Lecture 7: Shell

Xin Liu

xl24j@fsu.edu

COP 4610 Operating Systems

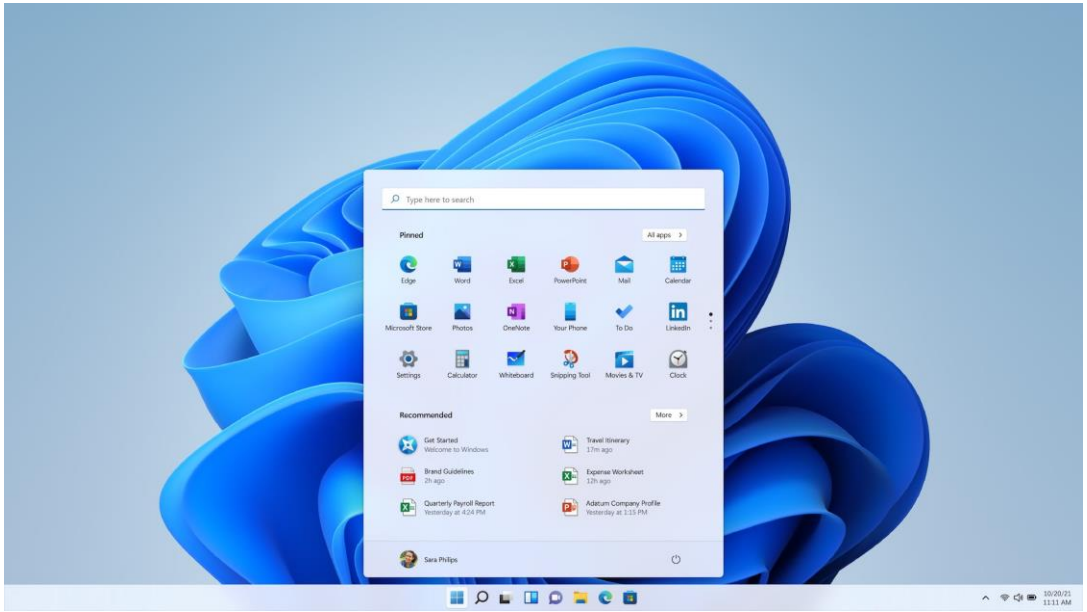
What Have We Learned?

- Building the World of Computer Systems
  - Hardware: Starts executing instructions (computation and I/O) from CPU Reset.
  - Firmware: Loads the operating system.
  - Operating System: The manager of state machines.
  - Initializes the first process (state machine).
    - Executes system calls.

- In the Entire System, Only One Program is Needed
  - **Docker/Virtual Machine**: A single program that can execute various applications.
  - Examples: vim (editor), dosbox (emulator), xeyes (GUI), etc.

•Wrapping the OS API for Users

- We need a program that "users can directly operate" to manage OS objects.
- This is the Shell
  - Kernel provides system calls; Shell provides the user interface.
  - The First Program to Directly Interact with Humans
  - Helps users create/manage processes (applications) and data files.



UNIX Shell

- The Great Design of the "Terminal" Era
  - The pinnacle of the Command-Line Interface (CLI).
- The Shell is a programming language that "translates user commands into system calls".
  - man sh (recommended reading!), bash, etc.
    - Examples: Q5 in HW3
  - We've always been programming, until the emergence of Graphical Shells (GUI).
    - Examples: Windows, Gnome, Symbian, Android.

1970

2020

The Quirky World of UNIX

- "Unix is user-friendly; it's just choosy about who its friends are."

- But if you think of the shell as a programming language, being "hard to use" doesn’t seem like much of a problem.

- After all, have you ever seen a programming language that’s "easy to use"?

A UNIX Shell

- Supported Basic Features:
  - Command execution: ls
  - Redirection: ls > a.txt
    - Using File Descriptors: A "pointer" to an open file
  - Piping: ls | wc -l
  - Background execution: ls &
  - Command combination: (echo a ; echo b) | wc –l

- The Shell Programming Language
  - Core Concept: Text Substitution
  - Building fast workflows based on text substitution
    - Redirection: cmd > file < file 2> /dev/null
    - Sequential execution: cmd1; cmd2, cmd1 && cmd2, cmd1 || cmd2
    - Piping: cmd1 | cmd2
    - Preprocessing: $(), <()
    - Variables/Environment Variables, Control Flow, etc.

Project 1 Hints

- Test code individually

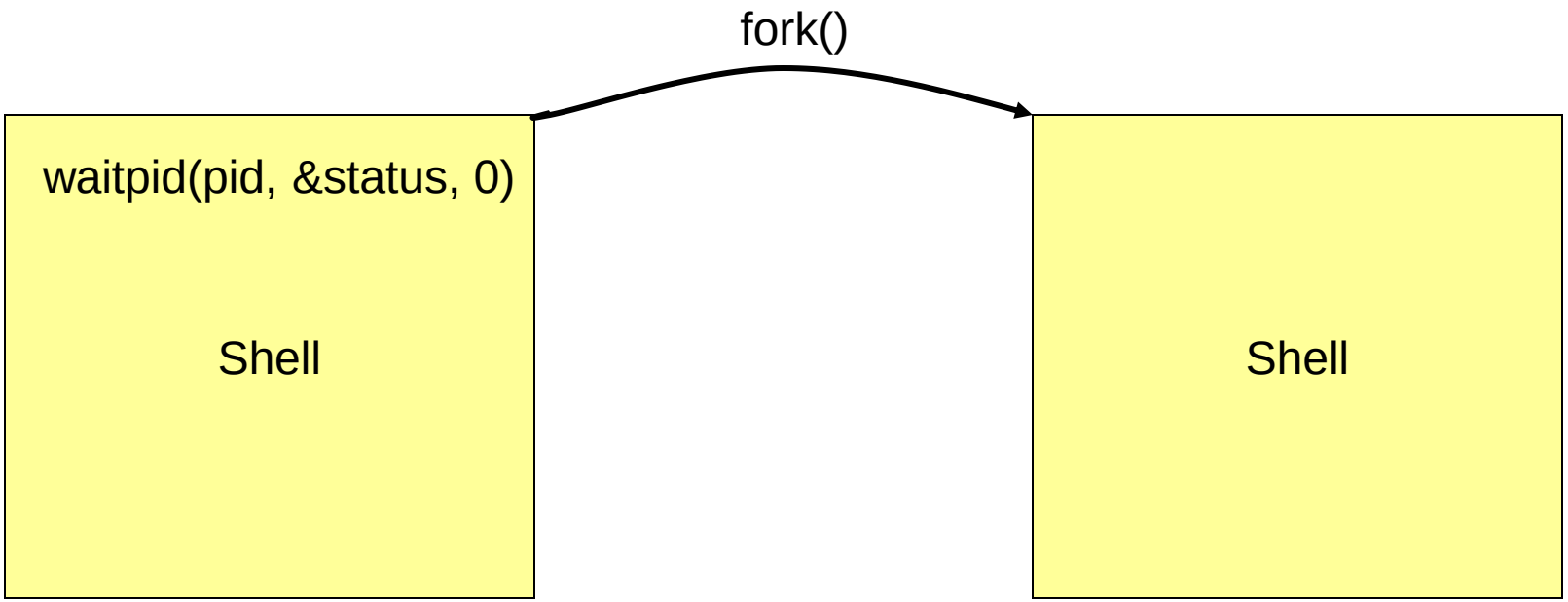
```
#ifdef TEST_PROMPT
        int main() {
        // Your test code for propt.c }
#endif
```

gcc -DTEST_PROMPT -o test_prompt src/prompt.c -Iinclude/ && ./test_prompt

- If your code relies on other codes:
  gcc -DTEST_EXPAND_TOKENS -o test_expand_tokens src/expand_tokens.c src/lexer.c -Iinclude && ./test_expand_tokens

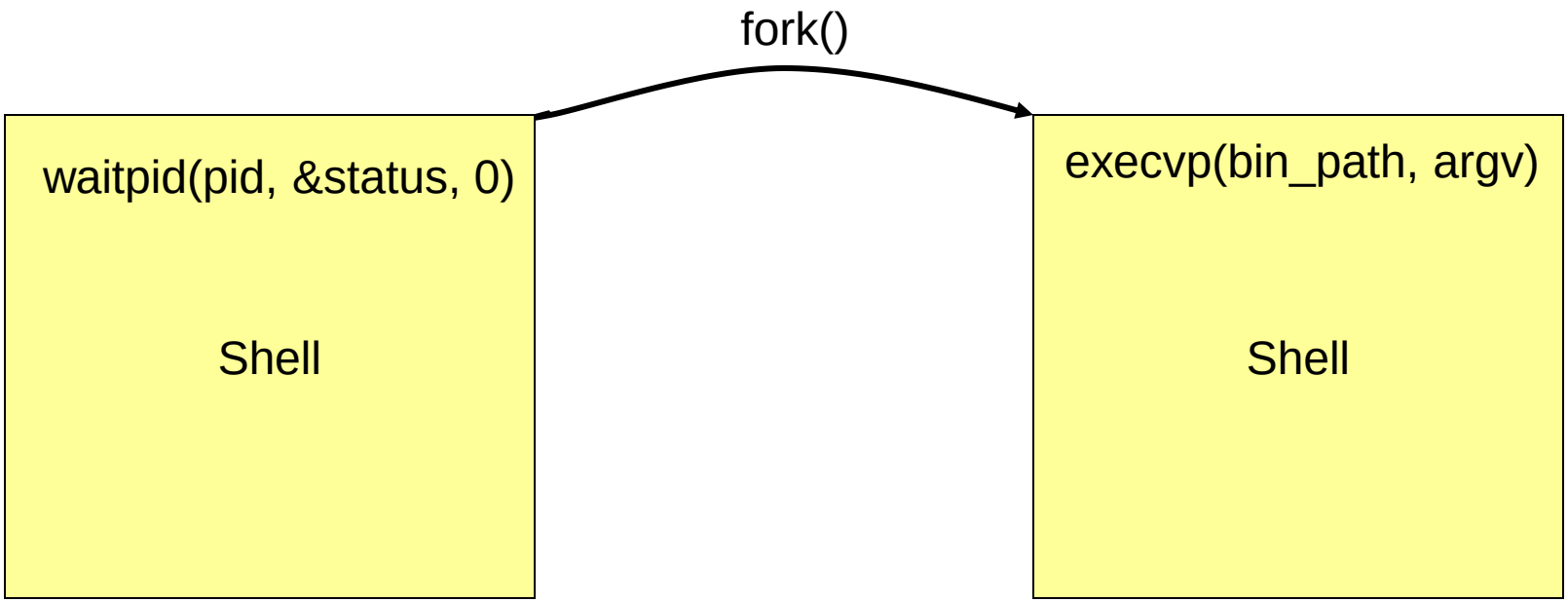
- You should create a Makefile for the whole project.

