ΕΠΛ421 - Προγραμματισμός Συστημάτων

Διάλεξη 15:

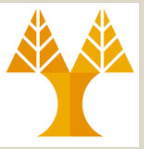
Επικοινωνία μεταξύ Διεργασιών (Inter-Process Communication (IPC))

Σωλήνες (Pipes) & FIFO

(Κεφάλαιο 15 - Stevens & Rago)

Δημήτρης Ζεϊναλιπούρ

Περιεχόμενο Διάλεξης

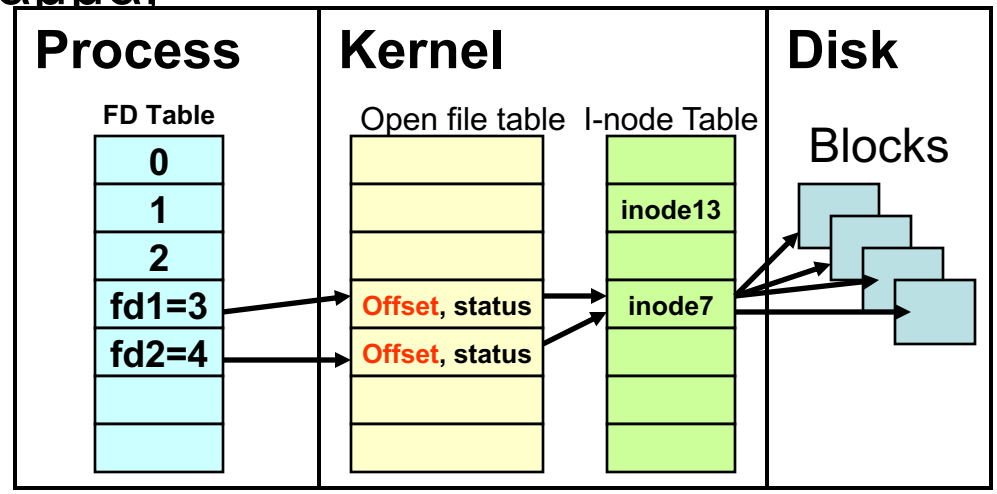


- Οι κλήσεις συστήματος dup, dup2.
- Διαδιεργασιακή Επικοινωνία – Μέρος Α
  A. Επικοινωνία μεταξύ Διεργασιών – Εισαγωγή και πρόβλημα.
  B. IPC1: Σωλήνες (Pipes)
  C. IPC2: FIFO (Named Pipes)

Διεργασίες και Αρχεία

- Προτού δούμε την επικοινωνία μεταξύ διεργασιών ας δούμε ξανά το θέμα των διεργασιών και των αρχείων.
- Θεωρήστε το ακόλουθο πρόγραμμα παρεμβαλλόμενων ενημερώσεων (*interceding update* scenario)
  - Στο UNIX τα ανοικτά αρχεία δεν κλειδώνονται αυτόματα αλλά μέσω κλήσεων **flock** (doesn't work over NFS) και **fcntl**
- Τι θα τυπώσει το πρόγραμμα;

```
#include <unistd.h>
#include <fcntl.h>
int main() {
  int fd1, fd2;
  fd1 = open("file1.txt", O_WRONLY | O_CREAT |
O_TRUNC, 0644);
  fd2 = open("file1.txt", O_WRONLY);
  write(fd1, "First Write\n", strlen("First Write\n"));
  write(fd2, "Second Write\n", strlen("Second Write\n"));
  close(fd1);
  close(fd2);
  return 0;
}
```



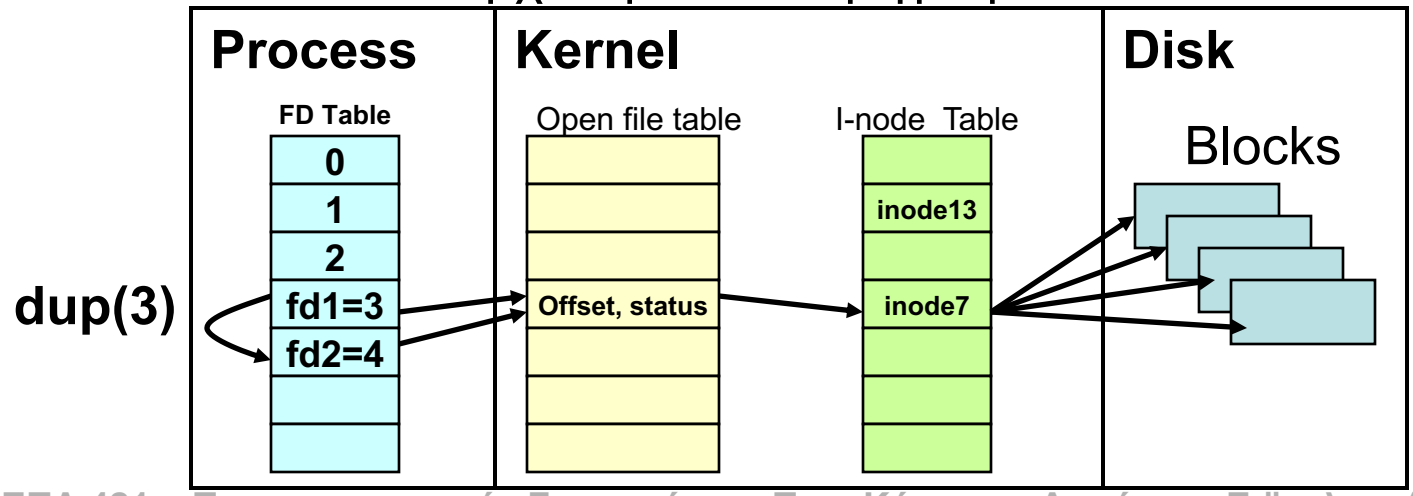
Έχουμε Write-after-Write πρόβλημα: Δηλ., τυπώνεται μόνο "Second write" διότι το Open File Table έχει δυο εγγραφές με διαφορετικά offset (αρχικά και τα δυο 0)

Η Κλήσεις Συστήματος dup(), dup2()

Μας επιτρέπουν να αντιγράφουμε ένα περιγραφέα αρχείου

int dup(int oldFD)
int dup2(int oldFD, int newFD)
Επιστρέφουν: τον νέο περιγραφέα (η dup2() επιστρέφει ουσιαστικά το newFD) ή -1 σε αποτυχία.

- Η dup βρίσκει το μικρότερο ελεύθερο περιγραφέα και τον αντιστοιχεί στο ίδιο ανοικτό αρχείο με το oldFD.
- Η dup2 δημιουργεί τον περιγραφέα newFD ο οποίος αντιστοιχεί στο ίδιο ανοικτό αρχείο με τον περιγραφέα oldFD.



