Operating Systems

(Sistemas Operativos)

Guide 6: Dup

University of Minho

2024 - 2025

File Interface

Duplicating file descriptors

#include <unistd.h>

- *int **dup**(int fd)*
  - **fd:** the file descriptor
  - Returns: a new file descriptor referring to the **same open file table entry as *fd***
    (-1 on errors)

Obs:

- chooses lowest-numbered available descriptor
- preserves original mode and position of *fd*

For more information: *$ man dup*

#include <unistd.h>

- *int **dup2**(int oldfd, int newfd)*
  - **oldfd:** the file descriptor
  - **newfd:** the file descriptor to refer to the same open file table entry as oldfd
  - Returns: the new file descriptor (*newfd*) or -1 on errors

Obs:

- **if *newfd* is open, *dup2* closes it implicitly**
- preserves original mode and position of *oldfd*

For more information: *$ man dup*

File Interface

Duplicating file descriptors

Imagine we want to redirect all writes done to STDOUT to a file.

How could we do this?

#include <unistd.h>

- *int **dup**(int fd)*
  - **fd:** the file descriptor
  - Returns: a new file descriptor referring to the **same open file table entr[illegible]** *(-1 on errors)*

Obs:

- chooses lowest-numbered available descriptor
- preserves original mode and position of *fd*

For more information: *$ man dup*

#include <unistd.h>

- *int **dup2**(int oldfd, int newfd)*
  - **oldfd:** the file descriptor
  - **newfd:** the file descriptor to refer to the same open file table entry as oldfd
  - Returns: the new file descriptor (*newfd*) or -1 on errors

Obs:

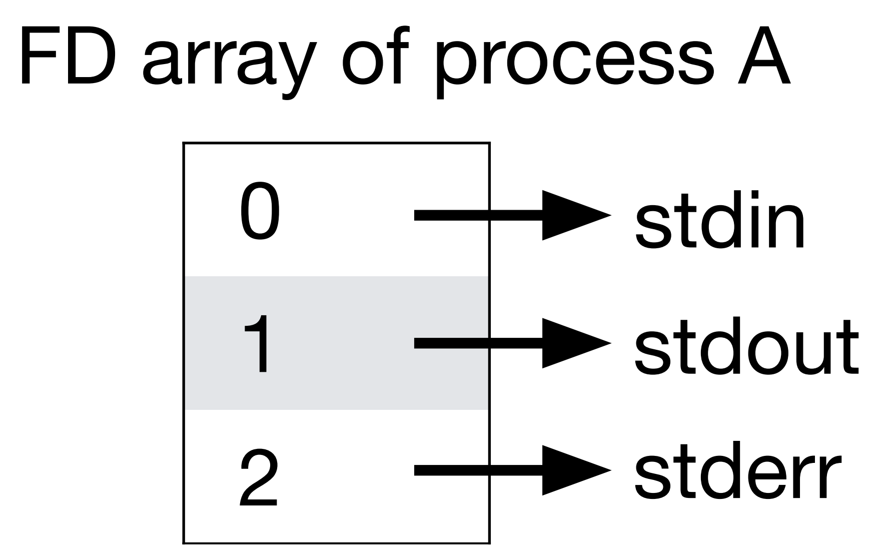
- **if** *newfd* **is open,** ***dup2*** **closes it implicitly**
- preserves original mode and position of *oldfd*

For more information: *$ man dup*

File Interface

Example 1: redirecting STDOUT to a file

```c
int main() {
  int fd = open(“foo.txt”, O_CREAT | O_APPEND | O_WRONLY, 0600);
  printf(“Opened fd=%d\n”, fd);
  dup2(fd, 1);
  close(fd);
  printf(“Redirected stdout to foo.txt\n”);
  // ...
  return 0;
}
```

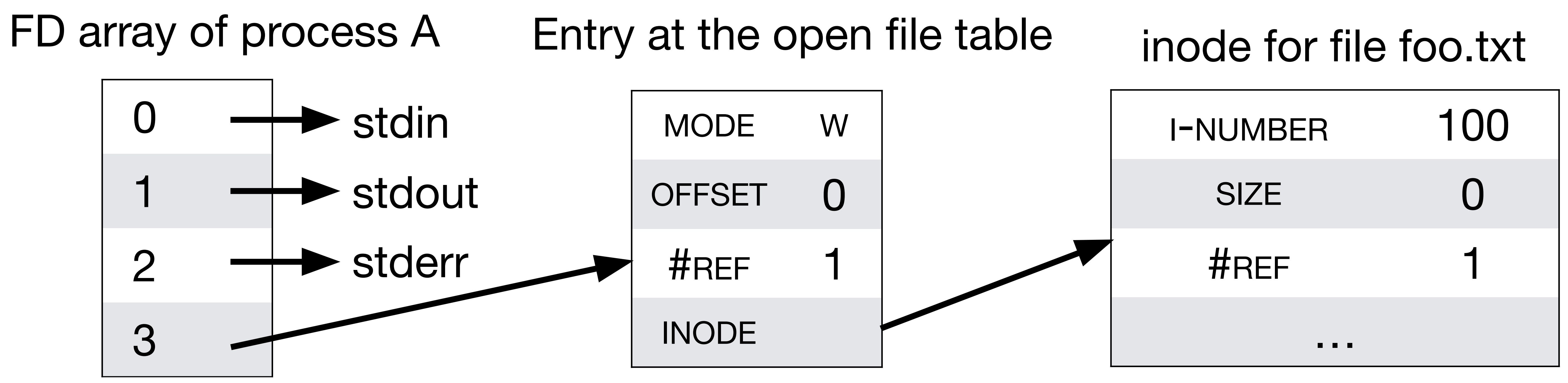


File Interface

Example 1: redirecting STDOUT to a file

```c
int main() {
  int fd = open("foo.txt", O_CREAT |
O_APPEND | O_WRONLY, 0600);
  printf("Opened fd=%d\n", fd);
  dup2(fd, 1);
  close(fd);
  printf("Redirected stdout to
foo.txt\n");
  // ...
  return 0;
}
```

