Βασικές λειτουργίες συστήματος πάνω σε αρχεία δεδομένων

Βασικές λειτουργίες (1)

- `int open(const char *path,`
  `int flags, mode_t perms)`
  - ανοίγει το αρχείο με όνομα `path` (αν συμπεριλαμβάνεται ένα μονοπάτι καταλόγων, αυτό πρέπει ήδη να υφίσταται)
- Η επιθυμητή πρόσβαση προσδιορίζεται μέσω `flags`
  - `O_RDONLY`,`O_WRONLY`,`O_RDWR`: διάβασμα, γράψιμο, και τα δύο
  - `O_APPEND`,`O_TRUNC`: προσθήκη στο τέλος, κόψιμο αρχείου
  - `O_CREAT`: δημιουργία αρχείου, αν δεν υπάρχει ήδη
  - `O_EXCL`: μαζί με `O_CREAT`, αποτυχία αν το αρχείο υπάρχει
- Οι άδειες πρόσβασης προσδιορίζονται μέσω `perms`
  - οκταδική τιμή (π.χ. `0700`) ή συμβολική σταθερά (π.χ. `S_IRWXU`)
  - λαμβάνονται υπόψη μόνο όταν δημιουργείται ένα **νέο** αρχείο

Περιγραφείς αρχείων (file descriptors)

- Σε περίπτωση επιτυχίας, η `open` επιστρέφει τον λεγόμενο **περιγραφέα αρχείου** (file descriptor)
- Οι περιγραφείς αρχείων είναι **ακέραιοι**
  - **δεν** υπάρχει ειδικός τύπος δεδομένων για αρχεία

- Ο περιγραφέας αρχείου που επιστρέφεται χρησιμοποιείται στην συνέχεια ως αναφορά (reference) στο αρχείο
- Όλες οι βασικές λειτουργίες αρχείων δέχονται ως παράμετρο έναν περιγραφέα αρχείου (ακέραιο)
  - π.χ. μια ακέραια τιμή που επέστρεψε η `open`

- Περισσότερα για τους περιγραφείς αρχείων, αργότερα ...

Βασικές λειτουργίες (2)

- `ssize_t read(int fd, void *buf, size_t n)`
  - **ανάγνωση** δεδομένων από το αρχείο
- `ssize_t write(int fd, void *buf, size_t n)`
  - **εγγραφή** δεδομένων στο αρχείο
- `off_t lseek(int fd, off_t pos, int whence)`
  - **μετακίνηση** θέσης ανάγνωσης/εγγραφής
- `int ftruncate(int fd, off_t length)`
  - **κόψιμο/επέκταση** του αρχείου
- `int fsync(int fd)`
  - **σύγχρονη αποθήκευση** δεδομένων στον δίσκο
- `int close(int fd)`
  - **κλείσιμο** του περιγραφέα αρχείου

```
int fd;
char str[]="hello world";

fd=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
write(fd,str,5);
write(fd,&str[5],6);
write(fd," :-)",4);
…
```

str

h	e	l	l	o		w	o	r	l	d	\0
68	65	6C	6C	6F	20	77	6F	72	6C	64	00

← ερμηνεία ως ASCII

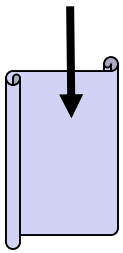
← τιμή bytes στην μνήμη

```
int fd;
char str[]="hello world";

fd=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
write(fd,str,5);
write(fd,&str[5],6);
write(fd," :-)",4);
…
```

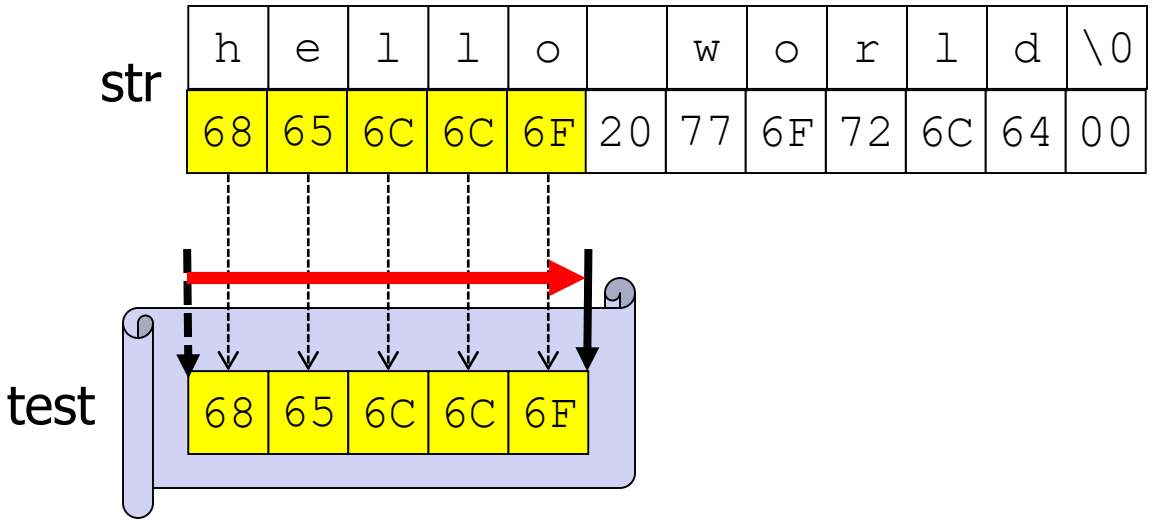
str

h	e	l	l	o		w	o	r	l	d	\0
68	65	6C	6C	6F	20	77	6F	72	6C	64	00

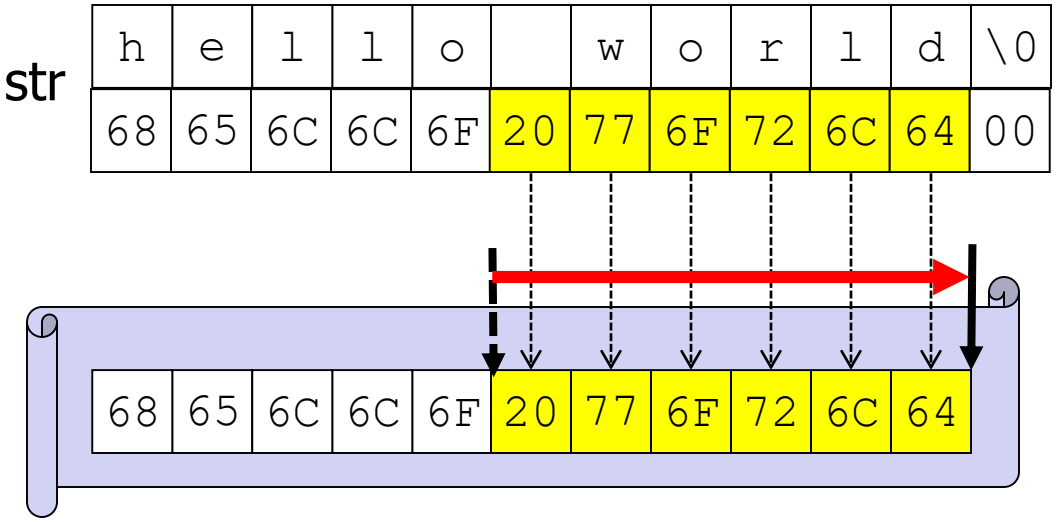
test



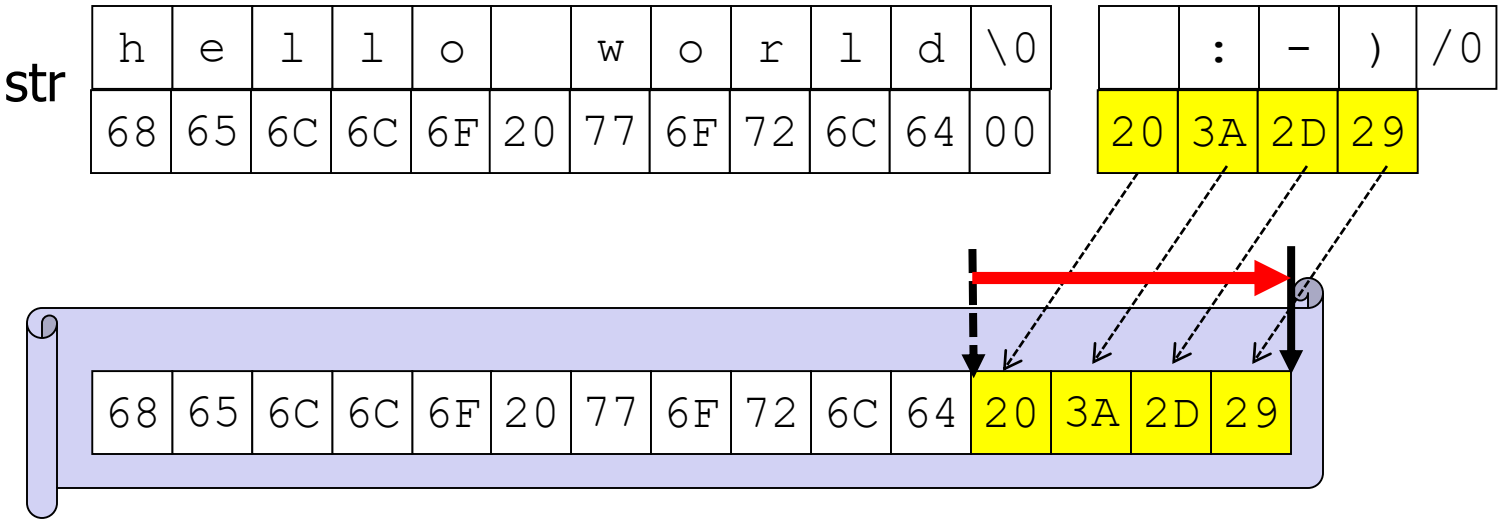
```
int fd;
char str[]="hello world";

fd=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
write(fd,str,5);
write(fd,&str[5],6);
write(fd," :-)",4);
…
```



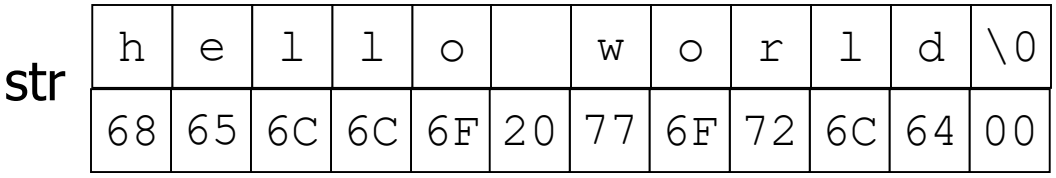
```
int fd;
char str[]="hello world";

fd=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
write(fd,str,5);
write(fd,&str[5],6);
write(fd," :-)",4);
…
```

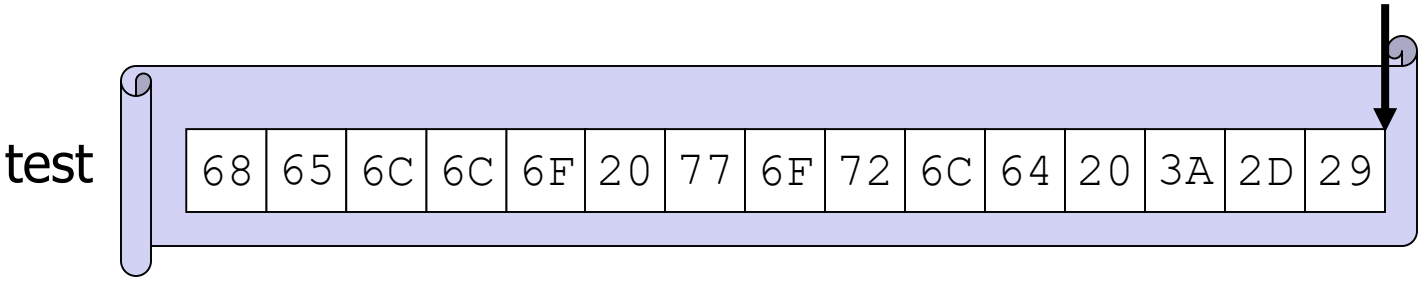


```
int fd;
char str[]="hello world";

fd=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
write(fd,str,5);
write(fd,&str[5],6);
write(fd," :-)",4);
…
```

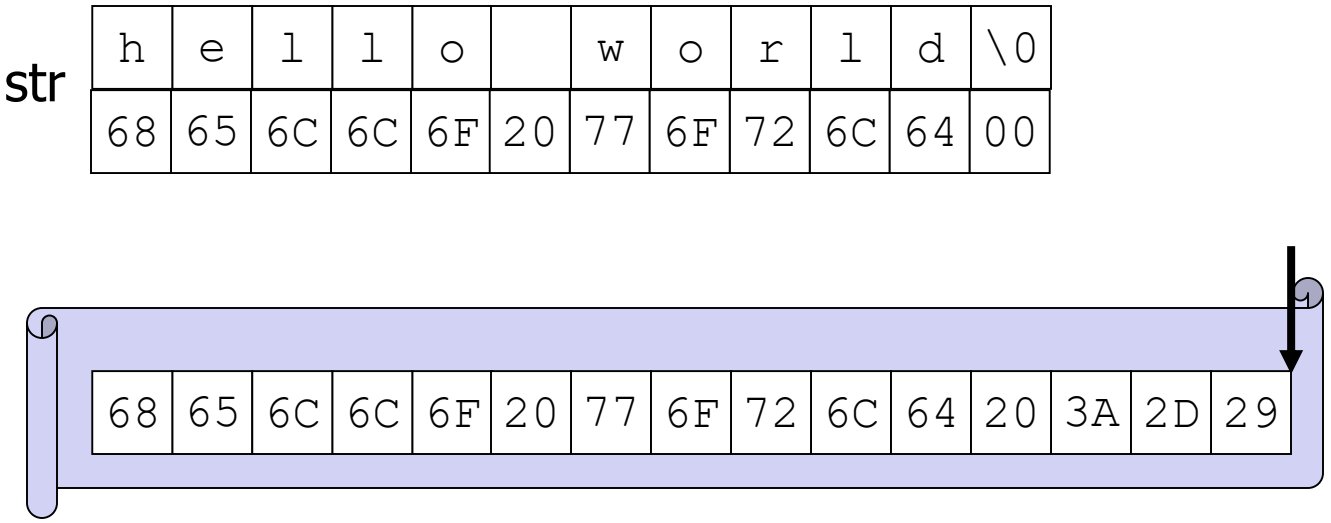


```
int fd;
char str[]="hello world";

fd=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
write(fd,str,5);
write(fd,&str[5],6);
write(fd," :-)",4);
…
```

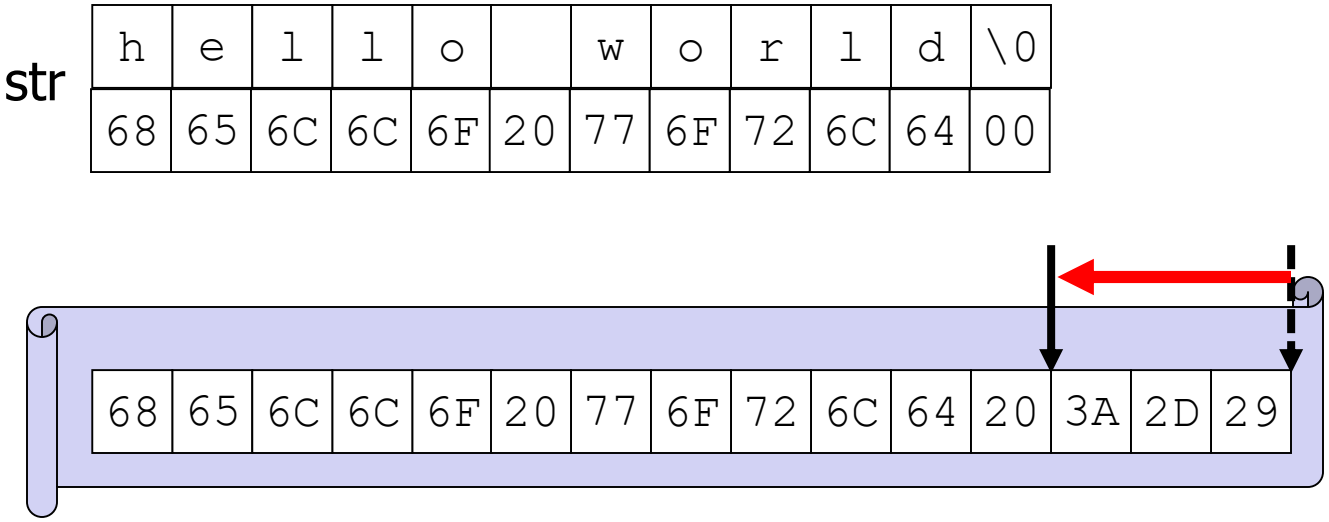




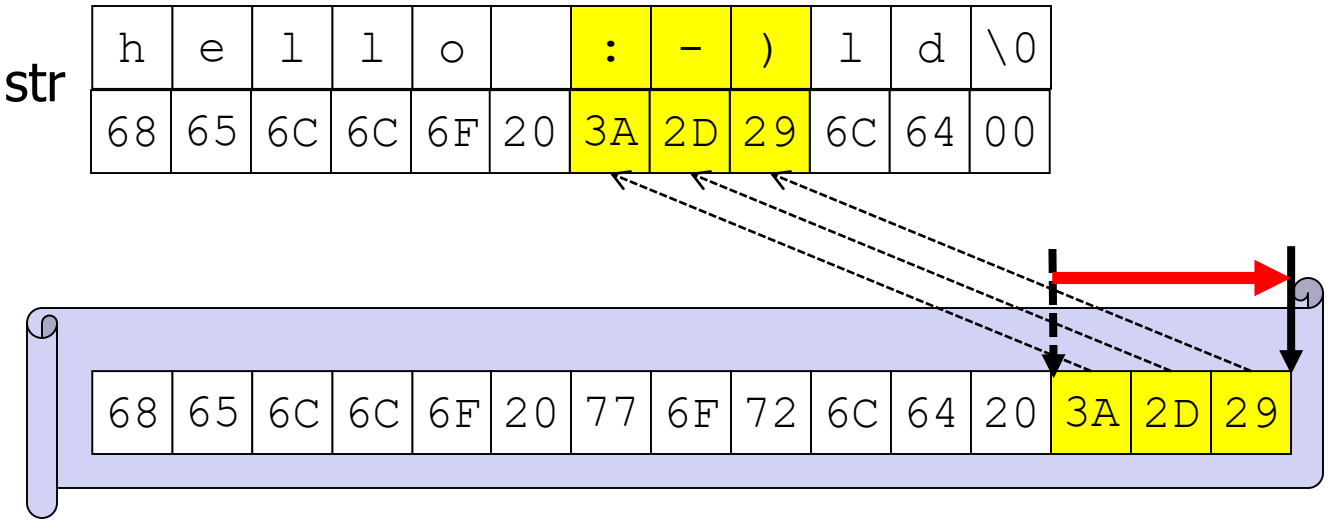
```
…
lseek(fd,(off_t)-3,SEEK_CUR);
read(fd,&str[6],3);
ftruncate(fd,5);
write(fd,str,5);
close(fd);
```



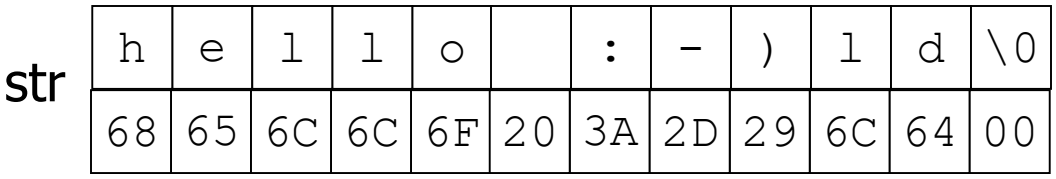
```
…
lseek(fd,(off_t)-3,SEEK_CUR);
read(fd,&str[6],3);
ftruncate(fd,5);
write(fd,str,5);
close(fd);
```

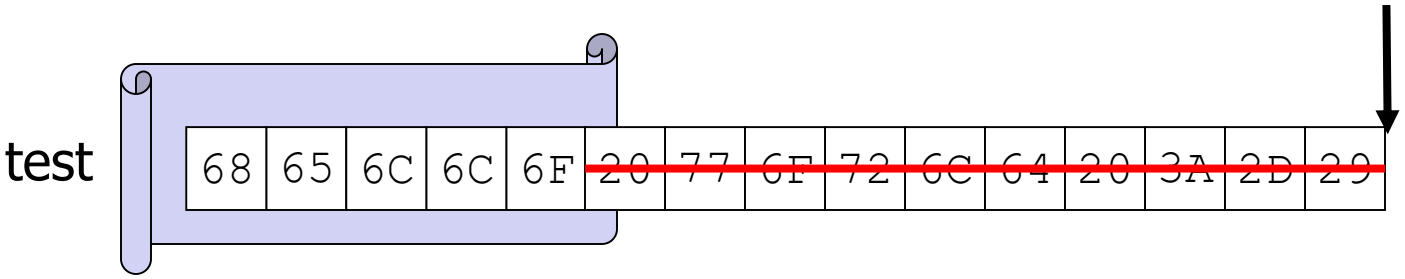


```
…
lseek(fd,(off_t)-3,SEEK_CUR);
read(fd,&str[6],3);
ftruncate(fd,5);
write(fd,str,5);
close(fd);
```