Foreground Execution

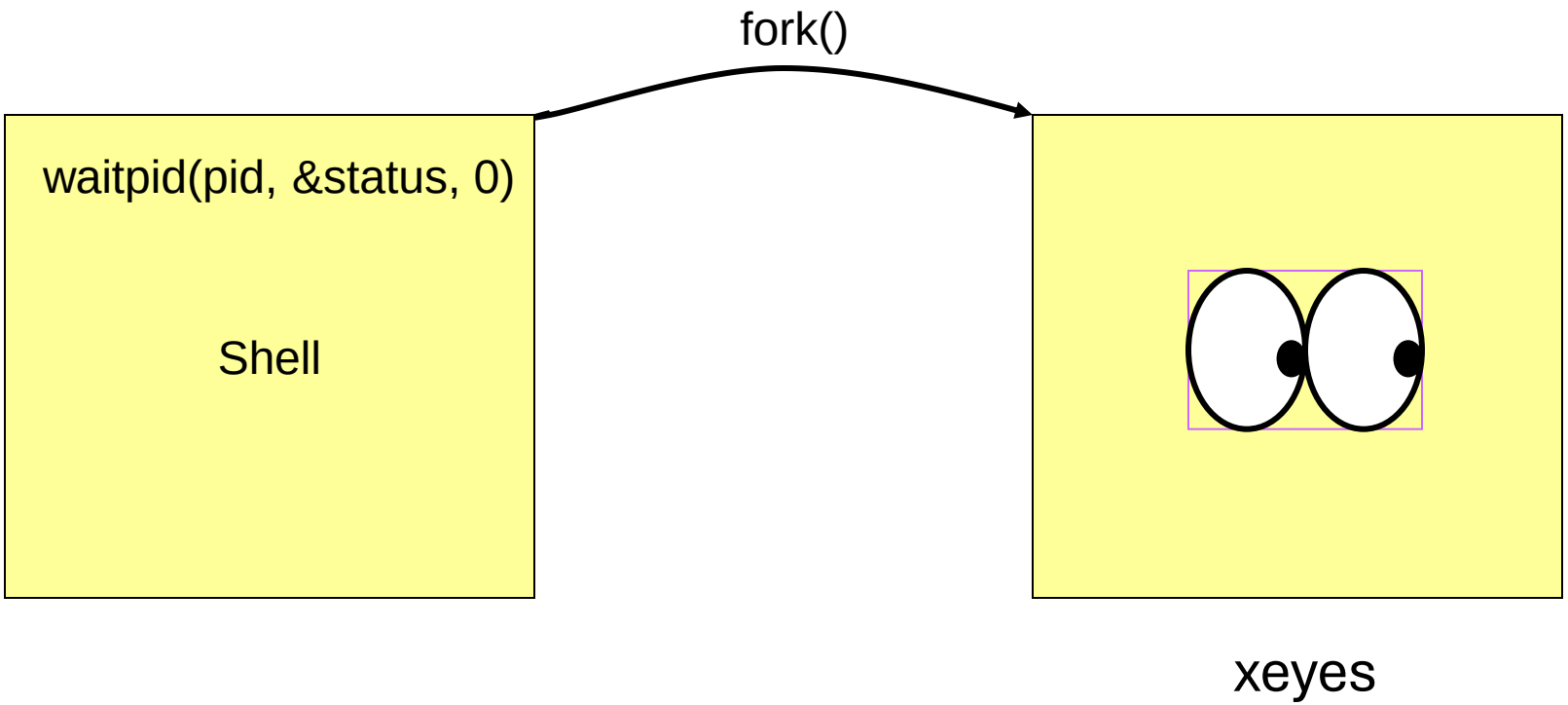


Type “xeyes” in your terminal

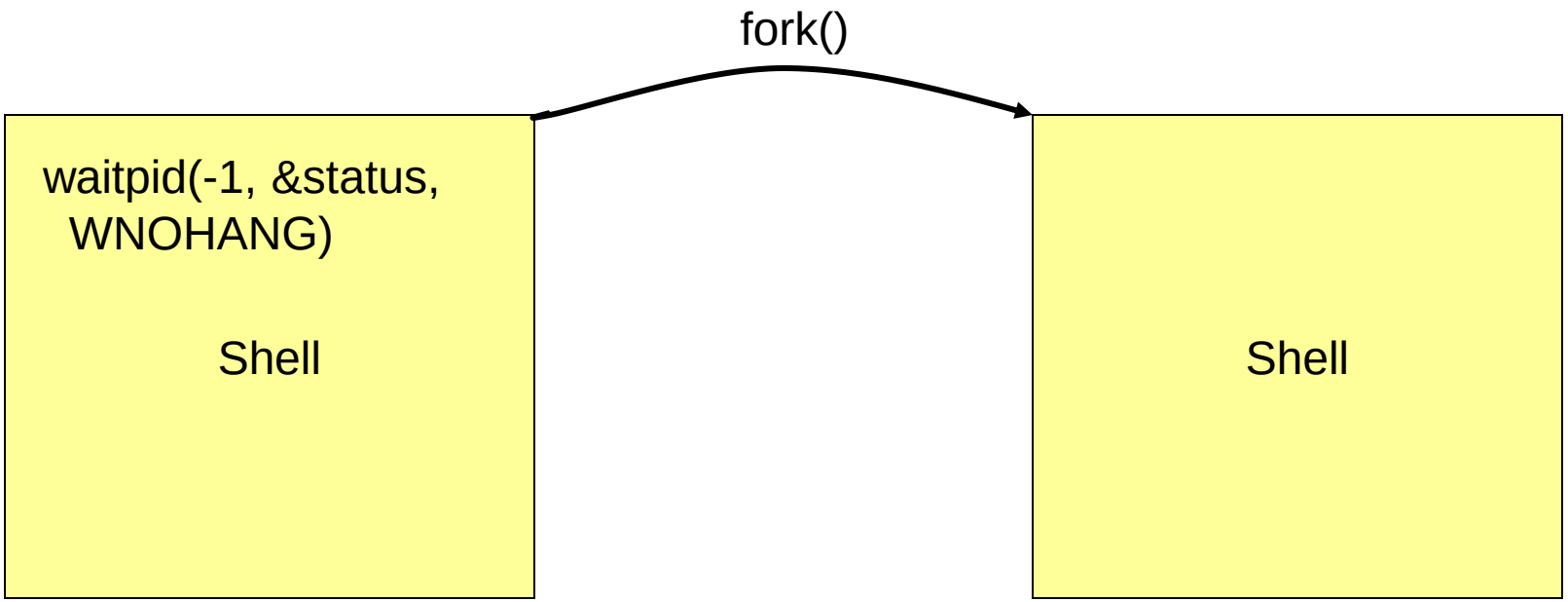
Foreground Execution



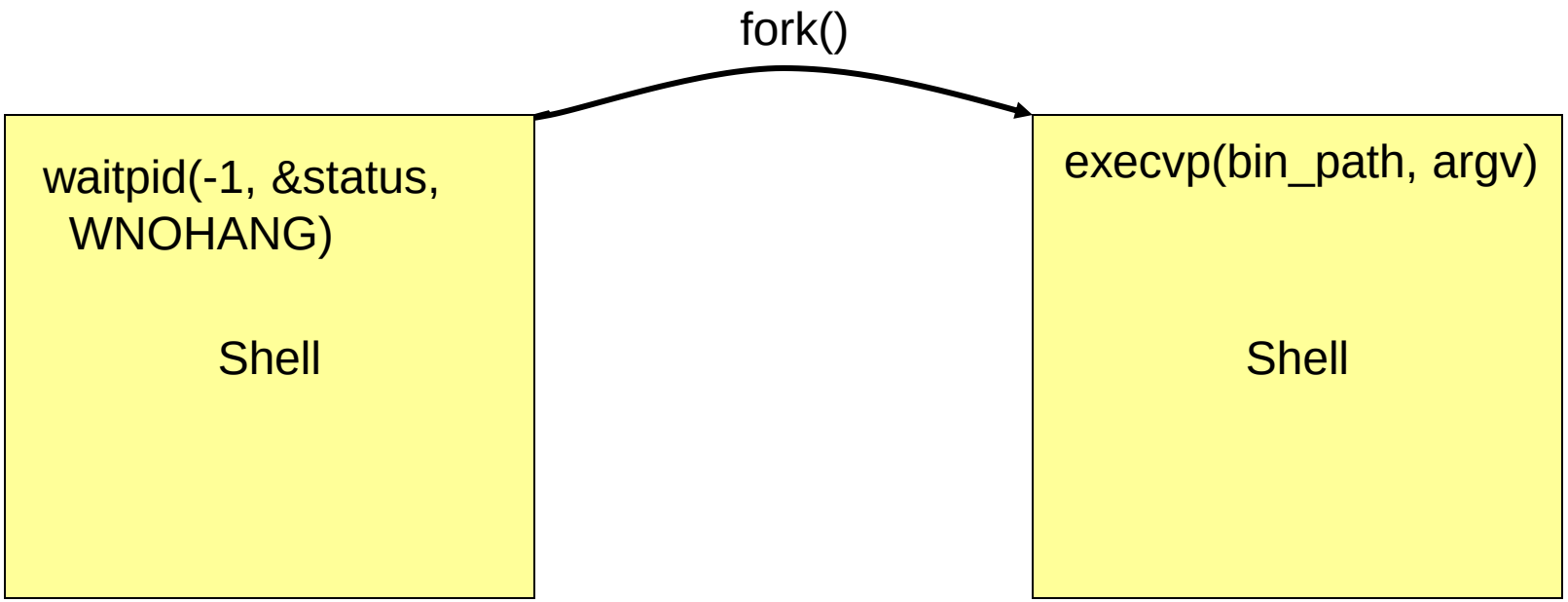
Foreground Execution



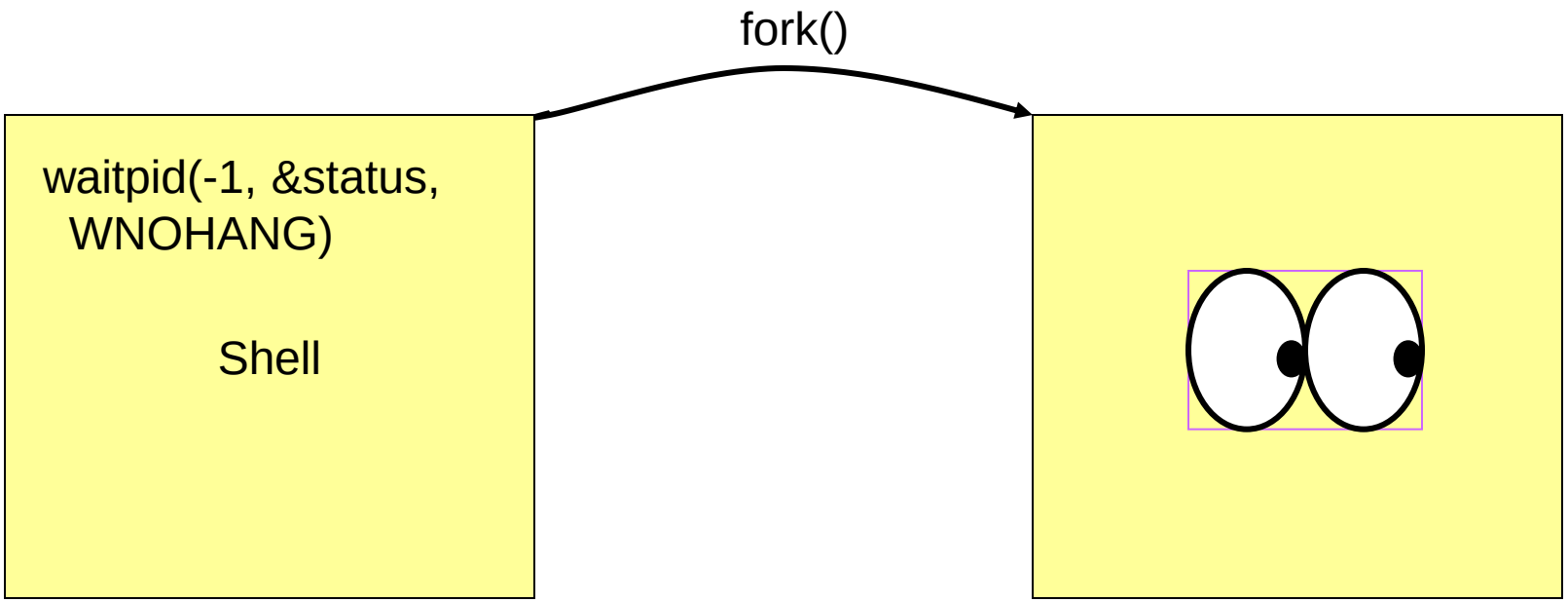
Background Execution



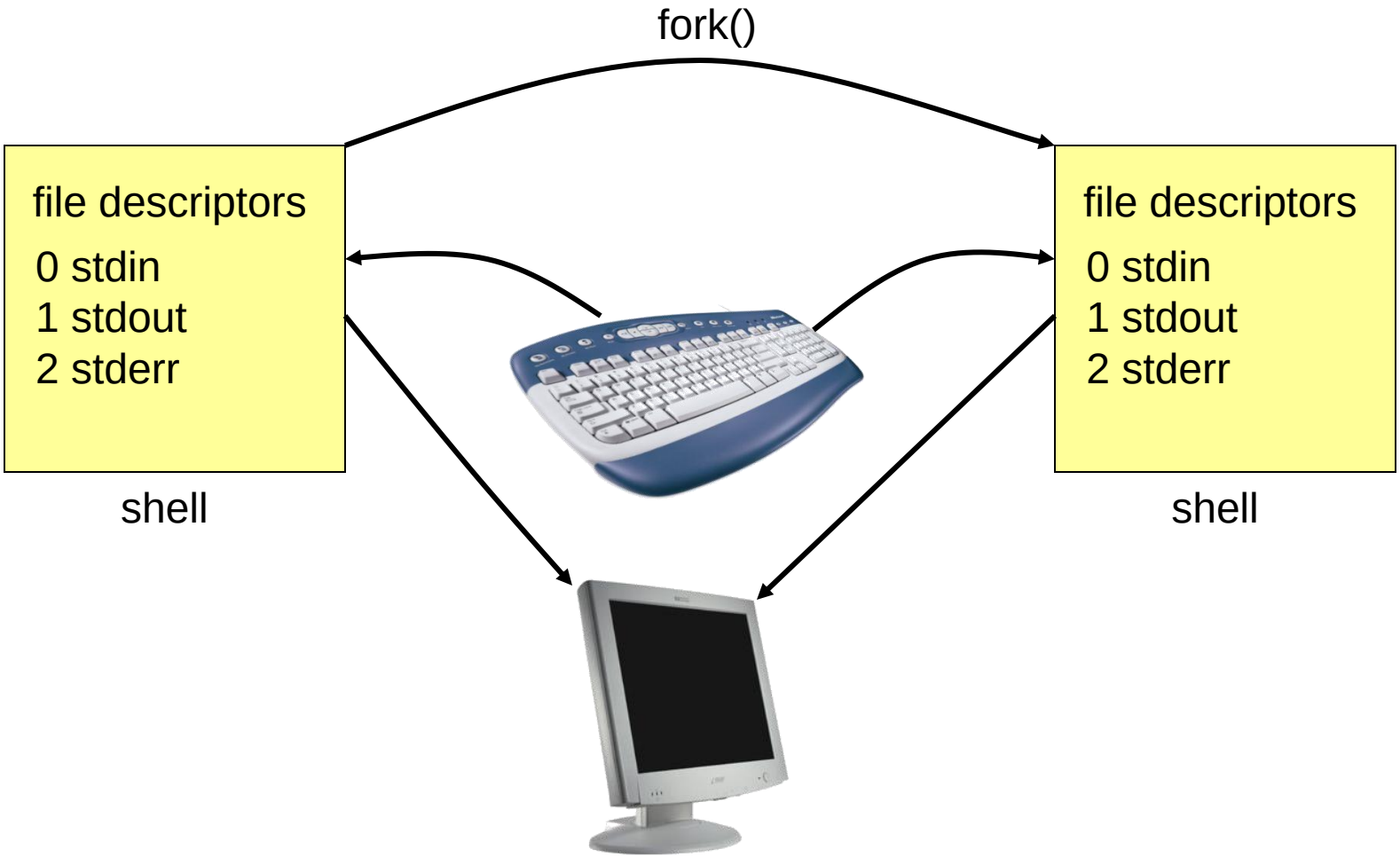