Opening file *foo.txt* for writing …

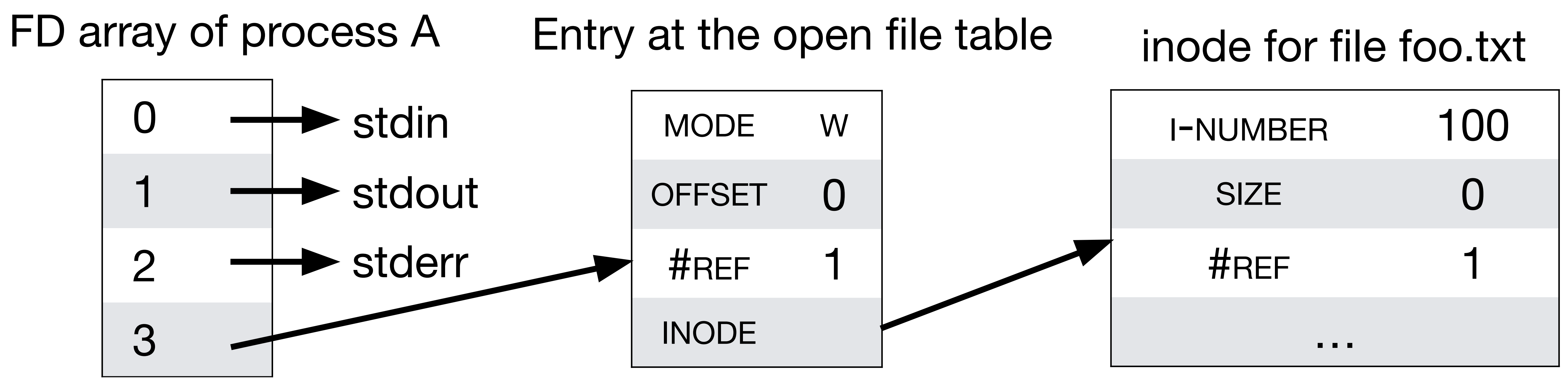


File Interface

Example 1: redirecting STDOUT to a file

```
int main() {
  int fd = open("foo.txt", O_CREAT | O_APPEND | O_WRONLY, 0600);
  printf("Opened fd=%d\n", fd);
  dup2(fd, 1);
  close(fd);
  printf("Redirected stdout to foo.txt\n");
  // ...
  return 0;
}
```

Writing to STDOUT …



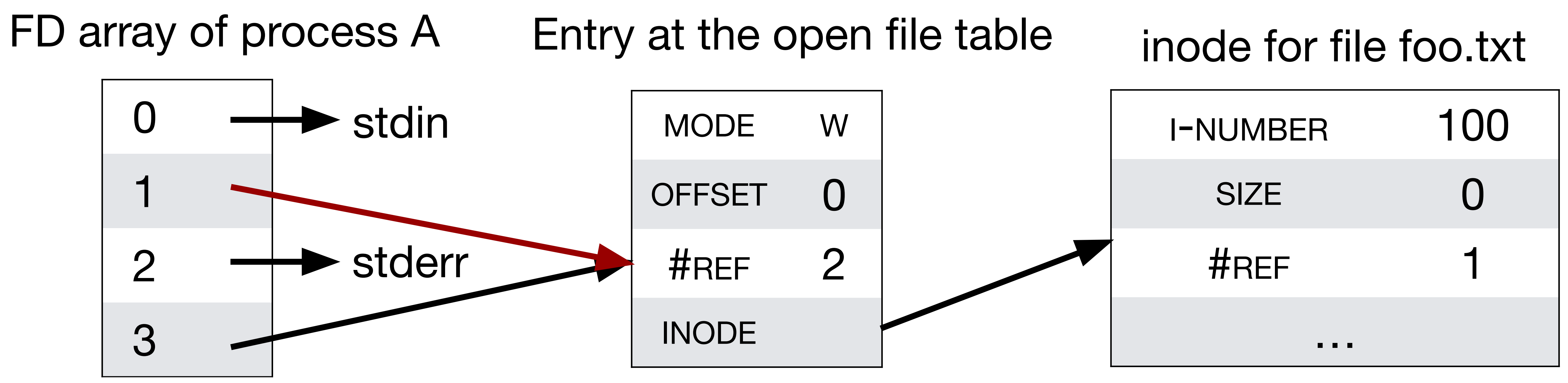
At this point, a ***write***(*1,...*) or a ***printf*** will result in data being written to the **stdout (display)**!