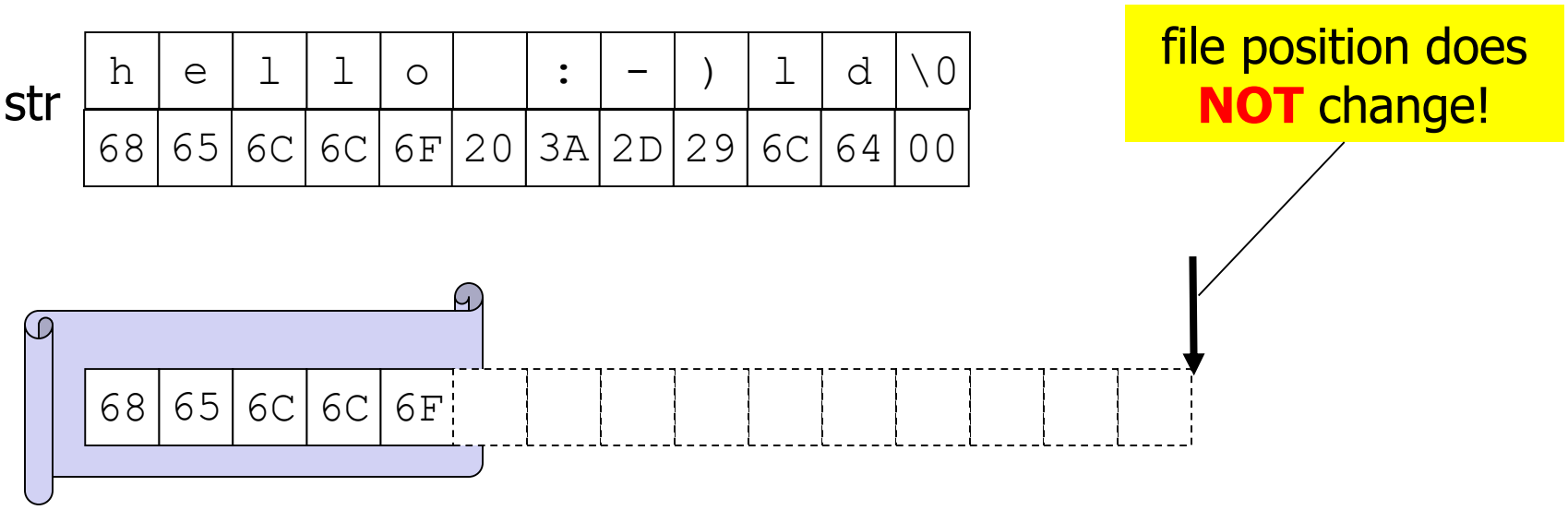


```
…
lseek(fd,(off_t)-3,SEEK_CUR);
read(fd,&str[6],3);
ftruncate(fd,5);
write(fd,str,5);
close(fd);
```

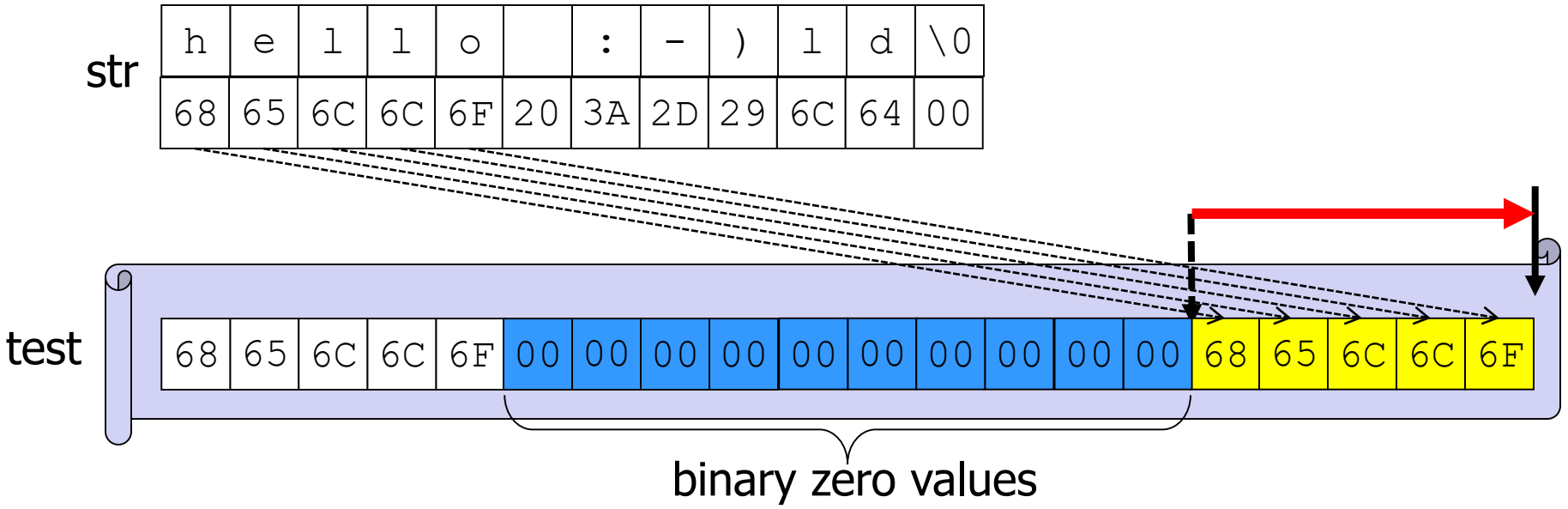




```
…
lseek(fd,(off_t)-3,SEEK_CUR);
read(fd,&str[6],3);
ftruncate(fd,5);
write(fd,str,5);
close(fd);
```



```
…
lseek(fd,(off_t)-3,SEEK_CUR);
read(fd,&str[6],3);
ftruncate(fd,5);
write(fd,str,5);
close(fd);
```



```
…
lseek(fd,(off_t)-3,SEEK_CUR);
read(fd,&str[6],3);
ftruncate(fd,5);
write(fd,str,5);
close(fd);
```

str

h	e	l	l	o		:	-	)	l	d	\0
68	65	6C	6C	6F	20	3A	2D	29	6C	64	00

test

68	65	6C	6C	6F	00	00	00	00	00	00	00	00	00	00	68	65	6C	6C	6F

Προσοχή!

- Οι `read`/`write` μπορεί να **μην** διαβάσουν/γράψουν όσα bytes ζήτησε ο προγραμματιστής
  - αυτό δεν είναι (απαραίτητα) λάθος ή πρόβλημα
- Πρέπει να **ελέγχεται** η τιμή επιστροφής, και αν χρειαστεί να **επαναληφθεί** η κλήση ώστε να διαβαστούν/γραφτούν τα υπόλοιπα bytes

- Οι `write`/`close` **δεν** δίνουν εγγυήσεις σχετικά με την αποθήκευση των δεδομένων στον δίσκο
  - το λειτουργικό κρατά δεδομένα σε «κρυφή» μνήμη και τα αποθηκεύει στο δίσκο **ασύγχρονα**/αργότερα – γιατί;
- Η **σύγχρονη** αποθήκευση δεδομένων στο μέσο αποθηκευσης (δίσκο) γίνεται με `fsync`

Απομάκρυνση αρχείου

- `int unlink(const char *path)`
  - καταργεί τον σύνδεσμο στο αρχείο
- Αν δεν υπάρχει άλλος σύνδεσμος σε αυτό το αρχείο, το αρχείο καθίσταται απροσπέλαστο και διαγράφεται
  - όταν κλείσει και ο τελευταίος περιγραφέας σε αυτό

- Δημιουργία προσωρινού αρχείου που διαγράφεται όταν το πρόγραμμα τερματιστεί ή κλείσει τον περιγραφέα

```
fd = open("tmp",O_RDWR|O_CREAT|O_EXCL,0077);
unlink("tmp");
…
close(fd);
```

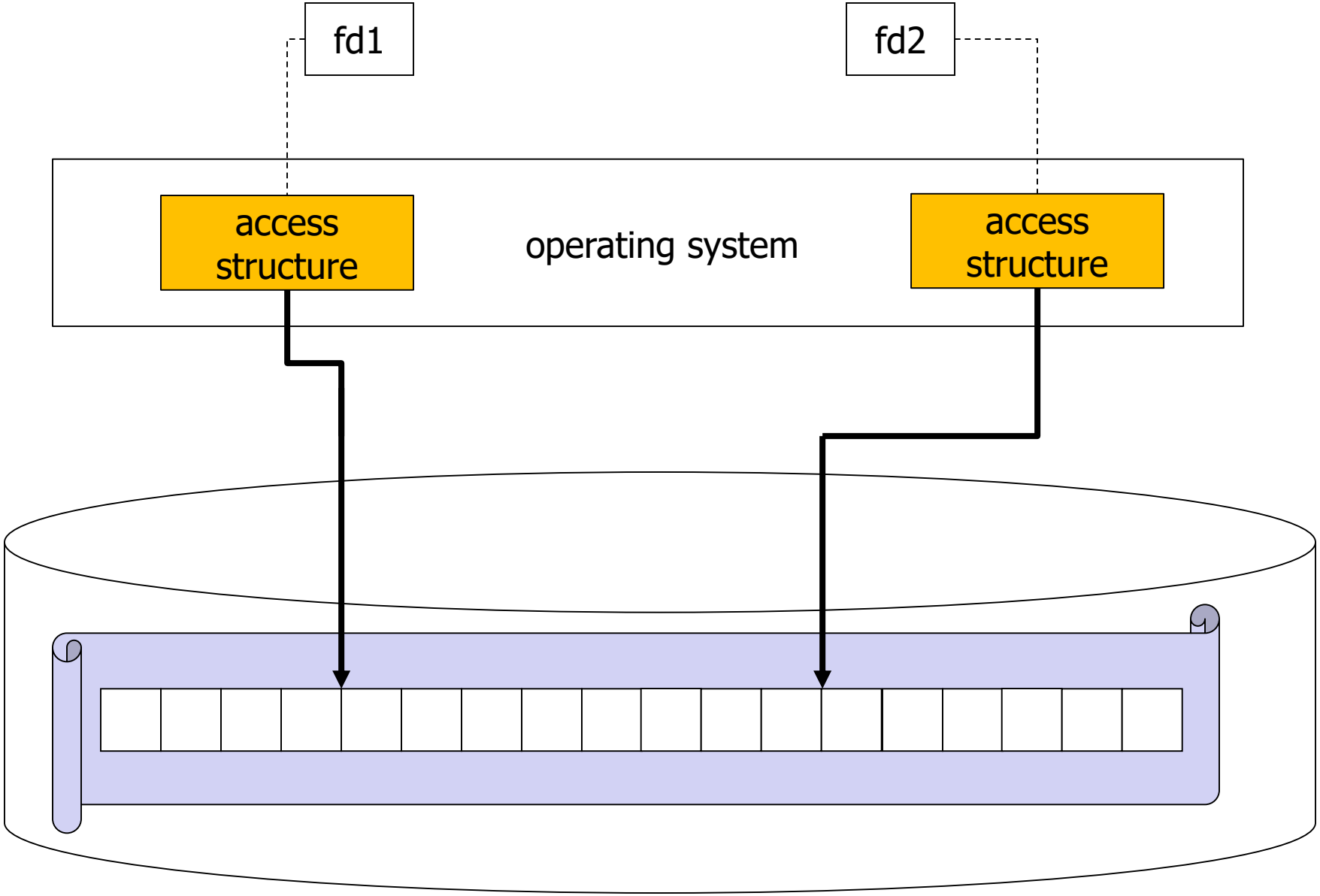
Καθιερωμένοι περιγραφείς αρχείων

- Υπάρχουν 3 συγκεκριμένοι περιγραφείς αρχείων, που έχουν καθιερωμένη/προκαθορισμένη χρήση
- **Καθιερωμένη είσοδος** (standard input)
  - `STDIN_FILENO` (`0`)
- **Καθιερωμένη έξοδος** (standard output)
  - `STDOUT_FILENO` (`1`)
- **Καθιερωμένη έξοδος λαθών** (standard error)
  - `STDERR_FILENO` (`2`)
- Αντιστοιχούν (συνήθως) στην «συσκευή τερματικού» (tty) που χρησιμοποιεί ο χρήστης του προγράμματος
  - `0` -> πληκτρολόγιο, `1`,`2` -> οθόνη
  - αυτά **δεν** είναι αρχεία

Περιγραφείς Αρχείων (file descriptors)

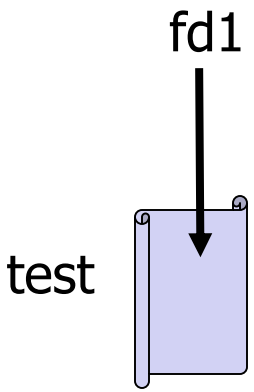
Περιγραφέας αρχείου

- Όταν ανοίγεται ένα αρχείο, το λειτουργικό επιστρέφει έναν ακέραιο **περιγραφέα** (file descriptor) μέσω του οποίου πραγματοποιούνται οι λειτουργίες πρόσβασης
  - δίνεται ως παράμετρος στις `read`, `write`, `close`, ...
- Ένα πρόγραμμα μπορεί να δημιουργεί/διατηρεί **ξεχωριστούς** περιγραφείς για το **ίδιο** αρχείο
- **Κάθε** περιγραφέας «έχει» την **δική του θέση** ανάγνωσης/εγγραφής πάνω στο αρχείο
- Λειτουργίες που εκτελούνται μέσω ενός περιγραφέα **δεν επηρεάζουν** την θέση ανάγνωση/εγγραφής των άλλων περιγραφέων

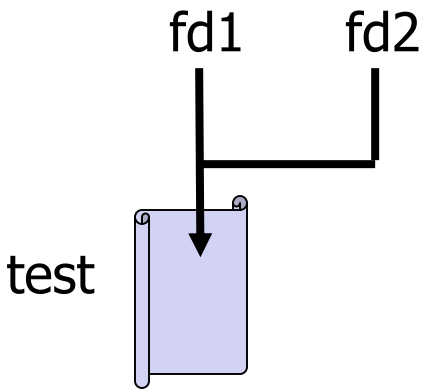

fd1
fd2
access structure
operating system
access structure

```
int fd1,fd2;

fd1=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2=open("test",O_RDWR,0);
write(fd1,"a boring lecture",16);
write(fd2,"a superb",8);
ftruncate(fd1,2);
write(fd2,"lecture",7);
close(fd1);
close(fd2);
```

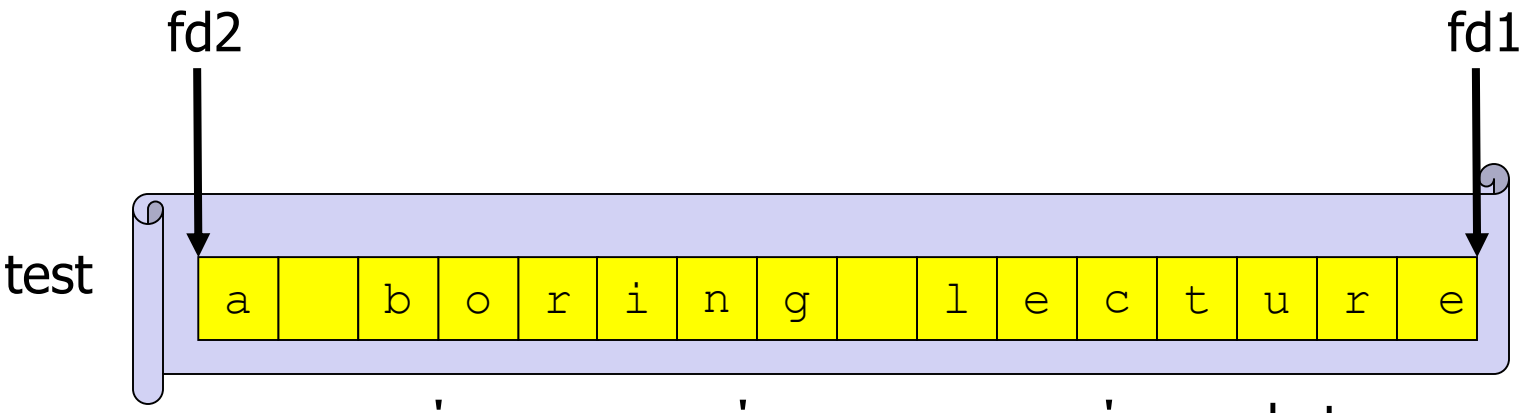
```
int fd1,fd2;

fd1=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2=open("test",O_RDWR,0);
write(fd1,"a boring lecture",16);
write(fd2,"a superb",8);
ftruncate(fd1,2);
write(fd2,"lecture",7);
close(fd1);
close(fd2);
```



```
int fd1,fd2;

fd1=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2=open("test",O_RDWR,0);
write(fd1,"a boring lecture",16);
write(fd2,"a superb",8);
ftruncate(fd1,2);
write(fd2,"lecture",7);
close(fd1);
close(fd2);
```

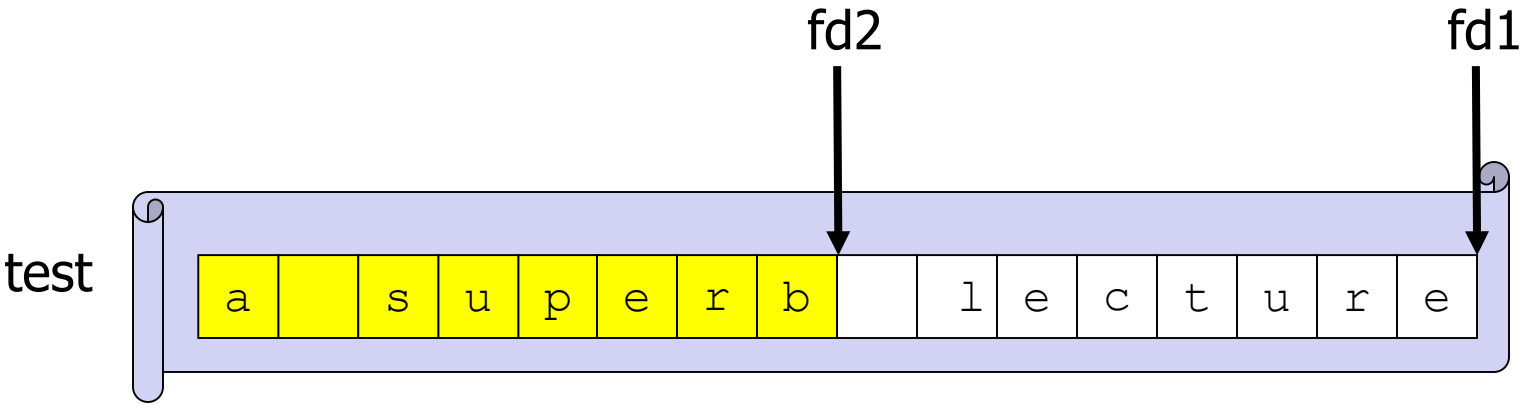


```
int fd1,fd2;

fd1=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2=open("test",O_RDWR,0);
write(fd1,"a boring lecture",16);
write(fd2,"a superb",8);
ftruncate(fd1,2);
write(fd2,"lecture",7);
close(fd1);
close(fd2);
```

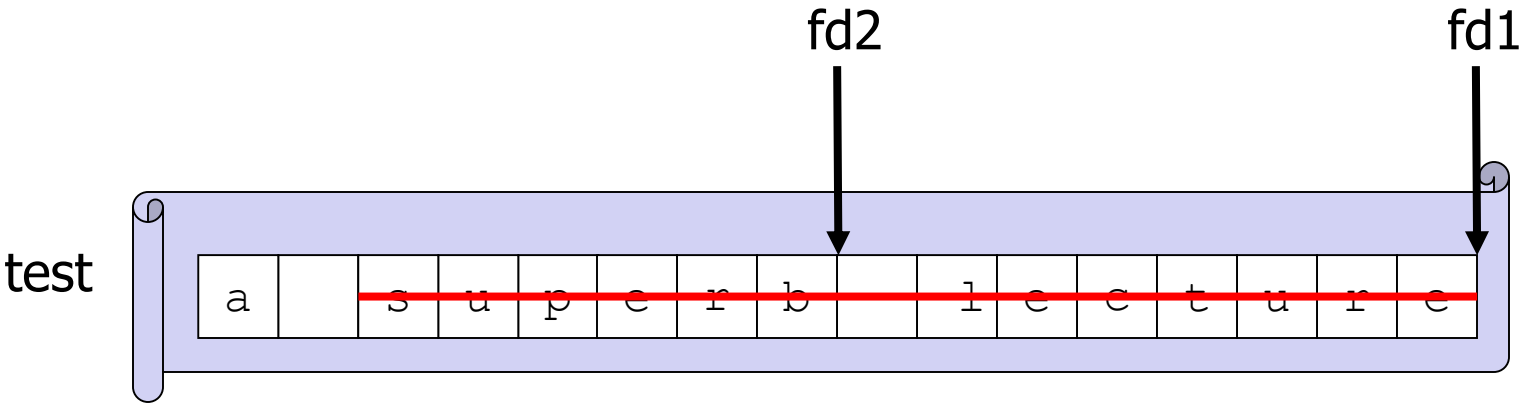


για να είναι πιο ευανάγνωστες οι τιμές των bytes,
εδώ παρουσιάζονται με ερμηνεία ASCII