Background Execution



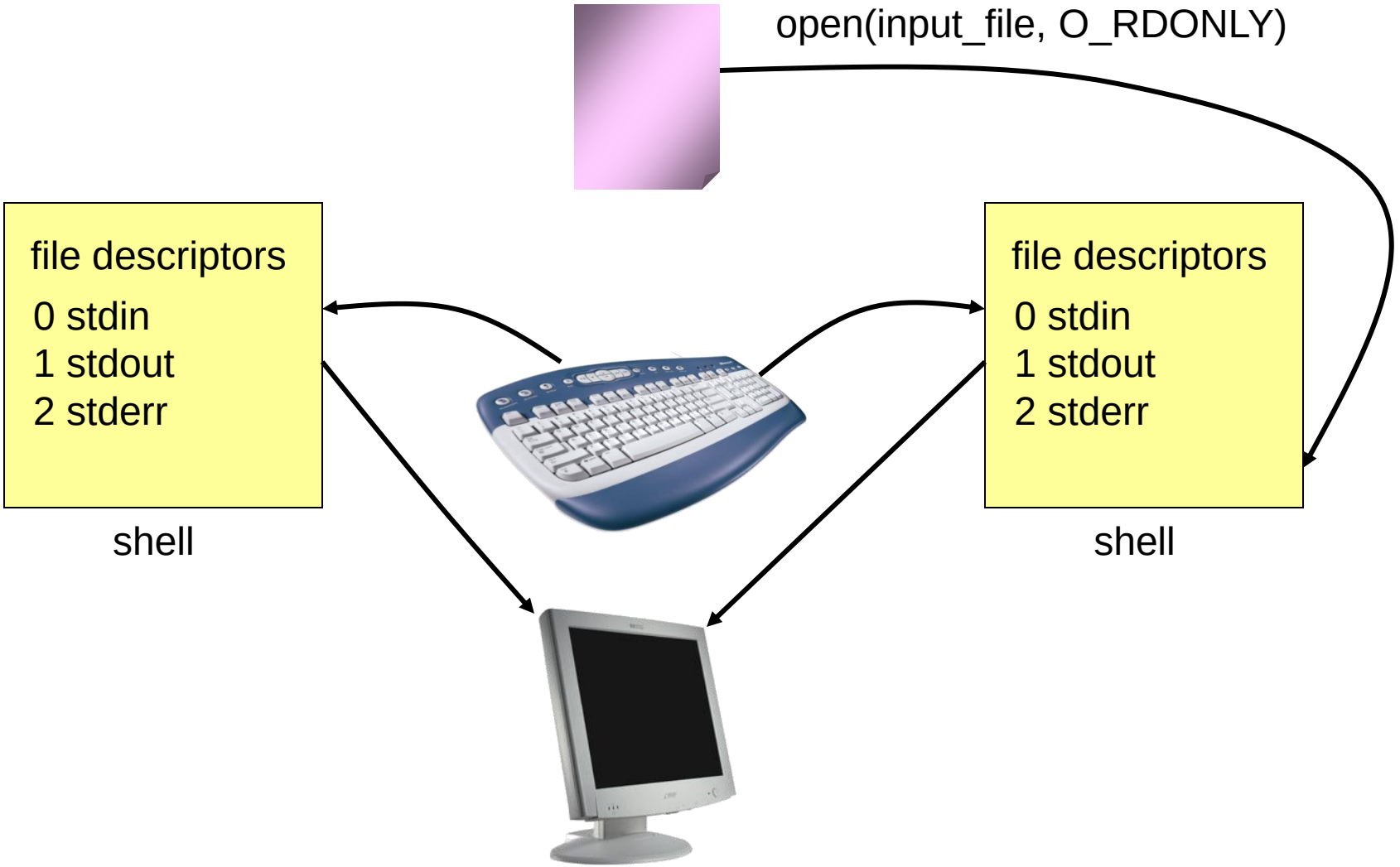
Background Execution



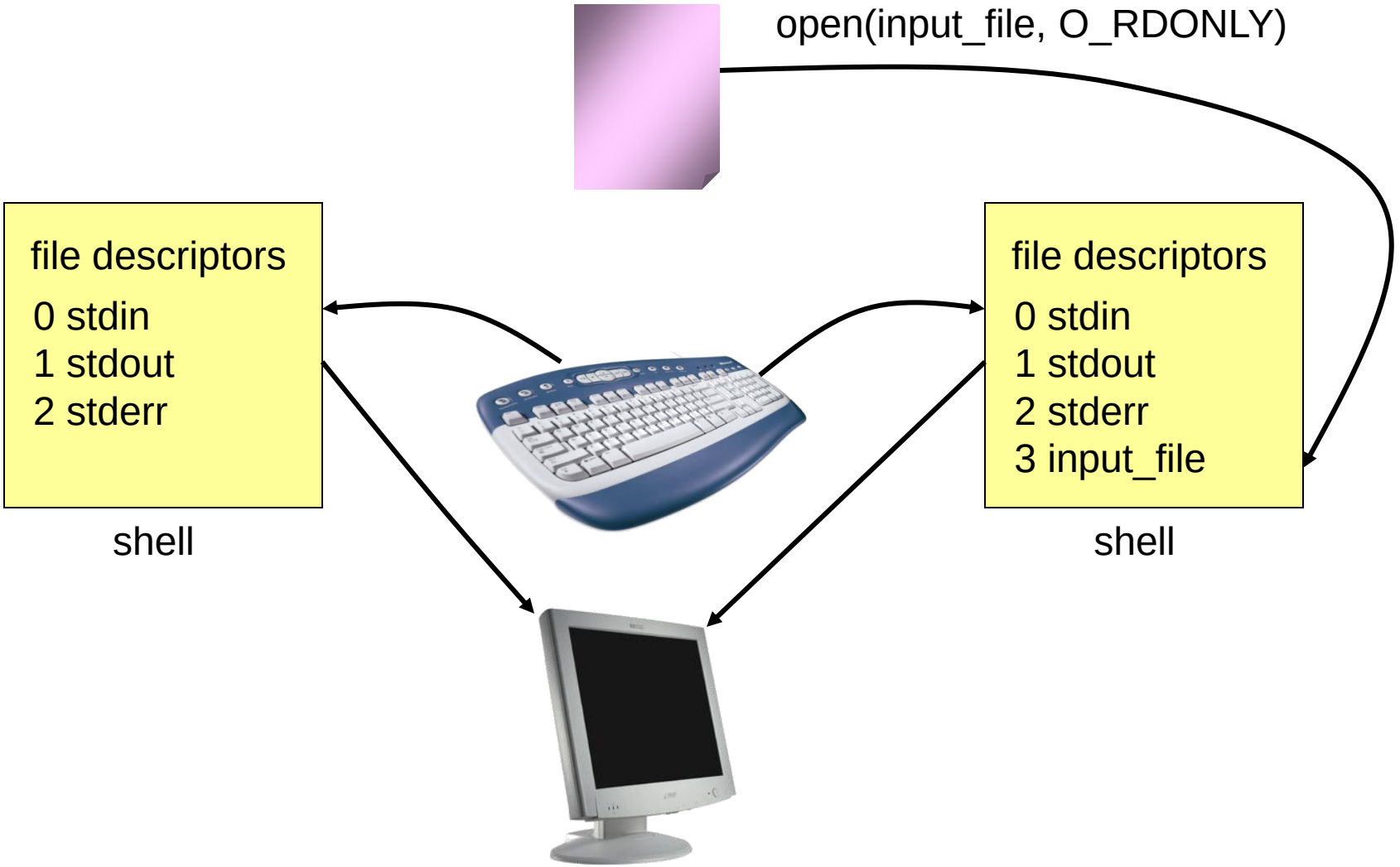
Input Redirection



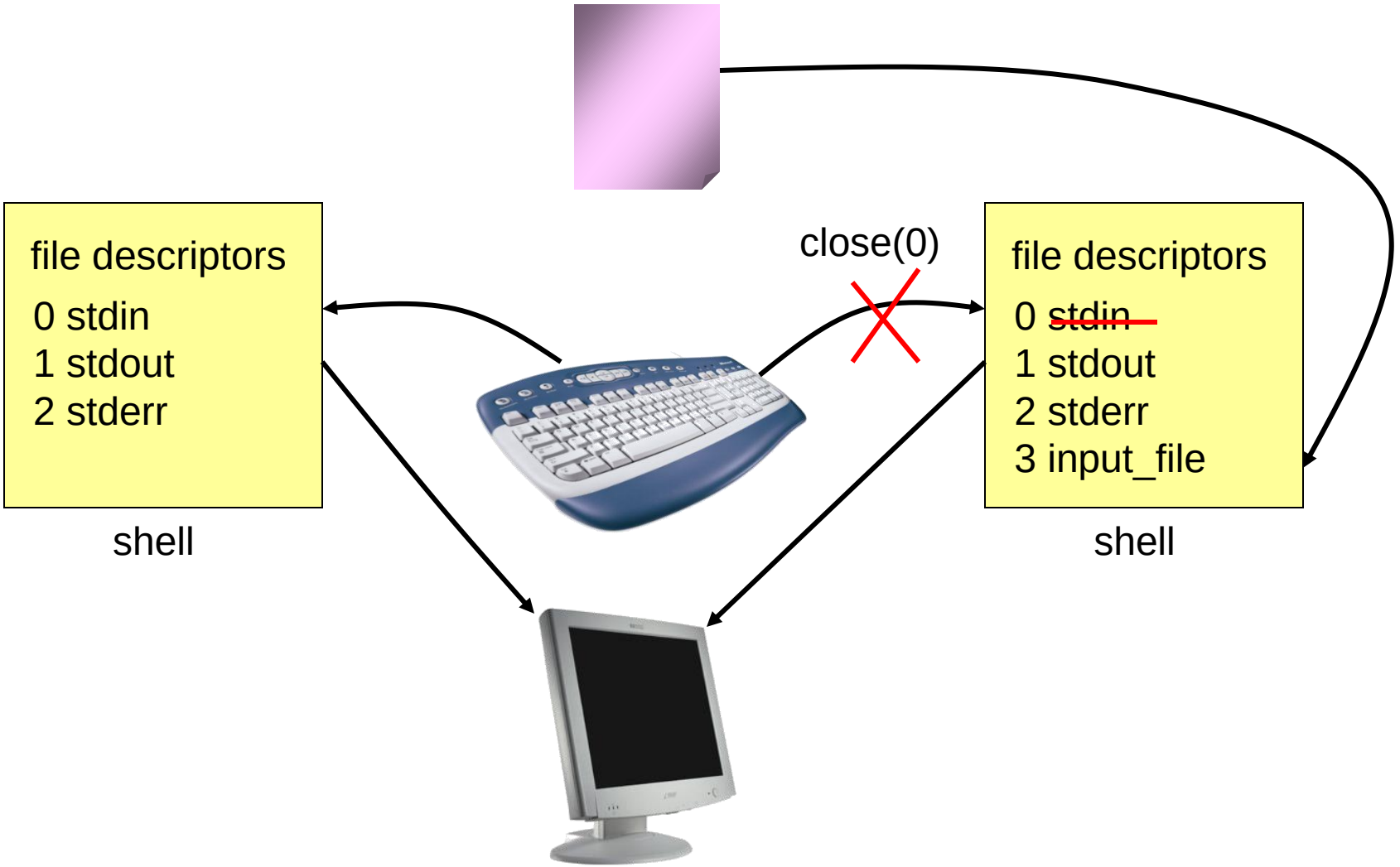
Input Redirection



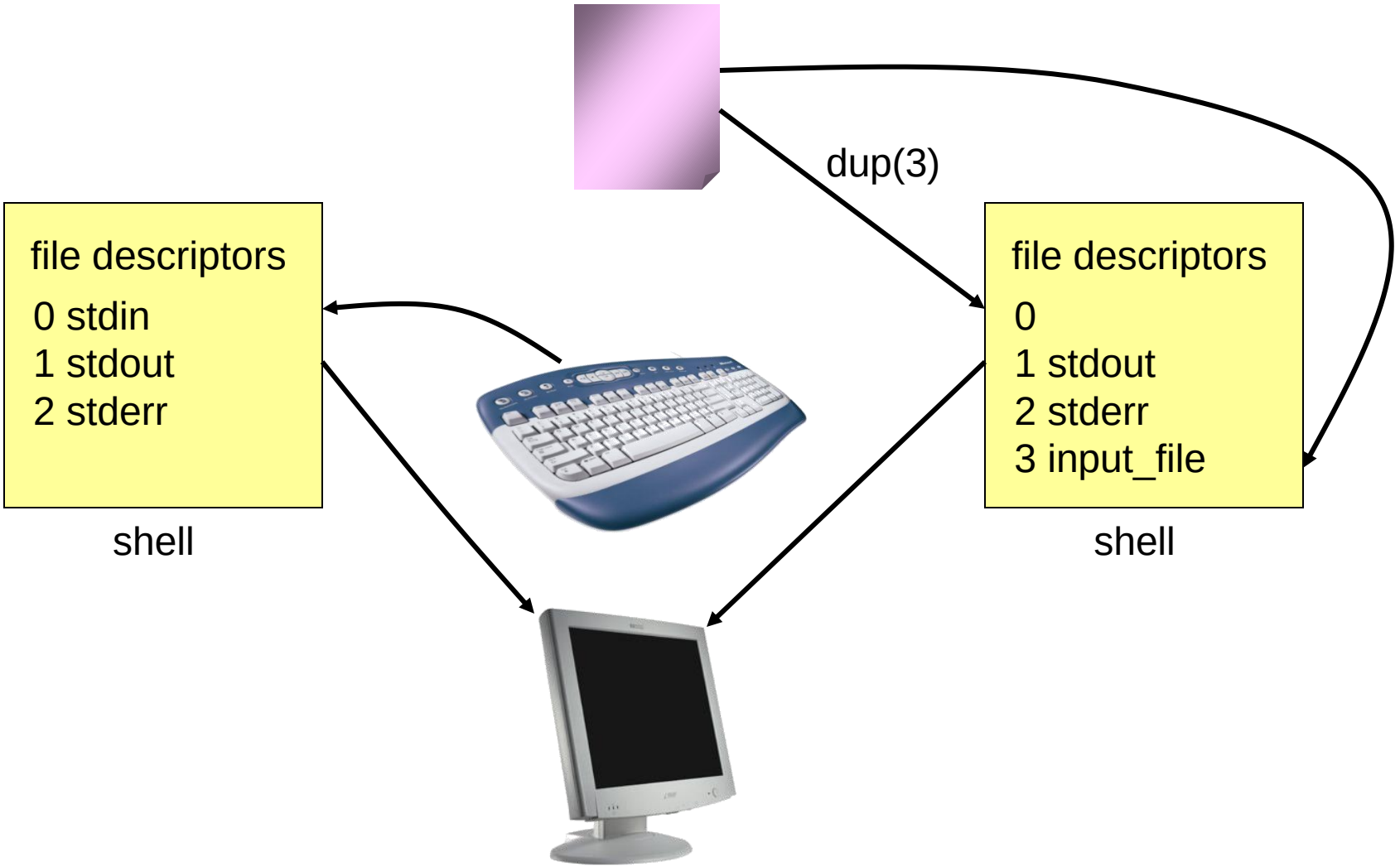
Input Redirection



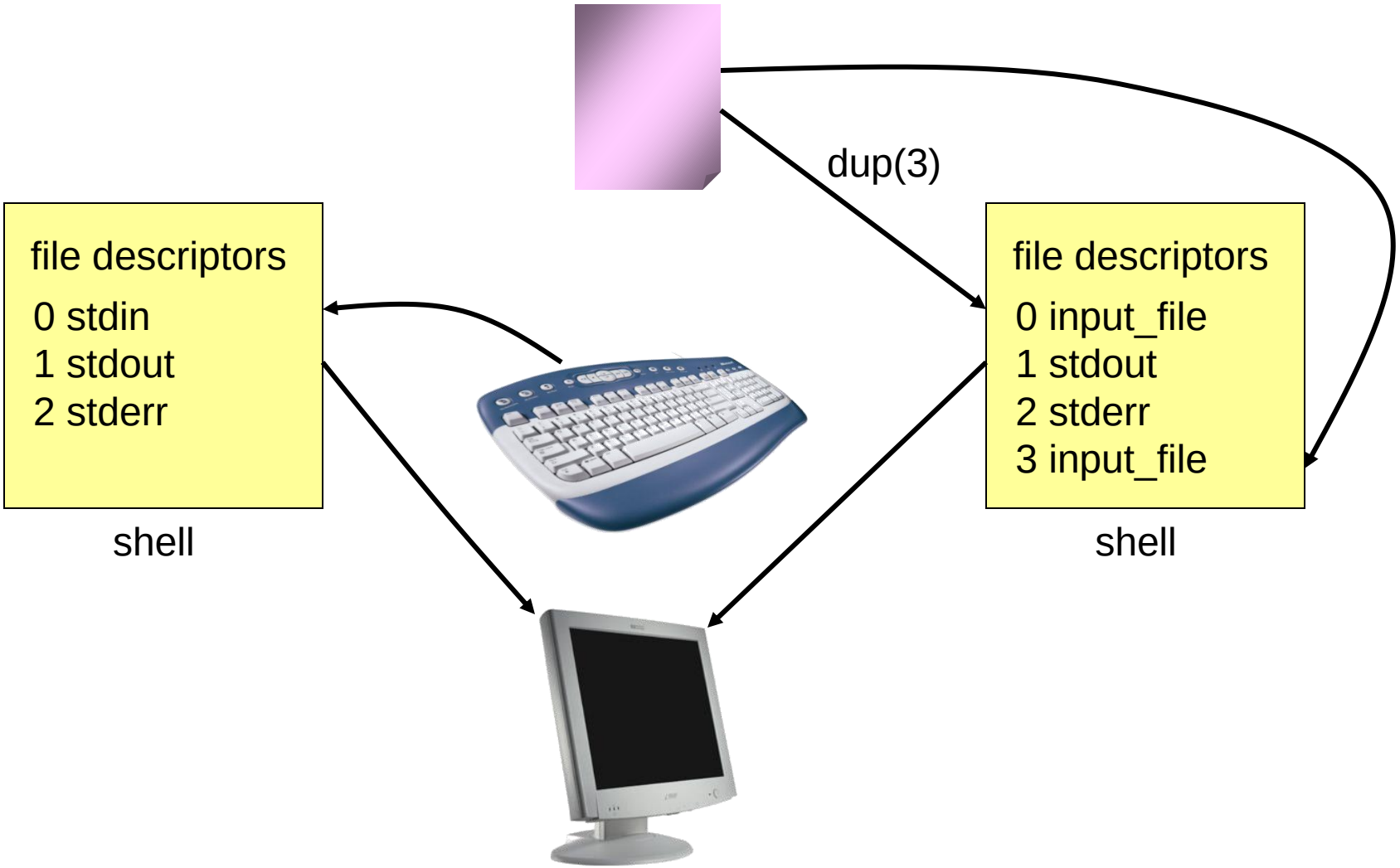
Input Redirection



