Διαδιεργασιακή επικοινωνία (inter-process communication IPC)

Συνεργασία ανάμεσα σε διεργασίες

- Για ασφάλεια/ανεξαρτησία, το ΛΣ εξασφαλίζει πλήρη **απομόνωση** ανάμεσα στις διεργασίες

- Τι γίνεται αν κάποιες διεργασίες επιθυμούν να συνεργαστούν / αλληλεπιδράσουν μεταξύ τους;

- Το ΛΣ παρέχει ειδικούς μηχανισμούς επικοινωνίας μέσω των οποίων διαφορετικές διεργασίες μπορούν να αλληλεπιδράσουν / στείλουν δεδομένα
  - inter-process communication (IPC) mechanisms

IPC mechanisms

- Αρχεία – files
  - βλέπε προηγούμενες διαλέξεις
- Αγωγοί – pipes
  - ροές bytes μιας κατεύθυνσης
- Υποδοχείς – sockets
  - unix domain sockets, Internet domain sockets
  - datagram sockets, stream sockets
- Ουρές μηνυμάτων – message queues
  - αποστολή/παραλαβή ομαδοποιημένων bytes
- Κοινόχρηστη μνήμη – shared memory
  - γράψιμο / διάβασμα απ' ευθείας στην / από την μνήμη

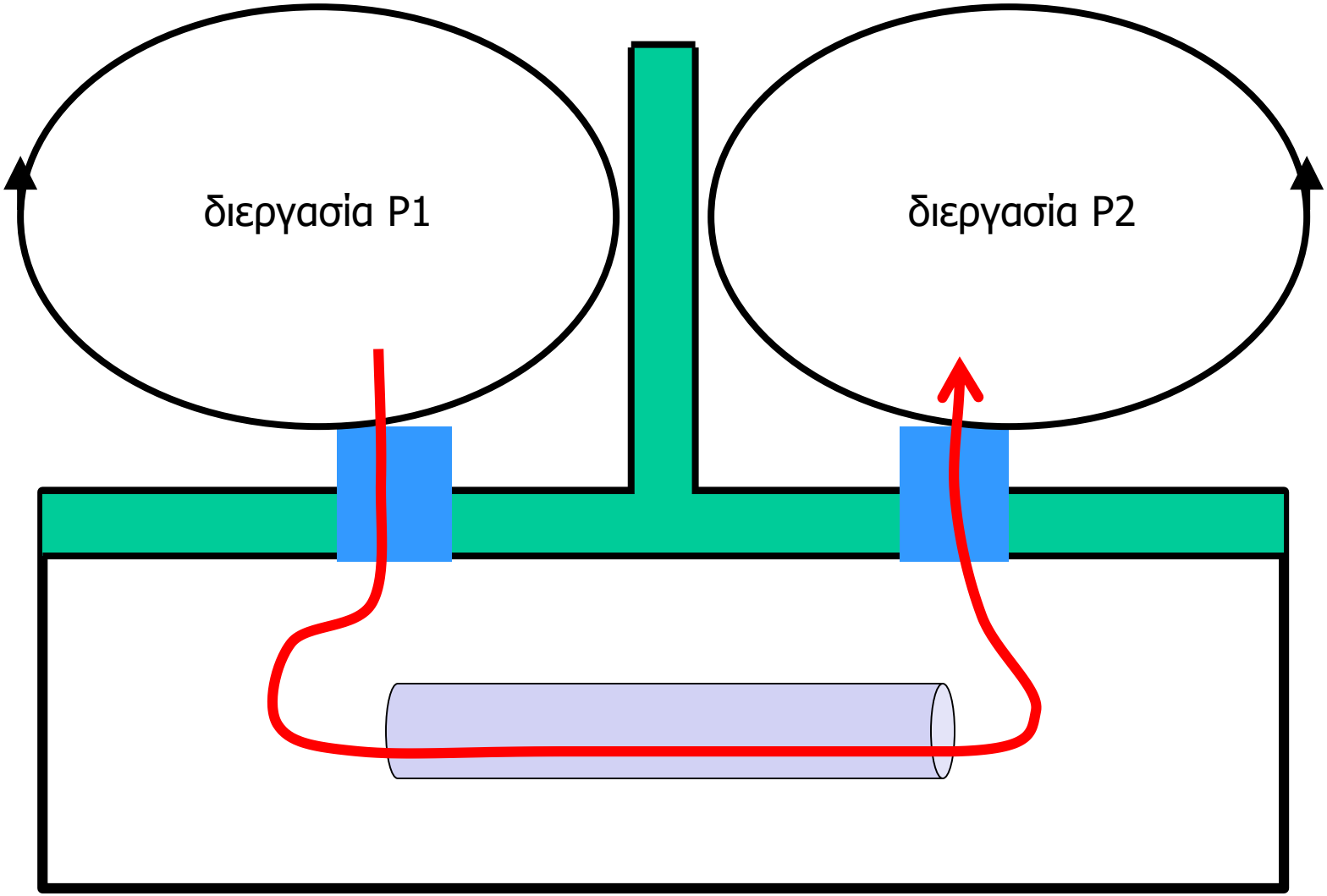
Αγωγοί/Σωλήνες (Pipes)

Τι είναι ένας αγωγός;

- **Μηχανισμός/συσκευή επικοινωνίας** διεργασιών
- Μετάδοση μιας **ροής** από bytes (σε μια κατεύθυνση)

- **First-in-first-out (FIFO)**: τα bytes βγαίνουν από τον αγωγό με την ίδια σειρά με την οποία μπήκαν
  - είναι μια **αποθήκη FIFO** μέσα στο λειτουργικό

- **Άκρο γραψίματος**: «μπαίνουν» τα δεδομένα
- **Άκρο διαβάσματος**: «βγαίνουν» τα δεδομένα


διεργασία P1
διεργασία P2

Προσπέλαση αγωγού

- Για κάθε ένα από τα δύο άκρα του αγωγού επιστρέφεται ξεχωριστός **περιγραφέας αρχείου**

- **Γράψιμο** στον αγωγό: `write`
- **Διάβασμα** από τον αγωγό: `read`
- **Κλείσιμο** περιγραφέα: `close`

- Ισχύουν πολλά από όσα ήδη γνωρίζουμε για το γράψιμο/διάβασμα δεδομένων σε/από αρχεία δίσκου
- Υπάρχουν όμως και σημαντικές **διαφορές** …

`read`, `write`, `close` σε αγωγό

- Η `read` **απομακρύνει** τα bytes από τον αγωγό
- Η `read` **μπλοκάρει** αν δεν υπάρχουν bytes μέσα στον αγωγό (μέχρι να γραφτούν bytes στον αγωγό)
  - ο αγωγός **δεν** έχει «τέλος» – όμως, βλέπε `close`
- Η `write` **μπλοκάρει** αν ο αγωγός έχει γεμίσει (μέχρι να διαβαστούν/απομακρυνθούν κάποια bytes)
  - ο αγωγός έχει **περιορισμένη χωρητικότητα**
- Η `close` έχει **παρενέργειες**
  - αν κλείσουν όλοι οι περιγραφείς ανάγνωσης του αγωγού (δεν υπάρχει κανείς να διαβάσει), η `write` αποτυγχάνει
  - αν κλείσουν όλοι οι περιγραφείς γραψίματος του αγωγού (δεν υπάρχει κανείς να γράψει), η `read` επιστρέφει 0
- **Δεν** υποστηρίζεται η `lseek` – γιατί;

