

מערכות הפעלה תרגול 8

Process communication Files, PIPE and FIFO

מתרגל-יורם סגל
yoramse@colman.ac.il

Contents

1

תקשורת בין תהליכים

2

pipe

3

FIFO

תכולת התירגולים עד כה T1-T5



תקשורת בין תהליכים

❖ אפשר דרך איתותים (signals)

- אבל אם רוצים להעביר מידע? (כאשר שולחים סיגנל ניתן להעביר רק את מספר הסיגנל ואי אפשר להעביר מידע נוסף ...)

❖ אפשר להעביר מידע דרך קבצים








■ בעיות:

- איטי.
- תהליכים שמעבירים ביניהם הרבה מידע, מה גודל הקובץ?
- איך נדע אם תהליך סיים לכתוב וסגר את הקובץ או שעדיין לא כתב? - כלומר בעיות סנכרון

Unix I/O Descriptor Table

המוסכמה היא ש fd -ים 0,1,2 כבר מוקצים:
standard input – 0
standard output – 1
standard error – 2

fd Descriptor Table

0	
1	
2	
3	
	
4	
	

Data structure for file 0



Data structure for file 1



Data structure for file 2



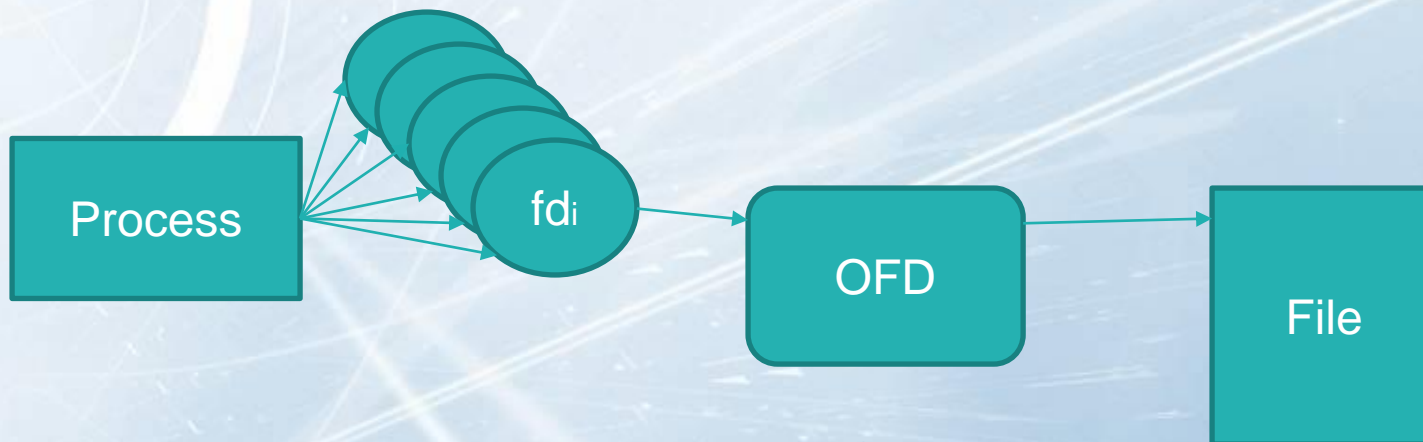
```
fd= open("file.txt")
```


File descriptors

❖ כדי לגשת לקובץ (בלינוקס) משתמשים ב
file descriptors (fd).

❖ לכל תהליך יש רשימה של fd-ים.

❖ ה fd הינו מספר המשויך למבנה (structure)
הנקרא Open File Description (OFD)



Open File Description (OFD)

❖ OFD הינו מבנה המכיל מידע על קובץ.

- Offset: איפה הכניסה הבאה לקובץ תהיה
- האם הקובץ ניתן לקריאה/כתיבה (לפי ההיררכיה).
- ועוד דגלים

יכולים להיות מספר fd-ים (של אותו תהליך)
המצביעים על OFD אחד.

תזכורת קבצים לעניין כתיבה וקריאה

כאשר תהליך מבצע קריאה מקובץ או כתיבה לקובץ המיקום הנוכחי בקובץ זז כמספר הבתים שנכתבו/נקראו.

O_RDWR: Open for reading and writing.

O_CREAT : If the file does not exist it will be created.

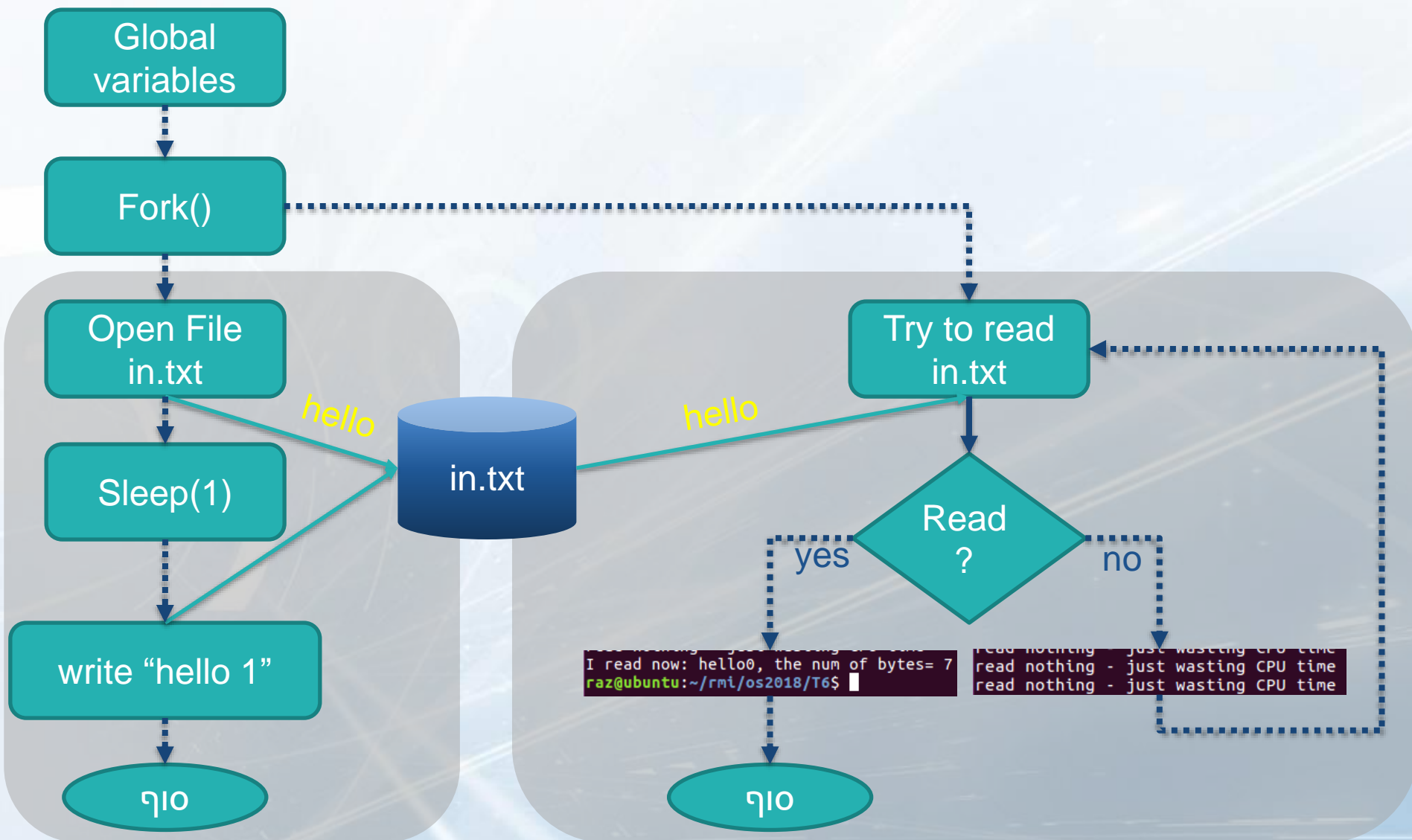
O_TRUNC If the file already exists and is a regular file and the open mode allows writing