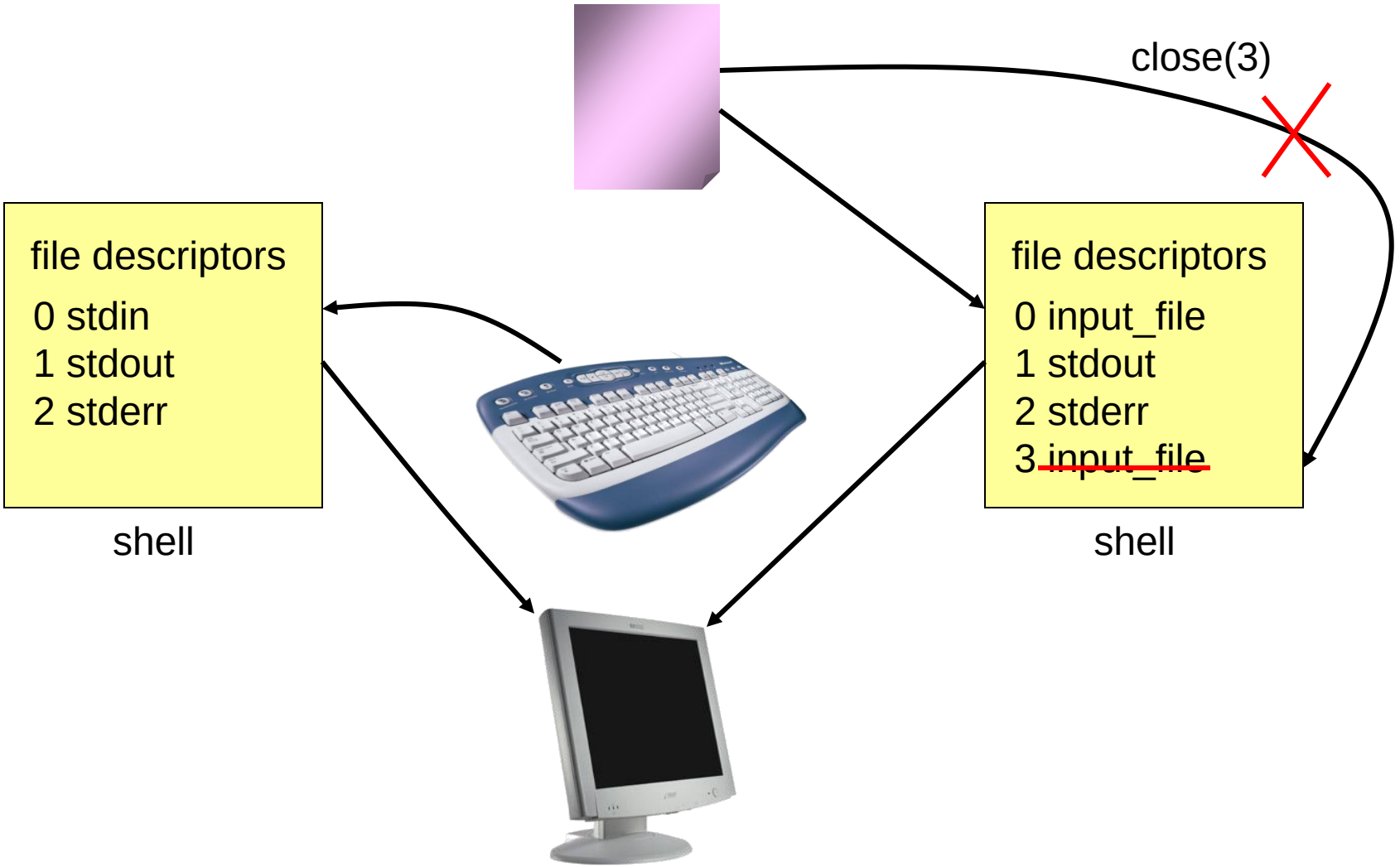
Input Redirection



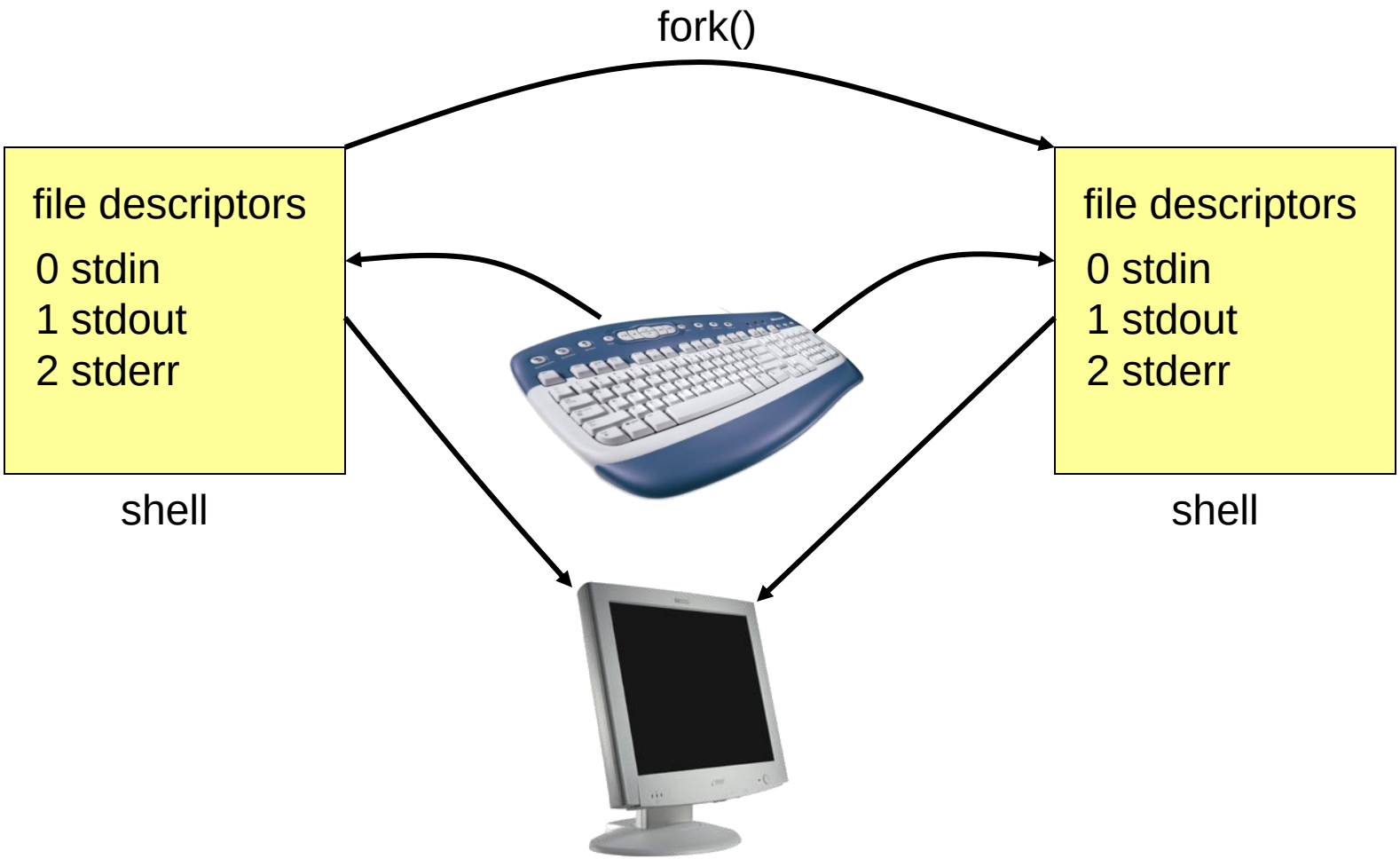
Input Redirection



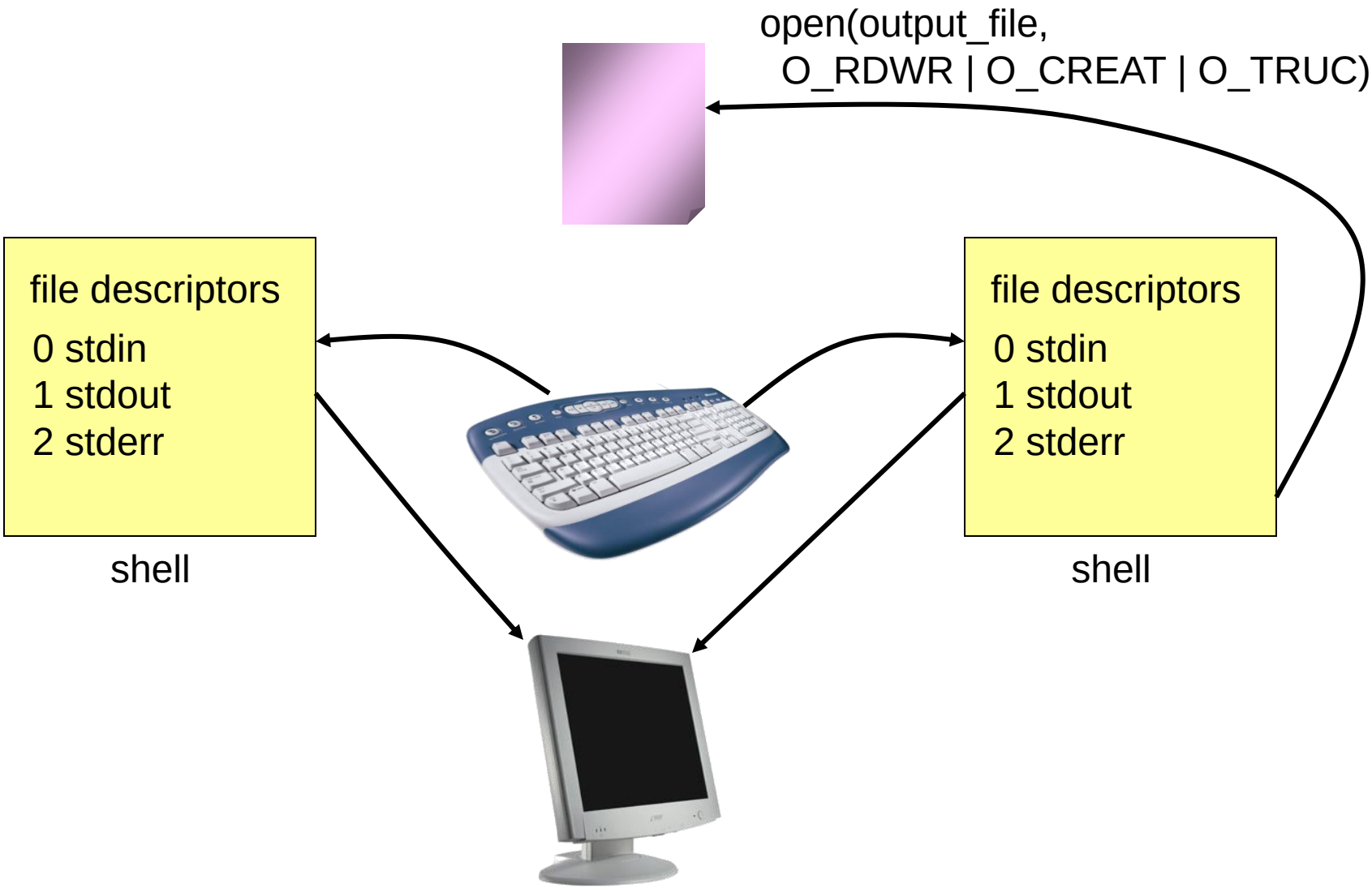
Input Redirection



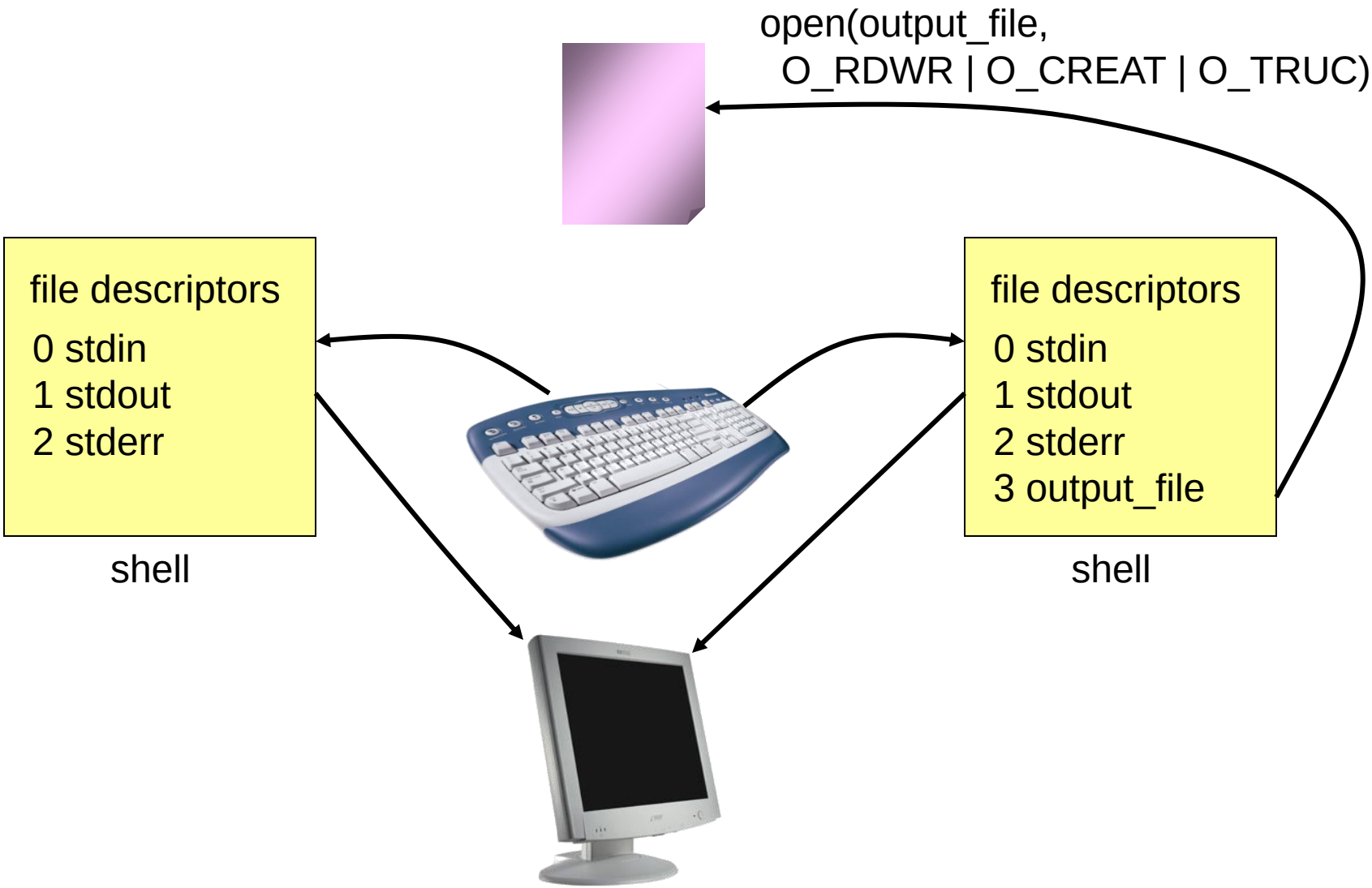
Output Redirection



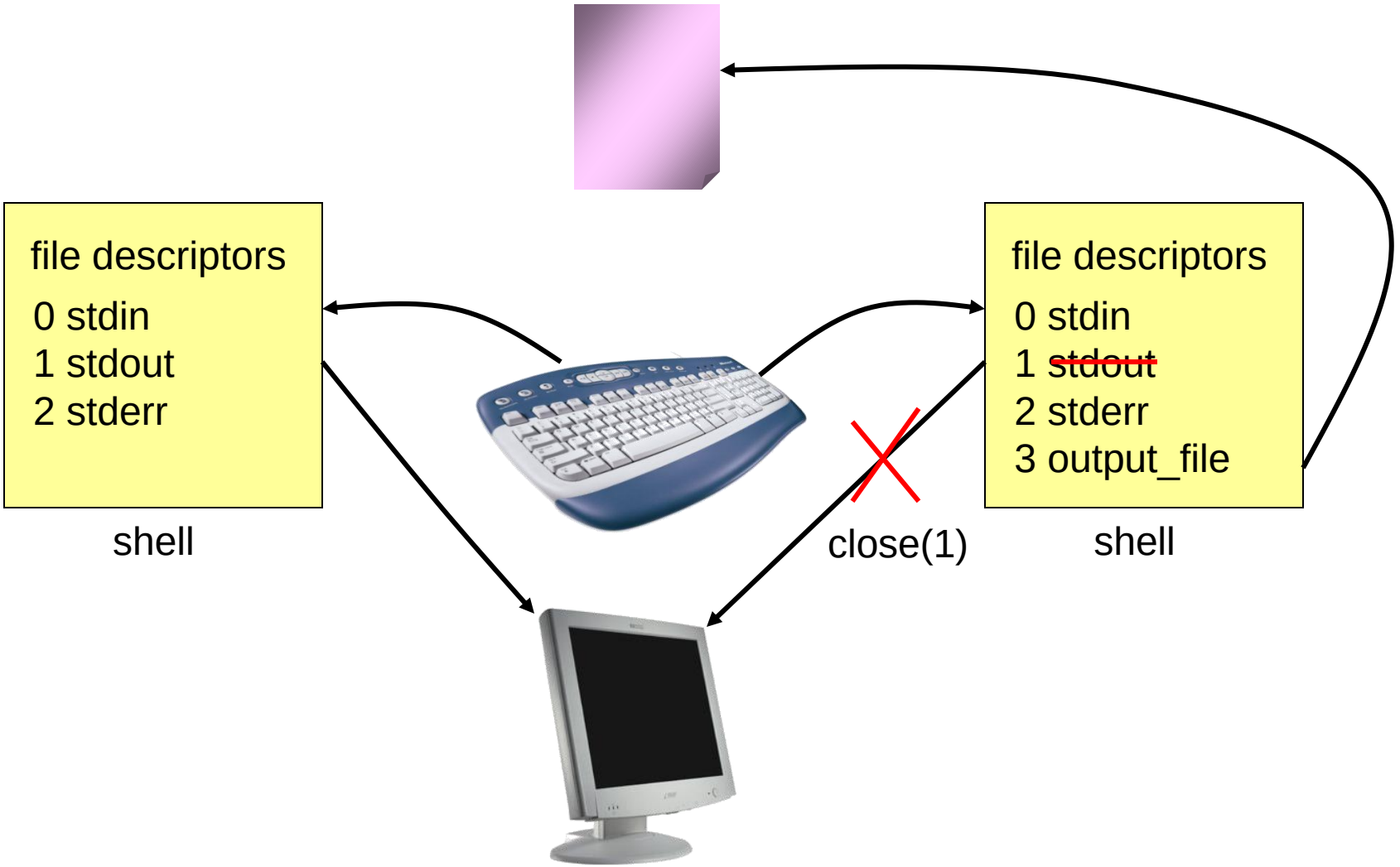
Output Redirection



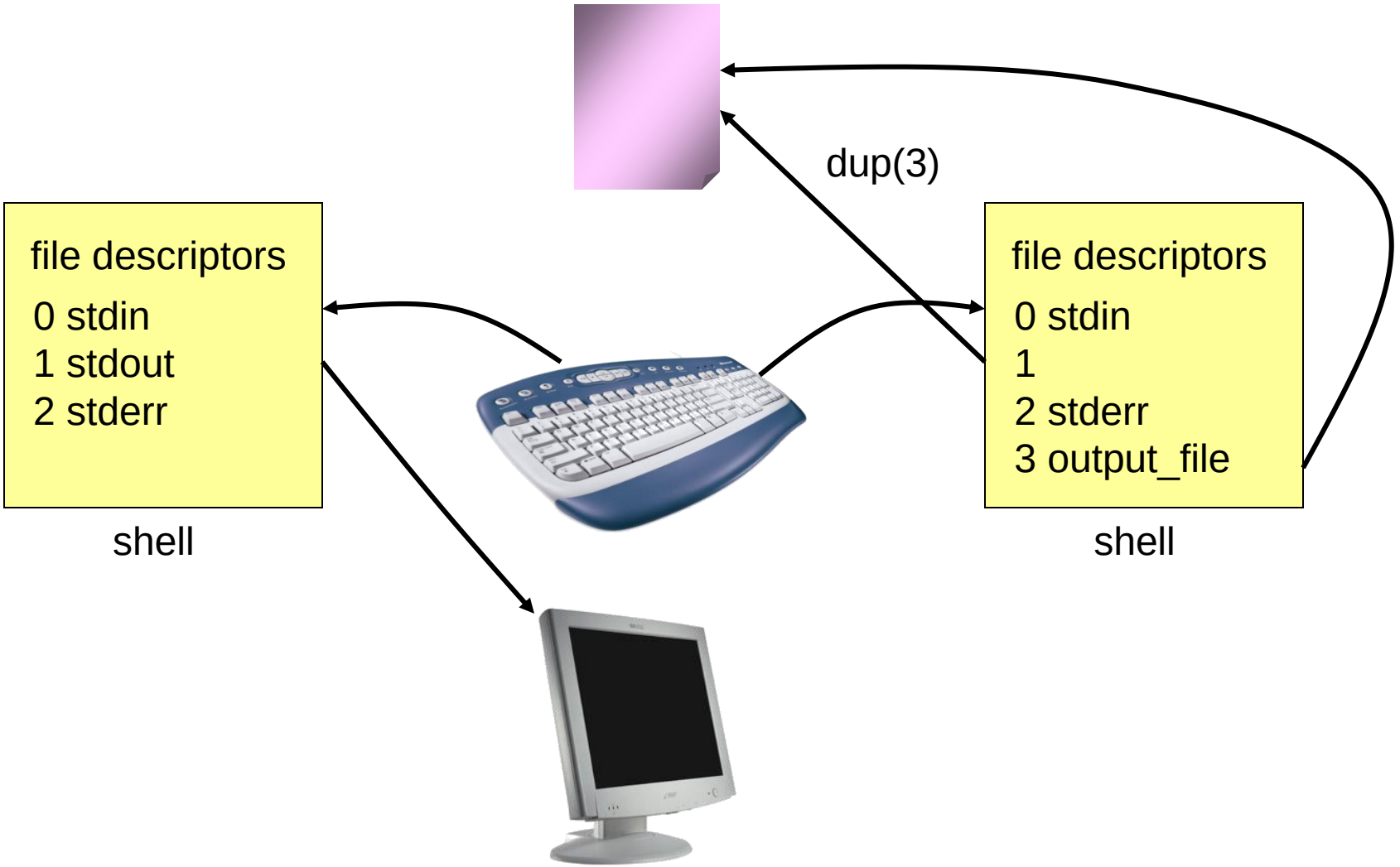