Ανώνυμοι αγωγοί

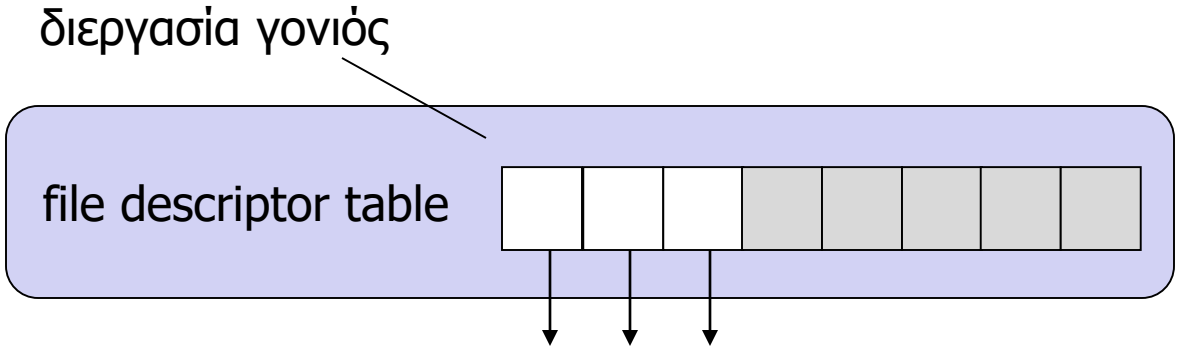
- `int pipe(int fd[2])`
- Δημιουργεί έναν **ανώνυμο** αγωγό

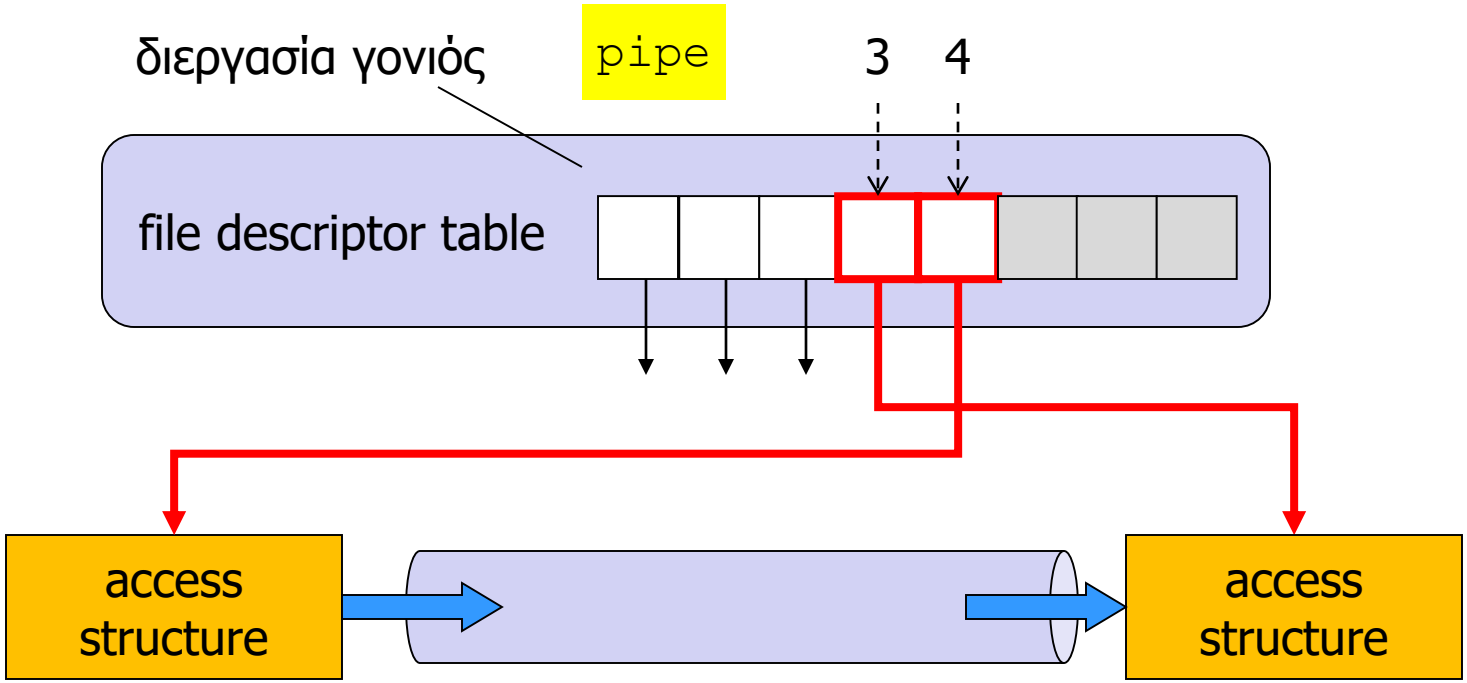
- Ο περιγραφέας για το **άκρο ανάγνωσης** αποθηκεύεται στο `fd[0]`
- Ο περιγραφέας για το **άκρο γραψίματος** αποθηκεύεται στο `fd[1]`

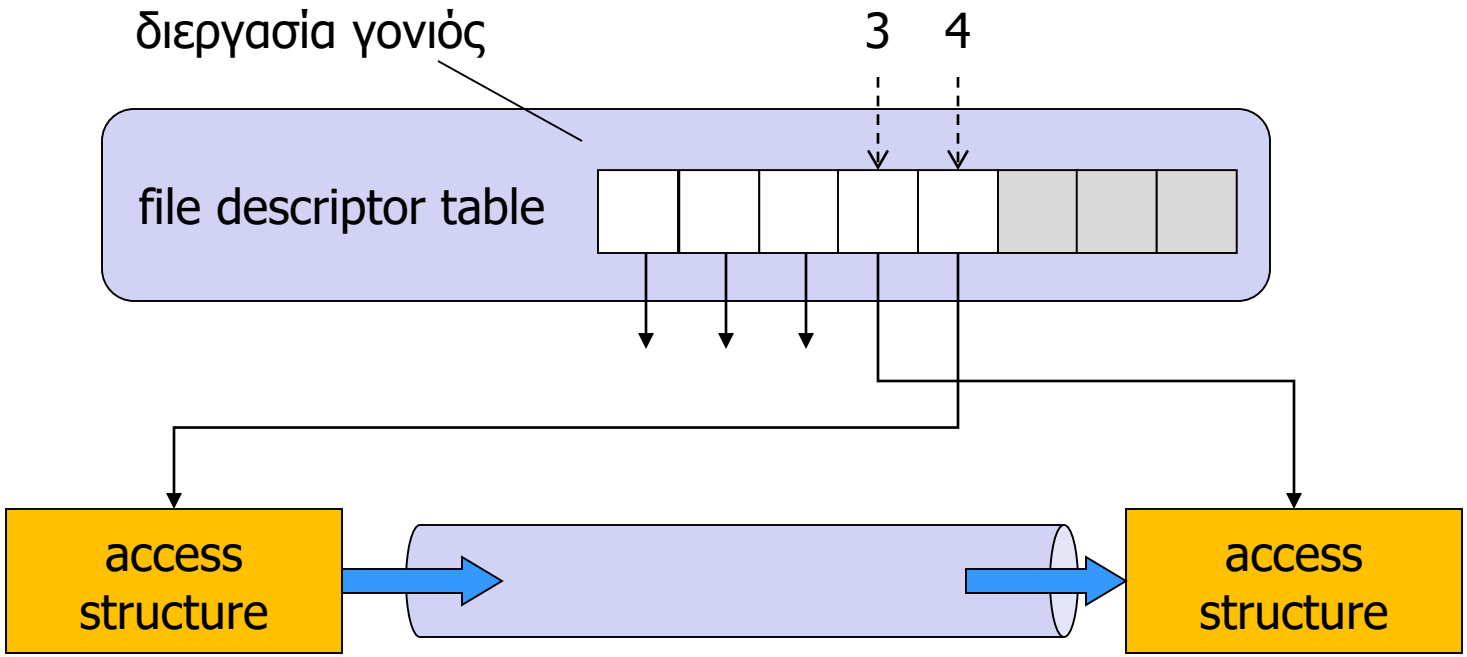
- Ο ανώνυμος αγωγός έχει **προσωρινή** υπόσταση
- Καταστρέφεται **αυτόματα** όταν κλείσει και ο τελευταίος περιγραφέας πάνω στον αγωγό