Παράδειγμα Κλήσης dup()

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

int main()
{
  int fd1, fd2;

  fd1 = open("file1.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
  fd2 = dup(fd1);
  write(fd1, "First Write\n", strlen("First Write\n"));
  write(fd2, "Second Write\n", strlen("Second Write\n"));
  close(fd1);
  close(fd2);
  return 0;
}
```

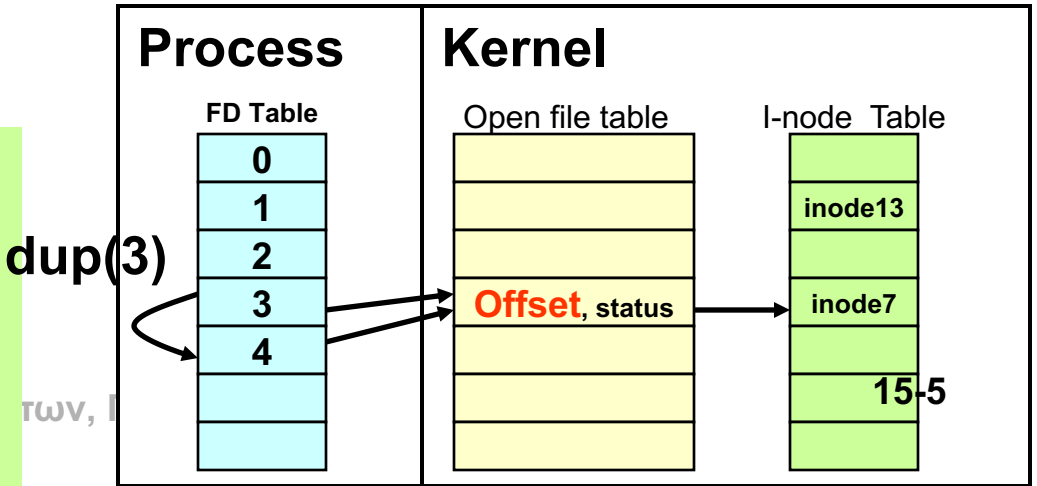
Τι θα τυπωθεί τώρα?

Θα τυπώσει ορθά

First Write

Second Write

Αυτό επειδή το offset του αρχείου είναι κοινό για τα δυο αρχεία, άρα κάθε εγγραφή αυξάνει το ίδιο Offset

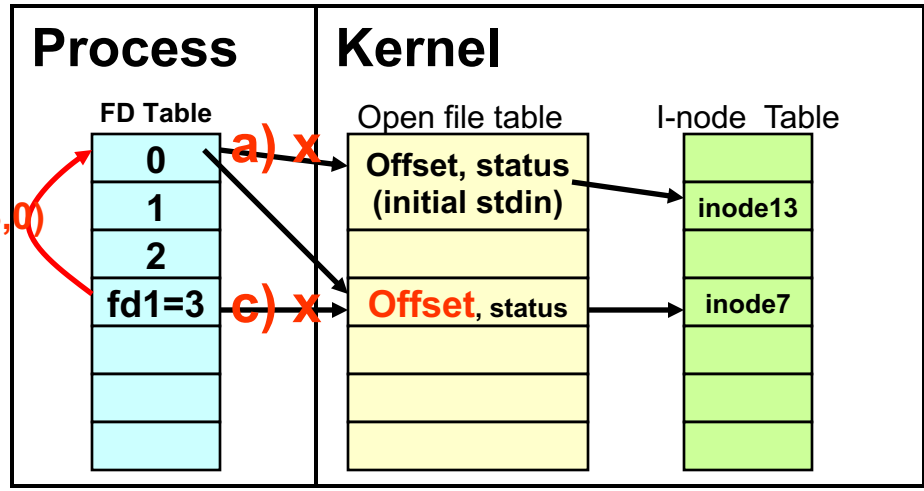


Παράδειγμα Κλήσης dup2()

Περιγραφή: Πολλές εντολές περιμένουν τα δεδομένα από το stdin (π.χ. Execlp). Εάν θέλουμε να κατευθύνουμε το περιεχόμενο ενός αρχείο σε μια τέτοια εντολή τότε το dup2() είναι πολύ χρήσιμη όπως δείχνει το παράδειγμα.

```
#include <unistd.h> // STDIN_FILENO
#include <stdio.h>  // BUFSIZE
#include <fcntl.h>  // O_RDONLY

int main()
{   int fd1, fd2;

    if ((fd1 = open("file.txt", O_RDONLY)) == -1) {
            perror("open"); exit(1);
    }

    close(STDIN_FILENO); // a) close FD#0 (optional step)
    fd2 = dup2(fd1, STDIN_FILENO); // b) Assign FD#3 into FD#0
    close(fd1);  // c) close FD#3 (optional step)

    printf("Now FD#0 reads from file.txt instead of STDIN!\n");

    // Since execlp expects the input from FD#0, it will sort whatever is currently
    // accessible through that descriptor (i.e., the content of file.txt)
    if (execlp("sort", "sort", NULL) == -1) {
            perror("execlp"); exit(1);
    }
   return 0;
}
```

b) dup2(3,0)



dup2(3,0) Αντικατάσταση περιγραφέα 0 με περιγραφέα 3, αντίστοιχο με την εντολή κελύφους exec 0<&3!

Η Κλήσεις dup(), dup2() με fork() Παράδειγμα ενός Shared Log file

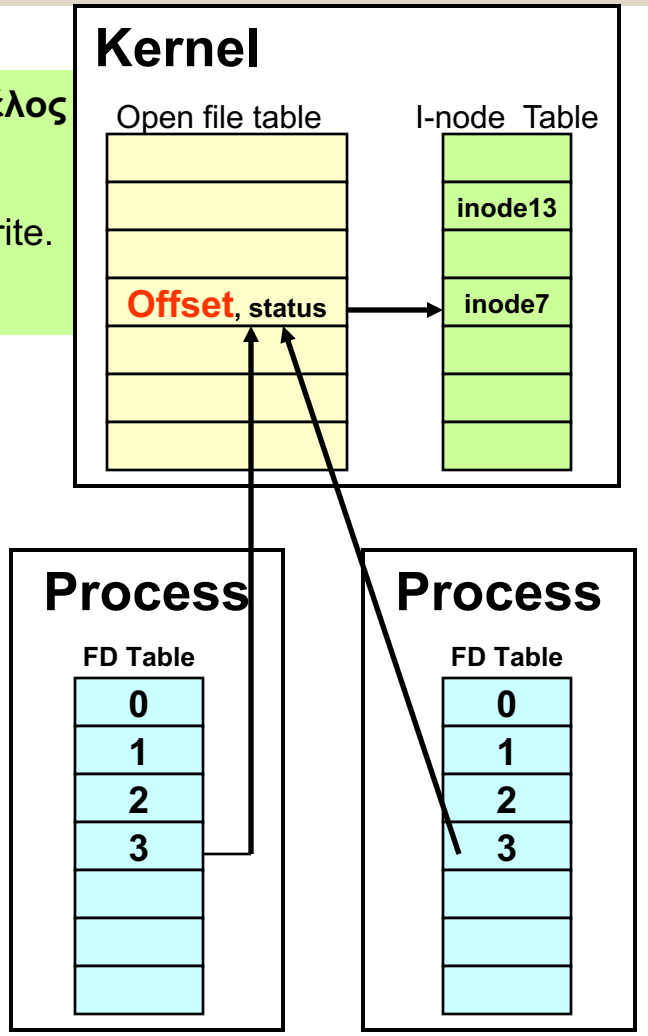
```
#include <fcntl.h>   // O_WRONLY | O_CREAT | O_TRUNC
#include <stdio.h>   // printf
#include <string.h>  // strcpy