File Interface

Example 1: redirecting STDOUT to a file

```c
int main() {
  int fd = open("foo.txt", O_CREAT | O_APPEND | O_WRONLY, 0600);
  printf("Opened fd=%d\n", fd);
  dup2(fd, 1);
  close(fd);
  printf("Redirected stdout to foo.txt\n");
  // ...
  return 0;
}
```

Redirecting STDOUT to file *foo.txt* …



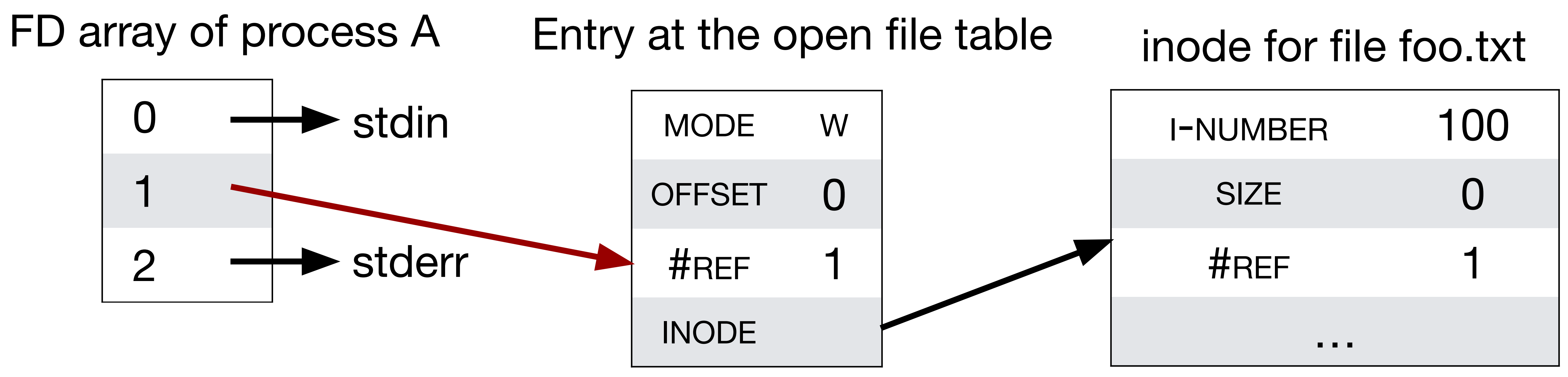
As of now, a ***write****(1,...)* or a ***printf*** will result in data being written to **file foo.txt!**

File Interface

Example 1: redirecting STDOUT to a file

```
int main() {
  int fd = open("foo.txt", O_CREAT |
O_APPEND | O_WRONLY, 0600);
  printf("Opened fd=%d\n", fd);
  dup2(fd, 1);
  close(fd);
  printf("Redirected stdout to
foo.txt\n");
  // ...
  return 0;
}
```

Closing unused file descriptors …

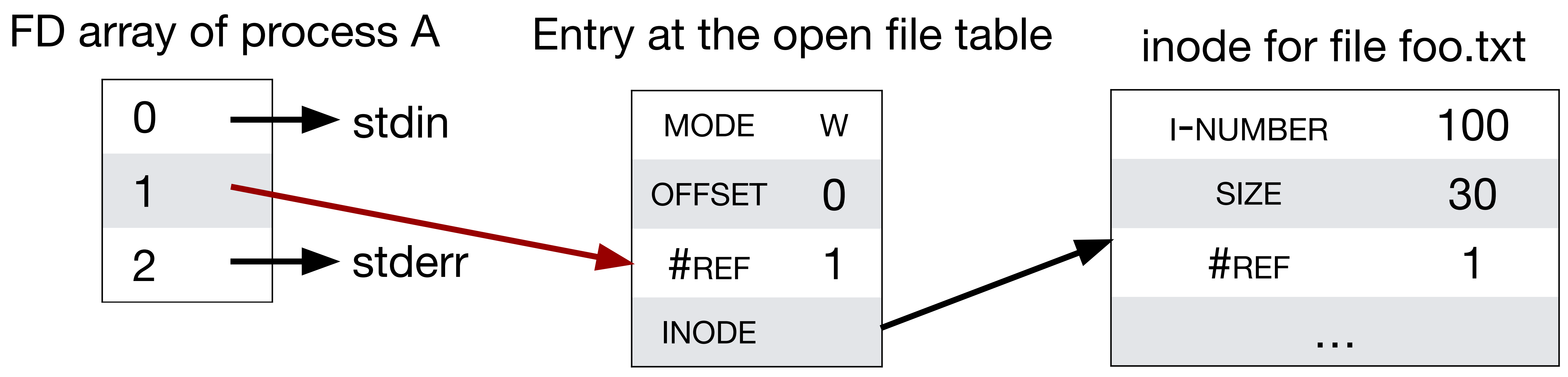


File Interface

Example 1: redirecting STDOUT to a file

```c
int main() {
  int fd = open(“foo.txt”, O_CREAT |
O_APPEND | O_WRONLY, 0600);
  printf(“Opened fd=%d\n”, fd);
  dup2(fd, 1);
  close(fd);
  printf(“Redirected stdout to
foo.txt\n”);
  // ...
  return 0;
}
```