(i.e., is O_RDWR or O_WRONLY) it will be truncated to length 0.

O_SYNC makes every write() to the file return only when the contents of the file have been written to disk. This provides the guarantee that when the system call returns the file data is on disk.

O_DSYNC provides a stronger guarantee. **It guarantees not only that the file data has been written to disk but also the file metadata.**

תקשורת בעזרת קבצים – T7_1.c



Random access

❖ כדי להגיע למקום מסויים בקובץ יש להעזר ב `syscall`

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

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

```
off_t lseek(int fd, off_t offset, int whence);
```

❖ *whence* יכול להיות:

- `SEEK_SET` – מתחילת הקובץ
- `SEEK_CUR` – מהמיקום הנוכחי
- `SEEK_END` – מסוף הקובץ

ערך החזרה: 1- במקרה של כשלון, ההיסט מתחילת הקובץ במקרה של הצלחה.

תקשורת בעזרת קבצים

❖ נוכל לשלב איתותים וקריאה\כתיבה מ\לקבצים על מנת להעביר מידע בין תהליכים.

❖ האיתותים נועדו על מנת שתהליך אחד יוכל לסמן לתהליך האחר שהוא סיים את פעולתו.

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

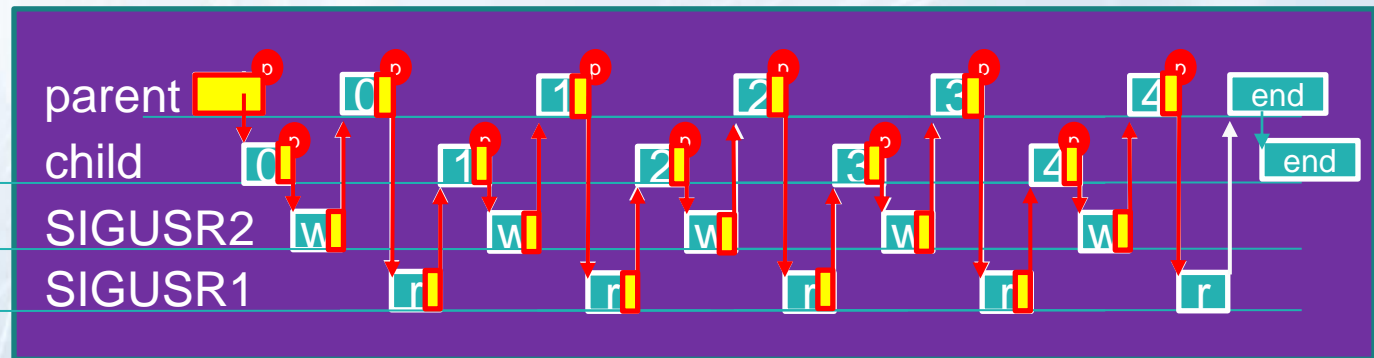
T7_2.c - תקשורת בעזרת קבצים

המטרה לסנכרן בין כתיבה לקובץ של הילד לבין הקריאה מהקובץ של האבא באמצעות :