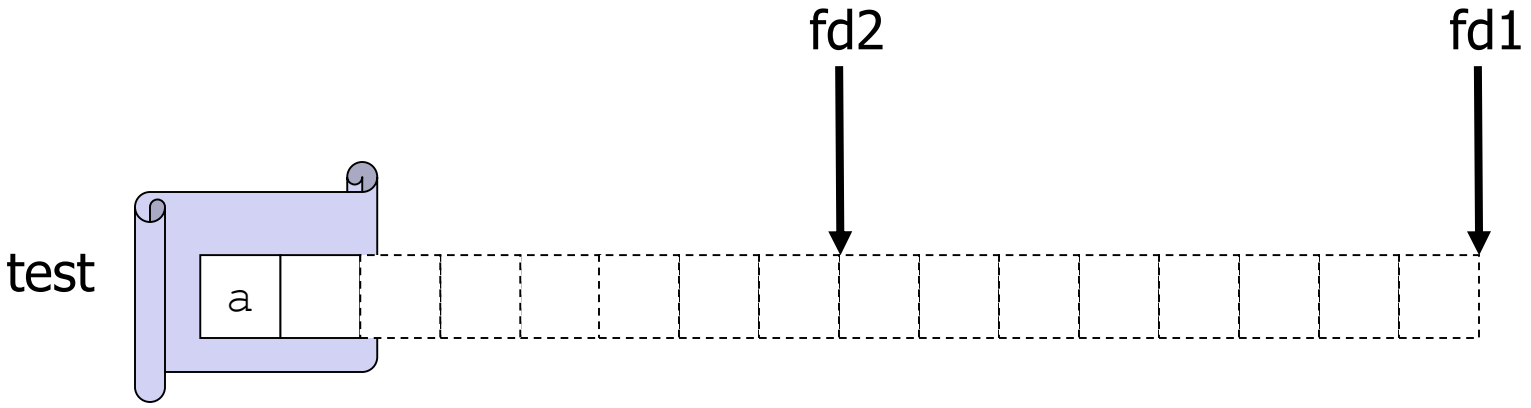
```
int fd1,fd2;

fd1=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2=open("test",O_RDWR,0);
write(fd1,"a boring lecture",16);
write(fd2,"a superb",8);
ftruncate(fd1,2);
write(fd2,"lecture",7);
close(fd1);
close(fd2);
```



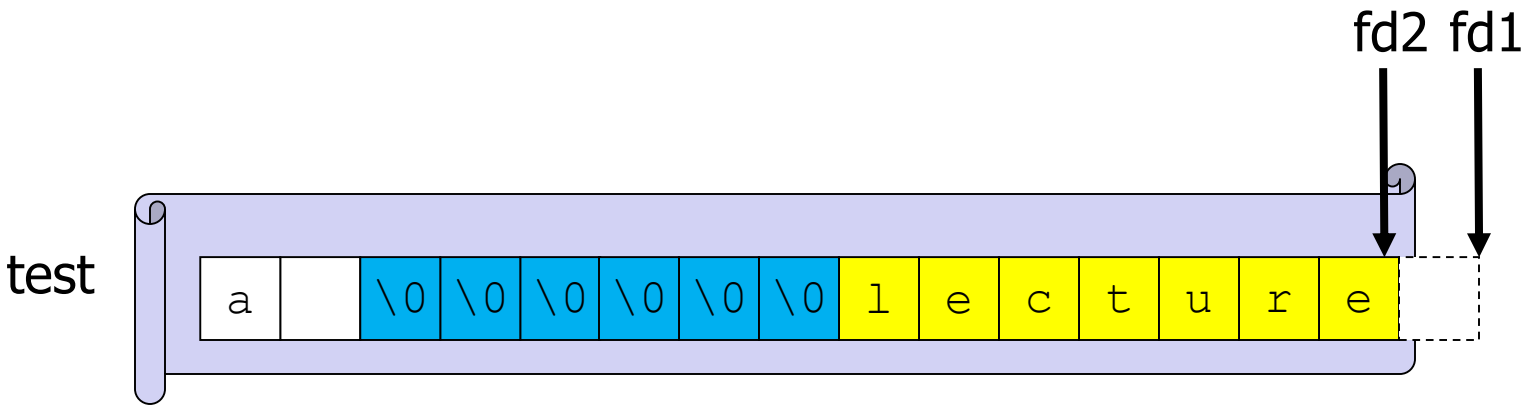
```
int fd1,fd2;

fd1=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2=open("test",O_RDWR,0);
write(fd1,"a boring lecture",16);
write(fd2,"a superb",8);
ftruncate(fd1,2);
write(fd2,"lecture",7);
close(fd1);
close(fd2);
```



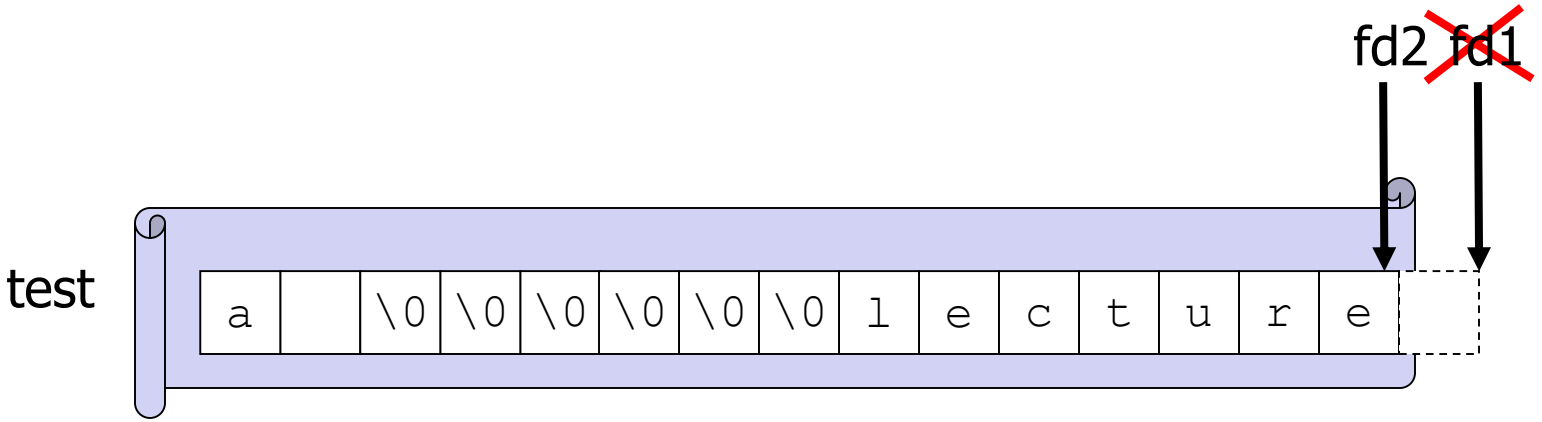
```
int fd1,fd2;

fd1=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2=open("test",O_RDWR,0);
write(fd1,"a boring lecture",16);
write(fd2,"a superb",8);
ftruncate(fd1,2);
write(fd2,"lecture",7);
close(fd1);
close(fd2);
```



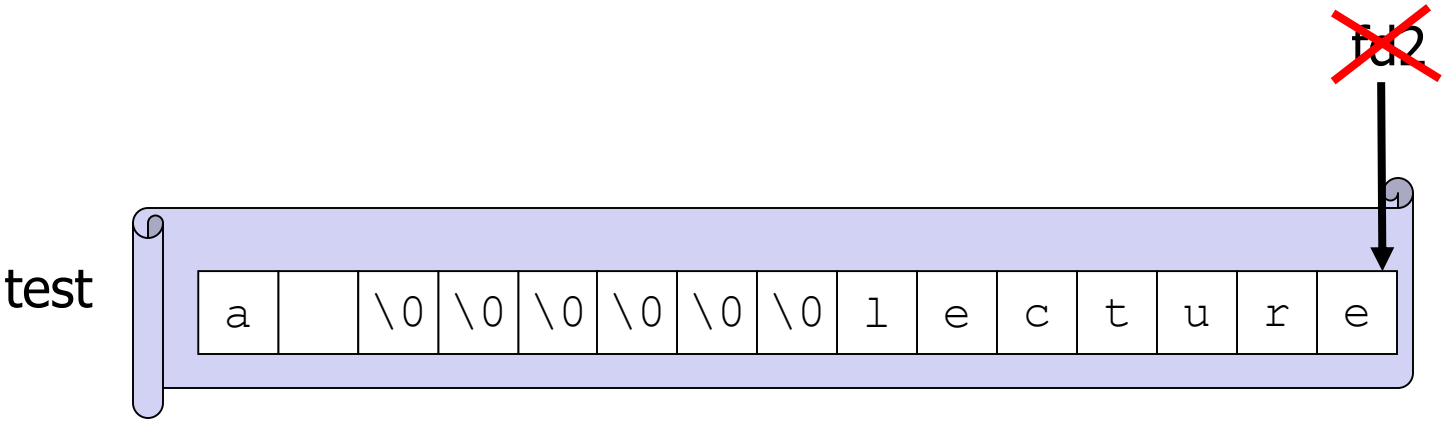
```
int fd1,fd2;

fd1=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2=open("test",O_RDWR,0);
write(fd1,"a boring lecture",16);
write(fd2,"a superb",8);
ftruncate(fd1,2);
write(fd2,"lecture",7);
close(fd1);
close(fd2);
```



```
int fd1,fd2;

fd1=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2=open("test",O_RDWR,0);
write(fd1,"a boring lecture",16);
write(fd2,"a superb",8);
ftruncate(fd1,2);
write(fd2,"lecture",7);
close(fd1);
close(fd2);
```



```
int fd1,fd2;

fd1=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2=open("test",O_RDWR,0);
write(fd1,"a boring lecture",16);
write(fd2,"a superb",8);
ftruncate(fd1,2);
write(fd2,"lecture",7);
close(fd1);
close(fd2);
```



Δομή πρόσβασης

- Όταν ανοίγει ένα αρχείο, το λειτουργικό δημιουργεί (εσωτερικά) μια ξεχωριστή **δομή πρόσβασης**
- Η δομή πρόσβασης περιέχει πληροφορία για το είδος πρόσβασης στο αρχείο (`mode` στην `open`) καθώς και για την τρέχουσα θέση ανάγνωσης/εγγραφής

- Το λειτουργικό διατηρεί (εσωτερικά) έναν **πίνακα από δείκτες** στις δομές πρόσβασης που έχουν δημιουργηθεί στο πλαίσιο κάθε προγράμματος
- Ο πίνακας αυτός **δεν** είναι άμεσα προσπελάσιμος από τον κώδικα της εφαρμογής (επίπεδο χρήστη)

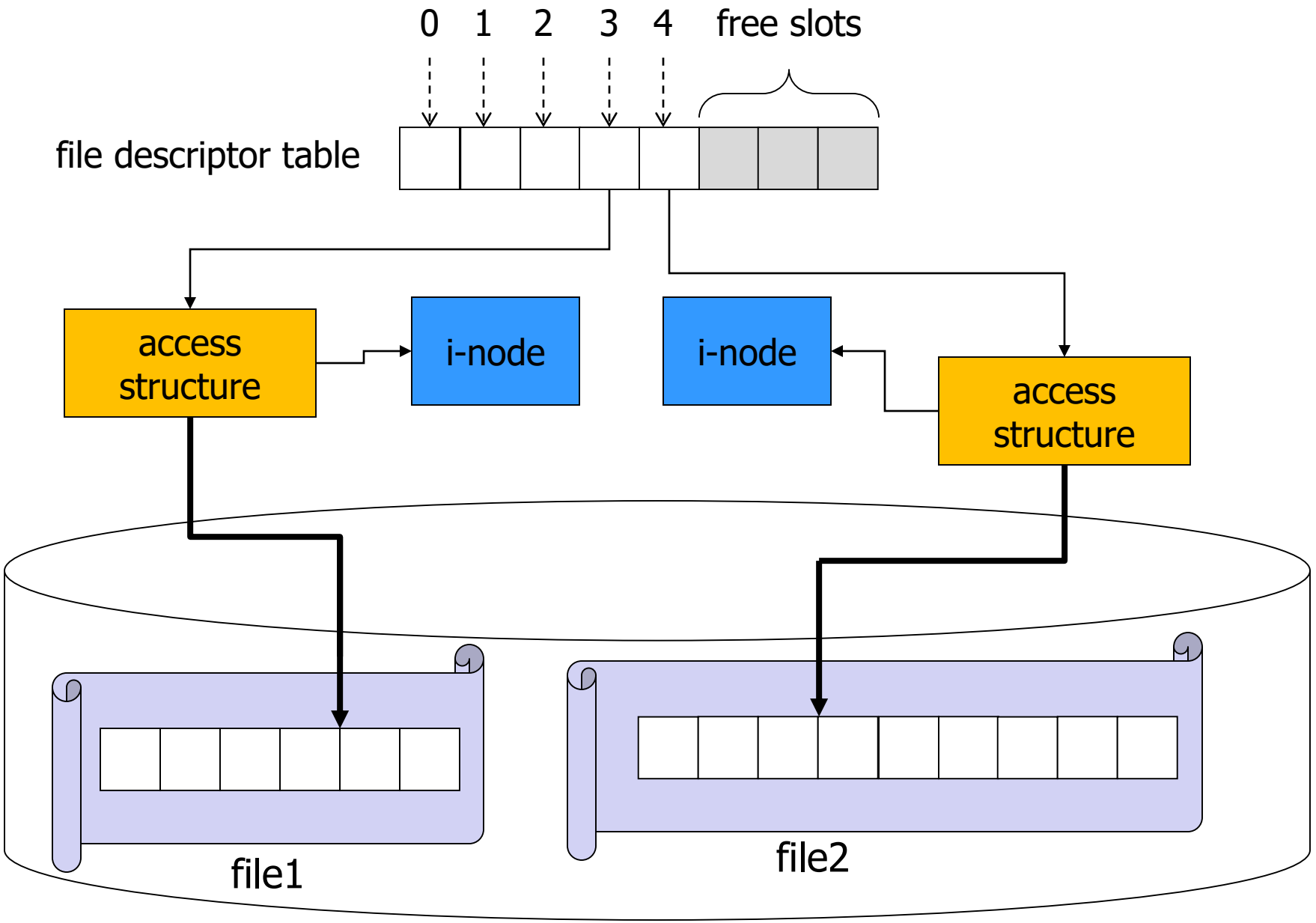
Τι είναι τελικά ο περιγραφέας αρχείου;

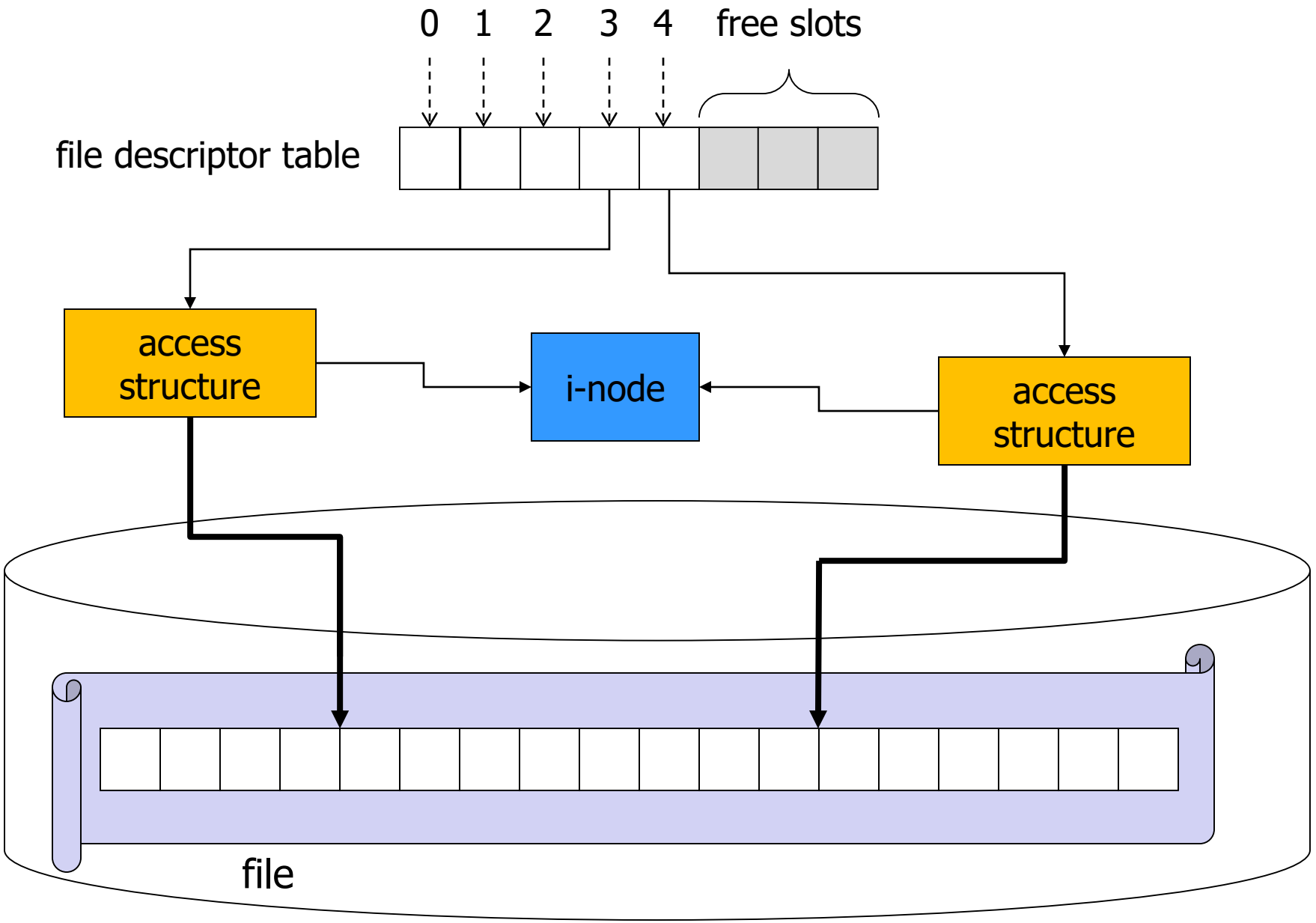
- Η **θέση στον πίνακα** όπου βρίσκεται ο δείκτης στην νέα δομή πρόσβασης που δημιουργήθηκε μέσω `open`
- Είναι ένας «απλός» ακέραιος

- Στις λειτουργίες `read`, `write`, `close` δεν πρέπει απαραίτητα να δοθεί μια τιμή που επέστρεψε η `open`
- Μπορεί να δοθεί μια **οποιαδήποτε** τιμή ακεραίου
  - με ευθύνη του προγραμματιστή
  - π.χ., προκαθορισμένες τιμές 0,1,2 για stdin, stdout, stderr
- Θα χρησιμοποιηθεί η (όποια) δομή πρόσβασης «τυχαίνει» να βρίσκεται σε αυτή τη θέση του πίνακα

Κόμβος πληροφορίας i-node

- Για κάθε ανοιχτό αρχείο, το λειτουργικό διατηρεί μια ξεχωριστή δομή (information node) όπου αποθηκεύονται διάφορες **πληροφορίες**
  - ιδιοκτήτης, ομάδα
  - άδειες πρόσβασης
  - μέγεθος, αριθμός μπλοκ
  - αριθμός συσκευής που βρίσκεται το αρχείο
  - χρονοσφραγίδα πρόσφατης πρόσβασης/αλλαγής
  - δεδομένα που κρατιούνται προσωρινά στην μνήμη
- Για κάθε ανοιχτό αρχείο, υπάρχει **μόνο ένα** i-node, στο οποίο συνδέονται όλες οι δομές πρόσβασης
- Το i-node μπορεί να προσπελαστεί μέσω των λειτουργιών `stat`, `lstat`, `fstat`


0
1
2
3
4
free slots
file descriptor table
access structure
i-node
i-node
access structure
file1
file2


0
1
2
3
4
free slots
file descriptor table
access structure
i-node
access structure
file