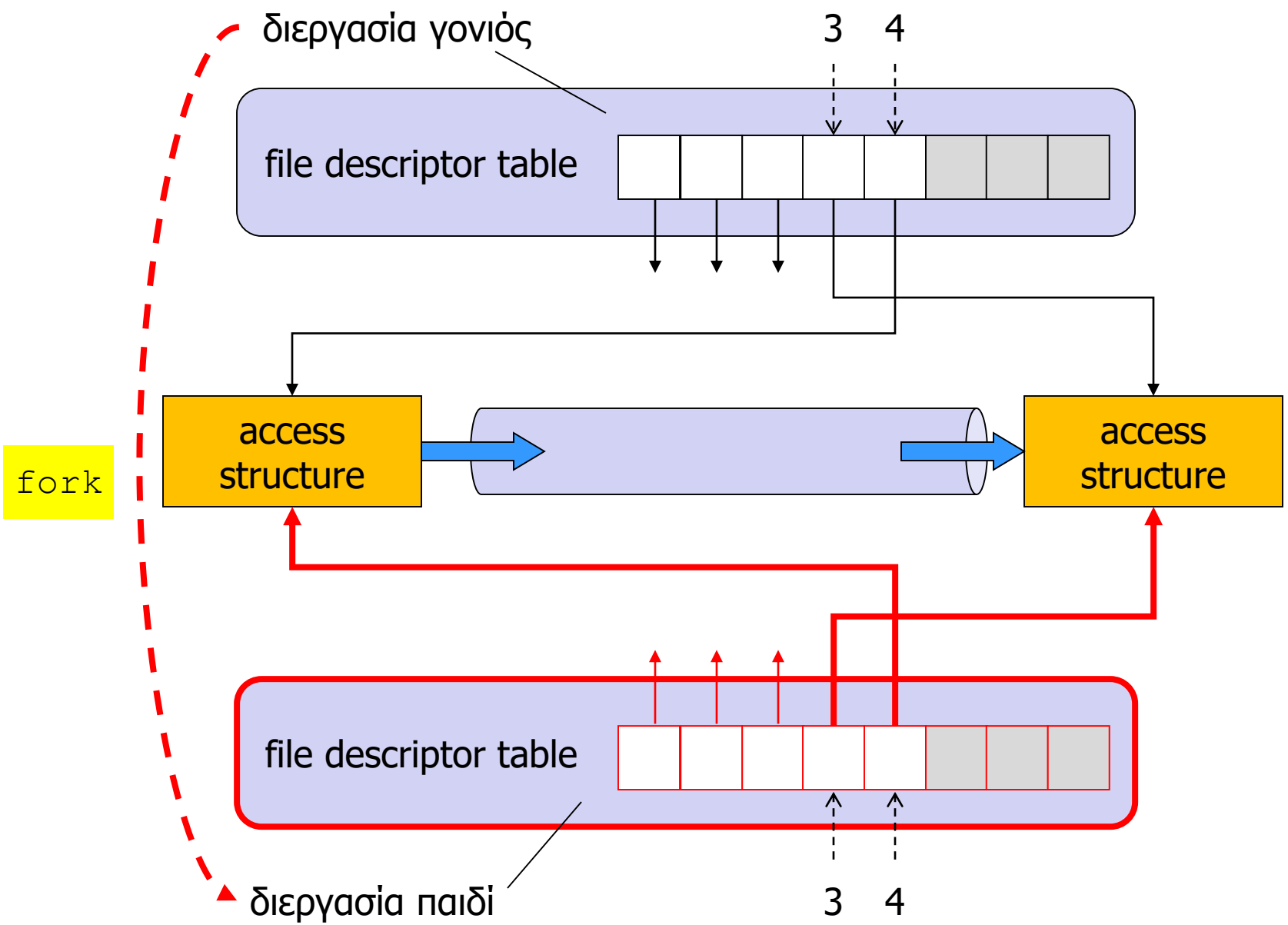
```
int main(int argc, char *argv[]) {
  int fd[2],n,i;

  n = atoi(argv[1]);
  pipe(fd);

  if (!fork()) {
    close(fd[0]); /* close pipe read end */
    for (i=0; i<n; i++) {
      res = write(fd[1],&i, sizeof(int));
      if (res != sizeof(int)) { return(1); }
    }
    return(0);
  }

  close(fd[1]); /* close pipe write end */
  while (1) {
    res = read(fd[0],&n,sizeof(int));
    if (res == 0) { break; }
    if (res != sizeof(int)) { return(1); }
    printf("parent: read %d\n",n);
  }
  return(0);
}
```

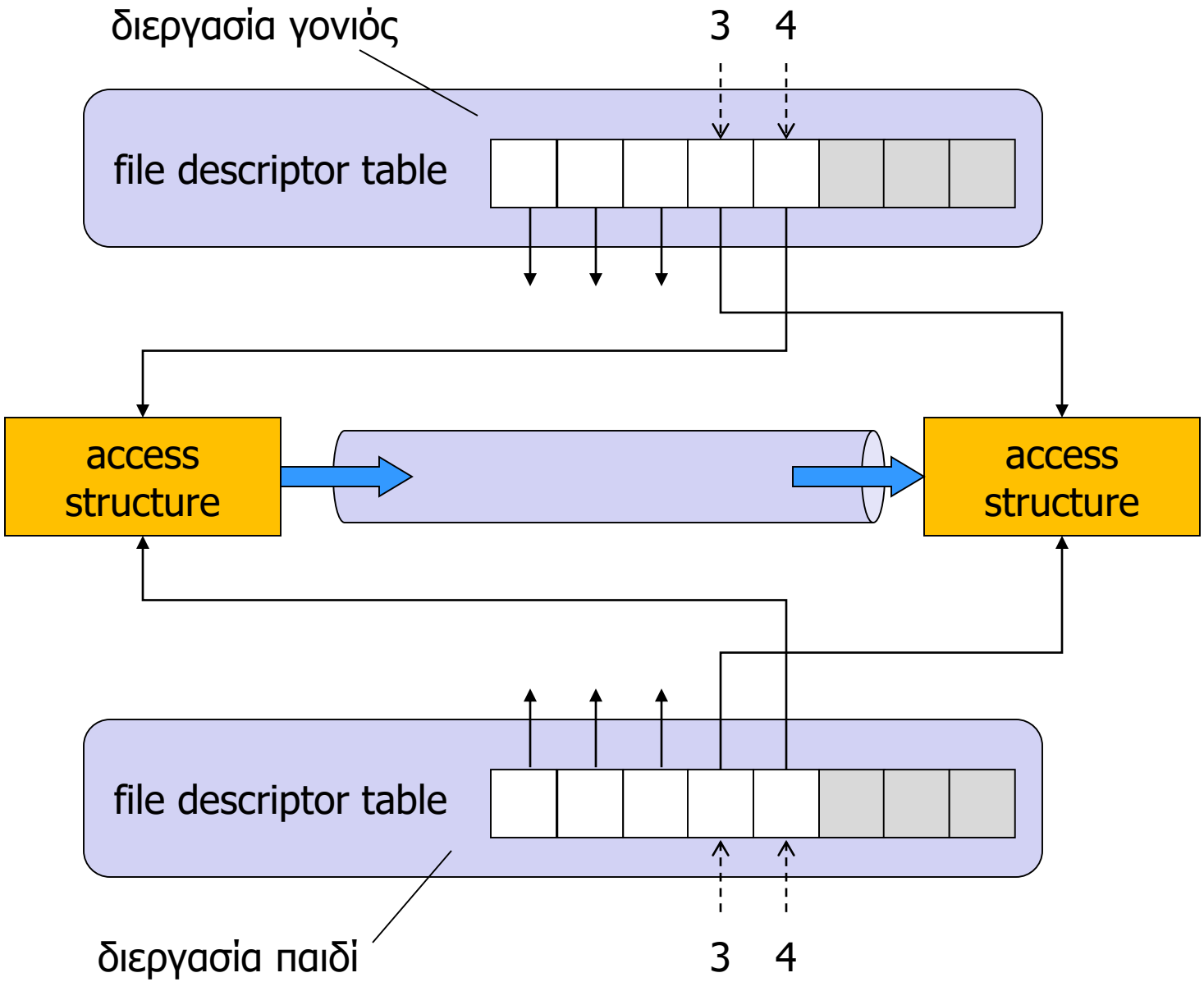
```
int main(int argc, char *argv[]) {
  int pid,fd[2],v1,v2,sum;

  v1 = atoi(argv[1]);
  v2 = atoi(argv[2]);

  pipe(fd);

  if (!(pid=fork())) {
    read(fd[0],&v1,sizeof(int));
    read(fd[0],&v2,sizeof(int));
    sum = v1+v2;
    write(fd[1],&sum,sizeof(int));
    return(0);
  }

  write(fd[1],&v1,sizeof(int));
  write(fd[1],&v2,sizeof(int));
  wait(pid,NULL,0);
  read(fd[0],&sum,sizeof(int));
  printf("parent: read %d\n",sum);
  return(0);
}
```

