצר pipe חדש ומאתחל 2 מצביעים בצורה הבאה:
  - fds[0] מצביע לקריאה
  - fds[1] מצביע לכתיבה

  ערך מוחזר:
  0 אם ה pipe נוצר בהצלחה
  1- אם קרתה שגיאה בזמן יצירת ה pipe

- **int close(int *fds*)** - סוגרת מצביע *fds* ומשחררת כל המשאבים אשר הוקצו בזמן יצירתו.

  ערך מוחזר:
  0 אם ה pipe נסגר בהצלחה
  1- אם קרתה שגיאה בזמן סגירת ה pipe

pipes

- נביט בתכנית הבאה –

כל מה שתוכנית עושה זה יוצרת pipe ולאחר מכן יוצרת תהליך חדש. מהסיבה שבזמן יצירת תהליך חדש כל התוכן של התהליך הנוכחי מועתק, לכן גם המצביעים של ה pipe ישוכפלו.

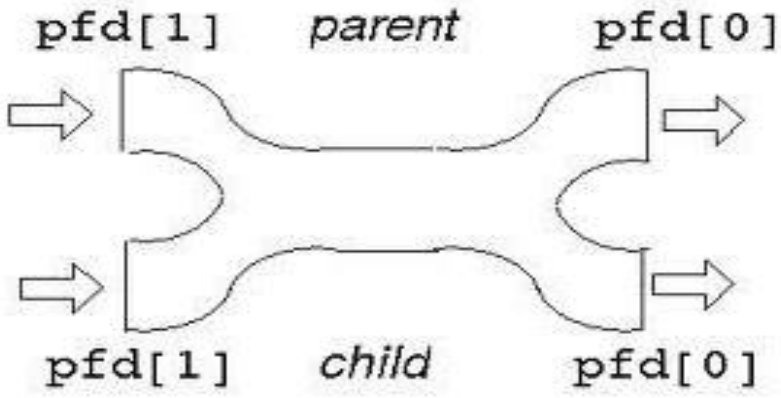
לכל אחד מהתהליכים (תהליך אב ותהליך בן) קיים מצביע משלו עבור קריאה וכתיבה אשר מחוברים בעזרת pipe יחיד.

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

int main(int argc, char *argc[])
{
        int pfd[2];

        pipe(pfd);

        fork();
        //do something
        ..............
        return 0;
}
```



Pipes - קריאה וכתיבה

- **ssize_t write(int *fd*, const void **buf*, size_t *count*)**
- כותבת *count* בתים לתוך מצביע *fd* מהזיכרון המתחיל מכתובת *buf*.
- ערך מוחזר:
  - אם הפעולה הצליחה אזי כמות הבתים אשר נכתבו.
  - במקרה של כישלון יוחזר 1-.

- **ssize_t read(int *fd*, const void **buf*, size_t *count*)**
- מנסה לקרוא *count* בתים מהמצביע *fd* לתוך זיכרון המתחיל ב *buf*. במידה ואין נתונים במצביע ברגע הנתון, הפעולה תמתין עד לרגע בו מישהו יכתוב לשם.
- ערך מוחזר:
  - כמות הבתים אשר נקראו, במידת ההצלחה.
  - במקרה של כישלון יוחזר 1-.

Pipes - דוגמא קוד + פלט

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main()
{
    pid_t pid;
    int myPipeFD[2];
    int ret;
    char buf[40];

    ret = pipe(myPipeFD); //יצירת ציור
    if(ret == -1){        //  בדיקה האם הצליח/נכשל
        perror("pipe error");
        exit(1);
    }

    pid=fork();  //יצירת תהליך בן
    if(pid == 0){
        printf("child process\n");
        write(myPipeFD[1] , "hello , the are someone?!\n" , 30); // בן כותב הודעה לתוך הציור
    }
    else{ // תהליך אב
        printf("parent process\n");
        read(myPipeFD[0] , buf , 30); //תהליך אבא קורא הודעה מתוך הציור
         printf("i am read message from buffer: %s\n" , buf); //ומדפיס אותה
    }
    return 0;
}
```