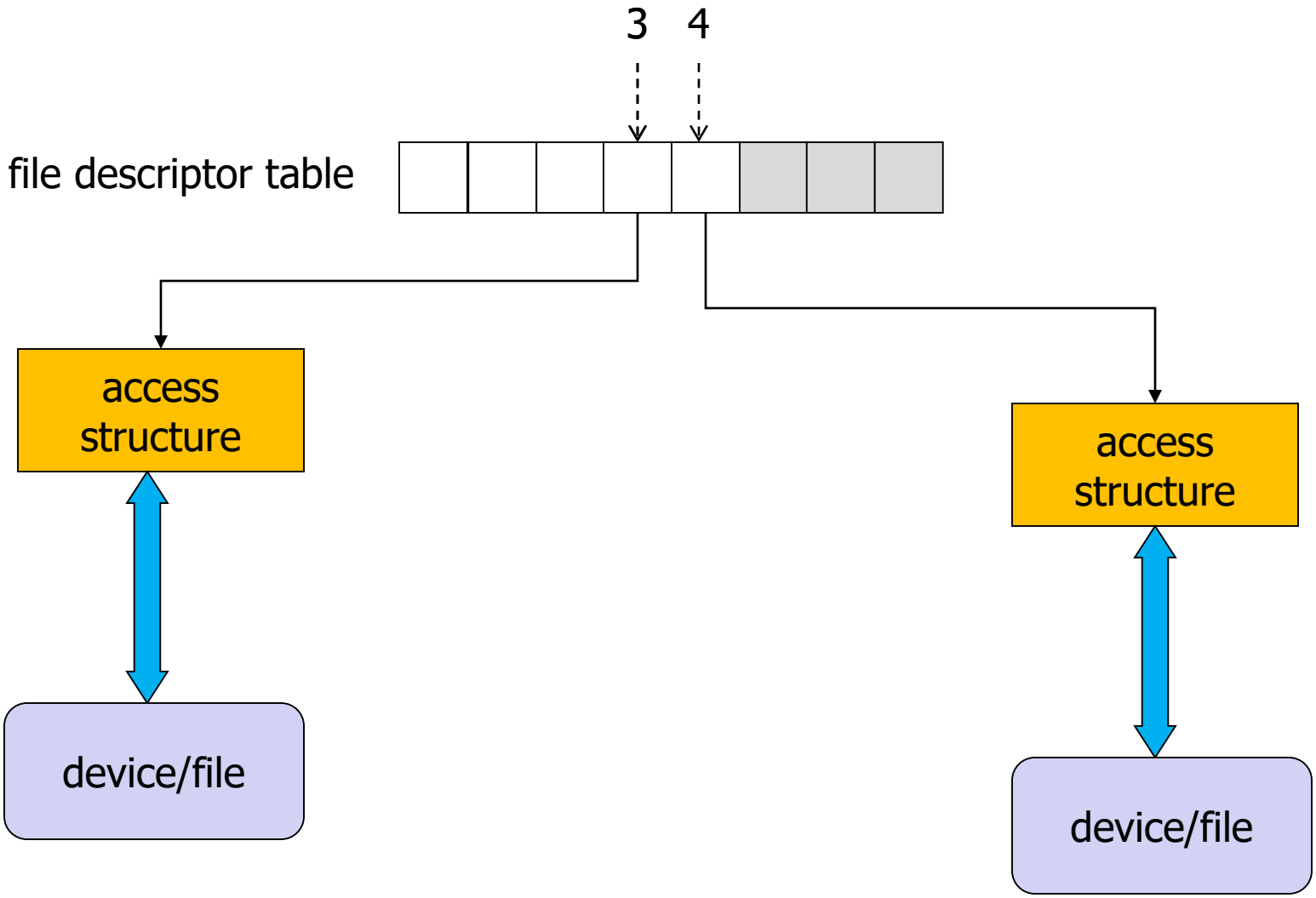
Αντιγραφή περιγραφέων αρχείων
&
Ανακατεύθυνση ΕΕ

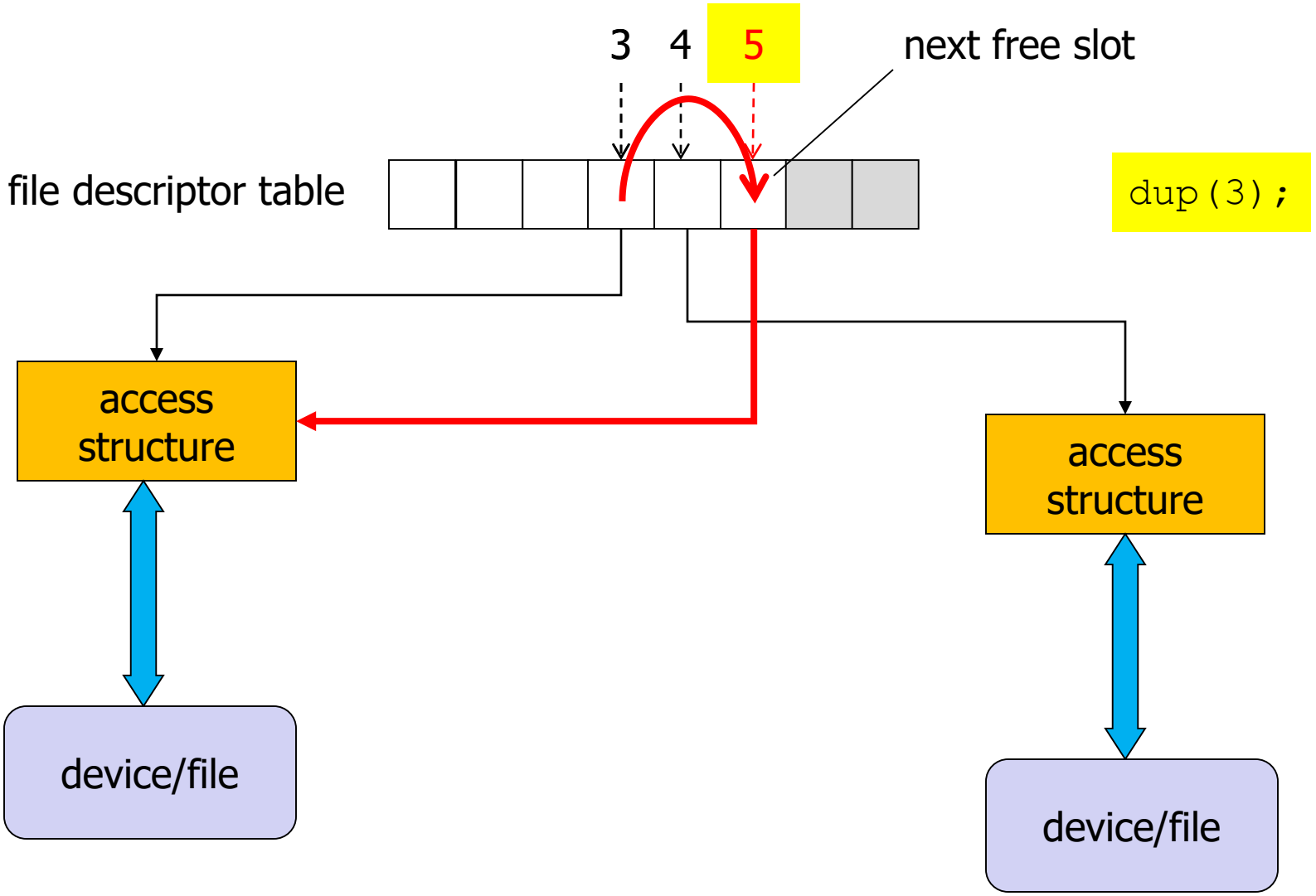
Αντίγραφα περιγραφέων

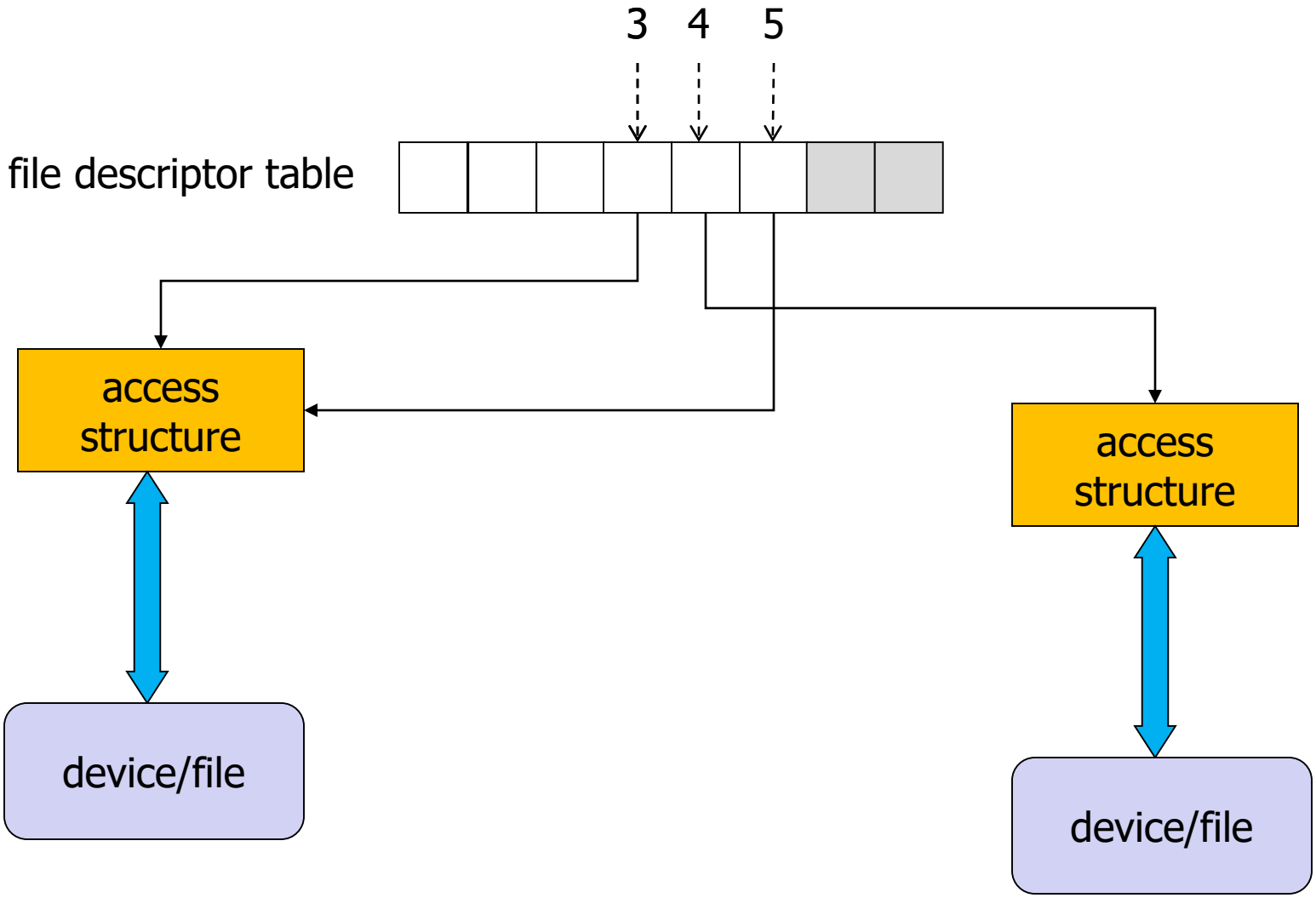
- `int dup(int fd)`: δημιουργεί ένα **αντίγραφο** του περιγραφέα `fd`, που δείχνει στην ίδια δομή πρόσβασης, και επιστρέφει την θέση του νέου περιγραφέα στον πίνακα των δομών πρόσβασης

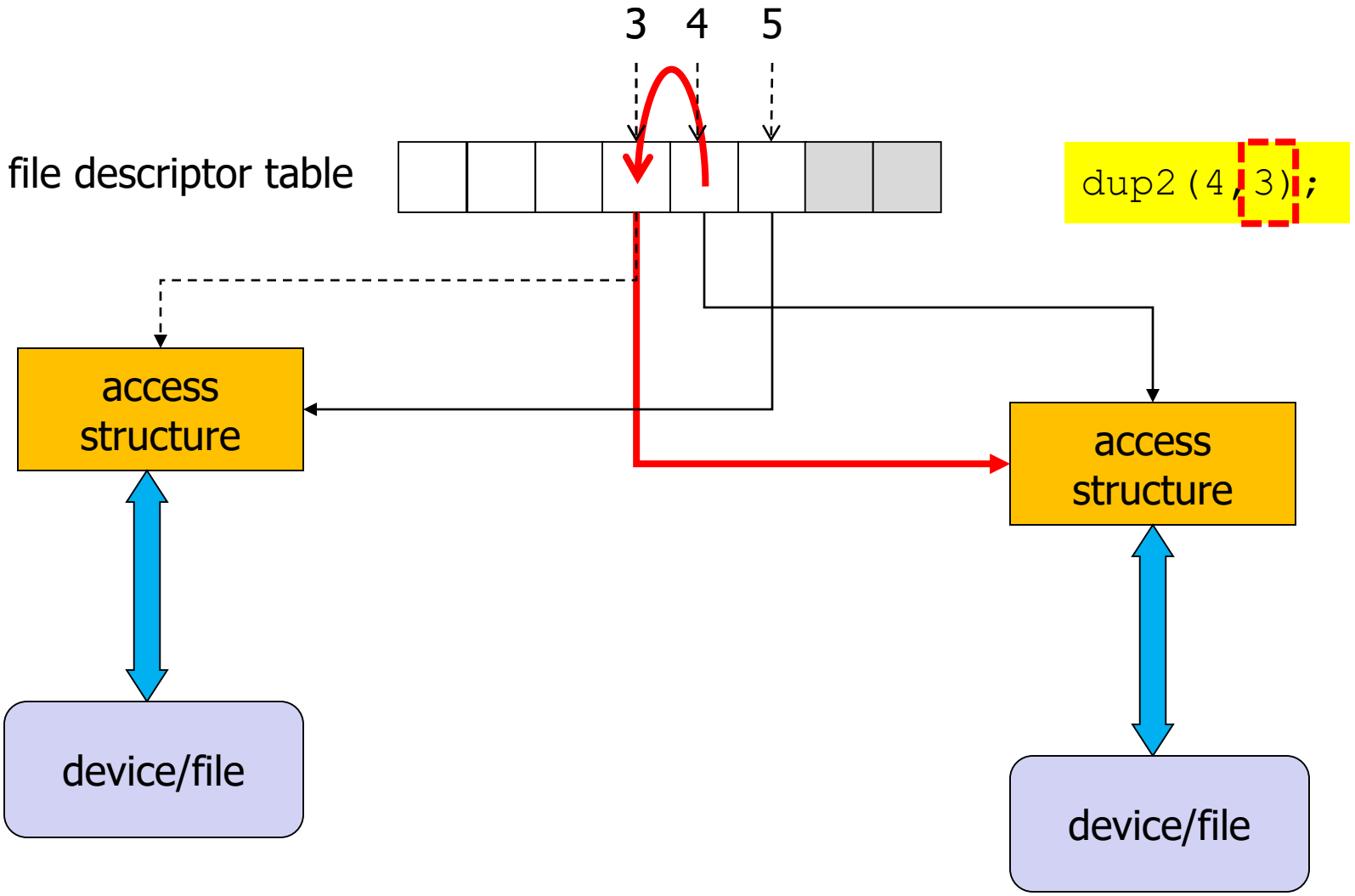
- `int dup2(int fd, int id)`: δημιουργεί ένα **αντίγραφο** του `fd`, στην θέση `id` του πίνακα
  - αν στην θέση `id` ήδη υπάρχει δείκτης σε κάποια άλλη δομή πρόσβασης, γίνεται **αντικατάσταση**

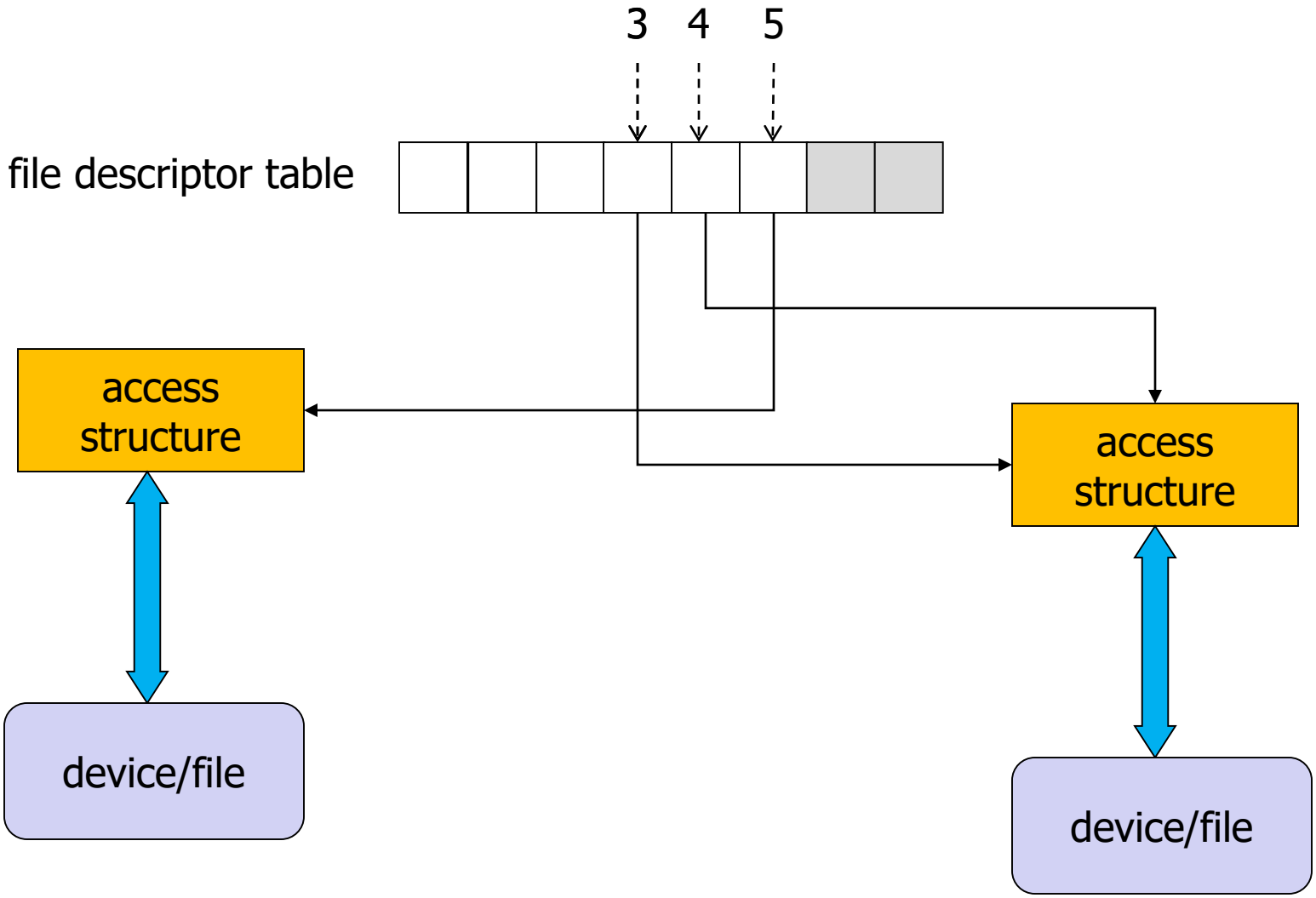
- Τα **αντίγραφα** δείχνουν στην **ίδια** δομή πρόσβασης
  - έχουν την **ίδια** θέση ανάγνωσης/εγγραφής


3
4
file descriptor table
access structure
access structure
device/file
device/file


3
4
5
next free slot
file descriptor table
dup(3);
access structure
access structure
device/file
device/file

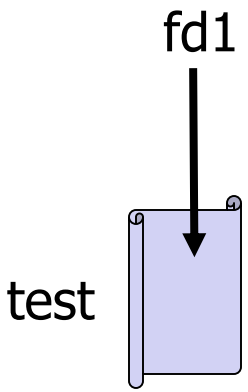

3
4
5
file descriptor table
access structure
access structure
device/file
device/file


3 4 5
file descriptor table
dup2(4,3);
access structure
access structure
device/file
device/file

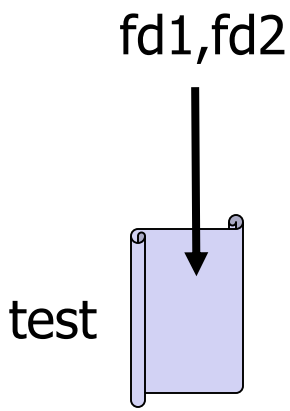

3
4
5
file descriptor table
access structure
access structure
device/file
device/file

```
int fd1,fd2;

fd1=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2=dup(fd1);
write(fd1,"have a nice",11);
write(fd2," day",4);
close(fd1);
close(fd2);
```