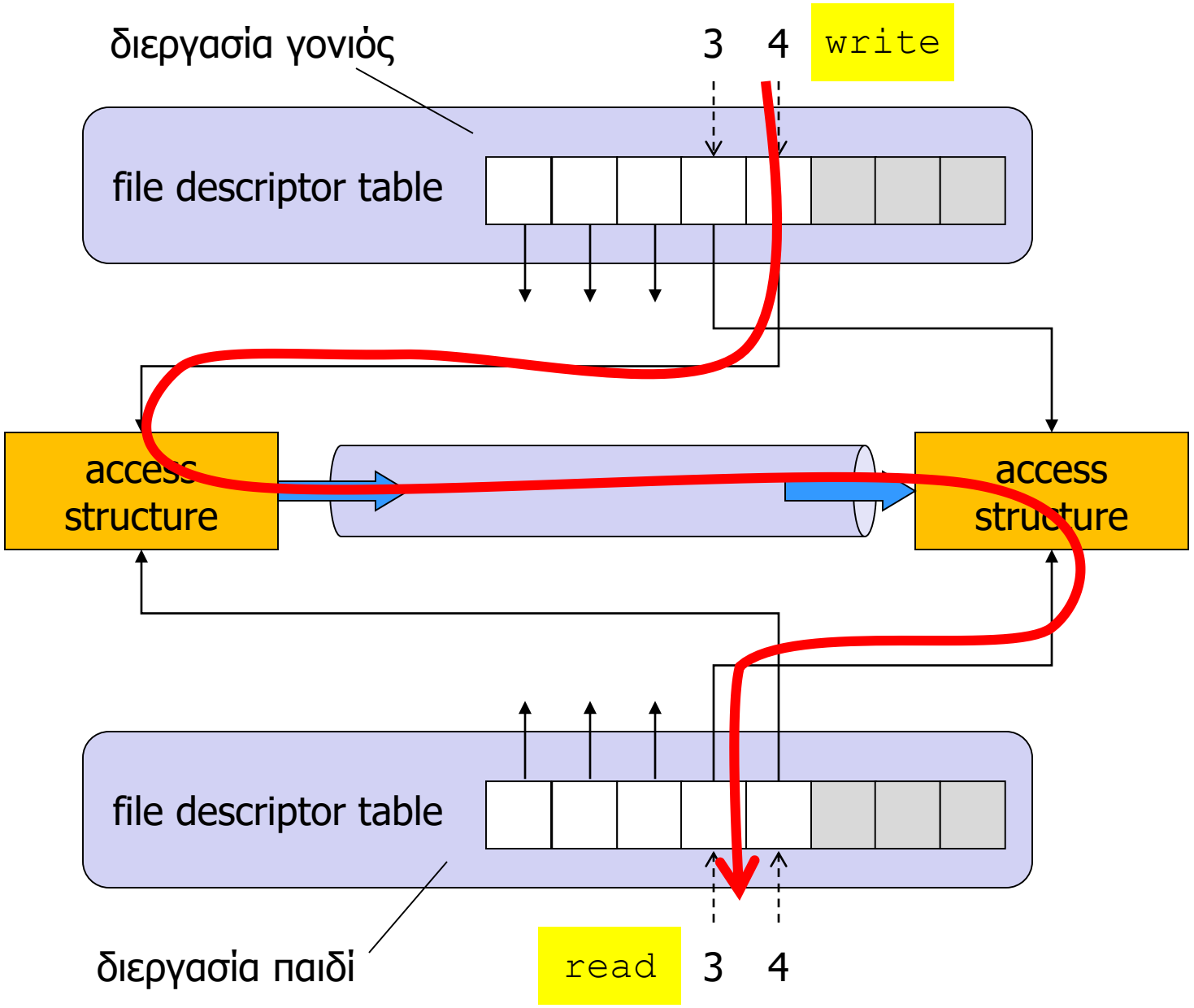

διεργασία γονιός
file descriptor table

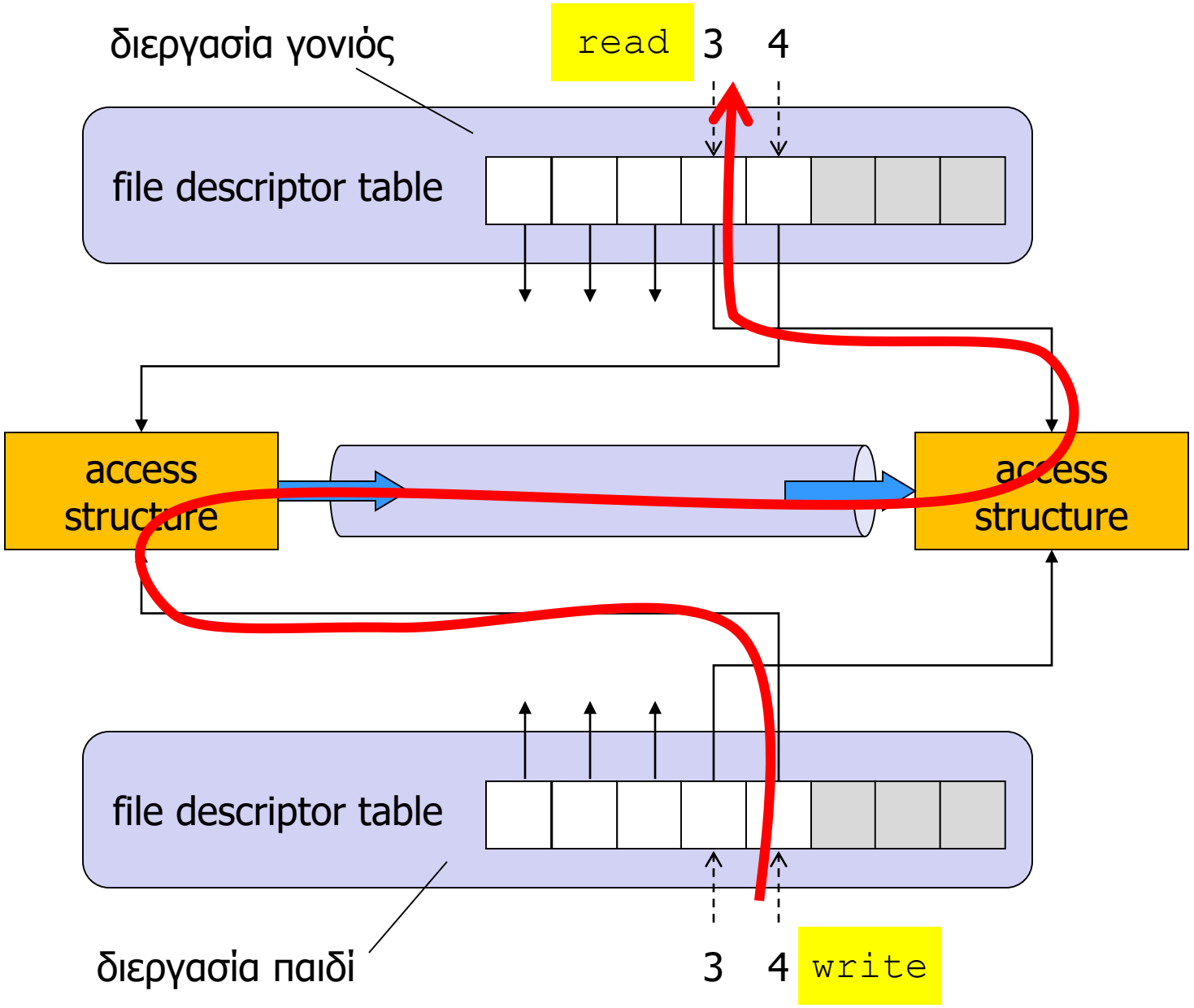

διεργασία γονιός
pipe
3
4
file descriptor table
access structure
access structure


διεργασία γονιός
3
4
file descriptor table
access structure
access structure


διεργασία γονιός
3
4
file descriptor table
fork
access structure
access structure
file descriptor table
διεργασία παιδί
3
4


διεργασία γονιός
3
4
file descriptor table
access structure
access structure
file descriptor table
διεργασία παιδί
3
4


διεργασία γονιός
3
4
write
file descriptor table
access structure
access structure
file descriptor table
διεργασία παιδί
read
3
4


διεργασία γονιός
read
3
4
file descriptor table
access structure
access structure
file descriptor table
διεργασία παιδί
3
4
write

