מעבדה 11 – מערכת הקבצים – שימוש בקריאות מערכת dup ו-pipe

שיטת העברת מידע ותאום הנפוץ ביותר ב-unix הוא העברת מידע דרך צינור pipe ומוכר לכל משתמש שמבצע פקודות כמו

```
ls –l | grep drwx
ls –l | more
```

לפני כן נקדים לתאר עוד קריאות מערכת שאנחנו צריכים.

לכל תהליך של תכנית C יש מספר כזה או אחר של קבצים/ ערוצי קלט/פלט שהם רשאים לפתוח בו זמנית. המוסכמה היא שעם תחילת ריצה של תכנית, ערוץ 0 מנותב ל-standard input (המקלדת בדרך כלל), ערוץ 1 מנותב ל-standard output (המסך בדרך כלל), ערוץ 2 מנותב ל-standard error (המסך בדרך כלל). אחד מאמצעי הגמישות של קוד unix הוא אפשרות לשנות את הניתוב הזה ללא שינוי הקוד של התוכנית עצמה.

למשל, פקודות נוסך

```
ls >filename.txt
ls >> filename.txt
sort < file1.txt > file2.txt
ls –l | more
ls | sort | more
```

או

```
time > filename.txt
```

ברמה הזו קוד הקלט/פלט (כלומר הפקודות שמשתמשים בהם) הינו כמעט בלתי תלוי באופי הערוץ עצמו. במלים אחרות הכתיבה למסך או לקובץ או להתקן אחר היא אותו דבר או כמעט אותו דבר.

קבצים או ערוצים מסוג אחר נפתחים בדרך כלל ע"י הפקודה open.

קריאה וכתיבה נעשית ע"י קריאות מערכת המעבירות רצפים של בתים באופן גולמי (ללא עיבוד) read ו-write לפי מספרי ערוצים.

לדוגמאות הבאות נכתוב ונקמפייל את תכנית Hello.c

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

void main(){
        printf("Hello World!\n");
}

>>cc Hello.c –o Hello
```

קריאת המערכת dup

קריאת המערכת dup מקבלת מספר סידורי של ערוץ ו"משכפלת אותו", כלומר מקצה מספר סידורי נוסף לאותו ערוץ.
התכונה המרכזית של הקריאה - מחויבות להשתמש במספר הערוץ הפנוי הקטן ביותר כאשר היא נענית לבקשה. במקרה של כישלון היא מחזירה 1-.

ההכרזה של הקריאה היא

int dup(int oldfd);

הדרך לנתב קלט/פלט של תוכניות, כמו שדיברנו לעיל, היא לפתוח את הקובץ ב-open, לסגור את הערוץ הרצוי (0,1,2) ולשכפל אותו ב-dup, מתוך הנחה שהקריאה תשתמש במספר הערוץ הרצוי לנו. אפשר לבצע את התהליך על כל שלושת הערוצים הסטנדרטיים.

דוגמא 1:

```
/* redirect1.c */
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

void fatal(char str[]){ fprintf(stderr, "%s\n", str);  exit(1);} // fatal

void sys_err(char str[]){ perror(str); exit(2);} // sys_err

int main(){

  if (fork() == 0) { /* Child */
        int fd1, fd2;

        if ( (fd1 = open("hello.txt", O_WRONLY | O_CREAT |O_TRUNC, 0644)) == -1)
                sys_err("open");
        if( close(1) == -1)   sys_err("close"); //סגירת ערוץ הפלט הסטנדרטי

        fd2 = dup(fd1);  // ניתוב פלט סטנדרטי (1) לקובץ
        if ( fd2 == -1 )  sys_err("dup");
        else
                if (fd2 != 1)      fatal("Unexpected dup result");

        execl("./hello", "hello", 0); // כרגע הפלט יופיע בקובץ במקום על מסך
   } /* if  fork*/

   printf("Parent terminating\n");
} /* main */
```

קריאת המערכת dup2

למעשה ישנה פקודת מערכת חדשה יחסית dup2 שמאפשרת לנו לבקש את מה שאנחנו רוצים ישירות.
ההכרזה על dup2 הינה:

```
int dup2(int oldfd, int newfd);
```

כאשר newfd זה מספר הערוץ שאנחנו רוצים עבור הכפיל. תשובת הפונקציה היא גם מספר הערוץ החדש. dup2 גם סוגרת את הערוץ אם יש צורך בכך, וכך נחסוך לנו הקריאה ל-close. הסיבה היחידה שלא להשתמש בהקשר הזה ב-dup2 היא שיש גרסאות, בעיקר ישנות, של unix שלא תומכות בה.

דוגמא 2:

```
/* redirect2.c */

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

void sys_err(char str[]){   perror(str);   exit(2);}

int main(){
        if (fork() == 0) {
                int fd1, fd2;

                if ( (fd1 = open("hello.txt", O_WRONLY | O_CREAT |O_TRUNC, 0644)) == -1)
                        sys_err("open");

                fd2 = dup2(fd1, 1);
                if ( fd2 == -1 )      sys_err("dup");

                execl("./hello", "hello", 0);

        } /* if fork */

        printf("Parent terminating\n");}
```

העברת מידע דרך צינור pipe

צינור הינו מנגנון העברת מידע מסונכרן בין 2 תהליכים דרך חוצץ בזיכרון. גודל החוצץ אינו סטנדרטי ותלוי, בדרך כלל, במערכת הפעלה ספציפית. בשביל להשתמש במנגנון הזה התהליכים חייבים להיות בעלי אב קדמון משותף (או אב ובן). לצינור יש קצה קריאה וקצה כתיבה. ניסיון לקרוא מצינור ריק עוצר את התהליך הקורא וגורם לו להמתין עד שיהיה לו מה לקרוא. ניסיון לכתוב לתוך צינור מלא גורם לתהליך הכותב להיעצר להמתנה עד שיתפנה מקום. ברגע שנתון בצינור נקרא, הוא "מתבטל".

צינור הוא משאב שמבקשים אותו בקריאת המערכת pipe. קריאת המערכת הזו מחזירה 2 מספרים שלמים המשמשים מספרי ערוץ לקריאה ולכתיבה. מספרי הערוצים הללו משמשים לכתיבה וקריאה ע"י קריאות המערכת write ו-read בהתאמה. ניתן גם לסגור את כל אחד מהקצוות ע"י קריאת המערכת close. במידה ואנחנו רוצים לנתב את הערוצים הסטנדרטיים לקצוות של צינור, הדבר אפשרי ונעשה תוך שימוש בקריאות המערכת dup או dup2.
צינור יחיד יהיה בדרך כלל להעברת מידע רק בכוון אחד ברגע נתון. כאשר משתמשים בצינור לעיתים קרובות שני התהליכים, משיקולי זהירות סוגר כל תהליך את אחד הקצוות, למרות שזה לא חובה. יצירת הצינור חייבת להיעשות ע"י האב הקדמון לפני היווצרות התהליכים המשתתפים (למעט האב הקדמון עצמו). המנגנון מתבסס על כך ש-fork משכפל את הניתוב של הערוצים ו-execl אינו משנה זאת.

קריאת המערכת pipe

קריאת המערכת pipe מבקשת הקצאה של צינור ומנתבת 2 מספרי ערוצים - אחד כקצה כתיבה והשני כקצה קריאה.

ההכרזה של pipe :

```
int pipe(int filedes[2]);
```

קריאת המערכת pipe מקבלת כפרמטר מערך של שני שלמים ומחזירה בכניסה הראשונה filedes[0] את ערוץ הקריאה מהצינור וב-filedes[1] את ערוץ הכתיבה. תשובת הפונקציה של pipe היא 0 במקרה של הצלחה ו-1- במקרה של כישלון.

בתור דוגמא נראה תוכנית המעבירה את המחרוזת "Hello World!" בית אחר בית דרך צינור, כאשר התוכנית השולחת נרדמת ל-3 שניות לאחר הפיצול (כלומר התוכנית הקוראת בטוח מקדימה אותה) והדבר עובד.

דוגמא 3:

```
/* pipe1.c - send information through pipe.  */
…
void sys_err(char str[]){ perror(str); exit(2);}

void child(int  pfd[]){
        int i;
        char msg[80];
```

הבן סוגר את ערוץ הפלט

```
        if (close(pfd[1]) == -1)
                sys_err("close");

        i = 0;
```

וקורא מערוץ הקלט תו אחרי תו שקודם רשם לשם האב

```
        do {
                if (read(pfd[0], &msg[i], 1) == -1)
                        sys_err("read");
                i++;
        } while (msg[i-1] != '\0');

        printf("Received  = %s\n", msg);
}

int main(){
        int  pfd[2], i;
        char str[] = "Hello World!\n";

        if (pipe(pfd) == -1)  sys_err("pipe");

        printf("pfd[0] = %d, pfd[1] = %d\n", pfd[0], pfd[1]);

        switch(fork()) {
                case -1:
                        Sys_err("fork");
                case 0:
                        child(pfd);
                default:
                        break;
        }

/* parent only */
```

האב סוגר את ערוץ הקלט

```
        if (close(pfd[0]) == -1)  sys_err("close");

        sleep(3);
```

ורושם (פולט) לערוץ הפלט תו אחרי תו

```
        for(i=0; i <= strlen(str); i++)

                if (write(pfd[1], &str[i], 1) == -1)  sys_err("write");

        return 0;
}
```

מהו פלט הרצת התכנית ?

דוגמא 4: התוכנית הבאה מעבירה את ההודעה במכה אחת