signal() - הרשמה

pause() – המתנה לסיום טיפול באיתות

kill() – שליחת איתות



parent

child

SIGUSR2

SIGUSR1

end

end

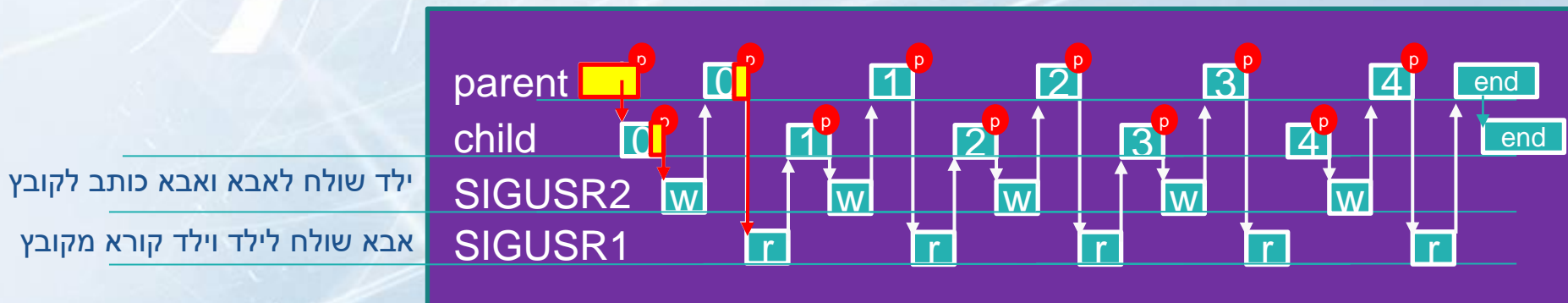
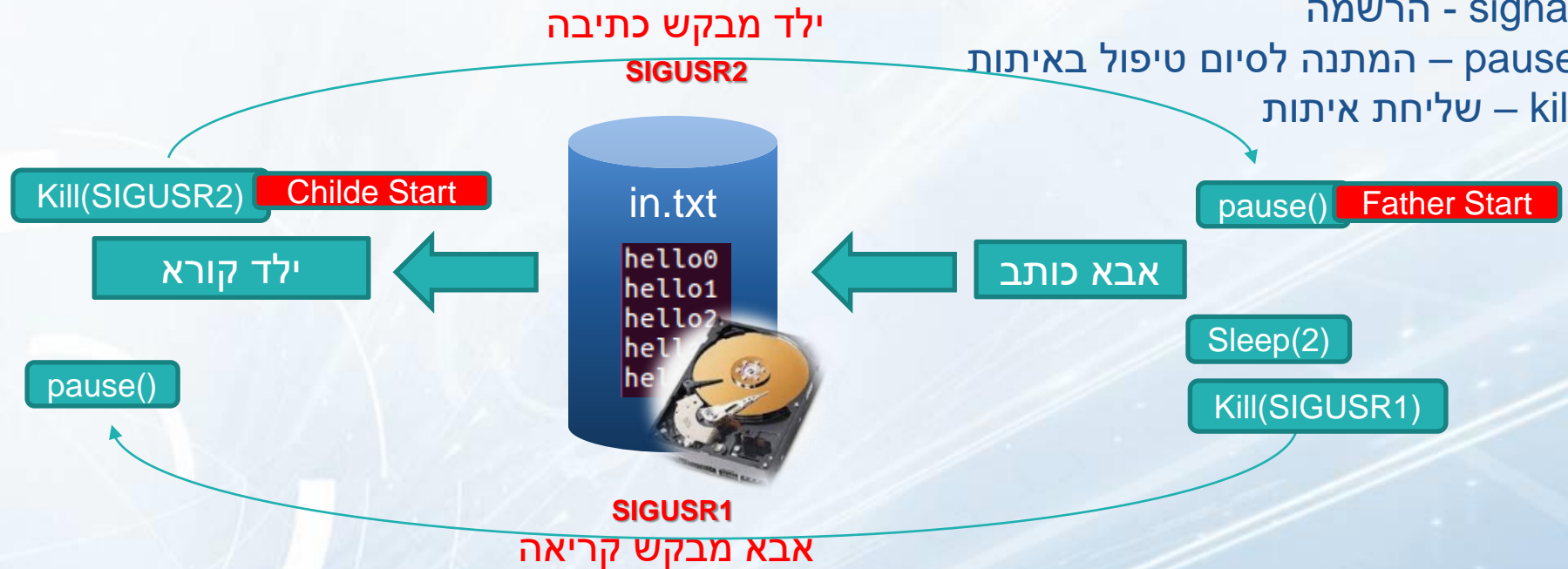
ילד שולח לאבא ואבא כותב לקובץ