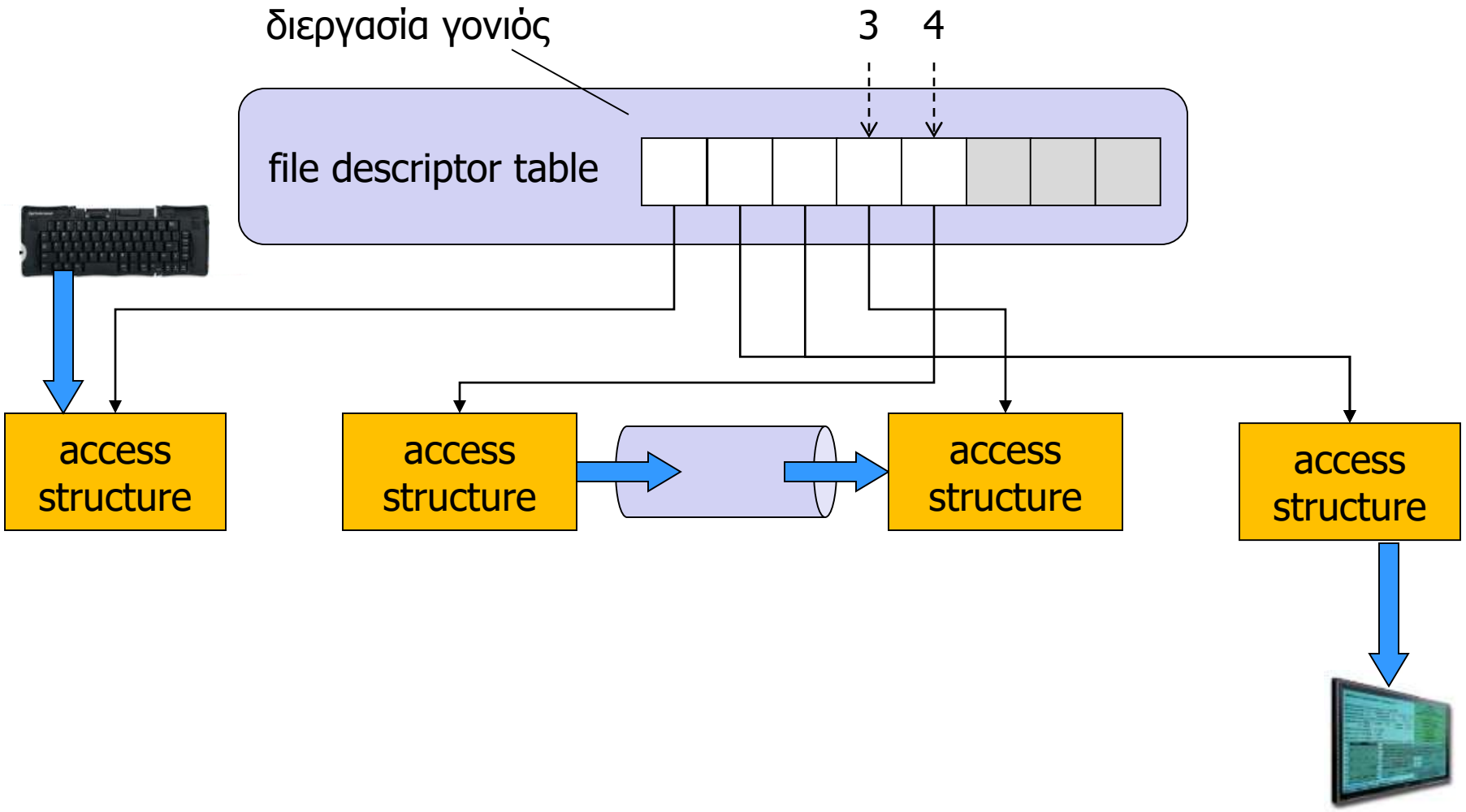
Ανακατεύθυνση εισόδου/εξόδου

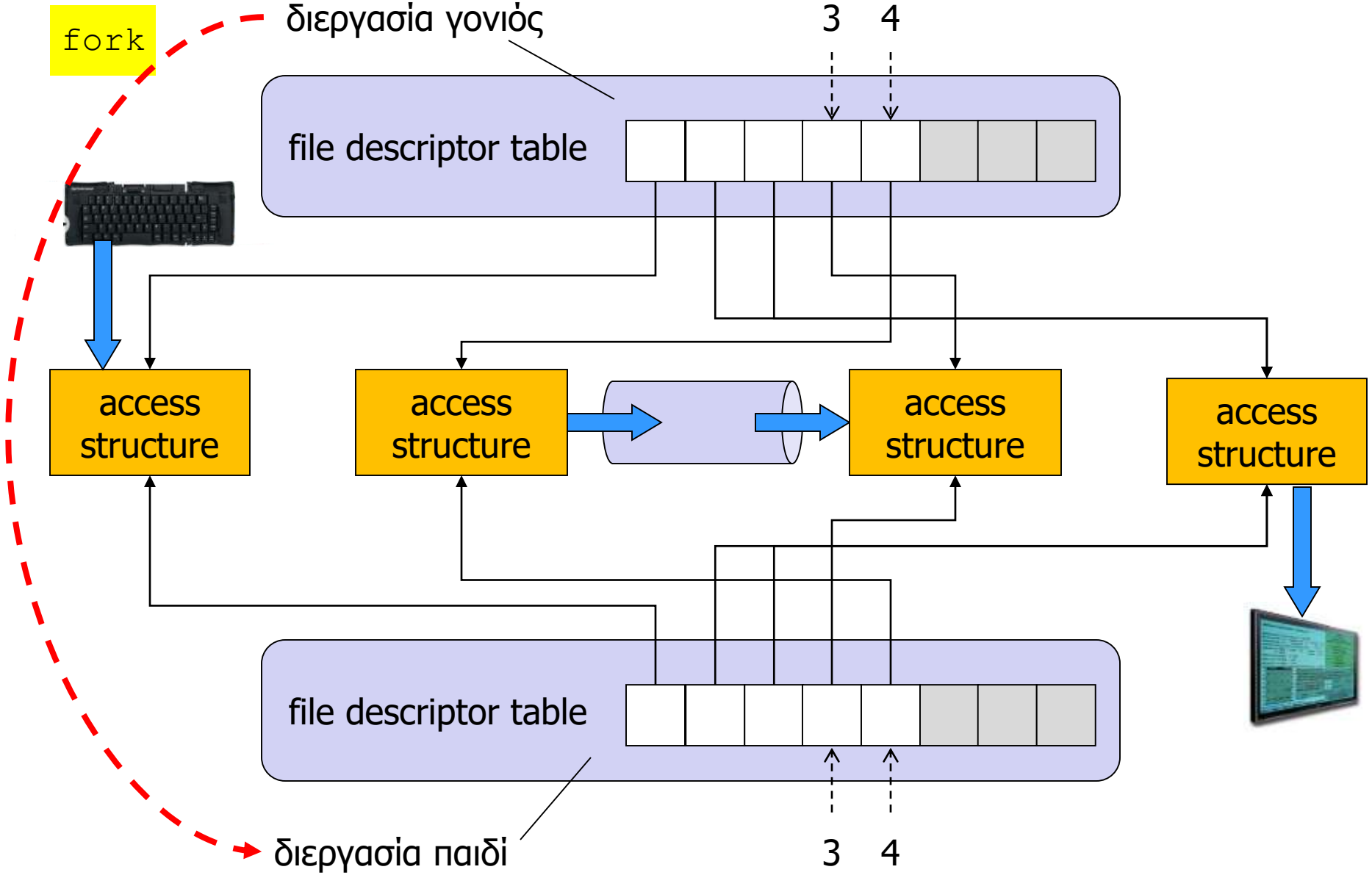
- Τα άκρα του αγωγού προσπελάζονται μέσω περιγραφέων αρχείων, όπως και ένα αρχείο