```
/* pipe2.c - send information through pipe.  */
….
void sys_err(char str[]){ perror(str); exit(2);}

void child(int  pfd[]){
        char msg[80];

        if (close(pfd[1]) == -1)    sys_err("close");

        if (read(pfd[0], msg, 80) == -1)   sys_err("read");

        printf("Received  = %s\n", msg);
}

void  main(){
        int  pfd[2];
        int i;
        char str[] = "Hello World!\n";

        if (pipe(pfd) == -1)    sys_err("pipe");

        printf("pfd[0] = %d, pfd[1] = %d\n", pfd[0], pfd[1]);

        switch(fork())  {
                case -1:
                        sys_err("fork");
                case 0:
                        child(pfd);
                default:
                        break;
        }

/* parent only */
        if (close(pfd[0]) == -1)

                sys_err("close");

        sleep(3);

        if (write(pfd[1], str, strlen(str)+1) == -1)   sys_err("write");

}
```

<u>דוגמא 5</u>: התוכנית הבאה משתמשת בצינור למידע דו כיווני. שימו לב שהיינו צריכים לסנכרן ע"י sleep את שני התהליכים, בכדי שתהליך הבן לא יקרא את מה שהוא עצמו כתב

```
/* pipe3.c - send information through pipe.  */
...
void sys_err(char str[]){ perror(str); exit(2);}

void child(int  pfd[]){
        char msg[80];
        char str[] = "Good Morning Parent!\n";

        if (read(pfd[0], msg, 80) == -1) sys_err("read");
        printf("Received  = %s\n", msg);

        if (write(pfd[1], str, strlen(str)+1) == -1)  sys_err("write");
}

void  main(){
        int  pfd[2], i;
        char str[] = "Good Morning Child!\n",   msg[80]

        if (pipe(pfd) == -1)    sys_err("pipe");

        printf("pfd[0] = %d, pfd[1] = %d\n", pfd[0], pfd[1]);

        switch(fork())  {
                case -1:
                        sys_err("fork");
                case 0:
                        child(pfd);
                default:
                        break;
        }

/* parent only */

        sleep(3);

        if (write(pfd[1], str, strlen(str)+1) == -1)   sys_err("write");

        sleep(2);

        if (read(pfd[0], msg, 80) == -1) sys_err("read");

        printf("Received  = %s\n", msg);
}
```

מהו פלט הרצת התכנית?

<u>תרגיל 1:</u> נתבונן בתוכנית הבאה:

```
#include <stdio.h>
int main() {
  char string[1] = "c";
  int pipe1[2], pipe2[2];
  int pid;
  pipe(pipe1);
  pipe(pipe2);
  write(pipe1[1], string, sizeof(char));  // (*)
  pid = fork();
  if (pid == 0)
        while (1) {
                        read(pipe1[0], string, sizeof(char));
                        printf("a\n");
                        write(pipe2[1], string, sizeof(char));
        }
  else if (pid > 0)
        while(1)
                {
                        read(pipe2[0], string, sizeof(char));
                        printf("b\n");
                        write(pipe1[1], string, sizeof(char));
                }
        return 0;
}
```

א. מה הפלט של התוכנית? הסבר.
ב. מה יהיה הפלט אם נחליף את pipe1[1] בשורה (*) להיות pipe2[1]? הסבר.
ג. מה יהיה פלט התכנית אם נמחק את השורה (*)? הסבר.
ד. כתוב תוכנית בשפה C הרצה תחת מערכת הפעלה CentOS המבצעת אותו הדבר בדיוק כמו הקוד הנתון אבל ללא שימוש ב-pipe . הדפסות הנעשות בתוכנית הנתונה על ידי תהליכים שונים, יתבצעו גם בתוכנית החדשה שלך על ידי תהליכים שונים וסדר ההדפסות יהיה זהה לסדר ההדפסות בתוכנת הנתונה.
לא נדרש ליצור בתוכנית החדשה שלך את אותו מספר התהליכים כמו בתוכנית הנתונה.

<u>תרגיל 2</u>

פקודת date היא פקודה סטנדרטית המממשת שליחת התאריך והשעה למסך.

פקודת wc היא פקודה סטנדרטית לספירת שורות, מילים ותווים.

תחת מ"ה UNIX כתבו בשפת C תכנית אשר מבצעת את הפעולה הבאה:

```
date | wc
```

הפלט בהרצת התכנית שלכם נדרש להיות בדיוק:

```
In date there are:
Line:   Words:   Chars:
   1      6      29
```

או

```
   1      6      28
```