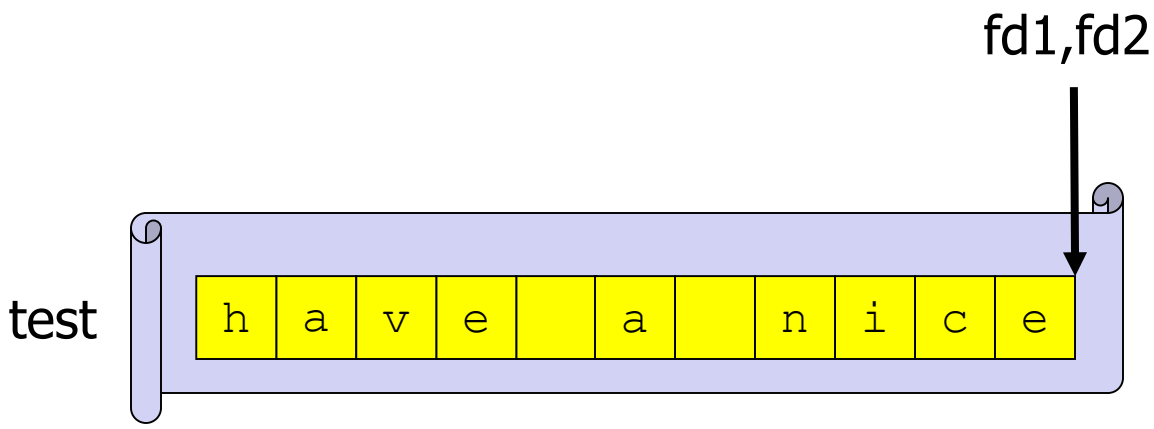
```
int fd1,fd2;

fd1=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2=dup(fd1);
write(fd1,"have a nice",11);
write(fd2," day",4);
close(fd1);
close(fd2);
```



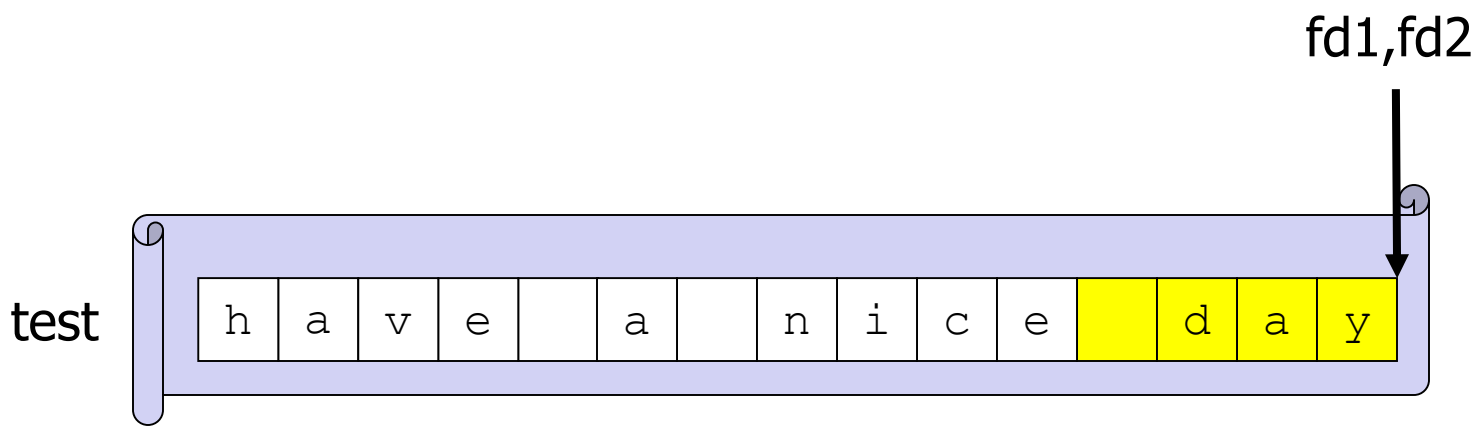
```
int fd1,fd2;

fd1=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2=dup(fd1);
write(fd1,"have a nice",11);
write(fd2," day",4);
close(fd1);
close(fd2);
```



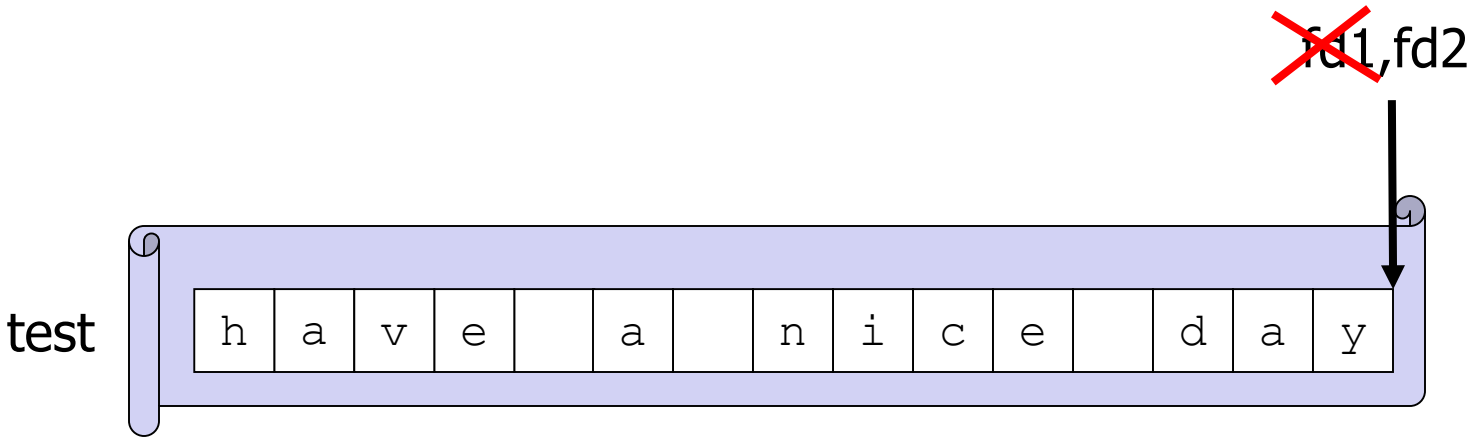
```
int fd1,fd2;

fd1=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2=dup(fd1);
write(fd1,"have a nice",11);
write(fd2," day",4);
close(fd1);
close(fd2);
```



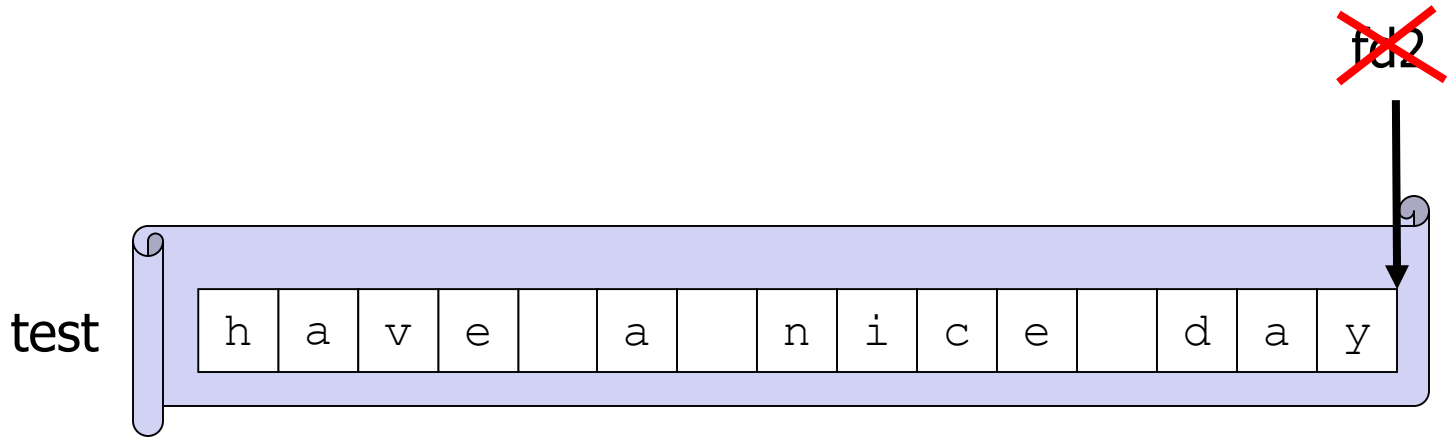
```
int fd1,fd2;

fd1=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2=dup(fd1);
write(fd1,"have a nice",11);
write(fd2," day",4);
close(fd1);
close(fd2);
```



```
int fd1,fd2;

fd1=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2=dup(fd1);
write(fd1,"have a nice",11);
write(fd2," day",4);
close(fd1);
close(fd2);
```



```
int fd1,fd2;

fd1=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2=dup(fd1);
write(fd1,"have a nice",11);
write(fd2," day",4);
close(fd1);
close(fd2);
```

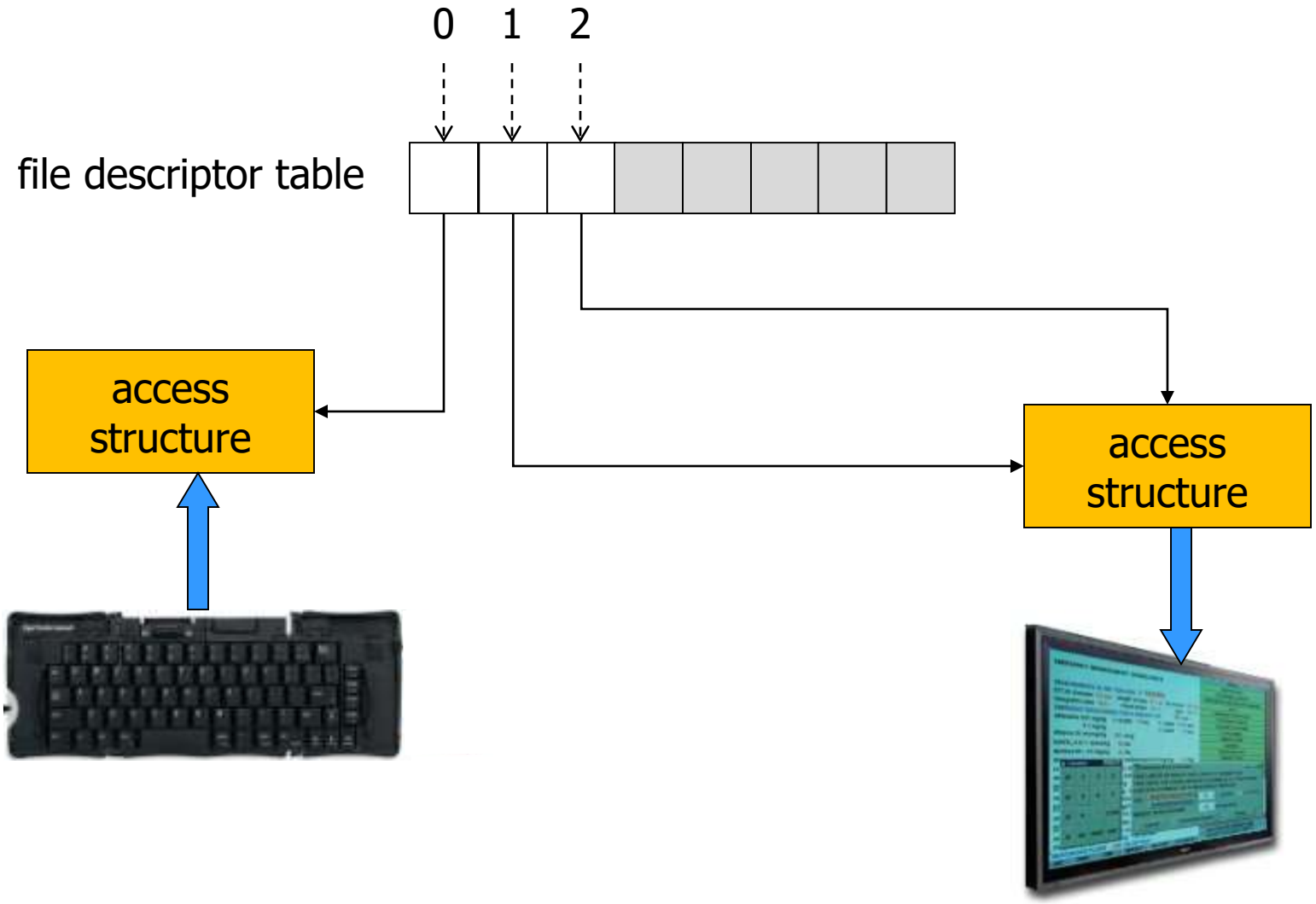


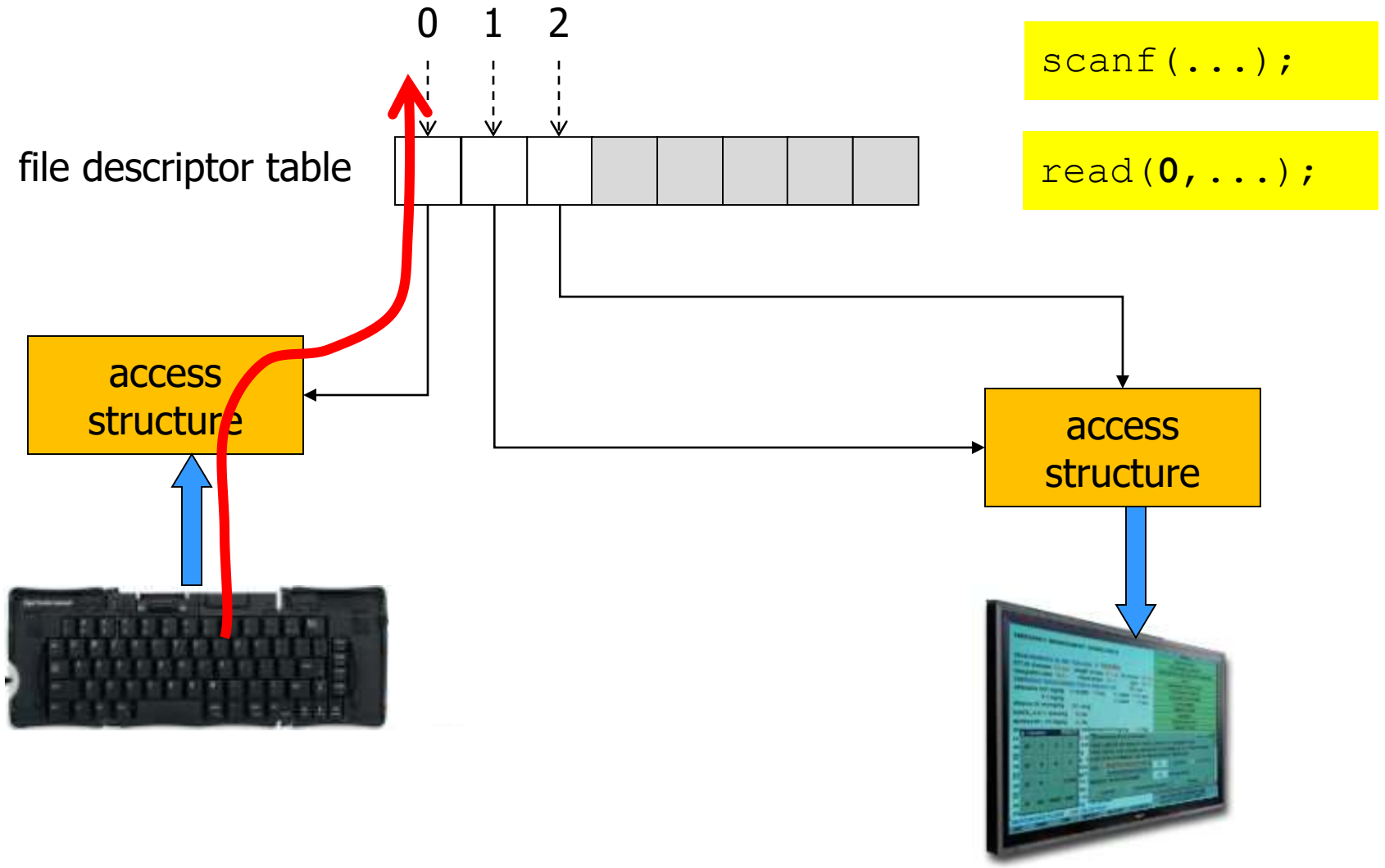
Είσοδος-έξοδος προγράμματος

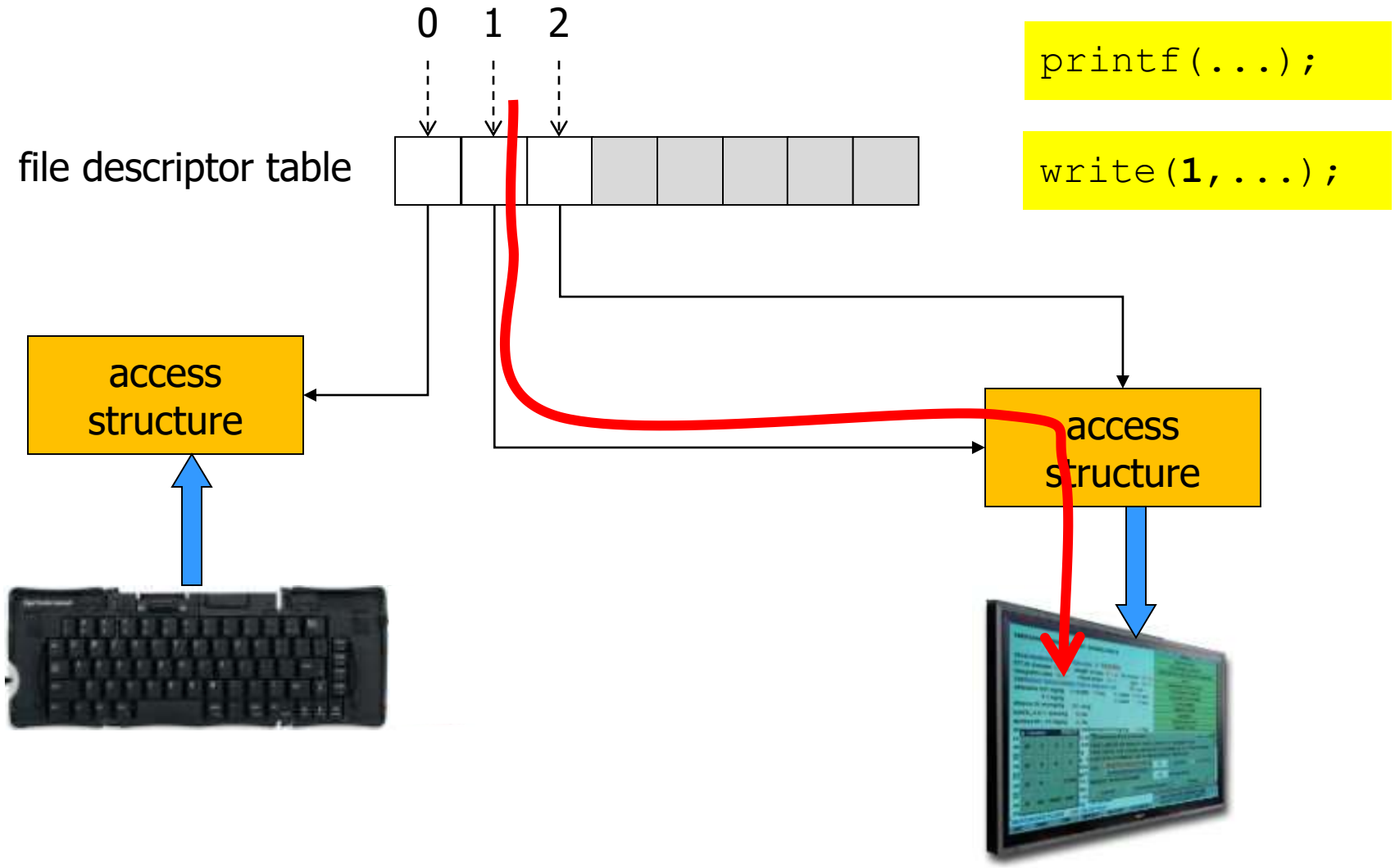
- **Κατά σύμβαση**, οι θέσεις 0, 1 και 2 του πίνακα δομών πρόσβασης παίζουν **ξεχωριστό** ρόλο
  - καθιερωμένη είσοδος, έξοδος, έξοδος λαθών

- Αρχικά, στις θέσεις αυτές αποθηκεύονται δομές πρόσβασης για την τερματική συσκευή μέσω της οποίας γίνεται η αλληλεπίδραση με τον χρήστη
  - 0 = είσοδος τερματικού = πληκτρολόγιο
  - 1,2 = έξοδος τερματικού = οθόνη