int main() {
    int pid;   int fd, bytes;
    char *line1 = "Parent write. ";
    char *line2 = "Child write. ";

    printf("Creating New Log \n");
    if ((fd = open("log.txt", O_WRONLY | O_CREAT | O_TRUNC, 0600)) == -1) {
            perror("open"); exit(1);
    }

    if ((pid = fork()) == -1) {   /* Check for error */
                perror("fork");  exit(1);
    }
    else if (pid == 0) {          /* The child process */
            while (1) {
                bytes = write(fd, line2, strlen(line2)); /* Data out */
            }
    }
    else {  /* The parent process */
             while(1) {
                bytes = write(fd, line1, strlen(line1)); /* Data out */
            }
     }
     exit(0);
}
```

Το write θα γίνεται πάντα στο τέλος του αρχείου!). Δηλ.,

....

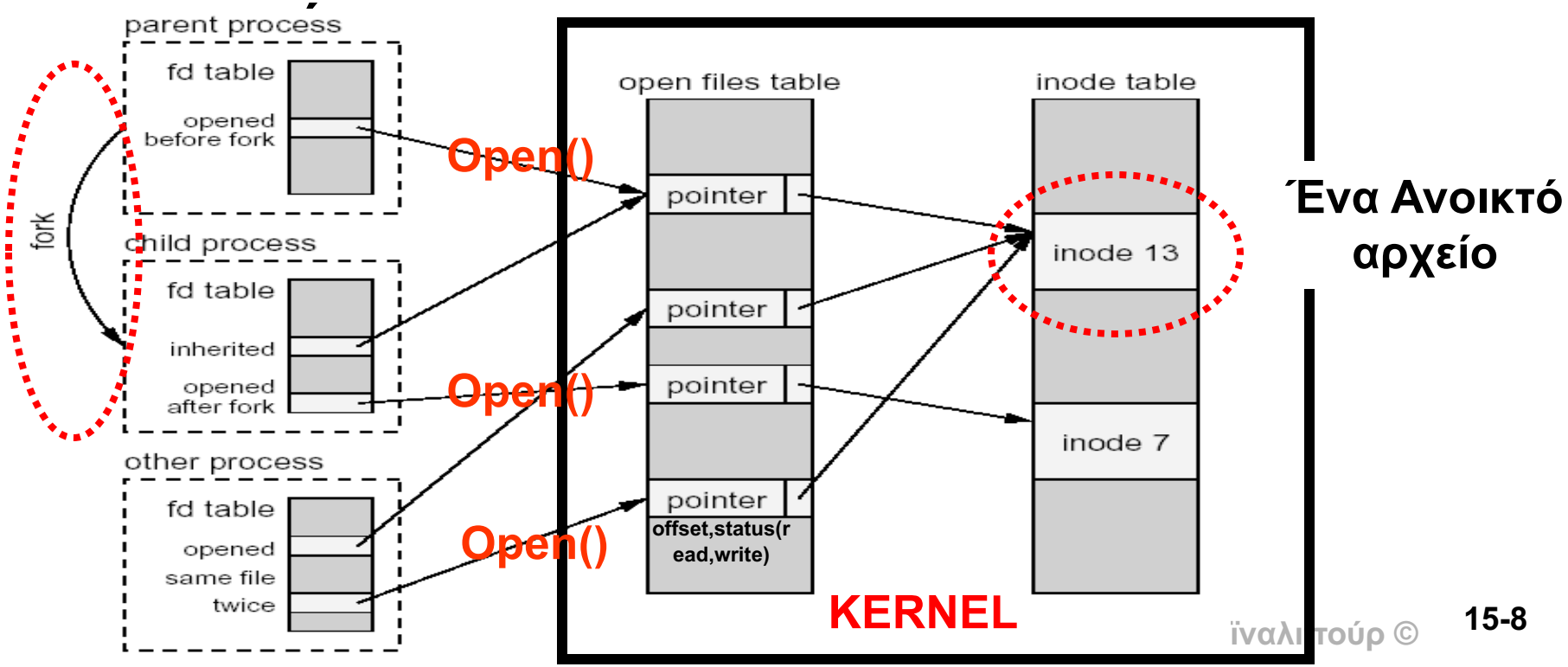
Child write. Child write. Parent write.
Parent write. Parent write.

....



Η Κλήση συστήματος dup()

- Έχουμε αναφέρει ήδη ότι όταν κάνουμε fork στο Unix τότε κοινοποιούνται μαζί με τα υπόλοιπα δεδομένα και οι περιγραφείς



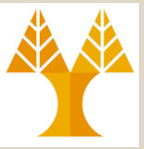
Επικοινωνία μεταξύ Διεργασιών
Το Πρόβλημα

- **Δυο διεργασίες δεν μοιράζονται το ίδιο memory space (stack, heap, etc).**
- **Επομένως δεν μπορούν να μοιράζονται δεδομένα, μεταβλητές, δομές δεδομένων, κτλ.**

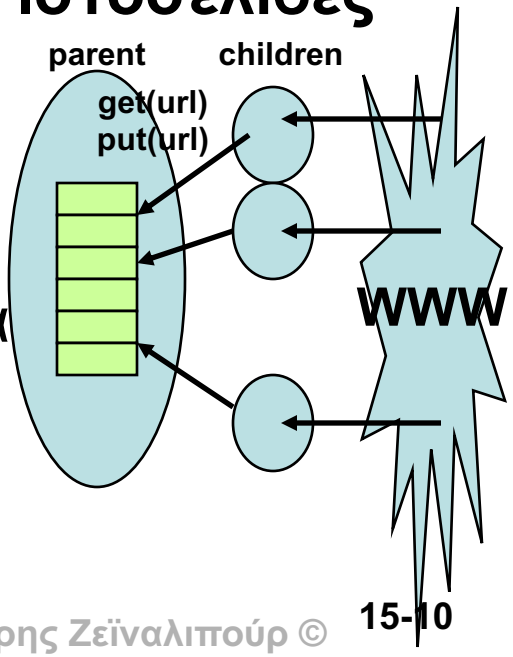
Μπορεί να πετύχουμε επικοινωνία με τεχνικές που μάθαμε έως τώρα;

- **Λύση 1;** Η μια διεργασία γράφει σε κάποιο αρχείο TEMP και η άλλη διαβάζει από εκεί.
  - **Πρόβλημα:** i) Ο δίσκος είναι αργός, ii) για να ανοίξει ο reader πρέπει ο writer να κλείσει το αρχείο – για να μην μπερδευτούν τα file offsets μέσα στον Kernel. Αυτό επιτρέπει μόνο γραμμικές παρά παράλληλες εκτελέσεις ☹.
- **Λύση 2;** Μπορεί να χρησιμοποιηθούν σήματα μαζί με τους κατάλληλους signal handlers.