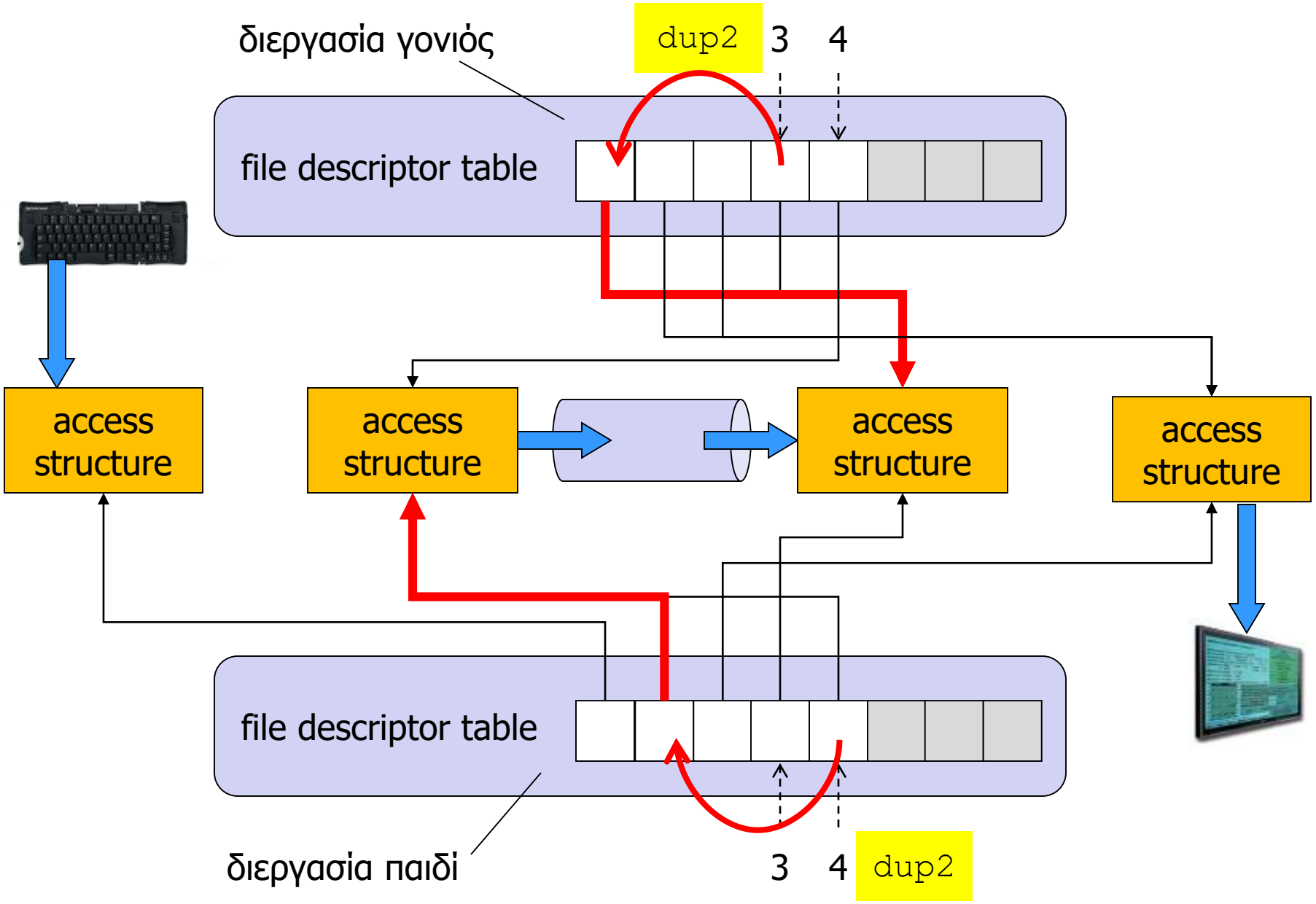
- Μπορεί να γίνει **ανακατεύθυνση** της καθιερωμένης Ε/Ε ενός προγράμματος σε έναν αγωγό, μέσω `dup2`
  - όπως γίνεται ανακατεύθυνση σε αρχείο