- Οι αντίστοιχες δομές πρόσβασης δημιουργούνται από το λειτουργικό / περιβάλλον εκτέλεσης, **χωρίς** να πρέπει να γραφτεί/εκτελεστεί κώδικας εφαρμογής

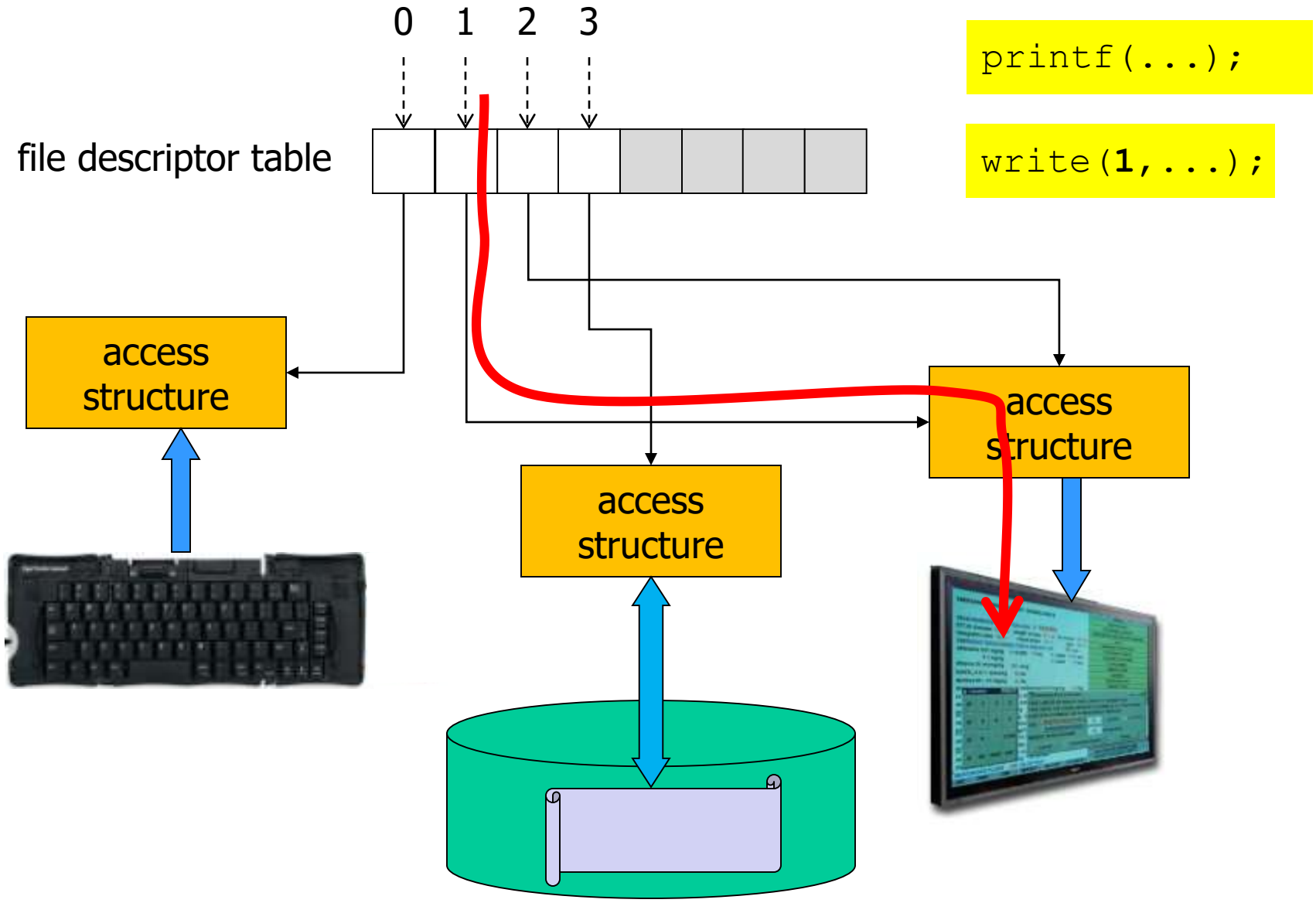

0
1
2
file descriptor table
access structure
access structure

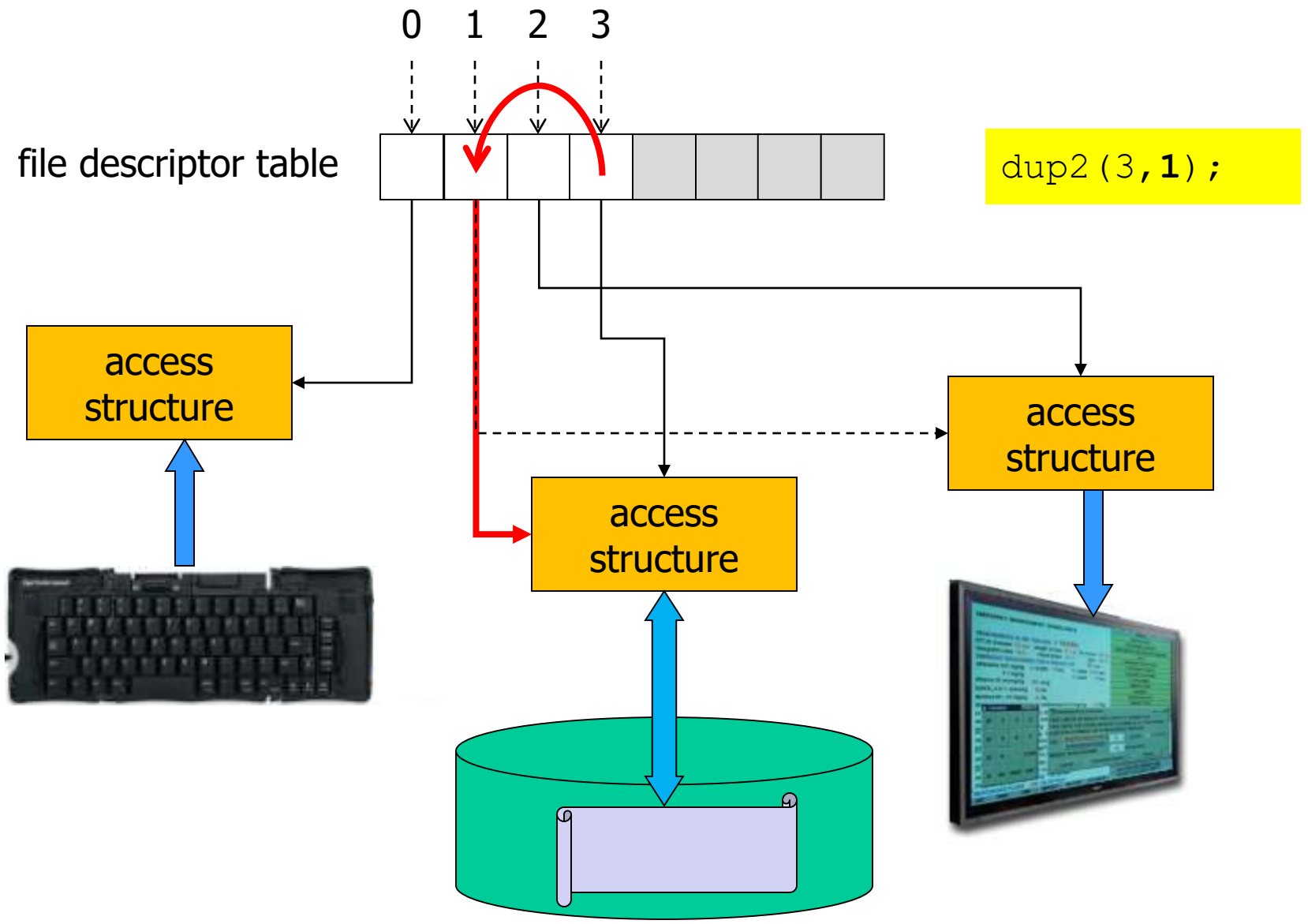

0
1
2
file descriptor table
scanf(...);
read(0,...);
access structure
access structure

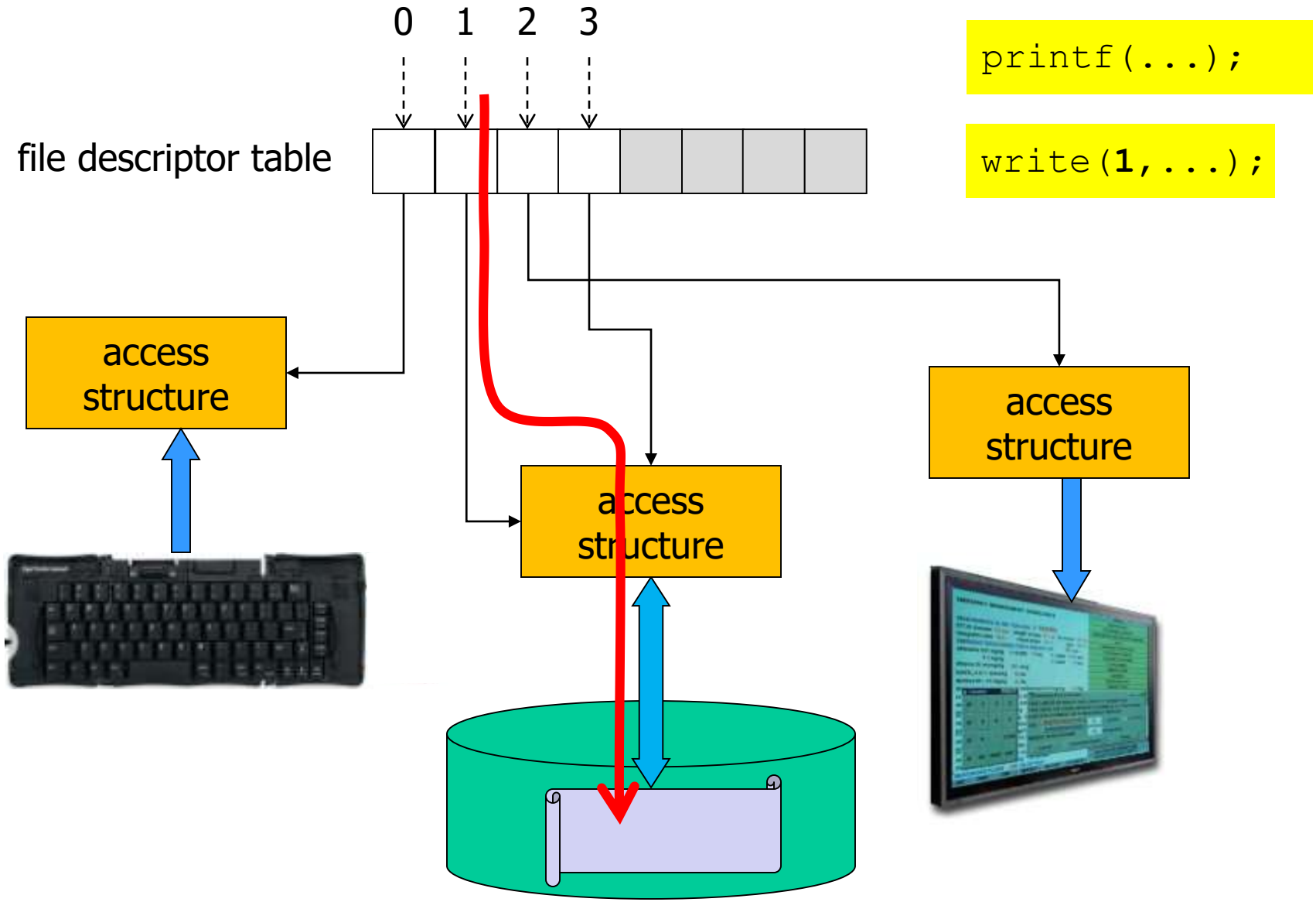

0
1
2
file descriptor table
printf(...);
write(1,...);
access structure
access structure

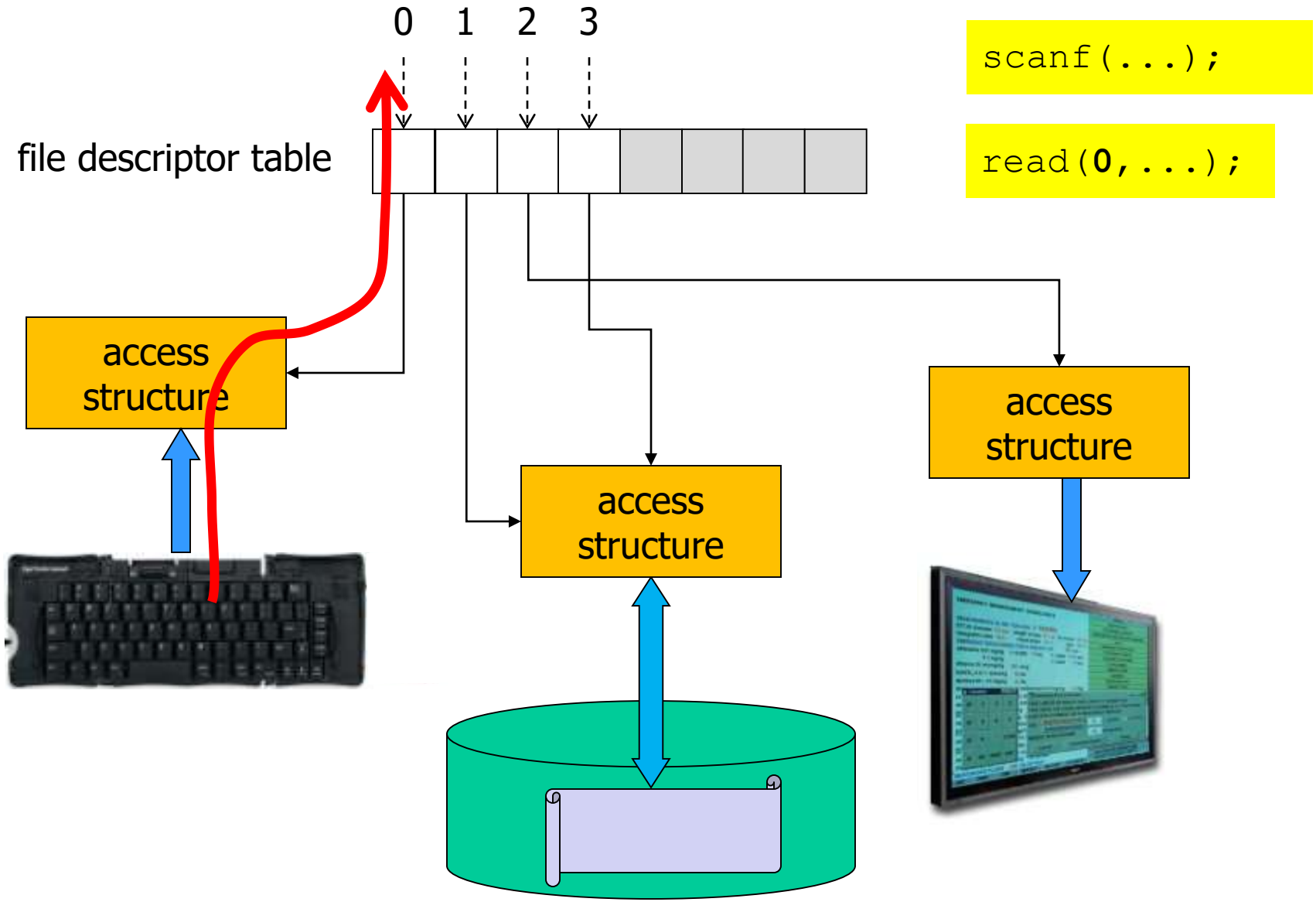
Ανακατεύθυνση εισόδου/εξόδου

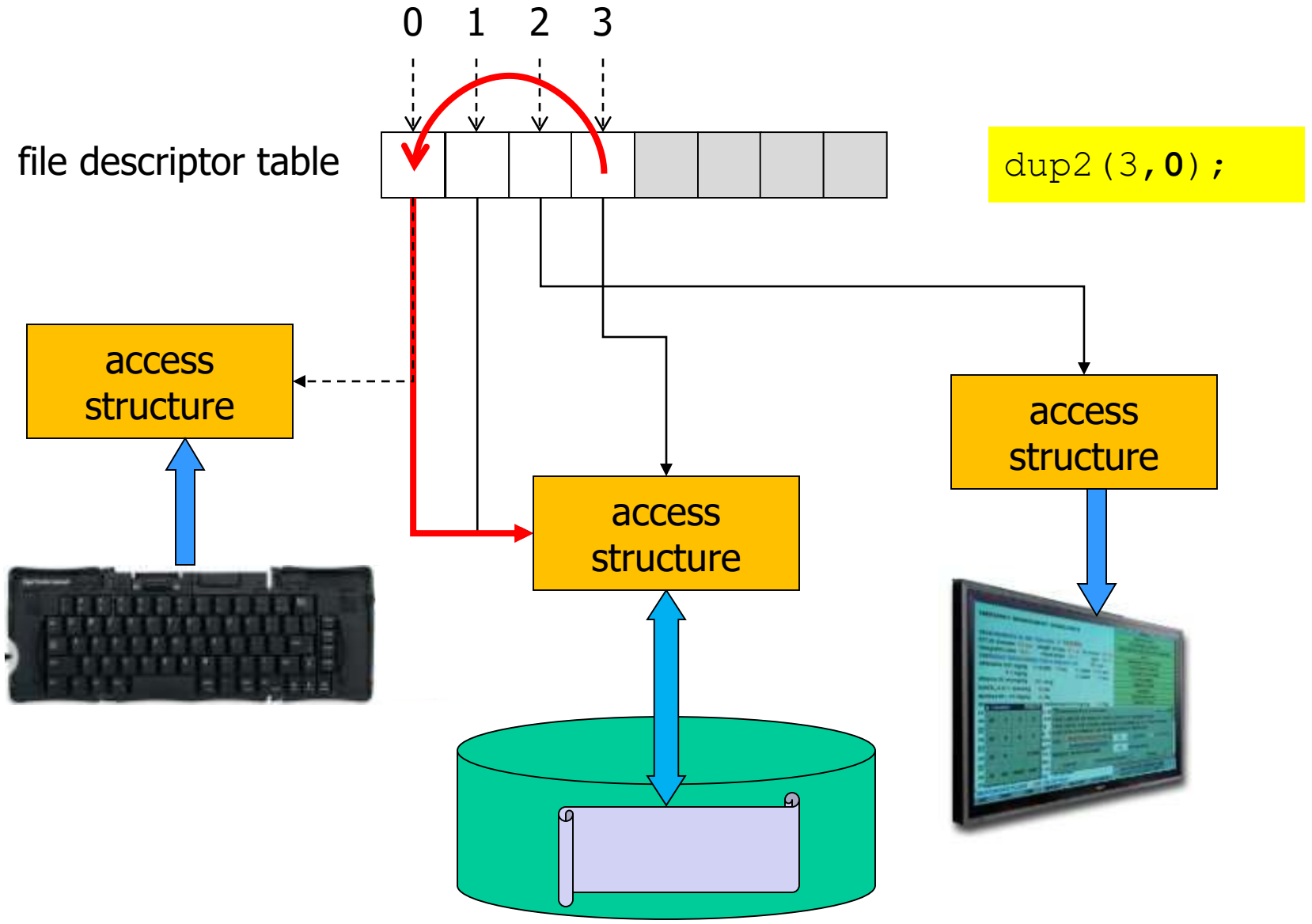
- Η είσοδος/έξοδος ενός προγράμματος **δεν είναι παγιωμένη** στην τερματική συσκευή του χρήστη
- Αυτό που είναι παγιωμένο (καθαρά ως σύμβαση) είναι ο ρόλος των δομών πρόσβασης που **«τυχαίνει»** να βρίσκονται στις θέσεις 0, 1, 2
- Μπορεί να γίνει **αντικατάσταση** των δομών πρόσβασης που βρίσκονται σε αυτές τις θέσεις
  - με `dup2`, **κατά την διάρκεια της εκτέλεσης**
  - οι νέες δομές πρόσβασης πρέπει να έχουν ανοίξει για διάβασμα και γράψιμο, αντίστοιχα
- Αυτό γίνεται **εν αγνοία** του όποιου κώδικα έχει **ήδη γραφτεί** να χρησιμοποιεί τους περιγραφείς 0, 1, 2
  - π.χ., μέσα από κλήσεις των `scanf`/`printf`