  - **Πρόβλημα:** Δεν μπορεί να αποσταλεί τίποτε πιο σύνθετο από ένα ακέραιο (π.χ., το exit code ή SIGUSR1, SIGUSR2 ☹)
- **Λύση 3;** Μπορεί να τεθεί η μεταβλητή (ή πίνακας ή δομή) πριν το fork().
  - **Πρόβλημα:** Μετά το fork() ο καθένας έχει το δικό του αντίτυπο επομένως πάλι δεν θα ξέρει ο ένας τα δεδομένα του άλλου μετά την κλήση της fork ☹

Επικοινωνία μεταξύ Διεργασιών
Εισαγωγή



- Έστω ότι θέλουμε να φτιάξουμε ένα **παράλληλο crawler** ο οποίος χρησιμοποιεί **50 διεργασίες** για να επιταχύνει τον **ρυθμό ανάκτησης ιστοσελίδων**.
- Προφανώς, η κάθε διεργασία πρέπει να γνωρίζει τα **URLs** που έχουν **ανακτήσει** οι άλλες διεργασίες για να μην ανακτώνται **πολλαπλές φορές κάποιες ιστοσελίδες**
- **Αυτό είναι ένα τυπικό πρόβλημα διαμοιρασμού πληροφορίας μεταξύ παράλληλων διεργασιών.**
- **Λύσεις:** Πάρα πολλές … όλες χρειάζονται κάποια μορφή διαδιεργασιακής επικοινωνίας … αυτό θα είναι το αντικείμενο μελέτης μας σε αυτή την ενότητα.



Επικοινωνία μεταξύ Διεργασιών Ποια είδη θα μελετήσουμε?

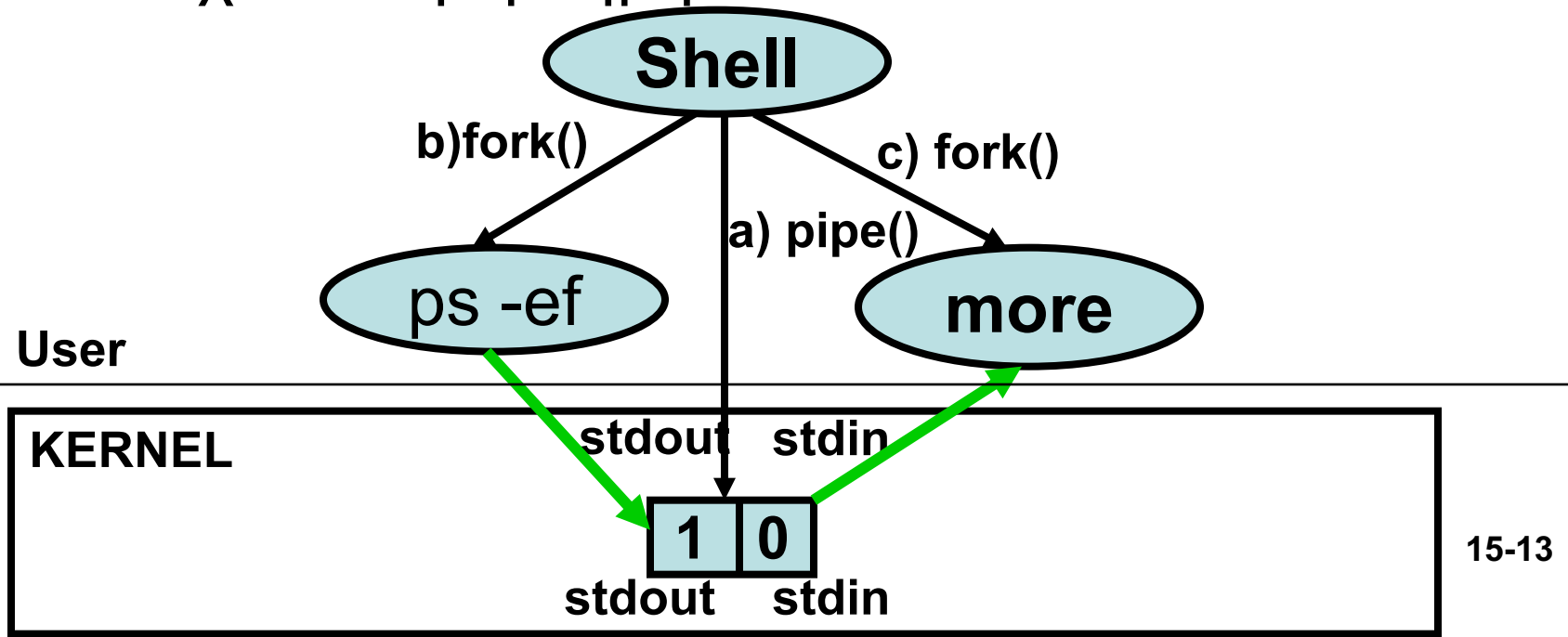
- **Μια Κατεύθυνση (Half Duplex) ανά πάσα στιγμή στον ίδιο Η/Υ**
  - **Α) Σωλήνες (Pipes)** – Διάλεξη 15 (15.2) – μεταξύ πατέρα⇔παιδί
  - **Β) FIFO (Named Pipes)** – Διάλεξη 15 (15.5) – μεταξύ οποιονδήποτε διεργασιών στον ίδιο Η/Υ
- **Προς Δυο Κατευθύνσεις (Full Duplex) στον ίδιο Η/Υ**
  - **Γ) Ουρές Μνημάτων (Message Queues)** – Διάλεξη 16
  - **Δ) Κοινή Μνήμη (Shared Memory)** – Διάλεξη 16
  - **Ε) Σηματοφόροι (Semaphores)** – Διάλεξη 16 – Μηχανισμός συγχρονισμού διεργασιών με μεταβλητές του πυρήνα για την αποκλειστική διαχείριση πόρων.
- **Full Duplex & Διεργασίες σε διαφορετικούς Η/Υ**
  - **Ζ) Υποδοχές (Sockets)** – Διάλεξη 17 & 18 – Network IPC - μεταξύ οποιονδήποτε διεργασιών που διαμοιράζονται κάποιο κοινό δίκτυο (θα επικεντρωθούμε μόνο σε TCP/IP sockets)