אבא שולח לילד וילד קורא מקובץ

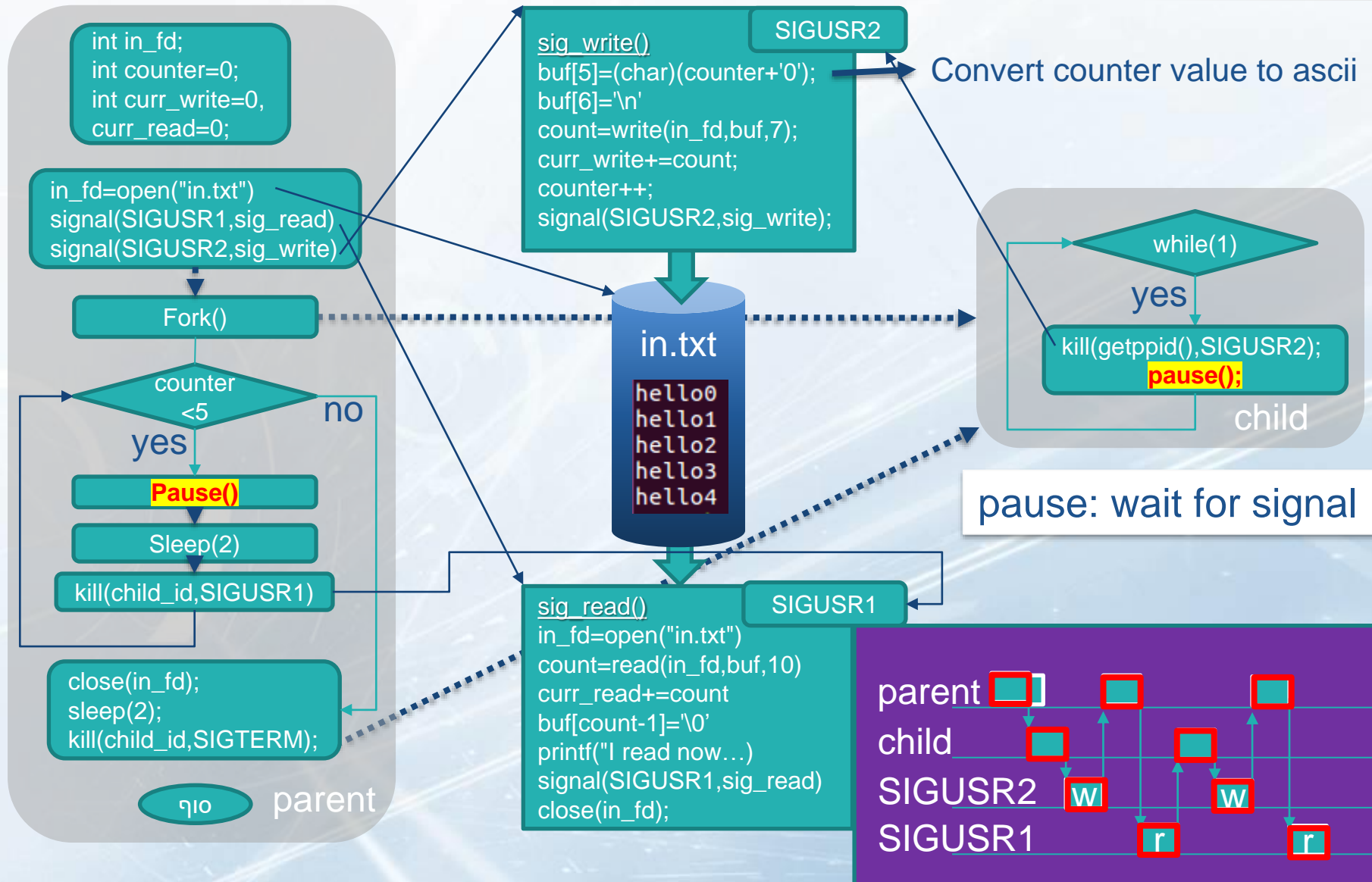
T7_2.c - תקשורת בעזרת קבצים

המטרה לסנכרן בין כתיבה לקובץ של הילד לבין הקריאה מהקובץ של האבא באמצעות :
signal() - הרשמה
ילד מרקיש כתיבה

pause() – המתנה לסיום טיפול באיתות
kill() – שליחת איתות



תקשורת בעזרת קבצים – T7_2.c



pipes

תקשורת בין תהליכים באמצעות קובץ בזיכרון

Write
fd= fields[1]

out



in

Read
fd= fields[0]



pipes

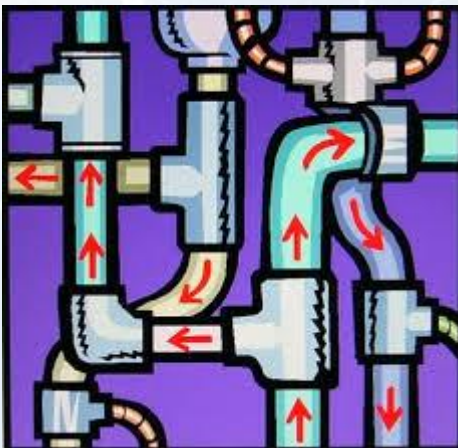
❖ pipes (צינורות) הם ערוצי תקשורת חד-כיווניים המאפשרים העברת נתונים לפי סדר FIFO (First-In-First-Out).

❖ pipes משמשים גם לסנכרון תהליכים, כפי שנראה בהמשך.

❖ המימוש של pipes ב-Linux הוא כאובייקטים של מערכת הקבצים, למרות שאינם צורכים שטח דיסק כלל ואינם מופיעים בהיררכיה של מערכת הקבצים.
❖ הגישה ל-pipe היא באמצעות שני descriptors: אחד לקריאה ואחד לכתיבה.

המשך Pipes

- ❖ היצירה של pipe היא באמצעות קריאת המערכת `pipe()`.
- ❖ ה-`pipe` הנוצר הינו פרטי לתהליך, כלומר אינו נגיש לתהליכים אחרים במערכת.
- ❖ הדרך היחידה לשתף `pipe` בין תהליכים שונים היא באמצעות קשרי משפחה
 - תהליך אב יוצר `pipe` ואחריו יוצר תהליך בן באמצעות `fork()` – לאב ולבן יש גישה ל-`pipe` באמצעות ה-`descriptors` שלו, המצויים בשניהם



- ❖ לאחר סיום השימוש ב-`pipe` מצד כל התהליכים (סגירת כל ה-`descriptors`) מפונים משאבי ה-`pipe` באופן אוטומטי.

יצירת pipe בפועל

❖ `#include <unistd.h>`

❖ `int pipe(int filedes[2]);`

שולחים לפונקציה מערך של 2 int.

הפונקציה מכניסה descriptors למערך fields

fields[0]: ה descriptor לקריאה מה pipe

fields[1]: ה descriptor לכתיבה ל pipe

קריאה מ pipe

❖ קריאה מ-pipe תחזיר:

- את כמות הנתונים המבוקשת אם היא נמצאת ב-pipe.
- **פחות מהכמות** המבוקשת אם זו הכמות הזמינה ב-pipe בזמן הקריאה.
- **0 (EOF)** כאשר כל ה-write descriptors נסגרו וה-pipe ריק.
- **תחסום את התהליך** אם יש כותבים (write descriptors) ל-pipe וה-pipe ריק. כאשר תתבצע כתיבה, יוחזרו הנתונים שנכתבו עד לכמות המבוקשת.

כתיבה ל pipe

❖ כתיבה ל-pipe תבצע:

■ אם יש מספיק מקום:

- אם יש קוראים פתוחים – כתיבת כל הכמות.
- אם אין קוראים פתוחים – כשלון **SIGPIPE**.

■ אם אין מספיק מקום למידע שצריך להכתב:

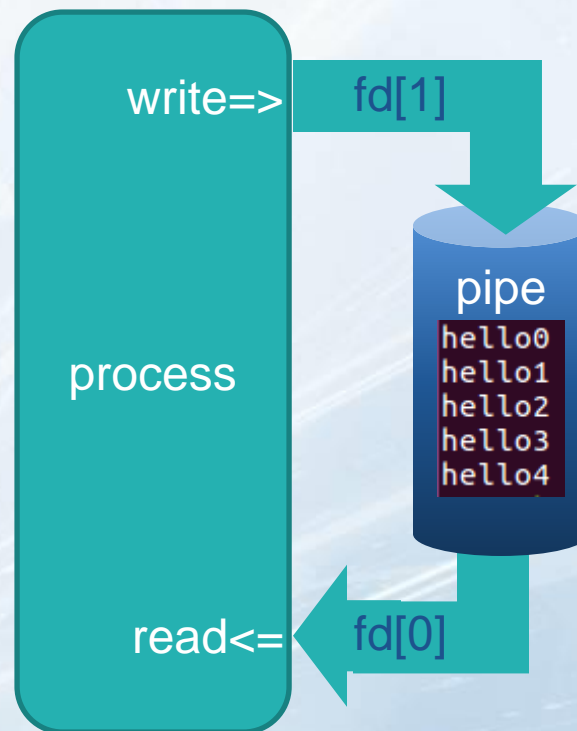
- אם יש קוראים פתוחים – הכתיבה תחסם עד שיתפנה מקום.
- אם אין קוראים פתוחים – כשלון **SIGPIPE**.