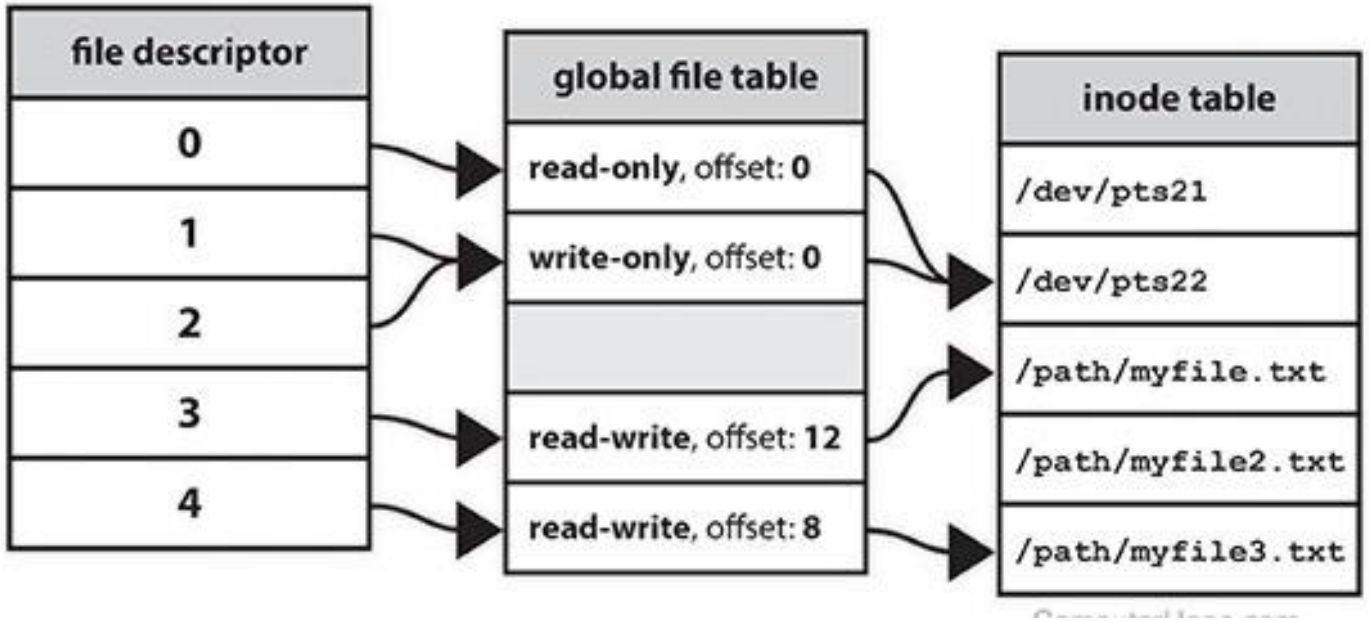
```
sysadmin@localhost:~$ ./a.out
parent process
child process
i am read message from buffer: hello , the are someone?!
```

pipes - תקשורת דו כיוונית

```
#define READ  0
#define WRITE 1
int main(int argc, char *argc[])
{
    int pid;
    int f_2_s[2]; //father to son
    int s_2_f[2]; // son to father
    pipe(f_2_s); // create pipe to communicate father to son
    pipe(s_2_f); // create pipe to communicate son to father
    switch(fork())  {
        case -1:
            printf("Unexpected error occured during creating a process\n");
            return 0;
        case 0:  //בן
            close(f_2_s[WRITE]); // נחסם - האופציה לכתוב לציור אבא - בשביל הבן.
            read(f_2_s[READ], &pid, sizeof(int)); // קריאה מהציור של האבא
            printf("Received message %s from parent\n", pid == getppid() ? "" : "not ");
            pid = getpid();
            close(s_2_f[READ]); // יכול רק לכתוב אליו , נחסם - אופציית קריאה מציור בן
            write(s_2_f[WRITE], &pid, sizeof(int));
            break;
        default: //אבא
            pid = getpid();
            close(f_2_s[READ]); // נחסם.יכול רק לכתוב לשם - אופציית קריאה מציור אבא
            write(f_2_s[WRITE], &pid, sizeof(int));
            close(s_2_f[WRITE]); // נחסם.יכול רק לקרוא ממנו - אופציית כתיבה לציור של הבן
            read(f_2_s[READ], &pid, sizeof(int));
            printf("Received message from process with pid=%d\n", pid);
            break;
    }
    return 0;
}
```