Α) Επικοινωνία με **Σωλήνες (Pipes)**

- Η απλούστερη μορφή επικοινωνίας μεταξύ διεργασιών.
- Η σωλήνα είναι ουσιαστικά ένα **memory buffer** μέσα στον πυρήνα!
- **Χρησιμοποιείται για διεργασίες με κοινό πρόγονο (ο οποίος έχει δημιουργήσει την σωλήνα).**
- Η επικοινωνία γίνεται **μόνο** προς μια κατεύθυνση **(half-duplex).**
- Μια διεργασία γράφει στο άκρο γραψίματος της σωλήνας (**write descriptor**) και η άλλη διαβάζει από το άκρο διαβάσματος (**read descriptor**).

Επικοινωνία με Σωλήνες **Pipes**

- Σωλήνες έχουμε χρησιμοποιήσει ήδη εκτενώς στα πλαίσια του προγραμματισμού κελύφους.
- Όταν εκτελούμε για παράδειγμα την διοχέτευση **‘ps –ef | more’** τότε έχουμε το ακόλουθα στοιχεία στην μνήμη



Η κλήση συστήματος pipe()

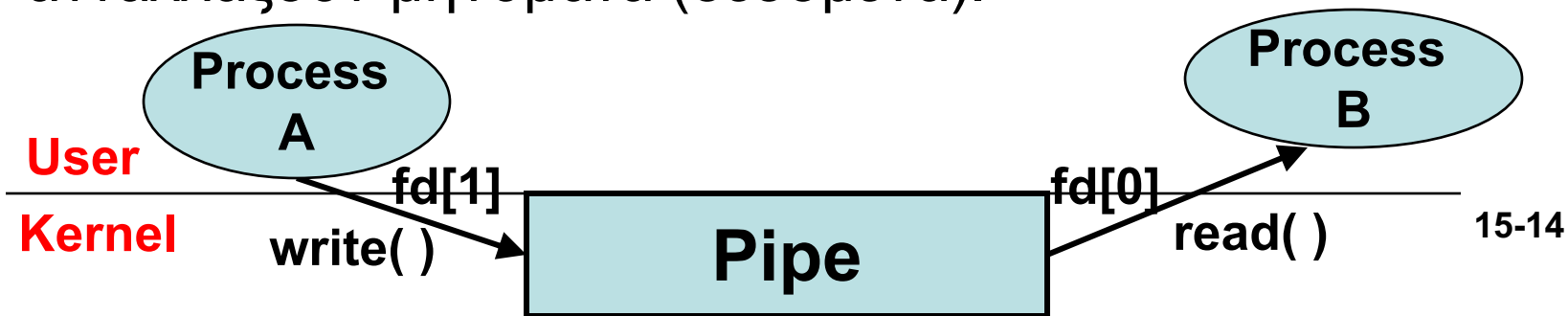
- Η δημιουργία σωλήνων στην γλώσσα C γίνεται με την κλήση συστήματος pipe().

#include <unistd.h>

int pipe(int filedesc[2]);

Επιστρέφει -1 σε περίπτωση λάθους ή 0 σε επιτυχία.

- Η pipe() δημιουργεί δυο file descriptors (οτιδήποτε γράφει μια διεργασία στο **fd[1]** μπορεί να διαβαστεί από την άλλη διεργασία στο **fd[0]**).
- Αυτό δίδει ένα μηχανισμό σε δυο διεργασίες να ανταλλάζουν μηνύματα (δεδομένα).



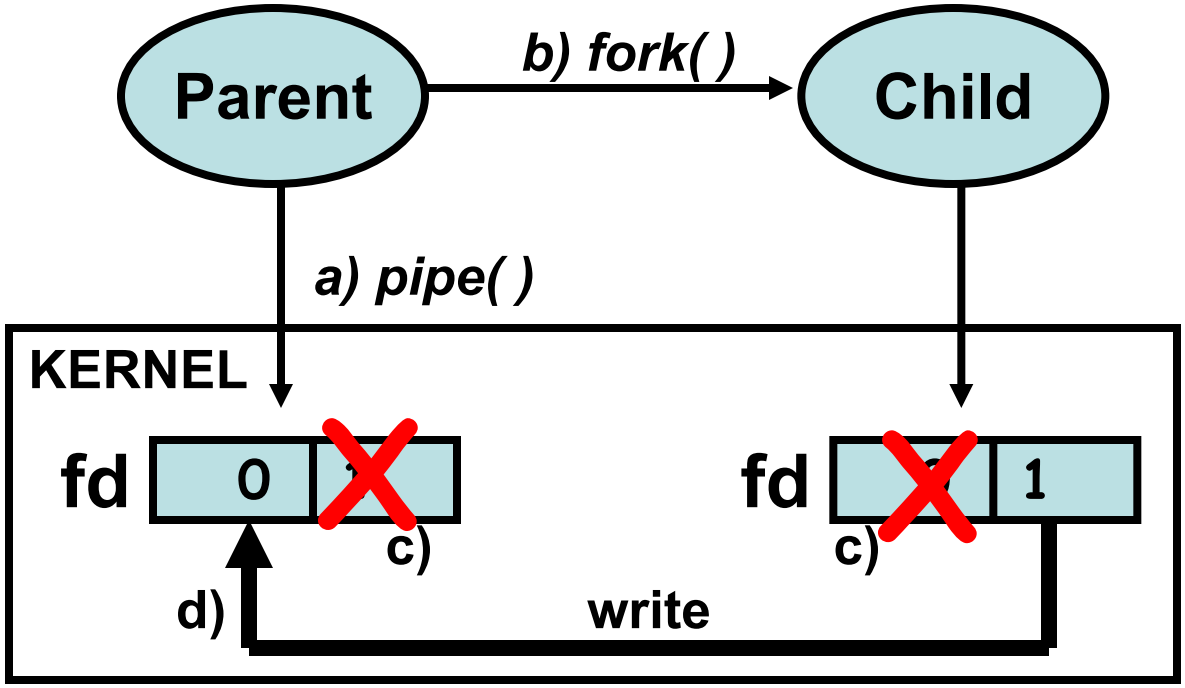
Παράδειγμα 1
Επικοινωνία **Παιδιού** => **Πατέρα**

Να γραφεί ένα πρόγραμμα C που να δημιουργεί την σωλήνα

παιδί => πατέρα

Στη συνέχεια, το παιδί γράφει στον πατέρα ένα string και ο πατέρας το εκτυπώνει.

Παράδειγμα 1
Επικοινωνία **Παιδιού => Πατέρα**



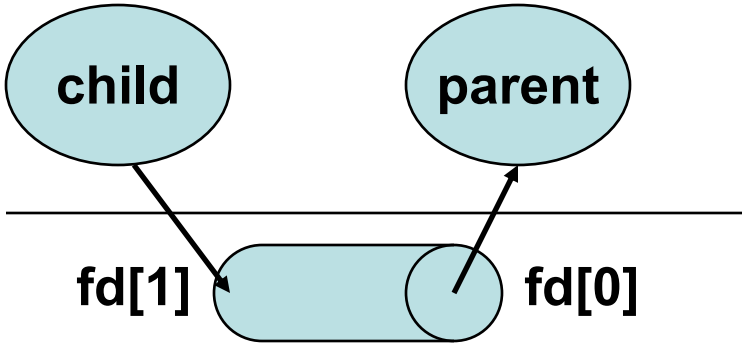
- Για να στείλει το παιδί (writer) κλείνει πρώτα το δικό του **fd[0]** (read) και στην συνέχεια γραφεί στο **fd[1]** (write).
- Αντίστοιχα ο πατέρας (reader) κλείνει το **fd[1]** (write) και διαβάζει από το δικό του **fd[0] (read)**

Παράδειγμα 1
Επικοινωνία **Παιδιού** => **Πατέρα**