❖ כתיבת פחות מ **PIPE_BUF** בתים (בדרך כלל 4096) נעשית בצורה אטומית.

❖ ה-pipe מוגבל בגודלו (4k בגרסאות kernel ישנות יותר ו64k בגרסאות החדשות).

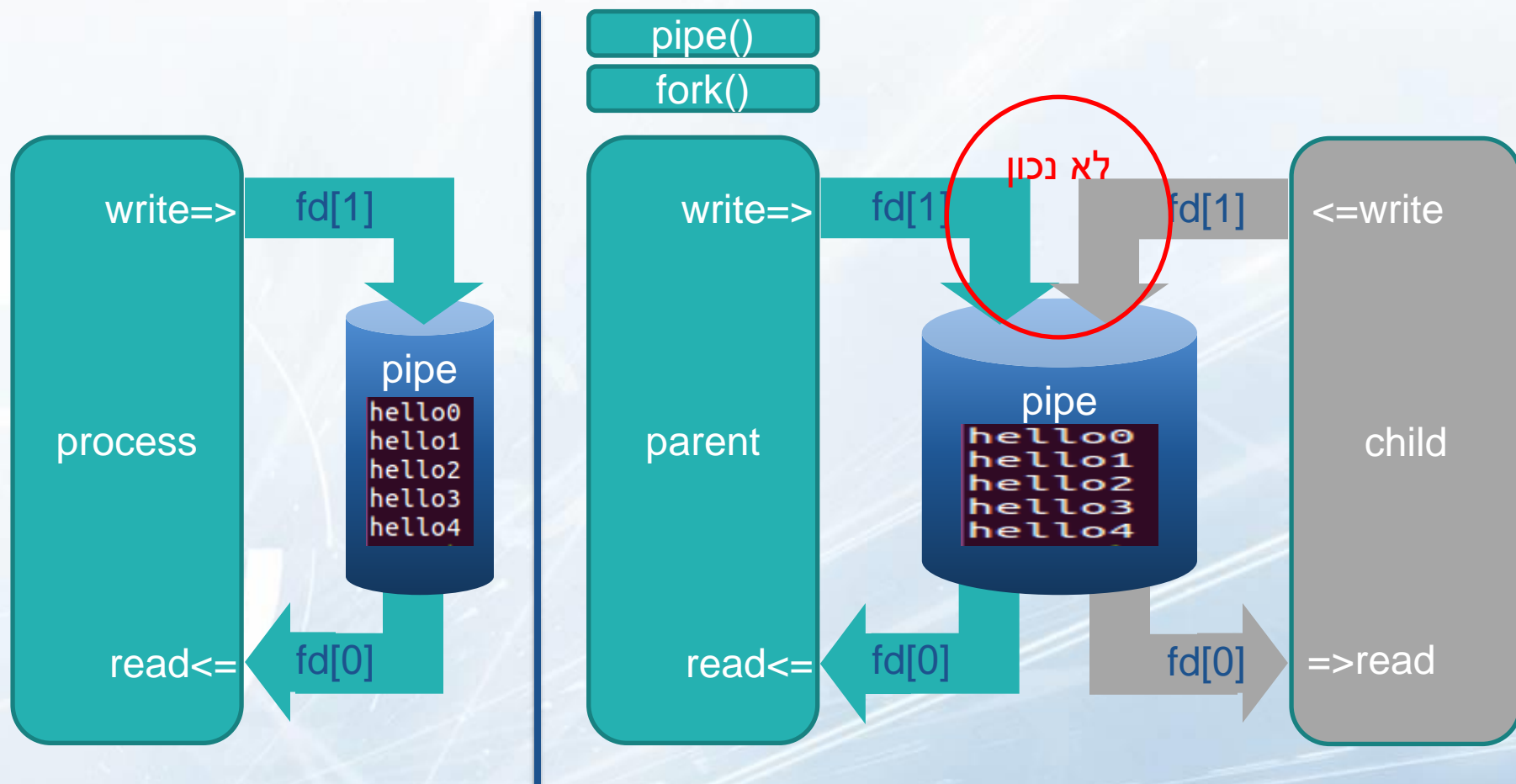
קריאה וכתיבה ל pipe

תוצר פתיחת PIPE



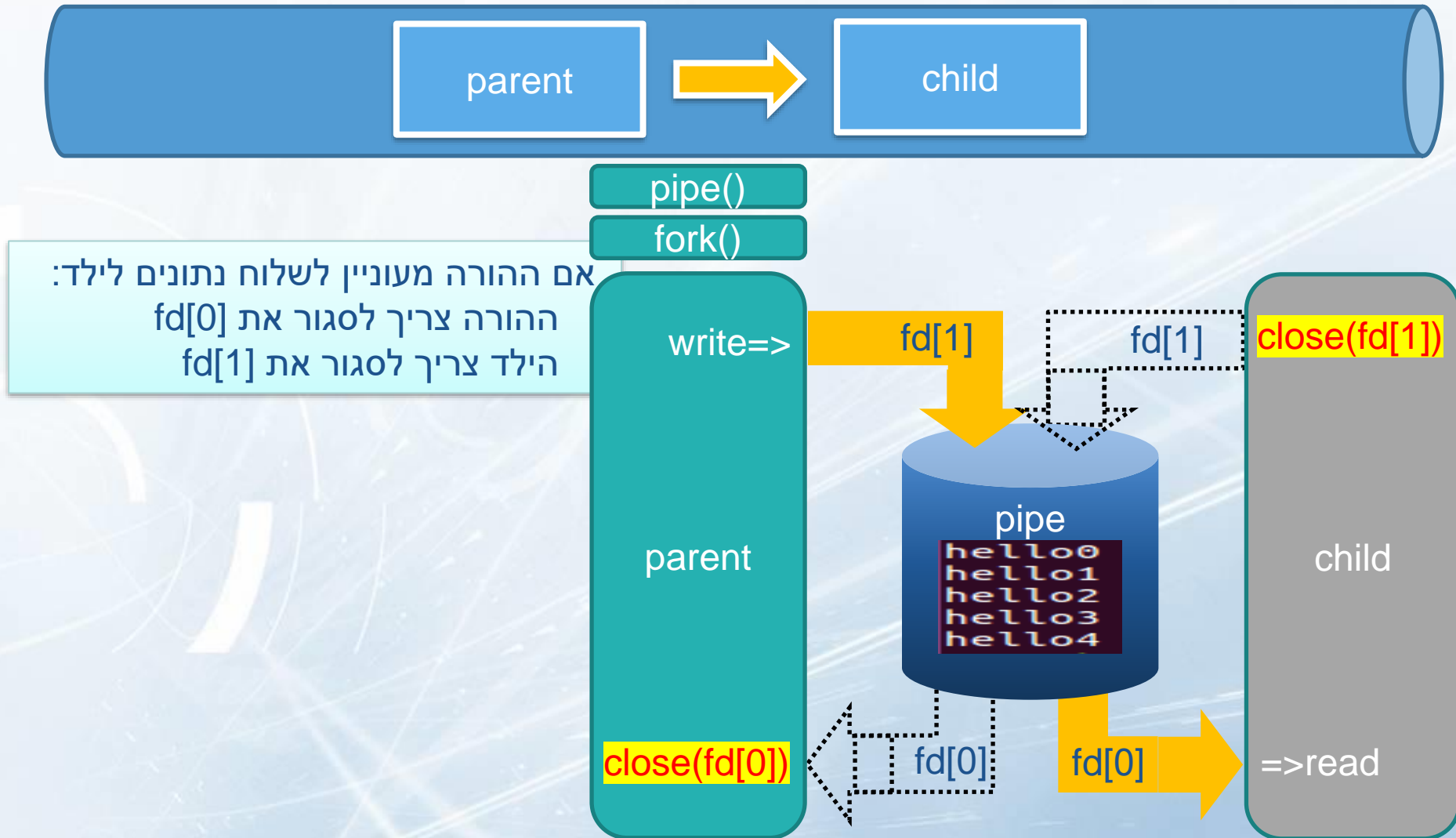
גודל PIPE קבוע
4k בגרסאות kernel ישנות