Output Redirection



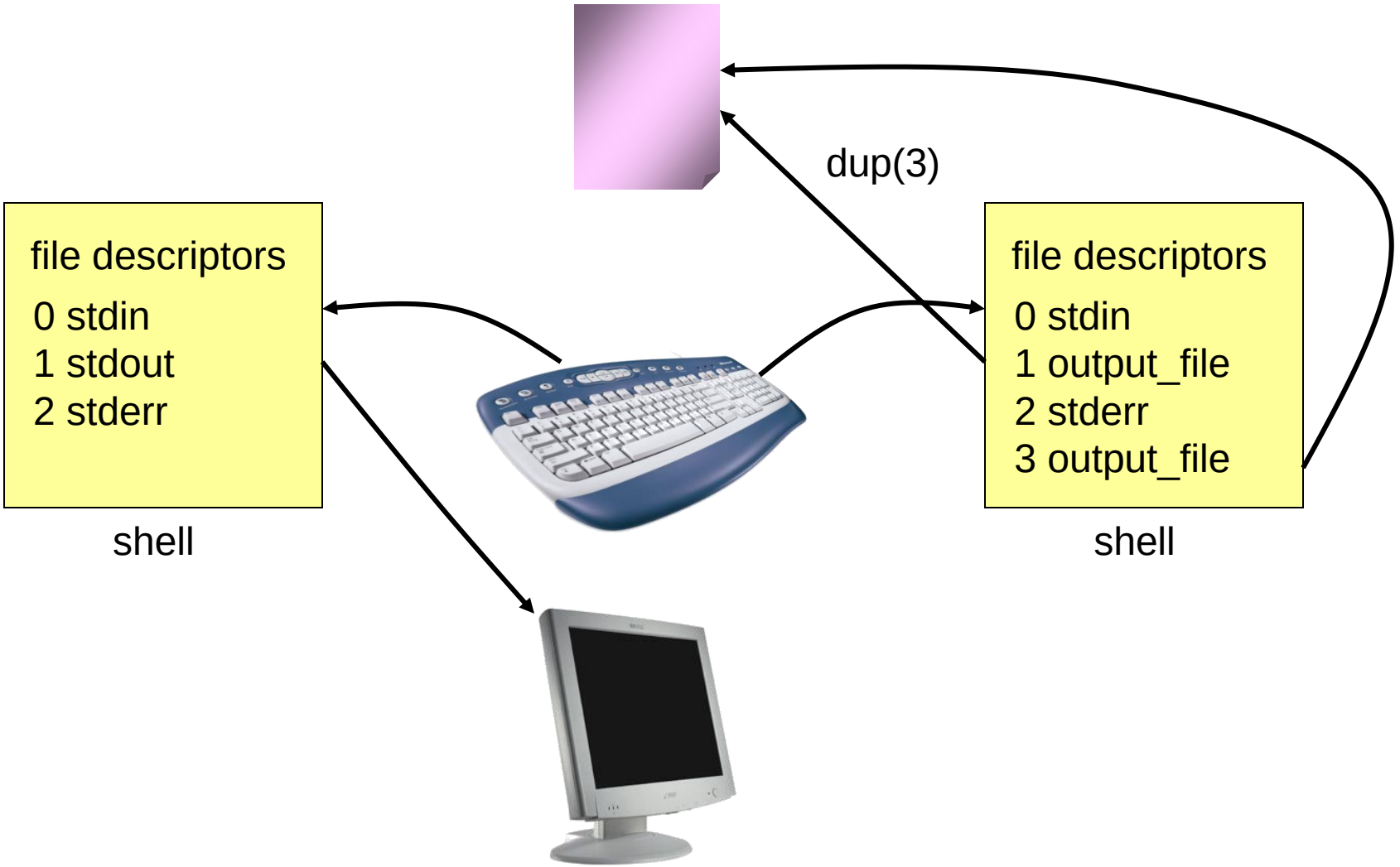
Output Redirection



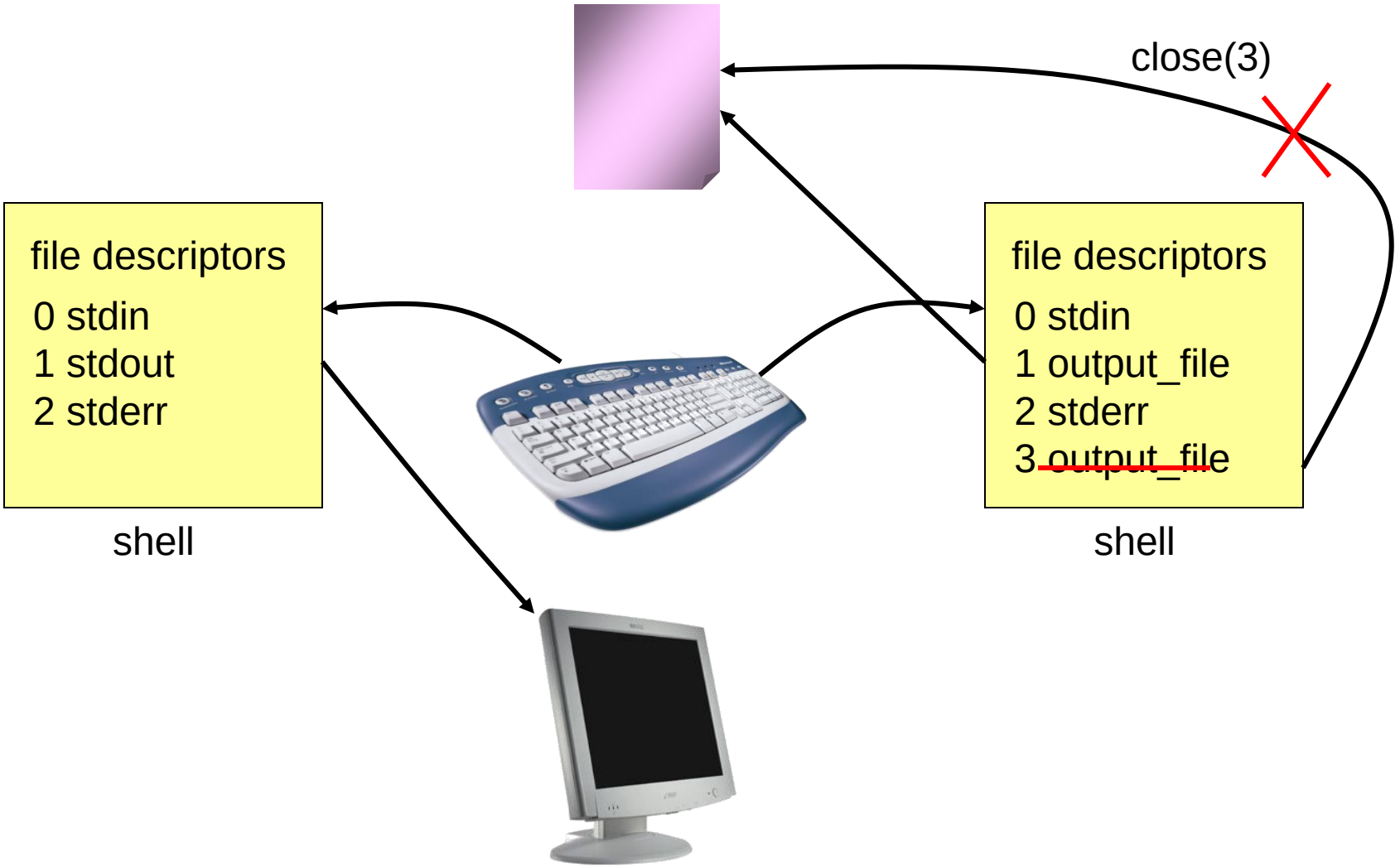
Output Redirection



Output Redirection



Output Redirection



Pipe

```
int p1_to_p2[2];
pipe(p1_to_p2);
```

file descriptors
0 stdin
1 stdout
2 stderr

shell

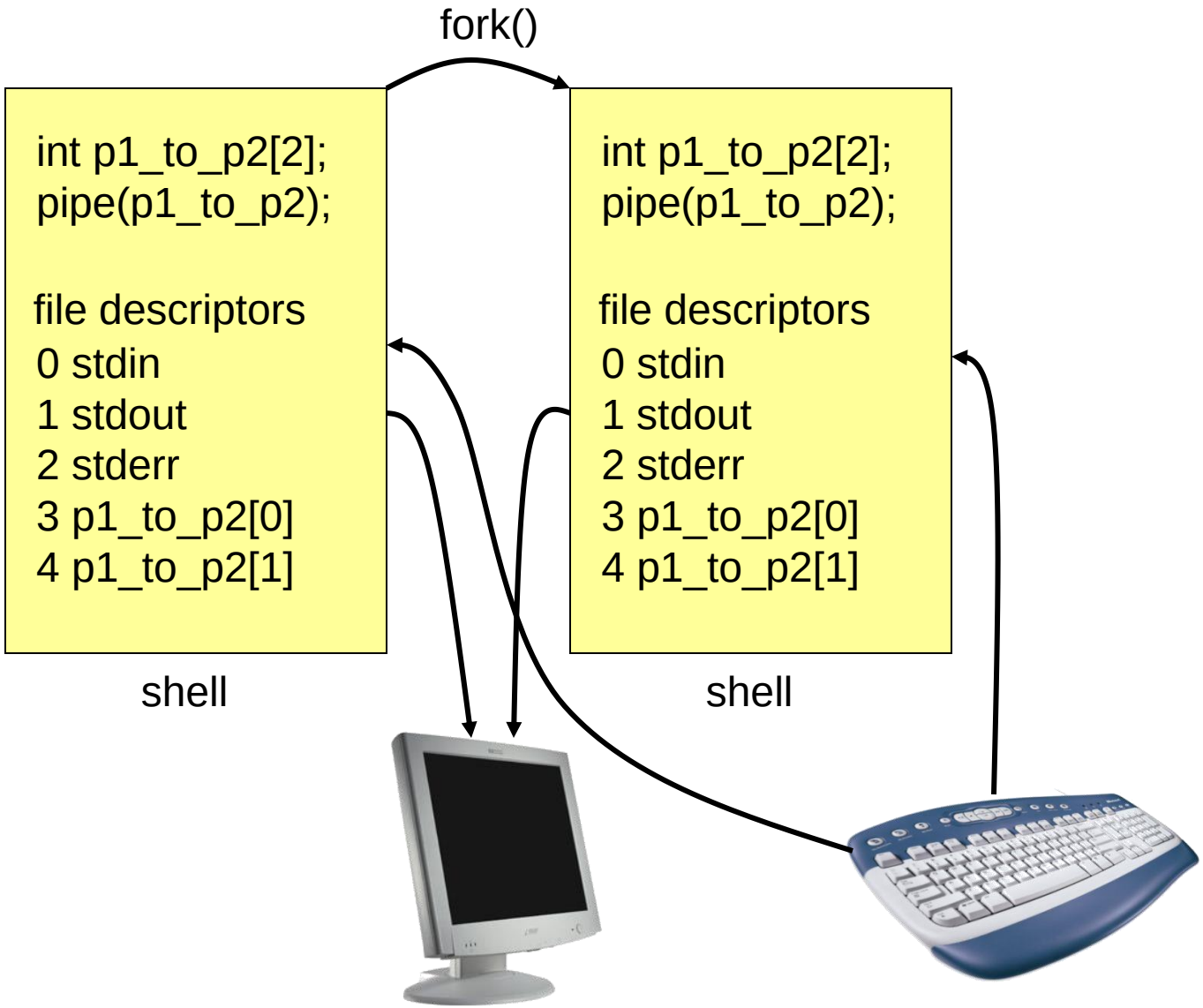
Pipe

```