```
#include <stdio.h> /* For printf */
#define READ 0 /* Read end of pipe */
#define WRITE 1 /* Write end of pipe */
char *phrase = "This is a test phrase.";
int main() {
        int pid, fd[2], bytes;
        char message[100];
        if (pipe(fd) == -1) { /* Create a pipe */
                perror("pipe");  exit(1);
        }
        if ((pid = fork()) == -1) { /* if Fork failed */
                perror("fork"); exit(1);
        }
        if (pid == 0) { /* Child (WRITER) Code*/
                close(fd[READ]); /* Close unused end */
                write(fd[WRITE], phrase, strlen(phrase)+1);
                close(fd[WRITE]); /* Close used end */
        }
        else { /* Parent (READER) Code */
                close(fd[WRITE]); /* Close unused end */
                bytes = read(fd[READ], message, sizeof(message));
                printf("Parent Read %d bytes: %s\n", bytes, message);
                close(fd[READ]); /* Close used end */
        }
}
```

Παράδειγμα Εκτέλεσης

```
$ ./pipe1
Parent read 23 bytes: This is a test phrase.
```



Επισημάνσεις

- Εάν δεν κάνουμε close το pipe τότε πάλι δουλεύει το πρόγραμμα (εφόσον οι file descriptors κλείνουν με την λήξη της κάθε διεργασίας)
- Από την στιγμή που γράφουμε στο pipe δεν μπορούμε πλέον να διαβάσουμε από το ίδιο pipe (και αντίστροφα)

Παράδειγμα 2
Επικοινωνία **Παιδιού** <=> **Πατέρα**

Να γραφεί ένα πρόγραμμα C που να δημιουργεί τις ακόλουθες σωλήνες

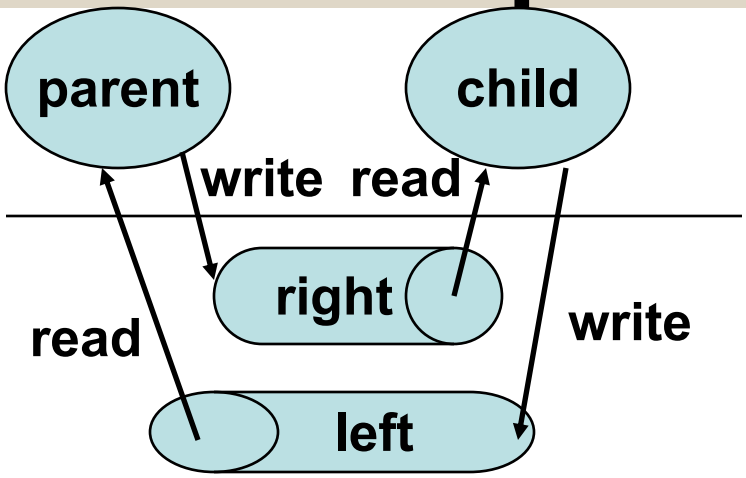
πατέρα => παιδί

παιδί => πατέρας

Στην συνέχεια ο πατέρας και το παιδί ανταλλάζουν ping (πατέρας) και pong (παιδί) εκτυπώνοντας κάθε φορά το μήνυμα που παραλαμβάνεται

Παράδειγμα 2
Επικοινωνία **Παιδιού <=> Πατέρα**

```
#include <stdio.h> /* For printf */
#define READ 0 /* Read end of pipe */
#define WRITE 1 /* Write end of pipe */
char *ping = "Ping"; char *pong = "Pong";
int main() {
        int pid, right[2], left[2], bytes;   char message[100];
        if ( (pipe(right) == -1) || (pipe(left) == -1) ) { /* Create two pipes */
                perror("pipe");  exit(1);
        }
        if ((pid = fork()) == -1) { /* Fork a child */
                perror("fork"); exit(1);
        }
        else if (pid==0)  { /* Child */
                close(right[WRITE]); close(left[READ]);
                while (1) {
                    bytes = read(right[READ], message, sizeof(message)); // (wait) read ping
                    printf("child received: %d bytes: %s\n", bytes, message);
                    write(left[WRITE], pong, strlen(pong)+1); // send pong
                    sleep(1);
                }
                close(right[READ]);  close(left[WRITE]);
        }
        else  { /* Parent  */
                close(right[READ]); close(left[WRITE]);
                while (1) {
                     write(right[WRITE], ping, strlen(ping)+1);   // send ping
                     bytes = read(left[READ], message, sizeof(message)); // (wait) read pong
                     printf("parent received: %d bytes: %s\n", bytes, message);
                     sleep(1);
                }
                close(right[WRITE]); close(left [READ]);
        }
}
```



Παράδειγμα Εκτέλεσης