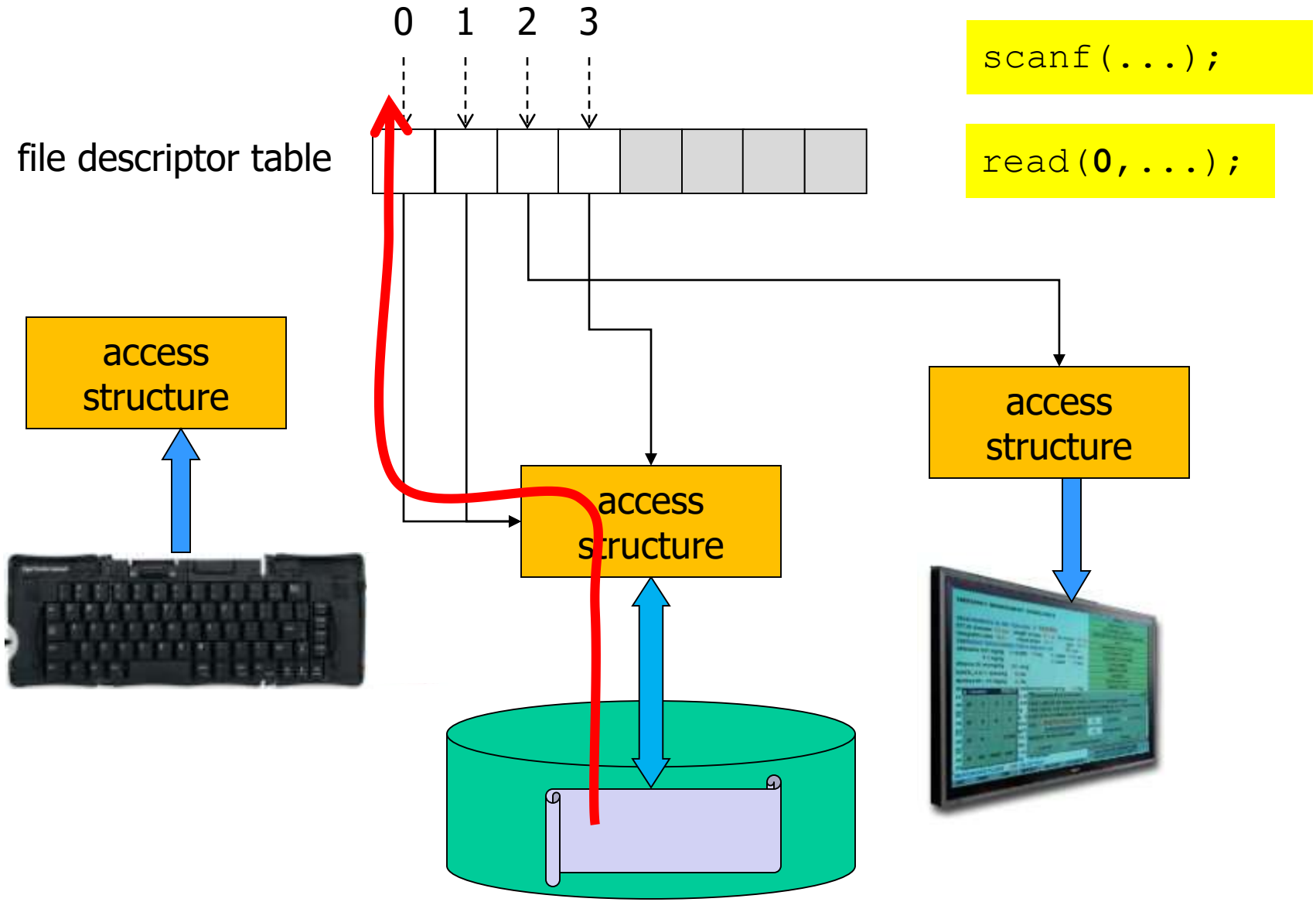

0 1 2 3
file descriptor table
printf(...);
write(1,...);
access structure
access structure
access structure


0
1
2
3
file descriptor table
dup2(3,1);
access structure
access structure
access structure


0
1
2
3
file descriptor table
printf(...);
write(1,...);
access structure
access structure
access structure


0 1 2 3
file descriptor table
scanf(...);
read(0,...);
access structure
access structure
access structure


0 1 2 3
file descriptor table
dup2(3,0);
access structure
access structure
access structure


0 1 2 3
file descriptor table
scanf(...);
read(0,...);
access structure
access structure
access structure

```
int fd, oldout, oldin;
char buf[16];

fd=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
printf("hello world\n");
oldout = dup(STDOUT_FILENO); // copy stdout
dup2(fd,STDOUT_FILENO); // redirect stdout to fd
printf("hello world again\n");
dup2(oldout,STDOUT_FILENO); // restore stdout
…
```

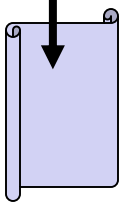
```
int fd, oldout, oldin;
char buf[16];

fd=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
printf("hello world\n");
oldout = dup(STDOUT_FILENO); // copy stdout
dup2(fd,STDOUT_FILENO); // redirect stdout to fd
printf("hello world again\n");
dup2(oldout,STDOUT_FILENO); // restore stdout
…
```

0

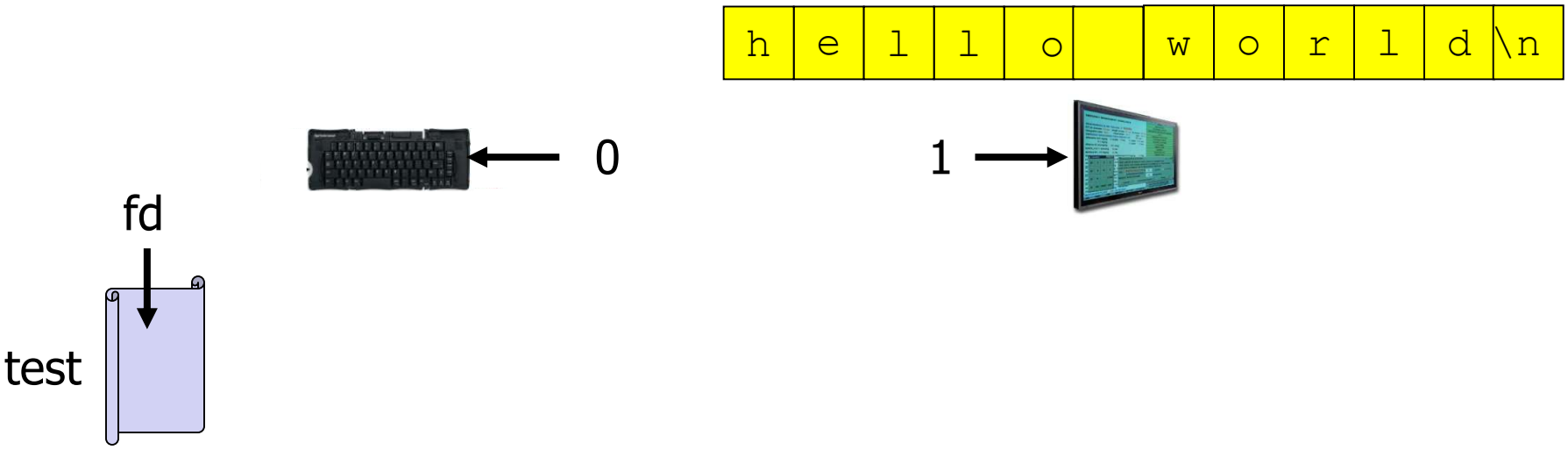
1

fd

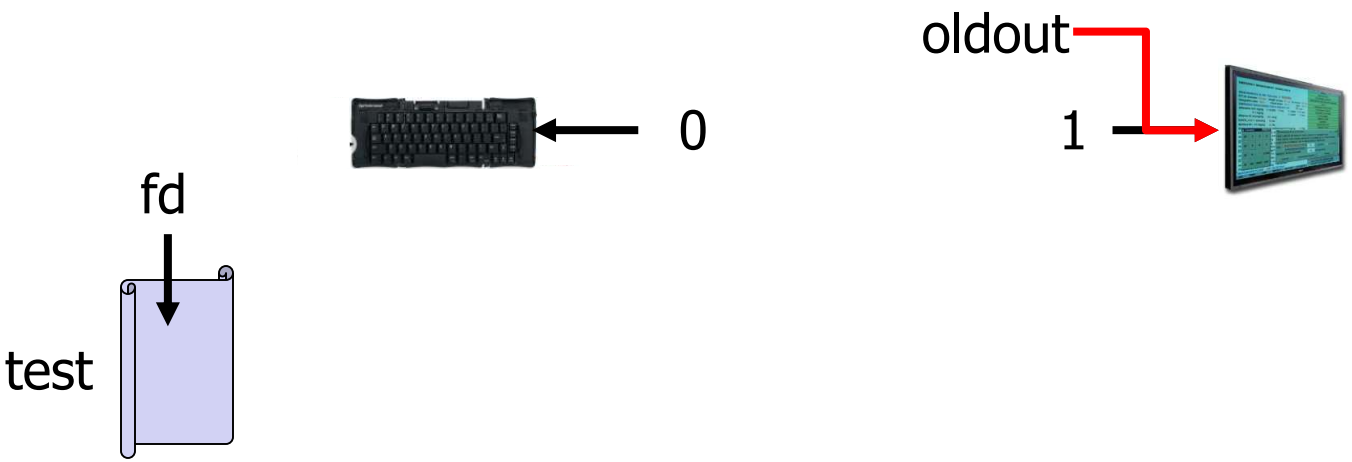
test



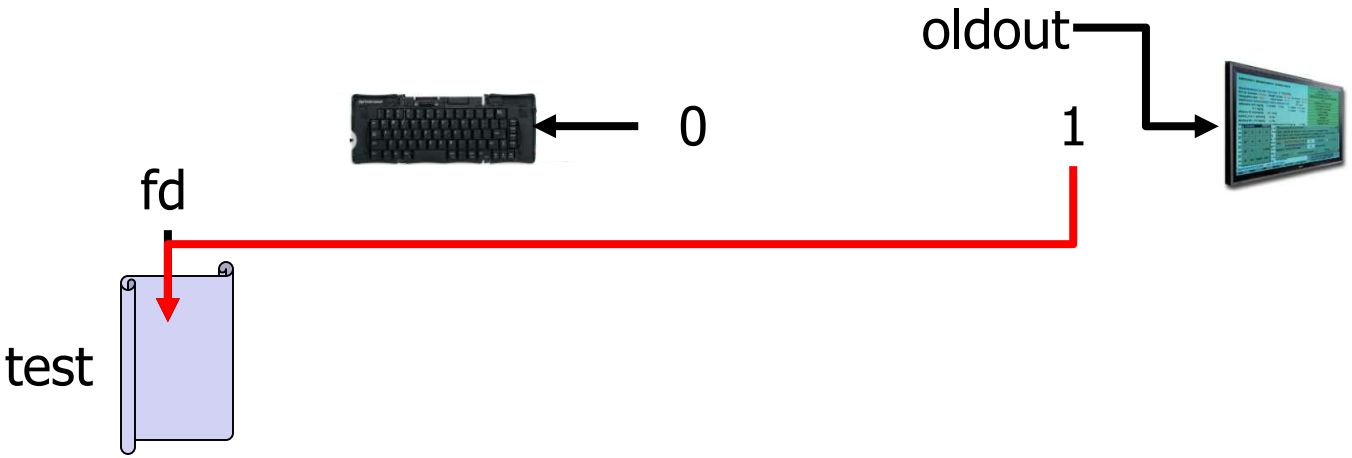
```
int fd, oldout, oldin;
char buf[16];

fd=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
printf("hello world\n");
oldout = dup(STDOUT_FILENO); // copy stdout
dup2(fd,STDOUT_FILENO); // redirect stdout to fd
printf("hello world again\n");
dup2(oldout,STDOUT_FILENO); // restore stdout
…
```



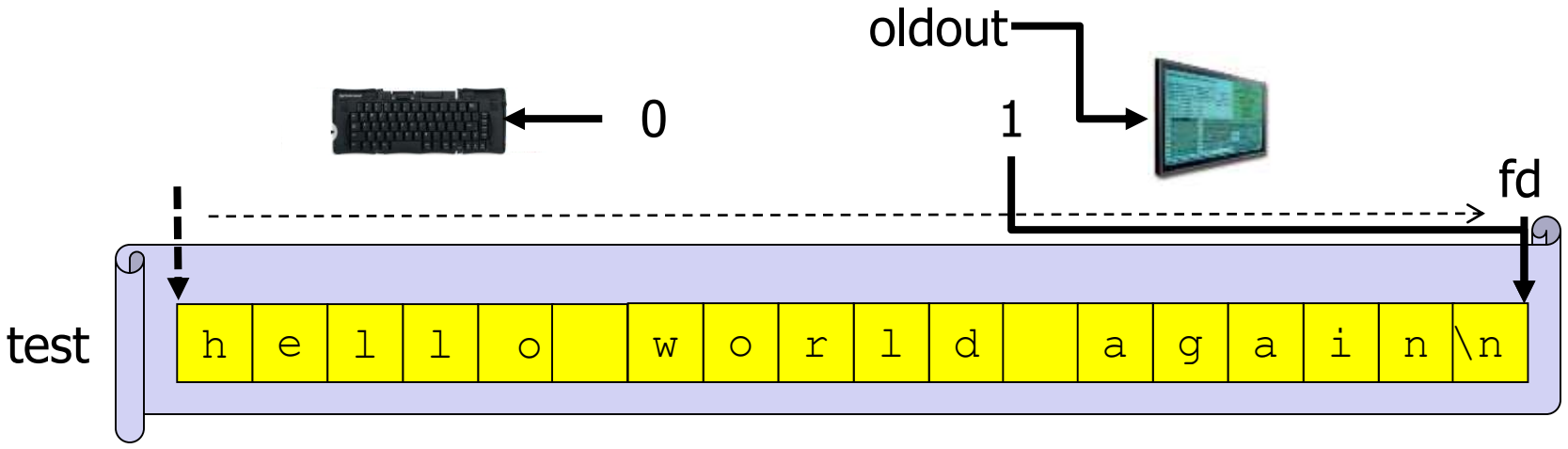
```
int fd, oldout, oldin;
char buf[16];

fd=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
printf("hello world\n");
oldout = dup(STDOUT_FILENO); // copy stdout
dup2(fd,STDOUT_FILENO); // redirect stdout to fd
printf("hello world again\n");
dup2(oldout,STDOUT_FILENO); // restore stdout
…
```



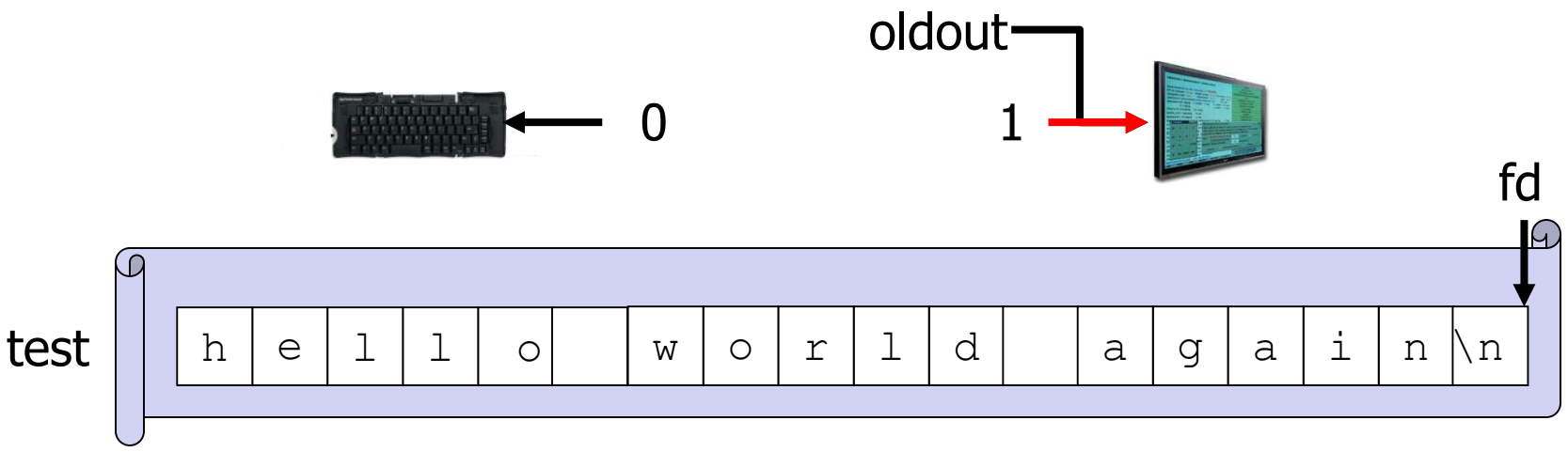
```
int fd, oldout, oldin;
char buf[16];

fd=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
printf("hello world\n");
oldout = dup(STDOUT_FILENO); // copy stdout
dup2(fd,STDOUT_FILENO); // redirect stdout to fd
printf("hello world again\n");
dup2(oldout,STDOUT_FILENO); // restore stdout
…
```



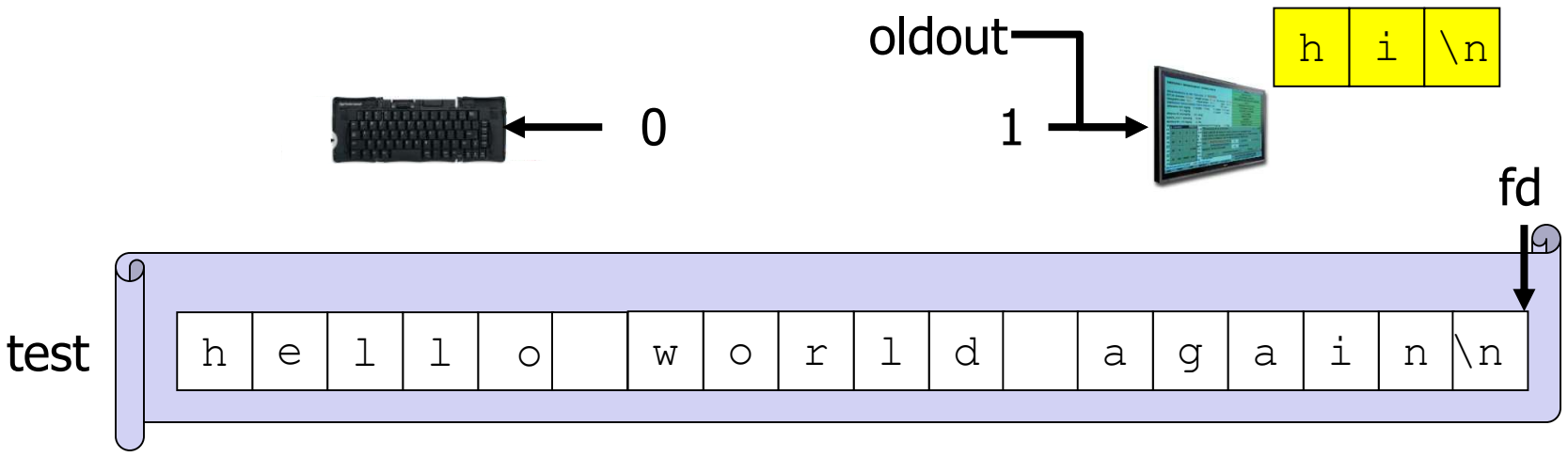
```
int fd, oldout, oldin;
char buf[16];

fd=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
printf("hello world\n");
oldout = dup(STDOUT_FILENO); // copy stdout
dup2(fd,STDOUT_FILENO); // redirect stdout to fd
printf("hello world again\n");
dup2(oldout,STDOUT_FILENO); // restore stdout
…
```



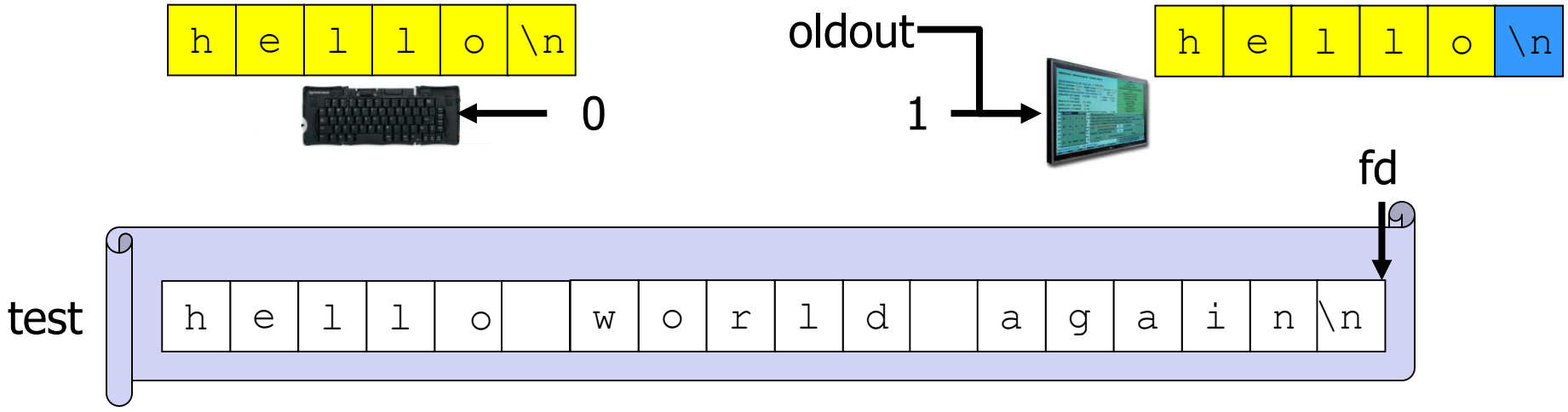
```
int fd, oldout, oldin;
char buf[16];

fd=open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
printf("hello world\n");
oldout = dup(STDOUT_FILENO); // copy stdout
dup2(fd,STDOUT_FILENO); // redirect stdout to fd
printf("hello world again\n");
dup2(oldout,STDOUT_FILENO); // restore stdout
…
```