int p1_to_p2[2];
pipe(p1_to_p2);
```

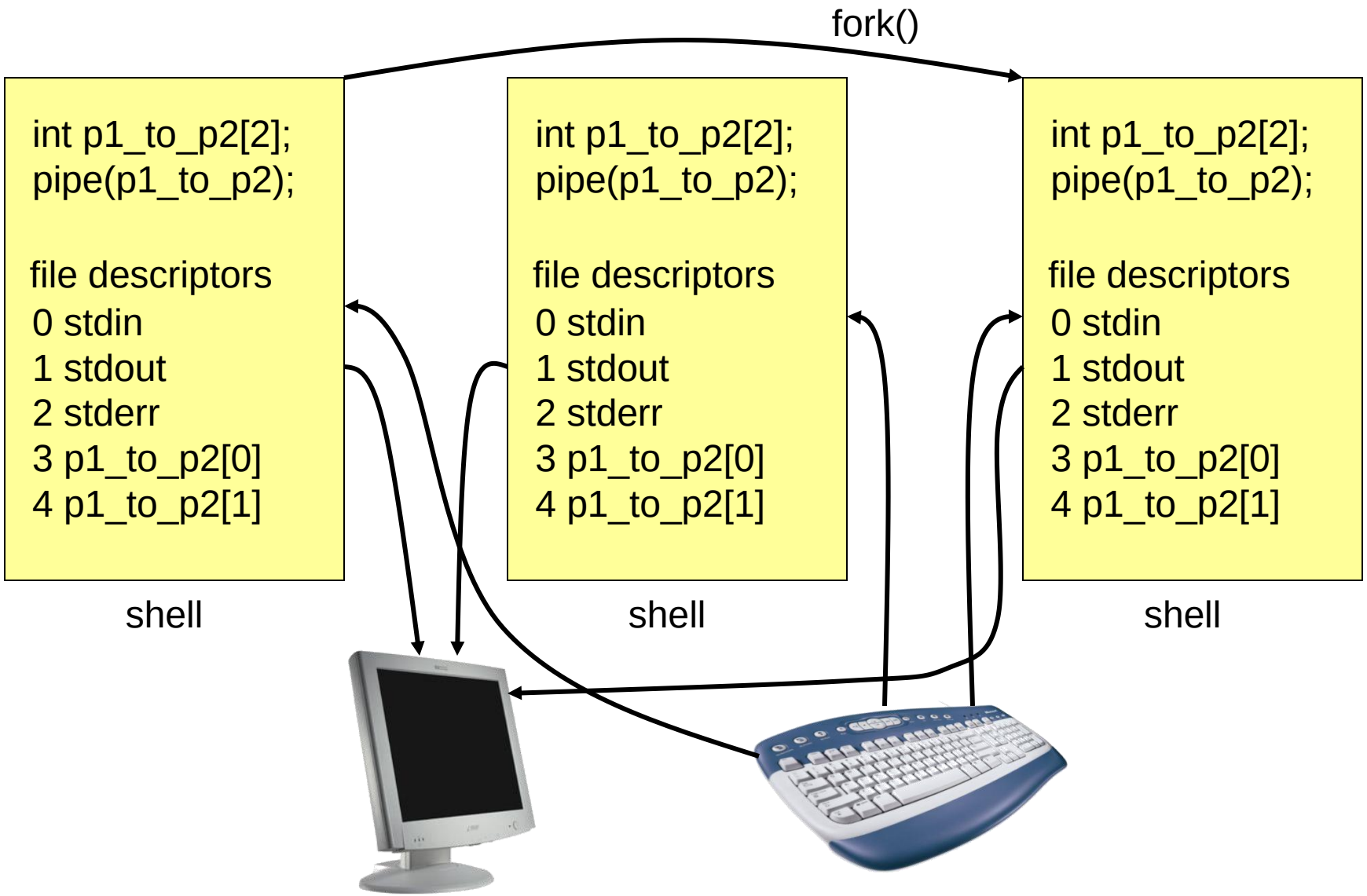
file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]

shell

Pipe


fork()
int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]
shell
int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]
shell

Pipe


fork()
int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]
shell
int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]
shell
int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]
shell

Pipe

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]

shell

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
~~1 stdout~~
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]