Writing to STDOUT …



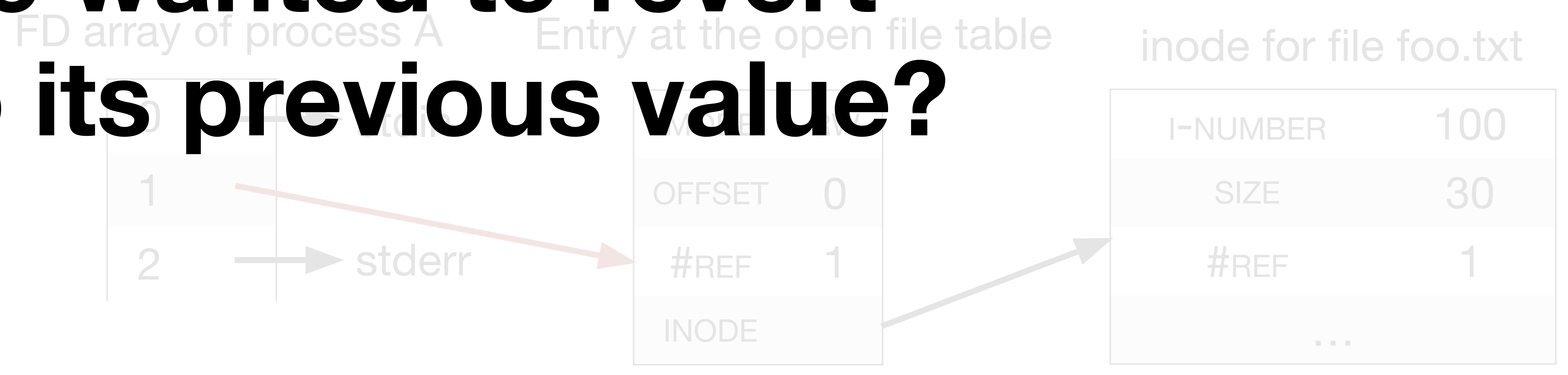
The ***printf*** will result in a write of 30 bytes to **file *foo.txt***.

File Interface

Example 1: redirecting STDOUT to a file

```
int main() {
  int fd = open("foo.txt", O_CREAT |
O_APPEND | O_WRONLY, 0600);
  printf("Opened fd=%d\n", fd);
  dup2(fd, 1);
  close(fd);
  printf("Redirected stdout to
foo.txt\n");
  // ...
  return 0;
}
```

Closing unused file descriptors …

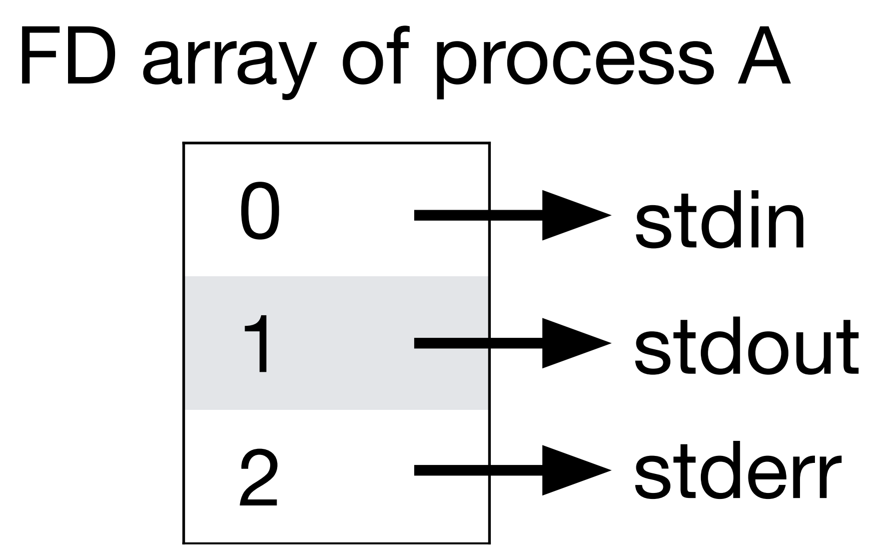


What if we wanted to revert STDOUT to its previous value?

File Interface

Example 2: redirecting STDOUT to a file, then restoring to its previous value

```c
int main() {
  int fd = open(“foo.txt”, O_CREAT | O_APPEND | O_WRONLY, 0600);
  int original_stdout = dup(1);
  dup2(fd, 1);
  close(fd);
  printf(“Redirected stdout to foo.txt\n”);
  // ...
  dup2(original_stdout, 1);
  close(original_stdout);
  printf(“Restored STDOUT\n”);
  return 0;
}
```