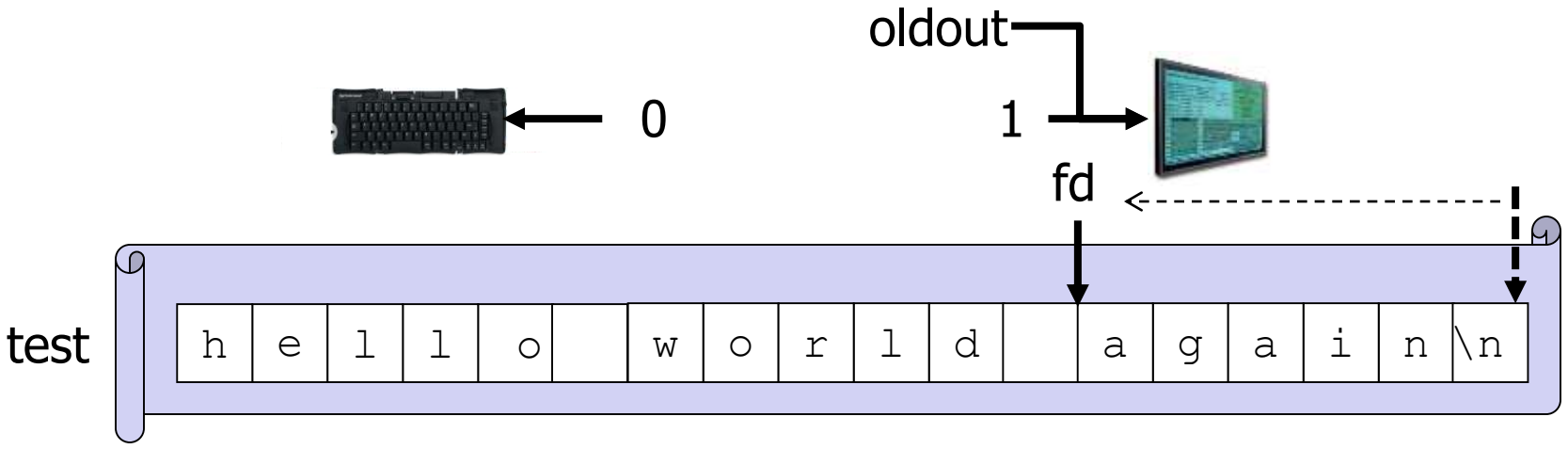
```
…
printf("hi\n");
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,12,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```



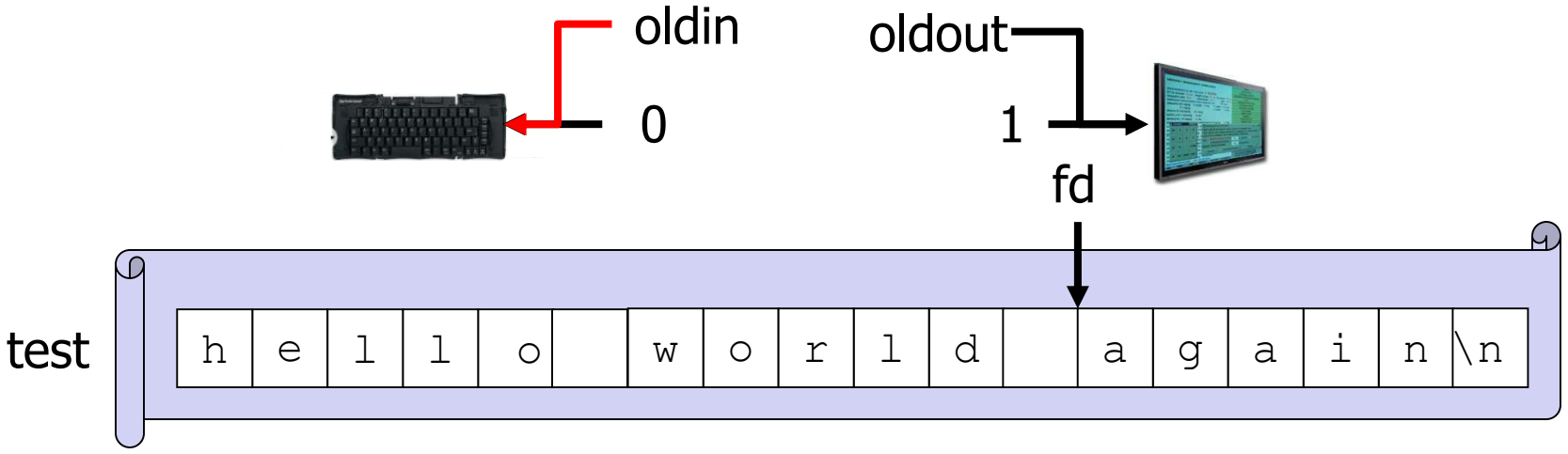
```
…
printf("hi\n");
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,12,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```



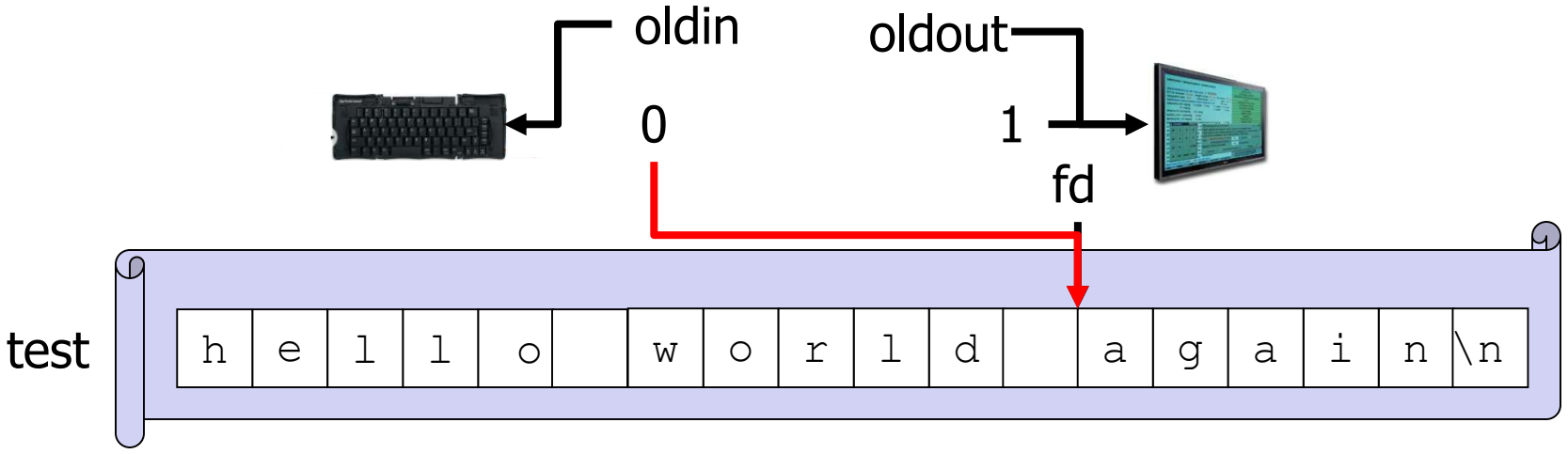
```
…
printf("hi\n");
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,12,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```



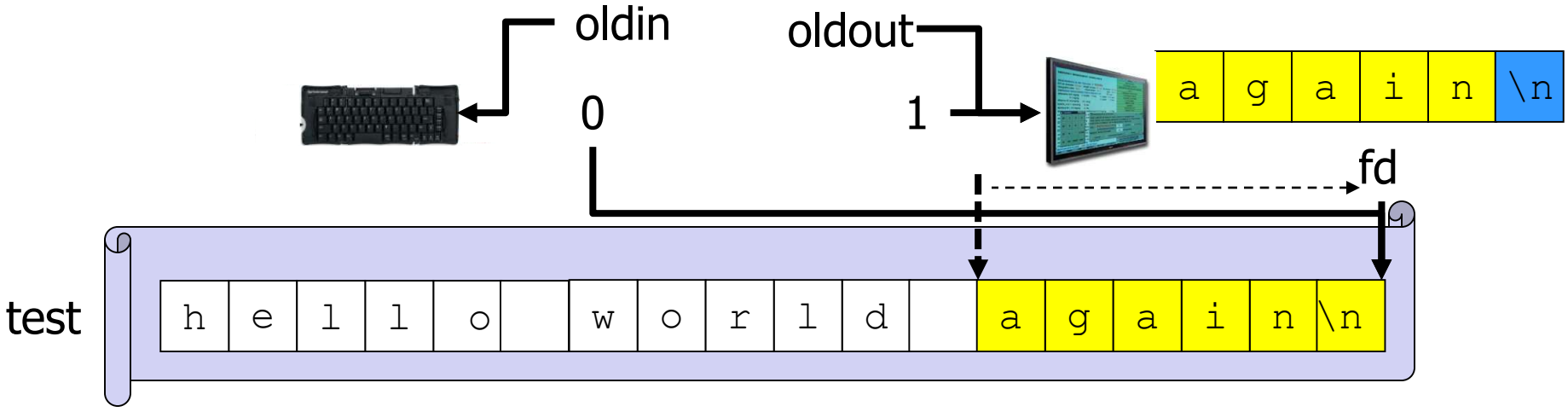
```
…
printf("hi\n");
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,12,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```



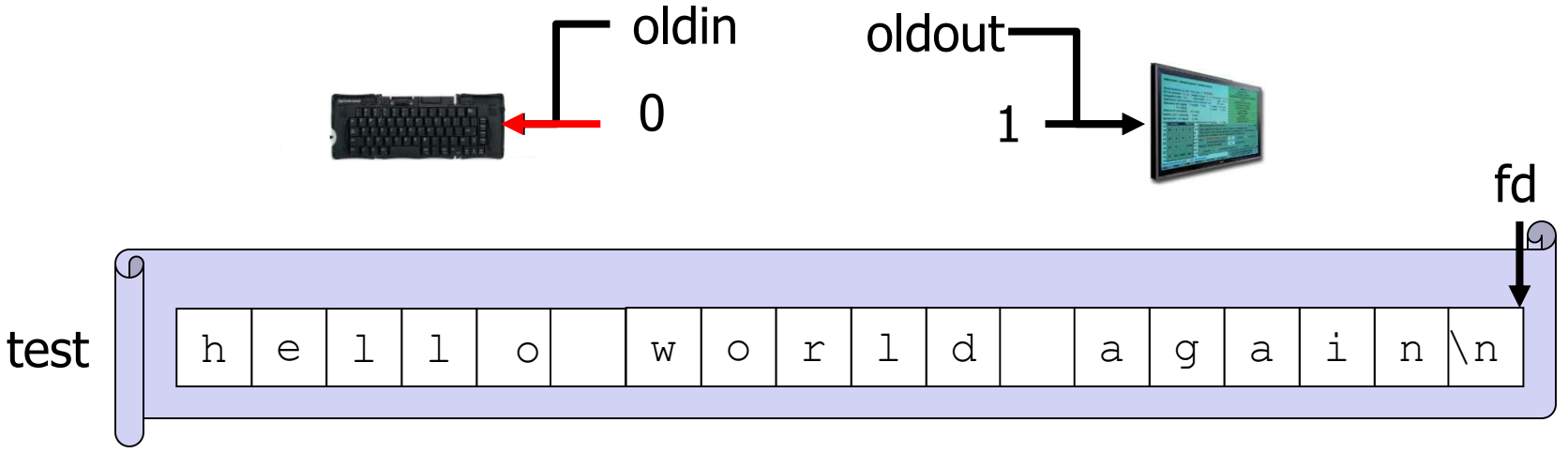
```
…
printf("hi\n");
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,12,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```



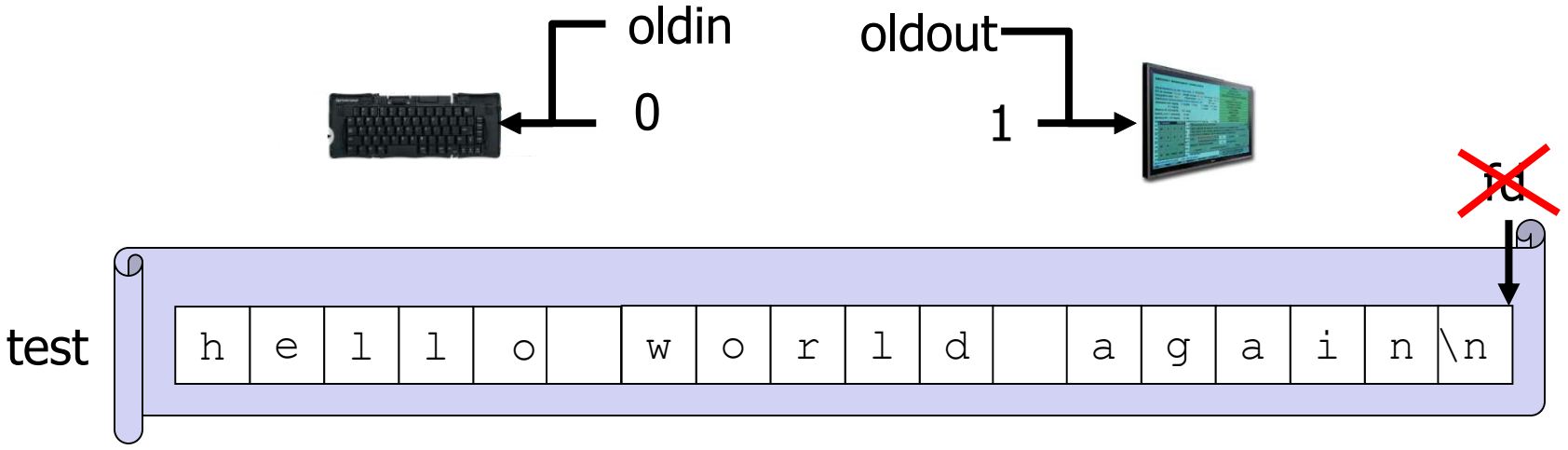
```
…
printf("hi\n");
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,12,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```



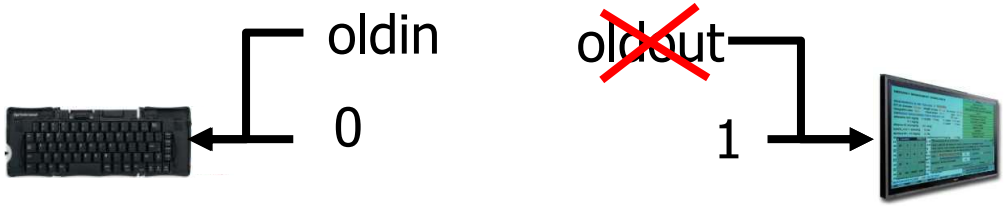
```
…
printf("hi\n");
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,12,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```

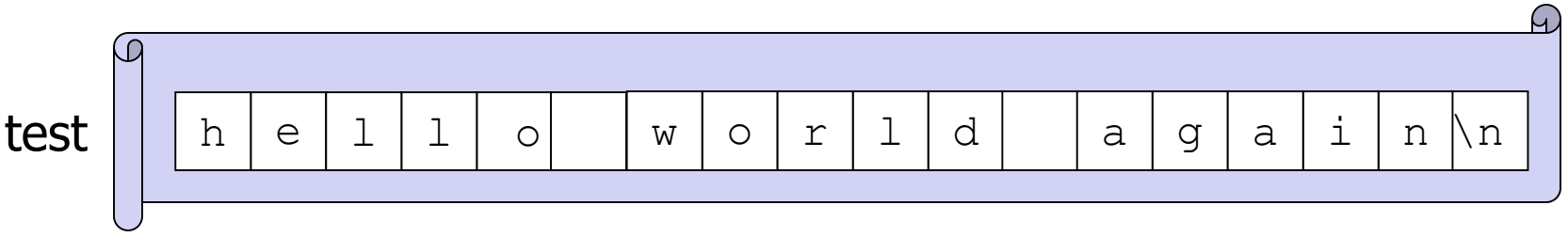


```
…
printf("hi\n");
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,12,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```

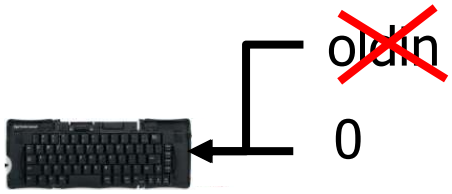


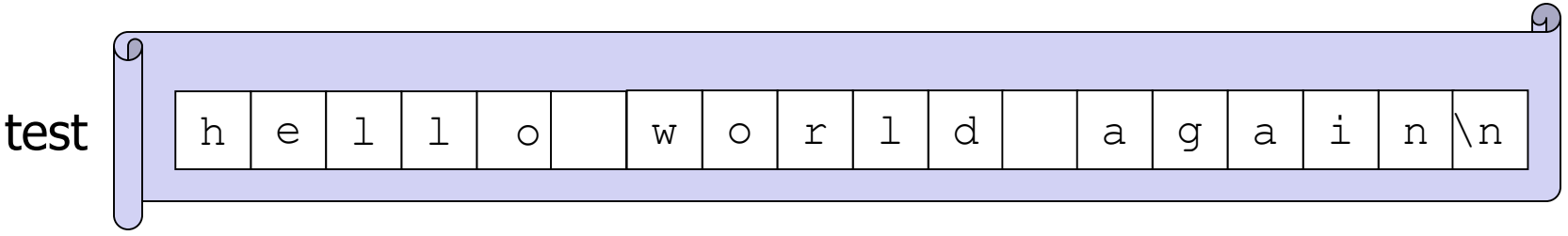
```
…
printf("hi\n");
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,12,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```



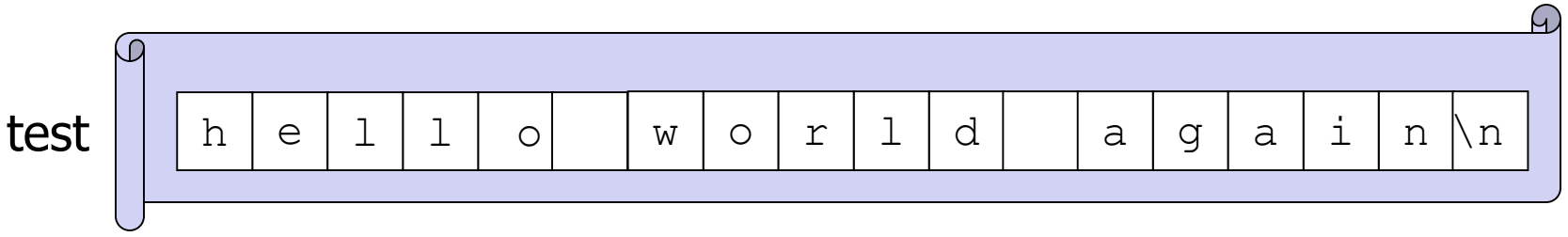


```
…
printf("hi\n");
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,12,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```





```
…
printf("hi\n");
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,12,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```



Ανακατεύθυνση και stdio

- Η χρήση της `stdio` με ανακατεύθυνση της εισόδου/εξόδου σε αρχεία **δεν** είναι εντελώς διαφανής
- Αν η `stdio` χρησιμοποιηθεί σε πρώτη φάση με την καθιερωμένη είσοδο/έξοδο να δείχνει στην συσκευή τερματικού, η πολιτική είναι **line buffered**
  - ακόμα και αν ακολουθήσει ανακατεύθυνση σε αρχείο
- Αν η `stdio` χρησιμοποιηθεί σε πρώτη φάση με την καθιερωμένη είσοδο/έξοδο να δείχνει ήδη σε αρχεία, η πολιτική είναι **fully buffered**
  - για να γραφτούν δεδομένα στο αρχείο πρέπει να ζητηθεί ρητά το άδειασμα της ενδιάμεσης αποθήκης με `fflush`

```
int fd;

fd=open("test",O_WRONLY|O_CREAT|O_TRUNC,S_IRWXU);
dup2(fd,STDOUT_FILENO); /* redirect stdout to fd */
printf("hello world\n");
…
```

fully buffered

```
int fd;

printf("something to set policy to line-buffered\n");
fd=open("test",O_WRONLY|O_CREAT|O_TRUNC,S_IRWXU);
dup2(fd,STDOUT_FILENO); /* redirect stdout to fd */
printf("hello world\n");
…
```

line buffered