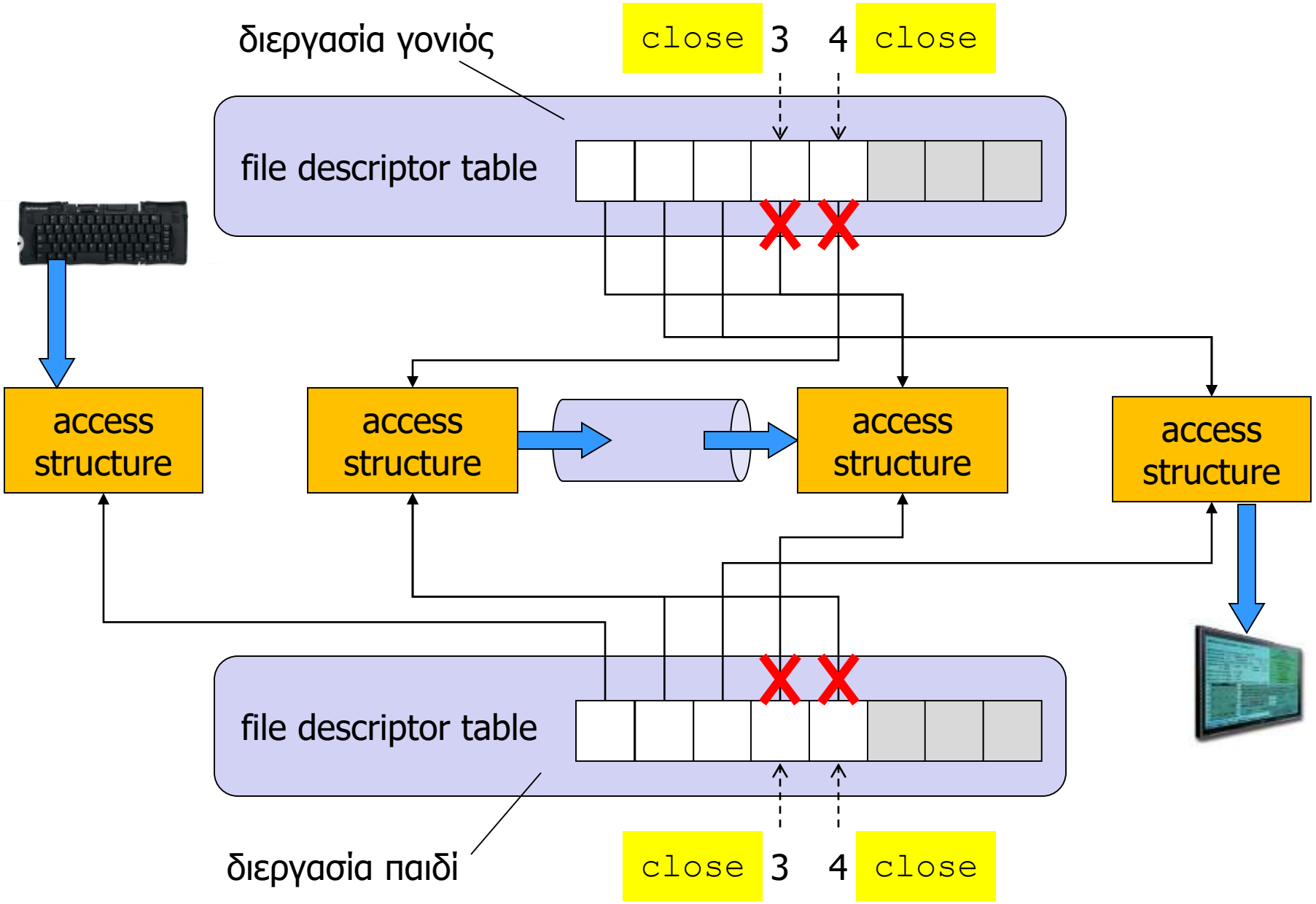
- Αντί το πρόγραμμα να διαβάζει δεδομένα από το τερματικό, μπορεί να τα διαβάζει από έναν αγωγό
- Αντί το πρόγραμμα να γράφει δεδομένα στο τερματικό, μπορεί να τα γράφει σε έναν αγωγό

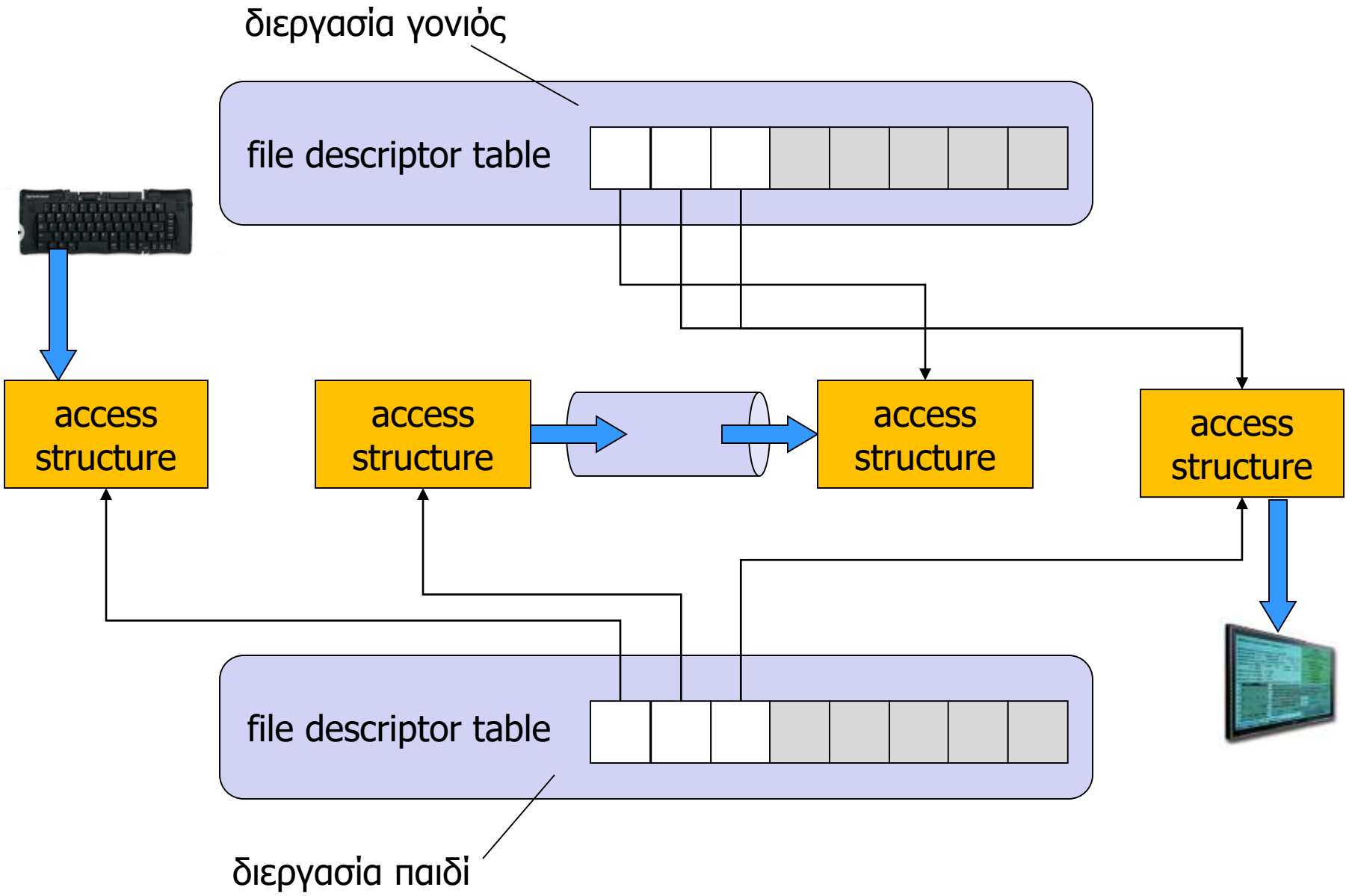
```
int main(int argc, char *argv[]) {
  int fd[2], i; char str[64];

  pipe(fd);

  if (!fork()) {
    dup2(fd[1],STDOUT_FILENO); /* redirect stdout */
    close(fd[0]); close(fd[1]); /* close pipe ends */
    for (i=1; i<argc; i++) {
      printf("%s\n",argv[i]);
    }
    return(0);
  }

  dup2(fd[0],STDIN_FILENO); /* redirect stdin */
  close(fd[0]); close(fd[1]); /* close pipe ends */
  while (scanf("%s",str) != EOF) {
    printf("%s\n",str);
  }
  return(0);
}
```