```
$ ./pingpong
child received: 5 bytes: Ping
parent received: 5 bytes: Pong
child received: 5 bytes: Ping
parent received: 5 bytes: Pong
```

Παράδειγμα 3

Να γραφεί ένα πρόγραμμα C που να συνδέει μέσω ενός σωλήνα την προκαθορισμένη έξοδο μιας εντολής με την προκαθορισμένη είσοδο μιας άλλης, να υλοποιηθεί δηλαδή μια σωλήνωση της μορφής

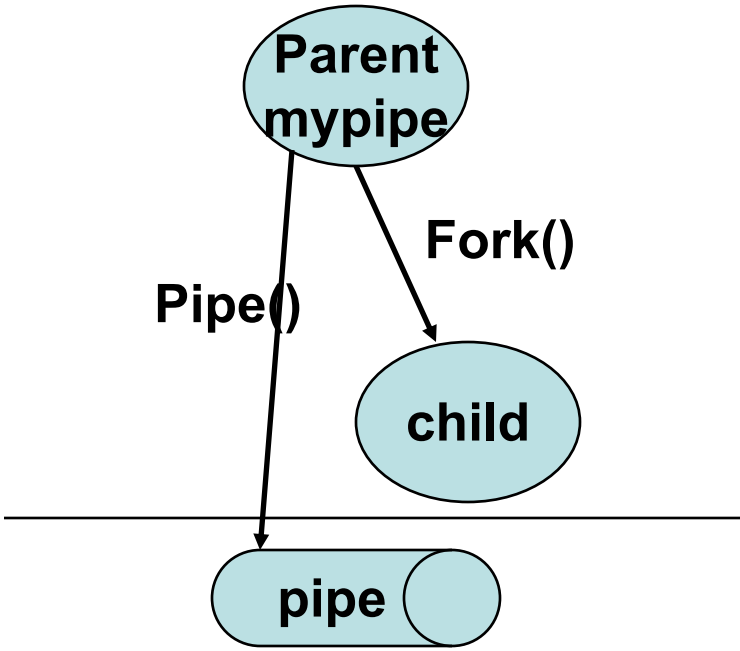
./mypipe ls sort

Το οποίο είναι ισοδύναμο με

ls | sort

Παράδειγμα 3

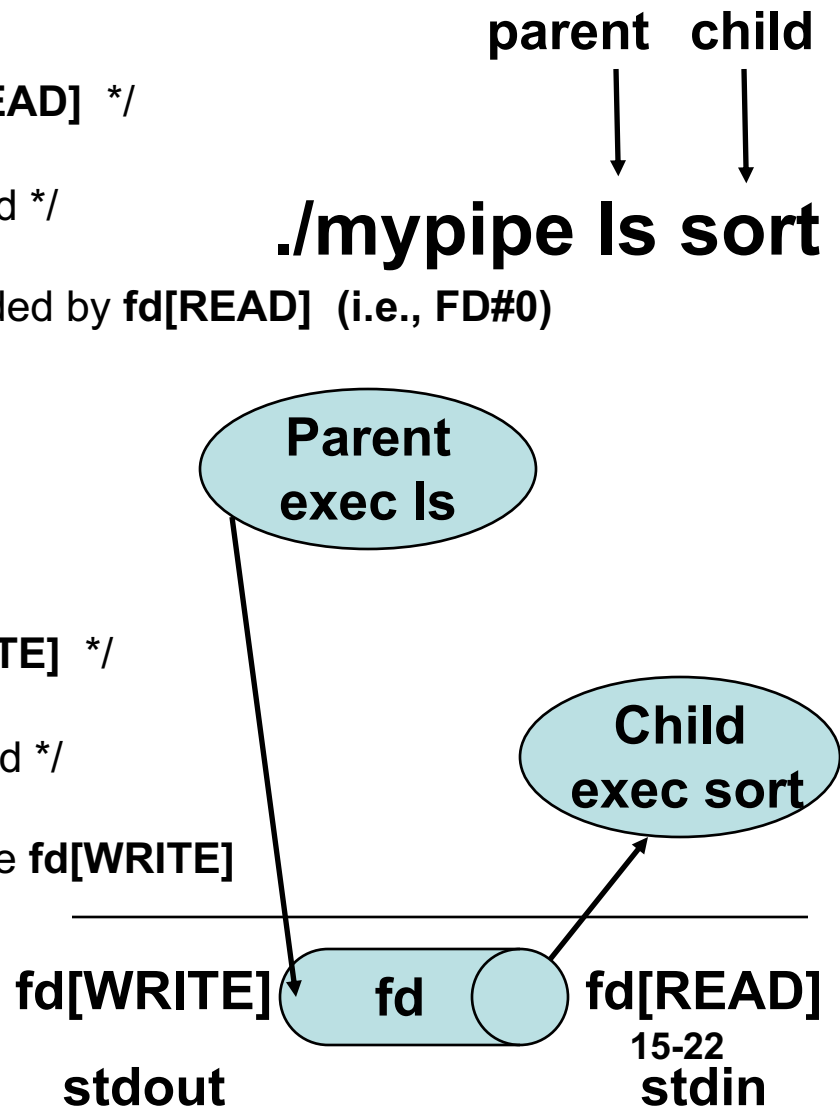
```
/* File: mypipe.c */
#include <stdio.h> /* For printf */
#define READ 0 /* Read end of pipe */
#define WRITE 1 /* Write end of pipe */

int main(int argc, char *argv[])
{
        int fd[2], pid;
        if (pipe(fd) == -1) {  /* Create a pipe */
             perror("pipe"); exit(1);
        }
        if ((pid = fork()) == -1) {  /* Fork a child */
             perror("fork");     exit(1);
        }
```



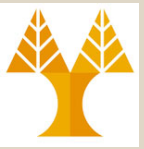
Συνέχεια επόμενη σελίδα…

Παράδειγμα 3

```
    else if (pid == 0) {          /* Child (READER) */
        close(fd[WRITE]);  /* Close unused end */
         /* Define that STDIN should come from fd[READ]  */
        dup2(fd[READ], 0);    /* 0 := fd[READ] */
        close(fd[READ]);      /* optional: Close read end */

        // The input of sort (argv[2]) will now be provided by fd[READ]  (i.e., FD#0)
        if (execlp(argv[2], argv[2], NULL) == -1) {
              perror("execlp");
        }
    }
    else { /* Parent (WRITER) */
        close(fd[READ]);
        /* Define that STDOUT should go into fd[WRITE]  */
        dup2(fd[WRITE], 1);  /* 1 := fd[WRITE] */
        close(fd[WRITE]);    /* optional: Close write end */

        // The output of ls (argv[1]) now goes into pipe fd[WRITE]
        if (execlp(argv[1], argv[1], NULL) == -1) {
              perror("execlp");
        }
    }
}
```



Επικοινωνία με Σωλήνες (Pipes)
Κάποιες Τελευταίες Λεπτομέρειες…



- Κάποια συστήματα υποστηρίζουν και **full-duplex pipes**, δηλ., μια διεργασία μπορεί να διαβάζει και να γράφει παράλληλα χωρίς να εγκαθιδρύει δυο σωληνες (Pipes), π.χ., αυτό ισχύει στο cygwin/windows.

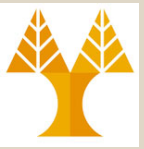
- Ωστόσο αυτό δεν ισχύει σε όλα UNIX (π.χ., Linux). Για λόγους **portability** (μεταφερσιμότητας) θα υποθέτουμε ότι οι σωλήνες είναι **ΠΑΝΤΟΤΕ half-duplex**.

- Όπως η **fopen()** προσφέρει διαχείριση αρχείων μέσω κλήσεων βιβλιοθήκης σε κανονικά αρχεία έτσι και η **popen()/pclose()** (Κεφ. 15.3) προσφέρει διαχείριση σωλήνων μέσω συναρτήσεων βιβλιοθήκης (παρά κλήσεων συστήματος).

C) Επικοινωνία με FIFO (Named Pipes)

- **Χρησιμοποιείται για επικοινωνία οποιονδήποτε διεργασιών στο ίδιο Η/Υ (όχι μόνο διεργασιών με τον ίδιο πρόγονο)**
- Η επικοινωνία γίνεται πάλι **μόνο** προς μια κατεύθυνση (**half-duplex**).
- Και πάλι πρόκειται για ένα memory buffer στον πυρήνα … αλλά αυτός είναι **προσπελάσιμος μέσω ενός ονόματος αρχείου** (για αυτό το FIFO ονομάζεται και **Named Pipe).**
- Προτού δούμε τα FIFOs στην C ας μελετήσουμε πως τα διαχειριζόμαστε μέσω του κελύφους.

Β) FIFOs στο Κέλυφος



Κέλυφος 1 – PID#2805

 # Δημιουργία Αρχείου FIFO

$mkfifo comm (το ίδιο με την πιο γενική εντολή "mknod comm p")

$ls -al comm

prw-r--r-- 1 dzeina faculty 0 Mar 7 13:20 comm

Δείχνει ότι είναι FIFO pipe

Τώρα θα γράψουμε στο FIFO

$cat /etc/profile > comm

Η εντολή εδώ κάνει block (δηλαδή περιμένει μέχρι να διαβάσει κάποιος τα δεδομένα)

Ανοίγουμε τώρα δεύτερο κέλυφος και γράφουμε…

Κέλυφος 2 – PID#2775

$cat < comm # Ανάγνωση από το αρχείο