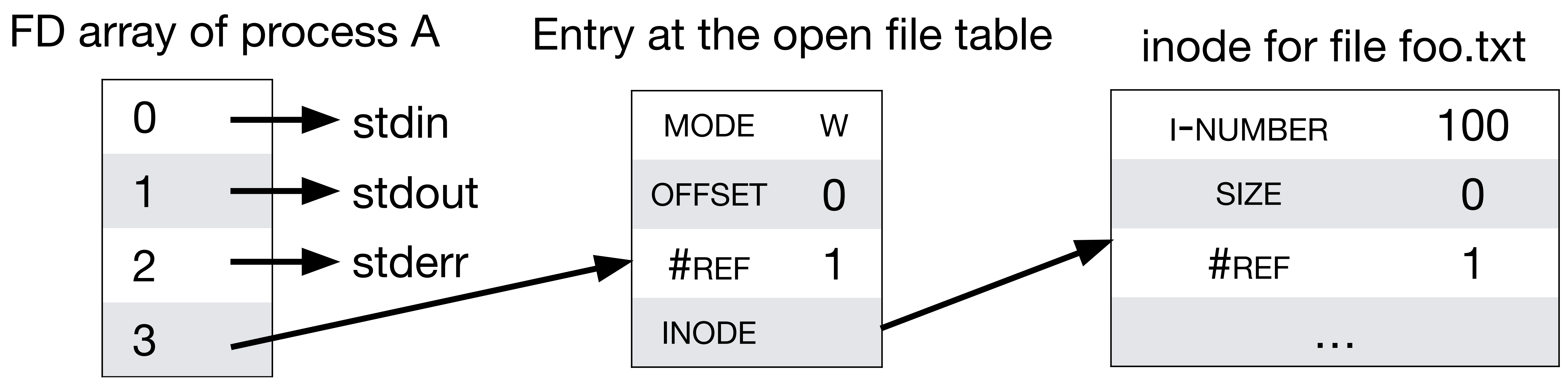


File Interface

Example 2: redirecting STDOUT to a file, then restoring to its previous value

```
int main() {
  int fd = open("foo.txt", O_CREAT | O_APPEND | O_WRONLY, 0600);
  int original_stdout = dup(1);
  dup2(fd, 1);
  close(fd);
  printf("Redirected stdout to foo.txt\n");
  // ...
  dup2(original_stdout, 1);
  close(original_stdout);
  printf("Restored STDOUT\n");
  return 0;
}
```

Opening file *foo.txt* for writing …

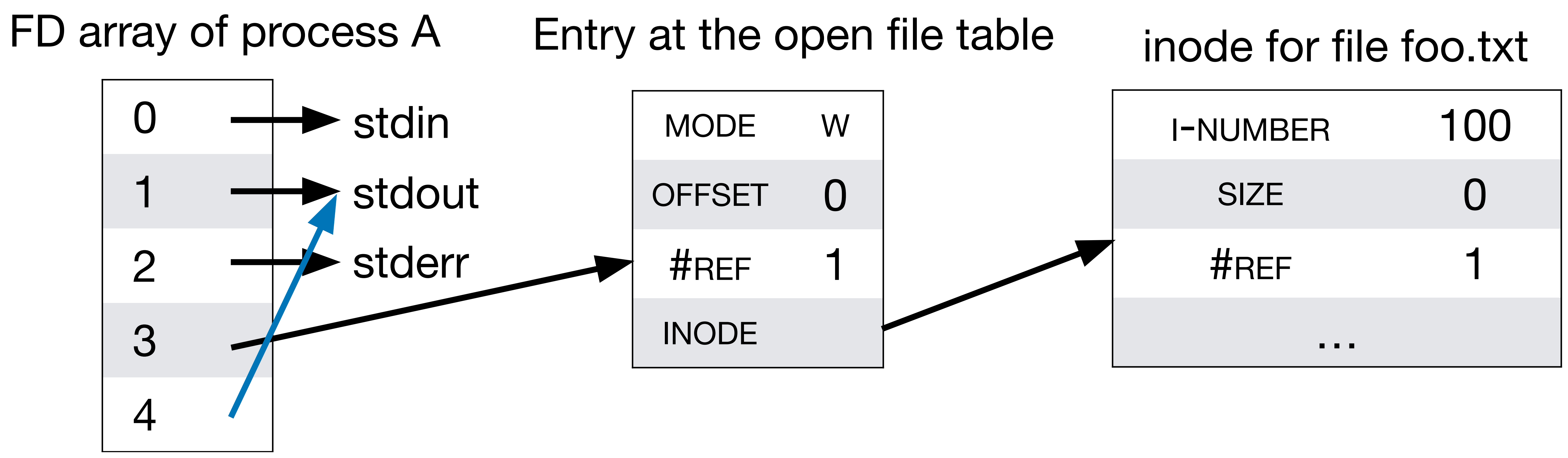


File Interface

Example 2: redirecting STDOUT to a file, then restoring to its previous value

```c
int main() {
  int fd = open("foo.txt", O_CREAT |
O_APPEND | O_WRONLY, 0600);
  int original_stdout = dup(1);
  dup2(fd, 1);
  close(fd);
  printf("Redirected stdout to
foo.txt\n");
  // ...
  dup2(original_stdout, 1);
  close(original_stdout);
  printf("Restored STDOUT\n");
  return 0;
}
```