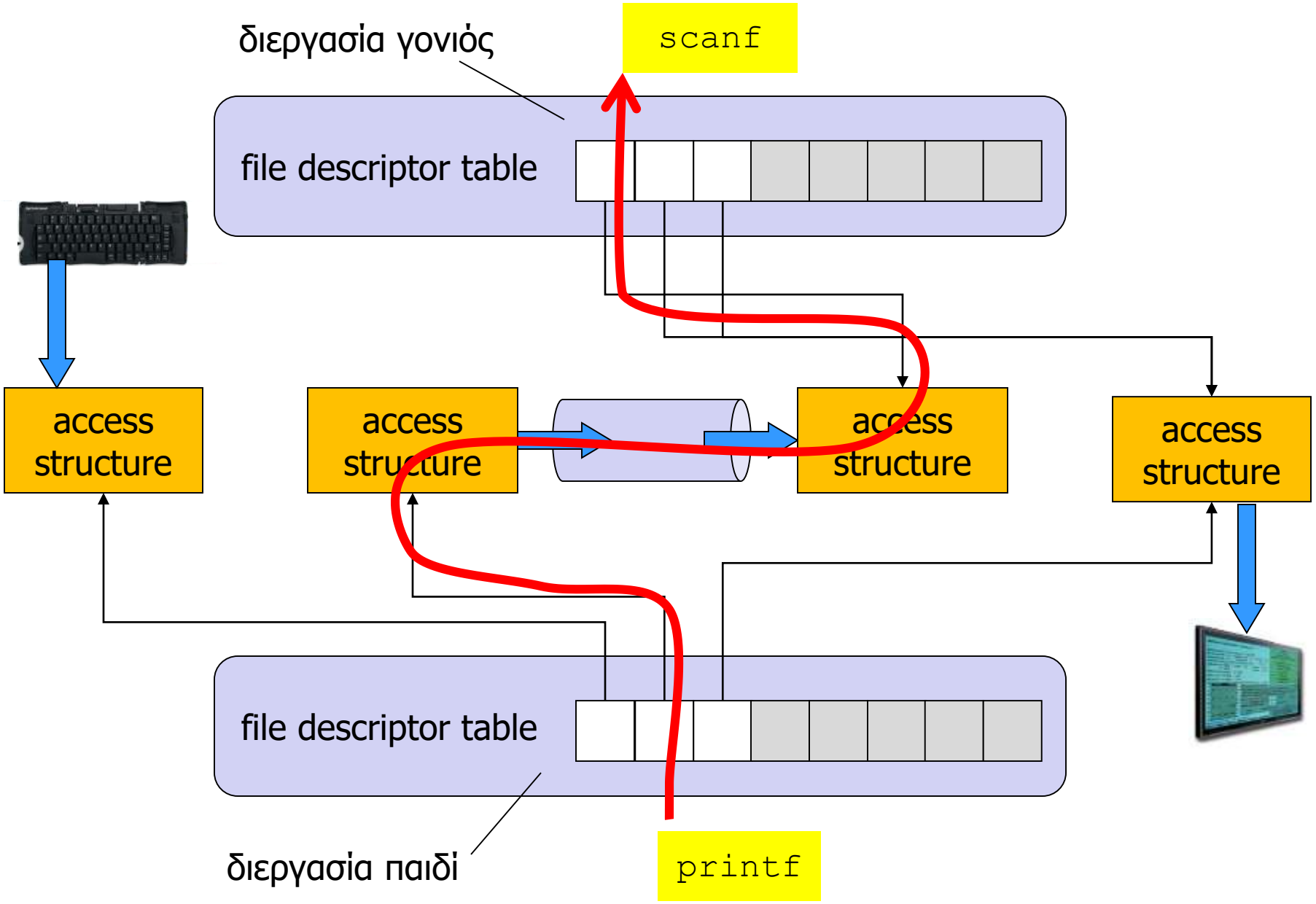

διεργασία γονιός
3
4
file descriptor table
access structure
access structure
access structure
access structure


fork
διεργασία γονιός
3
4
file descriptor table
access structure
access structure
access structure
access structure
file descriptor table
διεργασία παιδί
3
4


διεργασία γονιός
dup2
3
4
file descriptor table
access structure
access structure
access structure
access structure
file descriptor table
διεργασία παιδί
3
4
dup2


διεργασία γονιός
close 3 4 close
file descriptor table
access structure
access structure
access structure
access structure
file descriptor table
διεργασία παιδί
close 3 4 close


διεργασία γονιός
file descriptor table
access structure
access structure
access structure
access structure
file descriptor table
διεργασία παιδί


διεργασία γονιός
scanf
file descriptor table
access structure
access structure
access structure
access structure
file descriptor table
διεργασία παιδί
printf

Επώνυμοι αγωγοί

- `int mkfifo(cons char *path,mode_t perms);`
- Δημιουργεί έναν **επώνυμο** αγωγό
  - το όνομα και οι άδειες δίνονται όπως για ένα αρχείο
  - η κλήση **αποτυγχάνει** αν ο αγωγός υπάρχει ήδη
- Η δημιουργία περιγραφέων για τα άκρα γραψίματος και ανάγνωσης του αγωγού γίνεται μέσω της `open`
- Άκρο **ανάγνωσης**: με `O_RDONLY` (**μπλοκάρει** αν δεν υπάρχει ανοιχτό άκρο γραψίματος για τον αγωγό)
- Άκρο **γραψίματος**: με `O_WRONLY` (**μπλοκάρει** αν δεν υπάρχει ανοιχτό άκρο ανάγνωσης για τον αγωγό)
- Ο επώνυμος αγωγός έχει **μόνιμη** υπόσταση
- Καταστρέφεται **ρητά** με `unlink` όπως τα αρχεία

Χρησιμότητα επώνυμου αγωγού

- Προσυμφωνημένο «κανάλι/συχνότητα» επικοινωνίας
- Μπορεί να χρησιμοποιηθεί για την υλοποίηση σχημάτων επικοινωνίας client-server

- Οποιαδήποτε διεργασία μπορεί να ανοίξει έναν περιγραφέα στο άκρο ανάγνωσης γραψίματος του αγωγού

- Χρειάζεται κατάλληλος **συντονισμός** σε περίπτωση που πολλές διεργασίες επιθυμούν να γράψουν / διαβάσουν ταυτόχρονα στον / από τον αγωγό

server

```
int main(int argc, char *argv[]) {
  int fd,n; char str[N];

  mkfifo(argv[1],S_IRWXU); /* create named pipe */

  while (1) {
    fd = open(argv[1],O_RDONLY); /* open pipe read end */
    if (fd < 0) { perror("server: open"); return(1); }
    while ((n = read(fd,str,N-1)) > 0) {
      str[n] = '\0'; printf("server read: %s\n",str);
    }
    if (n < 0) { perror("server: read"); return(1); }
    printf("server: end of input\n");
    close(fd);
  }
}
```

client

```
int main(int argc, char *argv[]) {
  int fd,i;

  fd = open(argv[1],O_WRONLY);  /* open pipe write end */
  if (fd < 0) { perror("client: open"); return(1); }
  for (i=1; i<argc; i++) {
    write(fd,argv[i],strlen(argv[i]));
  }
  return(0);
}
```