DUP

File descriptor table •

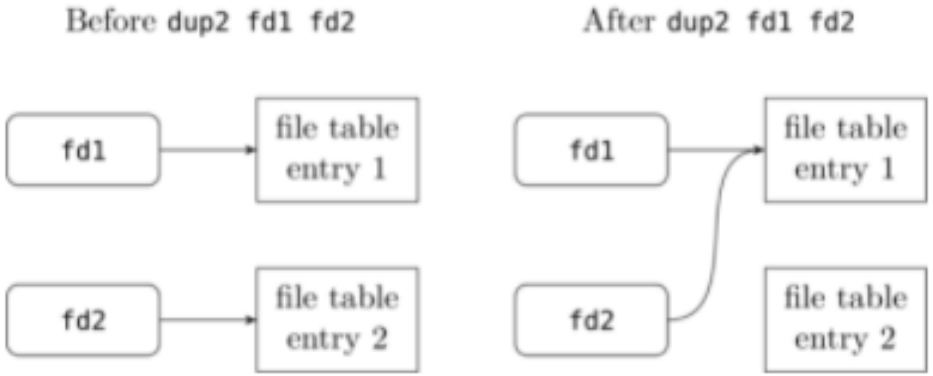


DUP

- **int dup2(int** *oldfd*, **int** *newfd***);** מעתיקה את המצביע מ *oldfd* ל *newfd*.

בעזרת פונקציה הנתונה נוכל להחליף את המצביעי קלט\פלט סטנדרטיים למקום אחר.

- stdin הקלט הסטנדרטי מספרו 0.
- stdout הפלט הסטנדרתי מספרו 1.
- stderr נתיב שגיאות מספרו 2.



DUP

- דוגמא -

```c
#include <unistd.h>
#include <stdio.h>
#define READ  0
#define WRITE 1

int main(int argc, char *argv[])
{
    int p[2];

    pipe(p);

    if (fork() == 0)
    {
        close(p[READ]);
        dup2(p[WRITE], WRITE); //dup
        execlp("ls", "ls", "-l", NULL);
    }

    if (fork() == 0)
    {
        close(p[WRITE]);
        dup2(p[READ], READ); //dup
        execlp("sort", "sort", "-r", NULL);
    }

    return 0;
}
```

התוכנית הנתונה מבצעת את הפקודה: ls –l | sort –r אשר ראינו במעבדות ראשונות את המשמעות שלה.

בדוגמא נוכחית ניתן לראות שלתהליך אחד אנו מחליפים פלט סטנדרטי אל תוך ה pipe כאשר WRITE במקרה זה מצביע לפלט סטנדרטי. בצורה דומה ניתן לראות שלתהליך שני אנו מחליפים את הקלט הסטנדרטי , גם כאן ה READ הוא מצביע לקלט סטנדרטי.

מטלה

- תרגיל –

יש לכתוב תכנית בה תהליך אבא יוצר תהליך בן.

תהליך אבא שולח לתהליך בן סיסמא סודית "12345" דרך צינור.

הבן קורא מהצינור ומבצע קריאה לתכנית חיצונית ECHO בשביל להדפיס את הסיסמא הסודית.