Creating a copy of STDOUT descriptor …

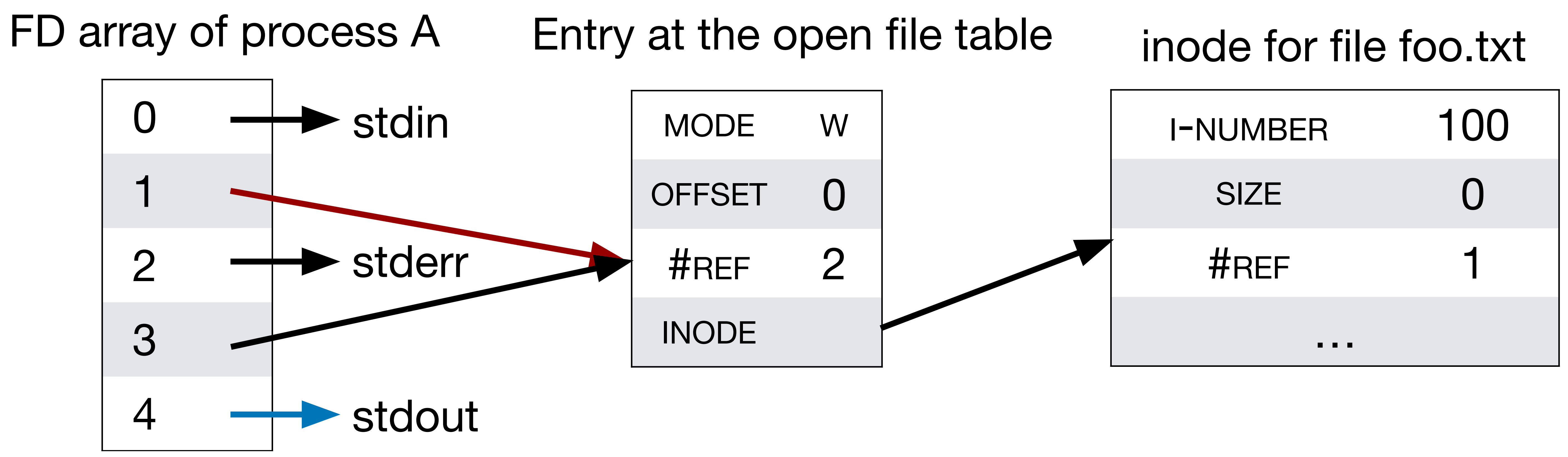


File Interface

Example 2: redirecting STDOUT to a file, then restoring to its previous value

```
int main() {
  int fd = open("foo.txt", O_CREAT |
O_APPEND | O_WRONLY, 0600);
  int original_stdout = dup(1);
  dup2(fd, 1);
  close(fd);
  printf("Redirected stdout to
foo.txt\n");
  // ...
  dup2(original_stdout, 1);
  close(original_stdout);
  printf("Restored STDOUT\n");
  return 0;
}
```

Redirecting STDOUT to file *foo.txt* …



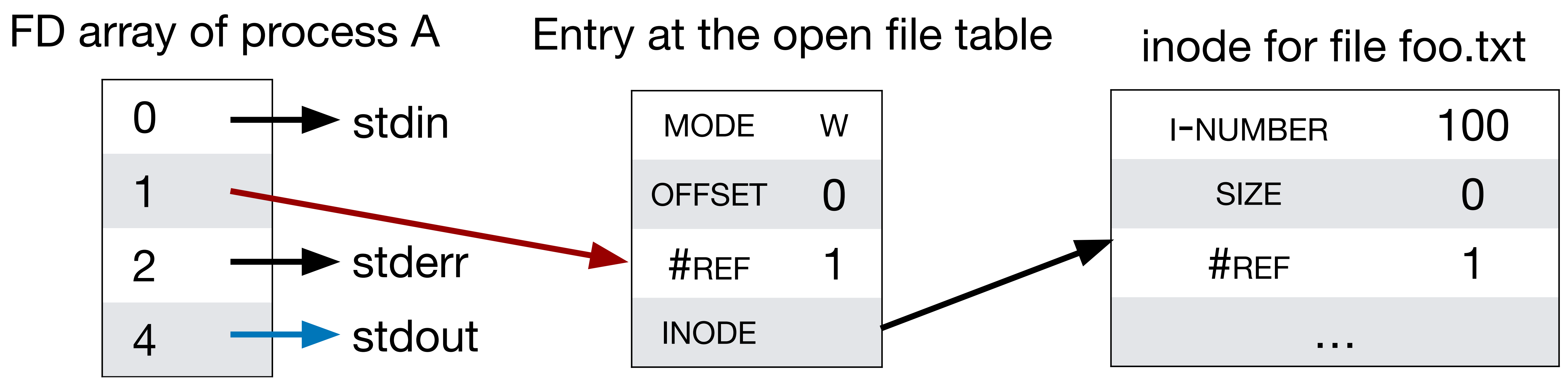
As of now, a ***write****(1,...)* or a ***printf*** will result in data being written to **file foo.txt!**

File Interface

Example 2: redirecting STDOUT to a file, then restoring to its previous value

```c
int main() {
  int fd = open("foo.txt", O_CREAT | O_APPEND | O_WRONLY, 0600);
  int original_stdout = dup(1);
  dup2(fd, 1);
  close(fd);
  printf("Redirected stdout to foo.txt\n");
  // ...
  dup2(original_stdout, 1);
  close(original_stdout);
  printf("Restored STDOUT\n");
  return 0;
}
```