64k בגרסאות החדשות

PIPE משותף בין הורה לילד

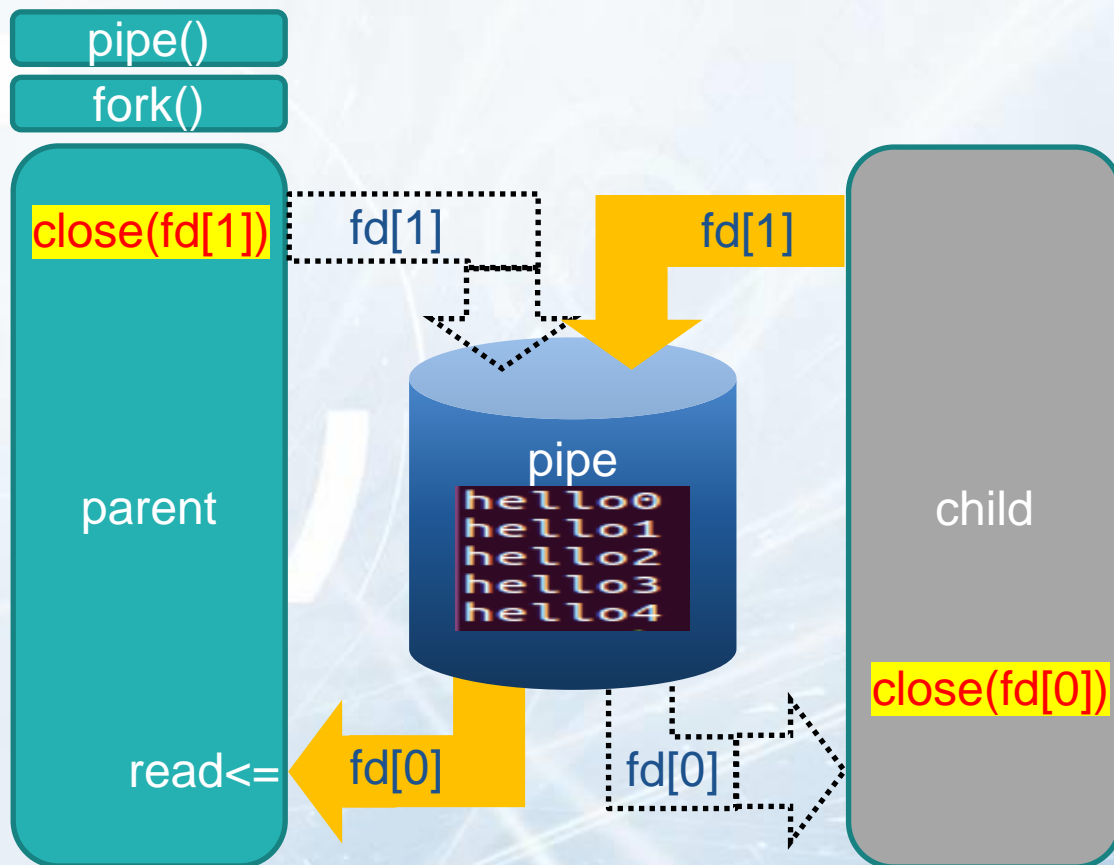
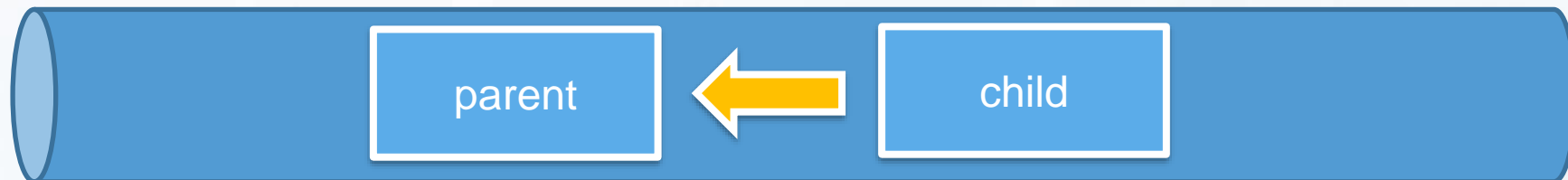


- מכיוון שה-descriptors (fd) משותפים בין ההורה לילד, עלינו תמיד לסגור את הקצה של הצינור שאינו בשימוש.
- הערה טכנית, EOF לעולם לא יוחזר אם הקצוות המיותרים של הצינור לא נסגרו במפורש.