```
# /etc/profile

# System wide environment and startup programs, for login setup
# Functions and aliases go in /etc/bashrc
pathmunge () {
        if ! echo $PATH | /bin/egrep -q "(^|:)$1($|:)" ; then
           if [ "$2" = "after" ] ; then
              PATH=$PATH:$1
....
```

Τώρα κάνουν unblock και τα δυο κελύφη

Παρατηρούμε ότι το κέλυφος-2 (νέα διεργασία) τύπωσε ότι είχε μέσα το αρχείο comm.

Επομένως πετύχαμε επικοινωνία μεταξύ δυο ξένων (όχι πατέρα-παιδί) διεργασιών

FIFO στην C

#include <sys/stat.h>

int mkfifo(char *pathname, mode_t mode)

Επιστρέφει -1 σε περίπτωση λάθους ή 0 σε επιτυχία.

- Η **mkfifo() δημιουργεί** το αρχείο: pathname (απόλυτο ή σχετικό μονοπάτι στο αρχείο) με δικαιώματα πρόσβασης mode (π.χ. 0777)
- Στην συνέχεια χρησιμοποιούμε τις χαμηλού επιπέδου κλήσεις συστήματος (open(), close(), read(), write(), unlink()) για να έχουμε πρόσβαση στο αρχείο.
- Ακολουθεί παράδειγμα χρήσης…

Παράδειγμα 4 – FIFO

Να γραφούν τα ακόλουθα προγράμματα

*α) Ένα **writer** που να δημιουργεί ένα **FIFO** pipe, εάν δεν υπάρχει ήδη, και στην συνέχεια να γράψει μέσα στο **FIFO 10 MB από κενά bytes (NULs).***

*β) Ένα **reader** ο οποίος να **διαβάζει** τα δεδομένα από το **FIFO** και εκτυπώνει πόσα bytes παραλήφθηκαν.*

./writer &

./reader

Παράδειγμα 4 – FIFO writer

/* fifo-writer.c: Writes data in the fifo */

```
#include <fcntl.h>  // O_WRONLY
#include <limits.h> // PIPE_BUF
#include <stdio.h>  // printf
#include <unistd.h> // F_OK

#define FIFO_NAME "my_fifo"
#define BUFFER_SIZE PIPE_BUF  // 4096
#define TEN_MEG 10485760

int main() {
        int fifoFD, bytes_sent = 0, sent = 0;
        char buffer[BUFFER_SIZE]={};

        /* check if FIFO_NAME file exists on disk*/
        if (access(FIFO_NAME, F_OK) == -1) {
                printf("Creating fifo file %s\n", FIFO_NAME);
                if (mkfifo(FIFO_NAME, 0777) == -1) {
                        perror("mkfifo"); exit(1);
                }
        }
        /* Now open the generated fifo file */
        printf("Producer %d opening %s\n", getpid(), FIFO_NAME);
        if ((fifoFD = open(FIFO_NAME, O_WRONLY)) == -1) {
                perror("open"); exit(1);
        }
        while(bytes_sent < TEN_MEG) {
                if ((sent = write(fifoFD, buffer, BUFFER_SIZE)) == -1) { /* Write Error */
                                perror("write"); exit(1);
                }
                bytes_sent += sent;
        }
        close(fifoFD); /* Close FIFO file*/
        return 0;
}
```

Έλεγχος ότι το FIFO αρχείο υπάρχει

Ανοίγουμε το FIFO για γραφή

Γράφουμε μέσα στο FIFO 10MB δεδομένων

Παράδειγμα 4 – FIFO reader

/* fifo-reader.c : Reads data from fifo file*/

```
#include <fcntl.h>  // O_WRONLY
#include <limits.h> // PIPE_BUF
#include <stdio.h>  // printf
#include <unistd.h> // F_OK

#define FIFO_NAME "my_fifo"
#define BUFFER_SIZE PIPE_BUF

int main() {
        int fifoFD, bytes_received = 0, received = 0;
        char buffer[BUFFER_SIZE];

        if ((fifoFD = open(FIFO_NAME, O_RDONLY)) == -1) {
                perror("open"); exit(1);
        }

        while((received = read(fifoFD, buffer, BUFFER_SIZE)) > 0) {
                bytes_received += received;
        }

        /* Now open the generated fifo file */
        printf("Consumer %d has received  %d bytes\n", getpid(), bytes_received);

        close(fifoFD); /* Close FIFO file*/
        return 0;
}
```

Ανοίγουμε το FIFO για ανάγνωση

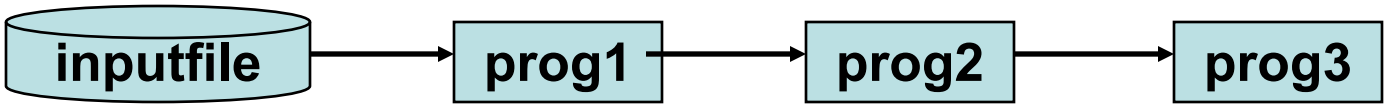
Διαβάζουμε από το FIFO όσα δεδομένα υπάρχουν

Εκτυπώνουμε τον αριθμό από bytes που έχουν διαβαστεί

Pipes vs. FIFO

- Τα **PIPEs** μπορούν να χρησιμοποιηθούν **μόνο** για **γραμμικές συνδέσεις** μεταξύ διεργασιών

  (π.χ. prog1 < inputfile | prog2 | prog3)



- Τα **FIFOs** μπορούν να δημιουργήσουν **μη-γραμμικές συνδέσεις** μεταξύ διεργασιών διότι έχουμε το filename:

  $mkfifo fifo1 fifo2 # δημιουργία 2 fifo αρχείων

  $prog3 < fifo1 & και $ prog4 < fifo2 &

  $prog1 < inputfile | tee fifo1 fifo2 | prog2

  # tee: copy stdin σε αρχεία fifo1,fifo2 και stdout