Closing unused file descriptors …

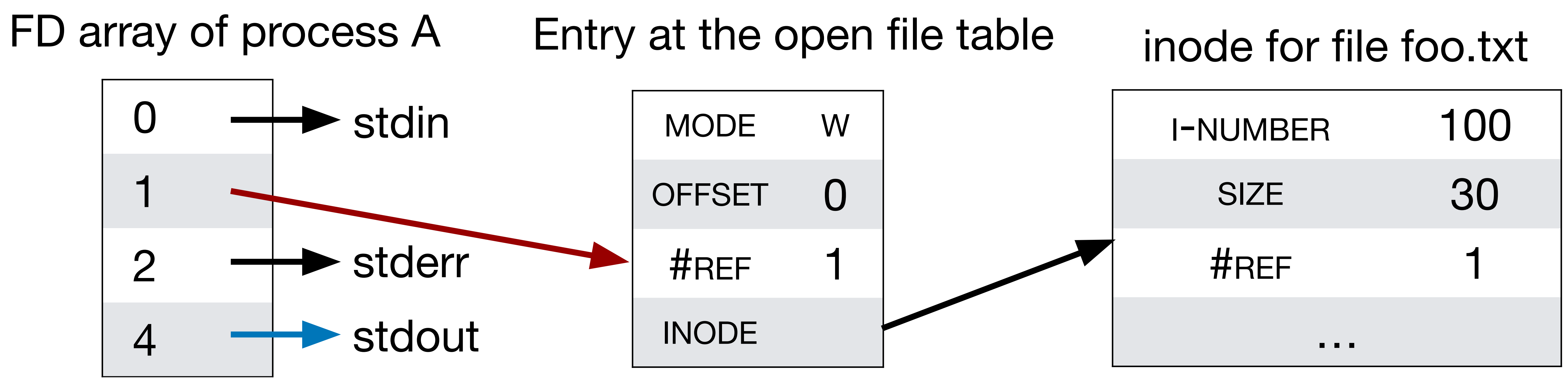


File Interface

Example 2: redirecting STDOUT to a file, then restoring to its previous value

```c
int main() {
  int fd = open("foo.txt", O_CREAT |
O_APPEND | O_WRONLY, 0600);
  int original_stdout = dup(1);
  dup2(fd, 1);
  close(fd);
  printf("Redirected stdout to
foo.txt\n");
  // ...
  dup2(original_stdout, 1);
  close(original_stdout);
  printf("Restored STDOUT\n");
  return 0;
}
```

Writing to STDOUT …



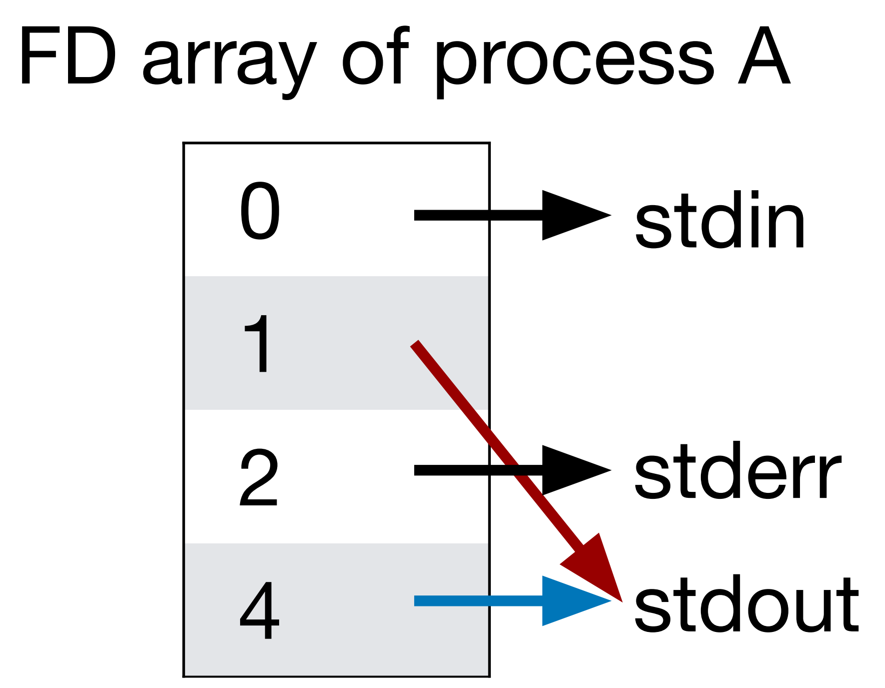
The ***printf*** will result in a write of 30 bytes to **file *foo.txt***.

File Interface

Example 2: redirecting STDOUT to a file, then restoring to its previous value

```c
int main() {
  int fd = open(“foo.txt”, O_CREAT |
O_APPEND | O_WRONLY, 0600);
  int original_stdout = dup(1);
  dup2(fd, 1);
  close(fd);
  printf(“Redirected stdout to
foo.txt\n”);
  // ...
  dup2(original_stdout, 1);
  close(original_stdout);
  printf(“Restored STDOUT\n”);
  return 0;
}
```

Redirecting STDOUT to its original value …



inode for file foo.txt

I-NUMBER	100
SIZE	30
#REF	0
…	

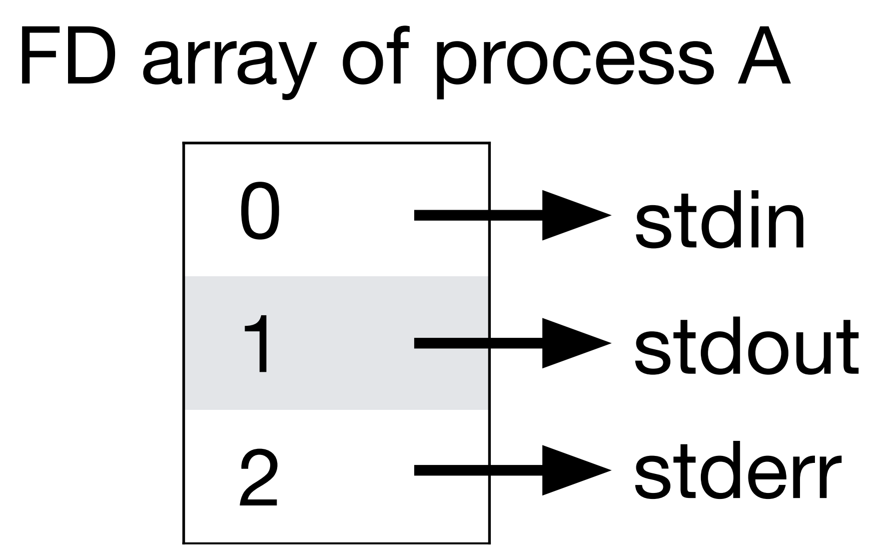
As of now, a ***write****(1,...)* or a ***printf*** will result in data being written again to **stdout (display)!**