PIPE משותף בין הורה לילד



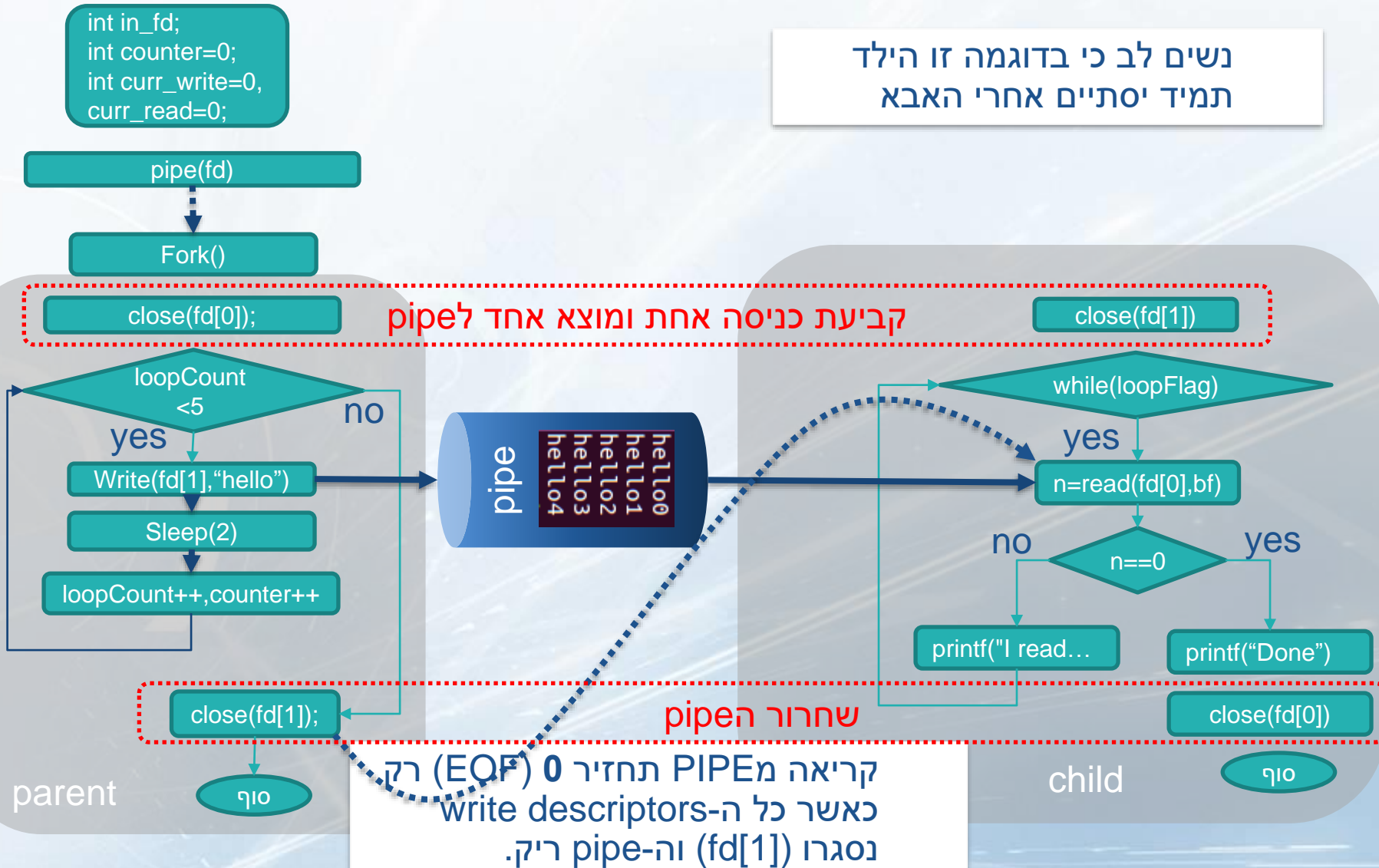
PIPE משותף בין הורה לילד



- אם ההורה מעוניין לקבל נתונים מהילד:
- על ההורה לסגור את ה-`fd[1]`
 - הילד צריך לסגור את ה-`fd[0]`