File Interface

Example 2: redirecting STDOUT to a file, then restoring to its previous value

```c
int main() {
  int fd = open(“foo.txt”, O_CREAT |
O_APPEND | O_WRONLY, 0600);
  int original_stdout = dup(1);
  dup2(fd, 1);
  close(fd);
  printf(“Redirected stdout to
foo.txt\n”);
  // ...
  dup2(original_stdout, 1);
  close(original_stdout);
  printf(“Restored STDOUT\n”);
  return 0;
}
```

Closing unused file descriptors …



inode for file foo.txt

I-NUMBER	100
SIZE	30
#REF	0
…	

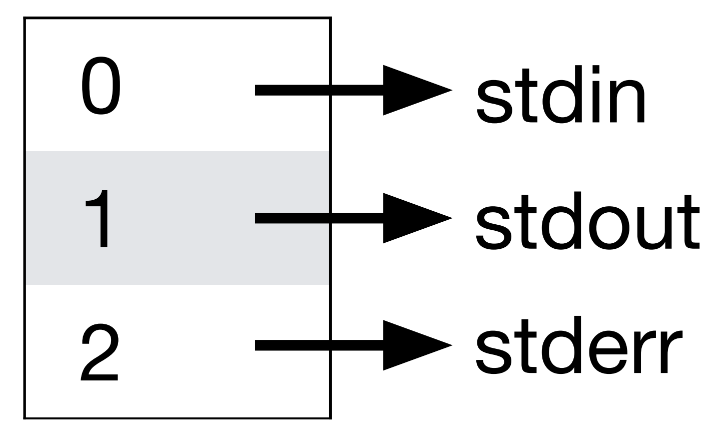
File Interface

Example 2: redirecting STDOUT to a file, then restoring to its previous value

```c
int main() {
  int fd = open(“foo.txt”, O_CREAT | O_APPEND | O_WRONLY, 0600);
  int original_stdout = dup(1);
  dup2(fd, 1);
  close(fd);
  printf(“Redirected stdout to foo.txt\n”);
  // ...
  dup2(original_stdout, 1);
  close(original_stdout);
  printf(“Restored STDOUT\n”);
  return 0;
}
```

Writing to STDOUT …

FD array of process A



inode for file foo.txt

I-NUMBER	100
SIZE	30
#REF	0
…	

The ***printf*** will result in a write of 17 bytes to **stdout (display)!**.

File Interface

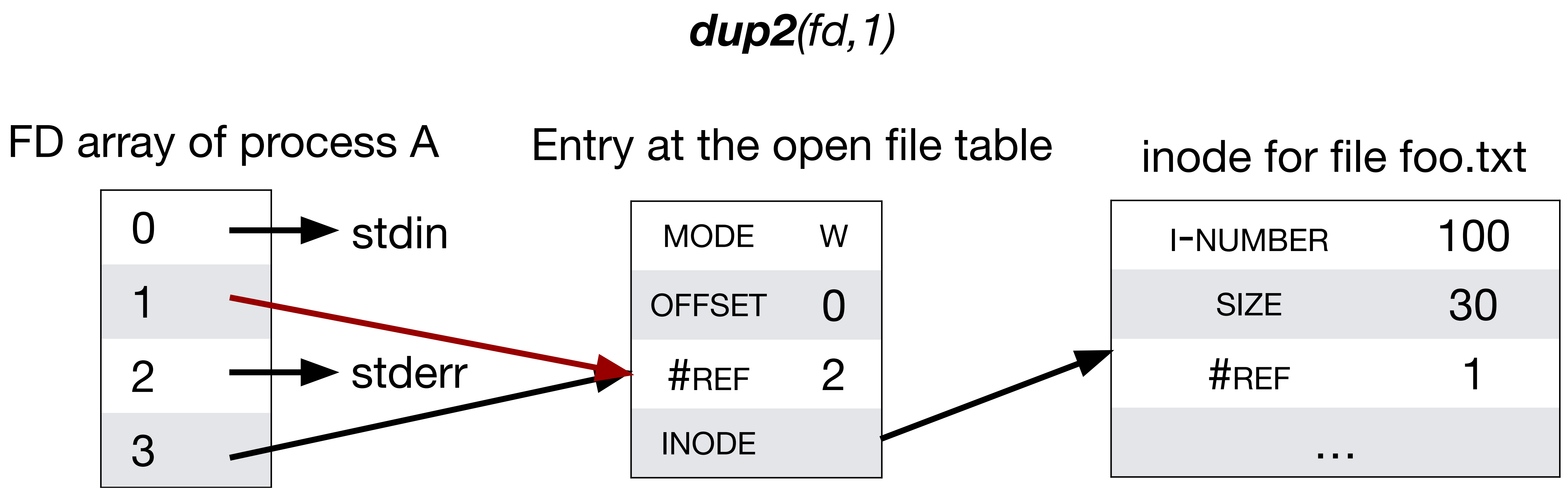
Important remarks – redirecting stdout to a file

- **The memory buffer size changes from a single line to the file system's default size** (typically 4KB).
  - *printf()*, etc. no longer perform line buffering.

- Use *fflush(stdout)* to flush buffered data to the file **(per operation)**

or

- Use setbuf(stdout, NULL) to disable buffering **(globally)**



***dup2**(fd,1)*
FD array of process A
0
1
2
3
stdin
stderr
Entry at the open file table
MODE W
OFFSET 0
#REF 2
INODE
inode for file foo.txt
I-NUMBER 100
SIZE 30
#REF 1
...