T7_3.c-PIPE

נשים לב כי בדוגמה זו הילד תמיד יסתיים אחרי האבא

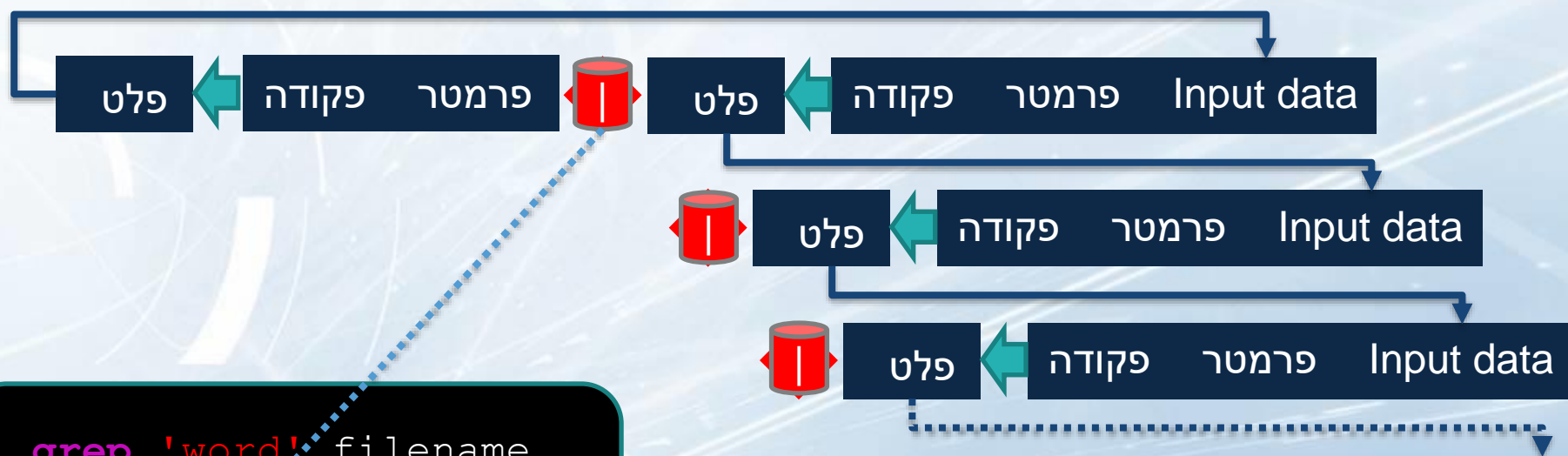


דוגמה c.4_T7

- ❖ שילוב של איתותים ו pipes
- ❖ נא לצייר סכמה מה עושה הקוד
- ❖ נא להריץ את הקוד
- ❖ נא לתאר מה ההדפסות המוצא

שרשור פקודות באמצעות PIPE

❖ כאשר רוצים שהקלט יבוא מתוך (או הפלט ישלח אל) תהליך אחר, מחליפים את הקלט או הפלט ב-pipe בין התהליכים



```
grep 'word' filename
```

```
$ ls -l | more
```

```
$ ls -l | grep 2018
```

FIFO

❖ הבעיה ב pipes:

- אם אנו רוצים שתהליכים שונים (לא קרובי משפחה) יתקשרו בניהם.
- היה נחמד אם ל pipe היה שם כמו שלקובץ יש שם.

FIFO

also known as named pipes

FIFO

❖ FIFO הוא למעשה pipe "ציבורי" שדרכו יכולים כל התהליכים במכונה לתקשר (כולם יכולים לגשת אליו).

❖ השימוש העיקרי של FIFO (או של כל אובייקט תקשורת בעל "שם") הוא כאשר תהליכים רוצים לתקשר דרך ערוץ קבוע מראש מבלי שיהיה ביניהם קשרי משפחה.

▪ למשל, כאשר תהליכי לקוח צריכים לתקשר עם תהליך שרת

❖ FIFO נוצר באמצעות קריאת המערכת `mknod()`, או באמצעות הפונקציה `mkfifo()` שמעניקה לו את שמו.

❖ **שם ה-FIFO הוא כשם קובץ במערכת הקבצים, למרות שאיננו קובץ כלל.**

▪ ה-FIFO מופיע במערכת הקבצים בשם שנבחר.

❖ שם ה-FIFO הוא השם של הקובץ כולל כל המסלול. כלומר יצור FIFO בתקייה מסויימת מייצר את הקובץ ה-FIFO באותה התקיה

FIFO

❖ לאחר היווצרו, ניתן לגשת ל-FIFO באמצעות פקודת `open()` ולעבוד איתו כרגיל (קריאה וכתיבה)

- ניתן לבצע הן קריאה והן כתיבה ל-FIFO דרך אותו descriptor (ערוץ תקשורת דו-כיווני)

❖ **סכרון:**

תהליך שפותח את ה-FIFO לקריאה בלבד **non** עד שתהליך נוסף יפתח את ה-FIFO לכתיבה, וההפך.

- פתיחת ה-FIFO לכתיבה וקריאה (`O_RDWR`) איננה חוסמת.

FIFO

❖ כאובייקטים ציבוריים רבים, FIFO אינו משוחרר אוטומטית לאחר שהשתמש האחרון בו סוגר את הקובץ, ולכן יש לפנותו בצורה מפורשת באמצעות פקודות או קריאות מערכת למחיקת קבצים (למשל, פקודת `rm` או הקריאה `unlink()`).

■ `rm` – מסיר קובץ או תקייה.

`unlink()` מוחק שם ממערכת הקבצים.

- אם שם זה היה הקישור האחרון לקובץ ובלי שום תהליכים המחזיקים את הקובץ, הקובץ נמחק והמרחב בו השתמשו זמין לשימוש חוזר.
- אם השם היה הקישור האחרון לקובץ אך יש תהליכים בהם עדיין הקובץ פתוח, אזי הקובץ יישאר קיים, עד שהקבצים יסגרו בתהליך ורק אז הוא ימחק.
- אם השם התייחס לקישור סמלי הקישור יוסר.
- אם השם התייחס לסוקט, או FIFO או התקן השם אומנם נמחק אך תהליכים עם האובייקט פתוח עשויים להמשיך להשתמש בו.

FIFO בפועל

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

```
int mknod (const char *pathname, mode_t mode, dev_t dev);
```

❖ קריאת המערכת `mknode()` יוצרת קודקוד מידע על אובייקט של מערכת הקבצים (קובץ רגיל, קובץ `device` או `named pipe` (FIFO)) שיקרא כשם הארגומנט הראשון (הנתיב) עם תכונות שיוגדרו ע"י `mode` ו `dev`.

הערות:

- ❖ הארגומנט `mode` מגדיר את ההרשאות של הקובץ וגם את סוג הקובץ.
- ❖ להגדרת FIFO אז `MODE` חייב להכיל `IFIFO_S` והיה שווה ל `0`.

FIFO בפועל

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

```
int mkfifo(const char *pathname, mode_t mode);
```

Mkfifo() - יוצר קובץ מסוג FIFO שיקרא לפי שם הנתיב שניתן.
הארגומנט mode מגדיר את ההרשאות של הFIFO.

FIFO בפועל

Define

Read permission bit
for the owner

Write permission bit for
the owner of the file

```
MYMODE (S_IFIFO | S_IRUSR | S_IWUSR)
```

```
mknod("myfifo.f",MYMODE,0);
```

Or

```
mkfifo("myfifo.f",0777);
```

```
fd=open("myfifo.f",O_RDONLY);
```

דוגמאות

❖ T7_5.c דוגמה פשוטה ל FIFO בין אב לבן מסונכרן

❖ דוגמה ל FIFO בין client לserver

■ T7_6.c – יצירת שני FIFO-ים

■ T7_7.c – יצירת CLIENT

■ T7_8.c – יצירת שרת