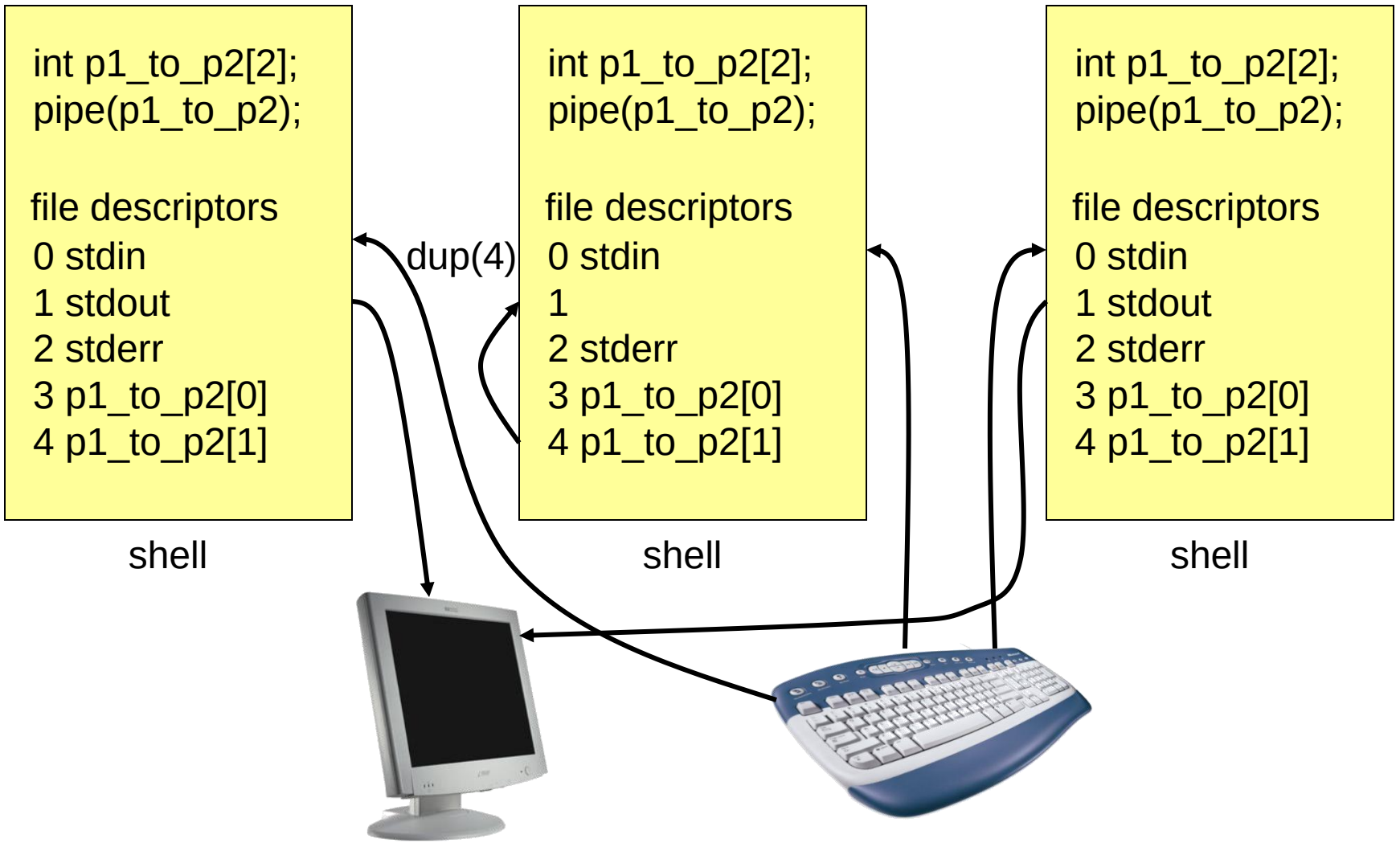
shell

int p1_to_p2[2];
pipe(p1_to_p2);

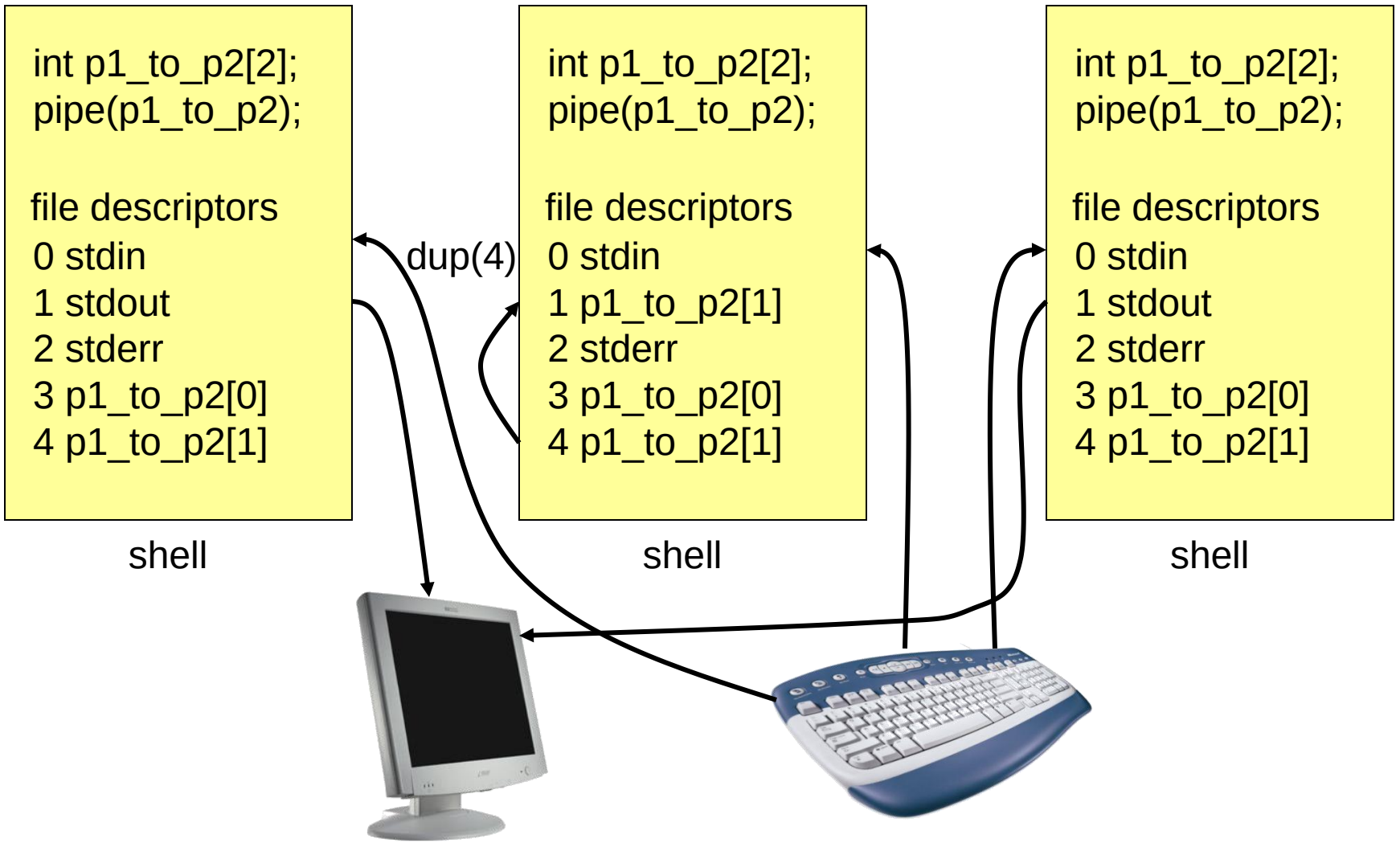
file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]

shell

Pipe


int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]
shell
dup(4)
int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0 stdin
1
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]
shell
int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]
shell

Pipe


int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]
shell
dup(4)
int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0 stdin
1 p1_to_p2[1]
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]
shell
int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]
shell

Pipe

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]

shell

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 p1_to_p2[1]
2 stderr
3 ~~p1_to_p2[0]~~
4 ~~p1_to_p2[1]~~

shell

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]

shell

Pipe

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]

shell

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 p1_to_p2[1]
2 stderr

shell

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]

shell

Pipe

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]

shell

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 p1_to_p2[1]
2 stderr

execvp(bin_path,
argv)

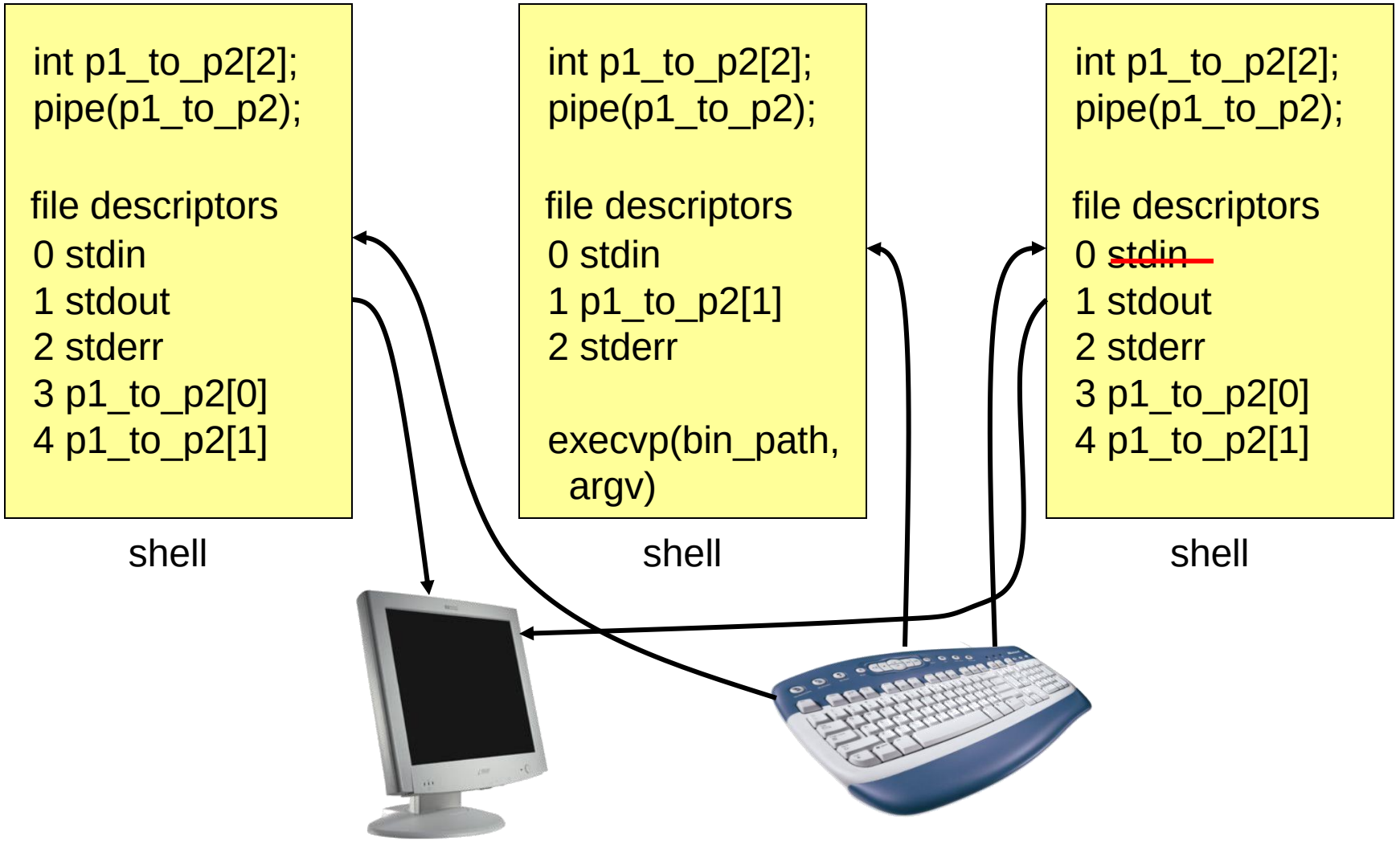
shell

int p1_to_p2[2];
pipe(p1_to_p2);

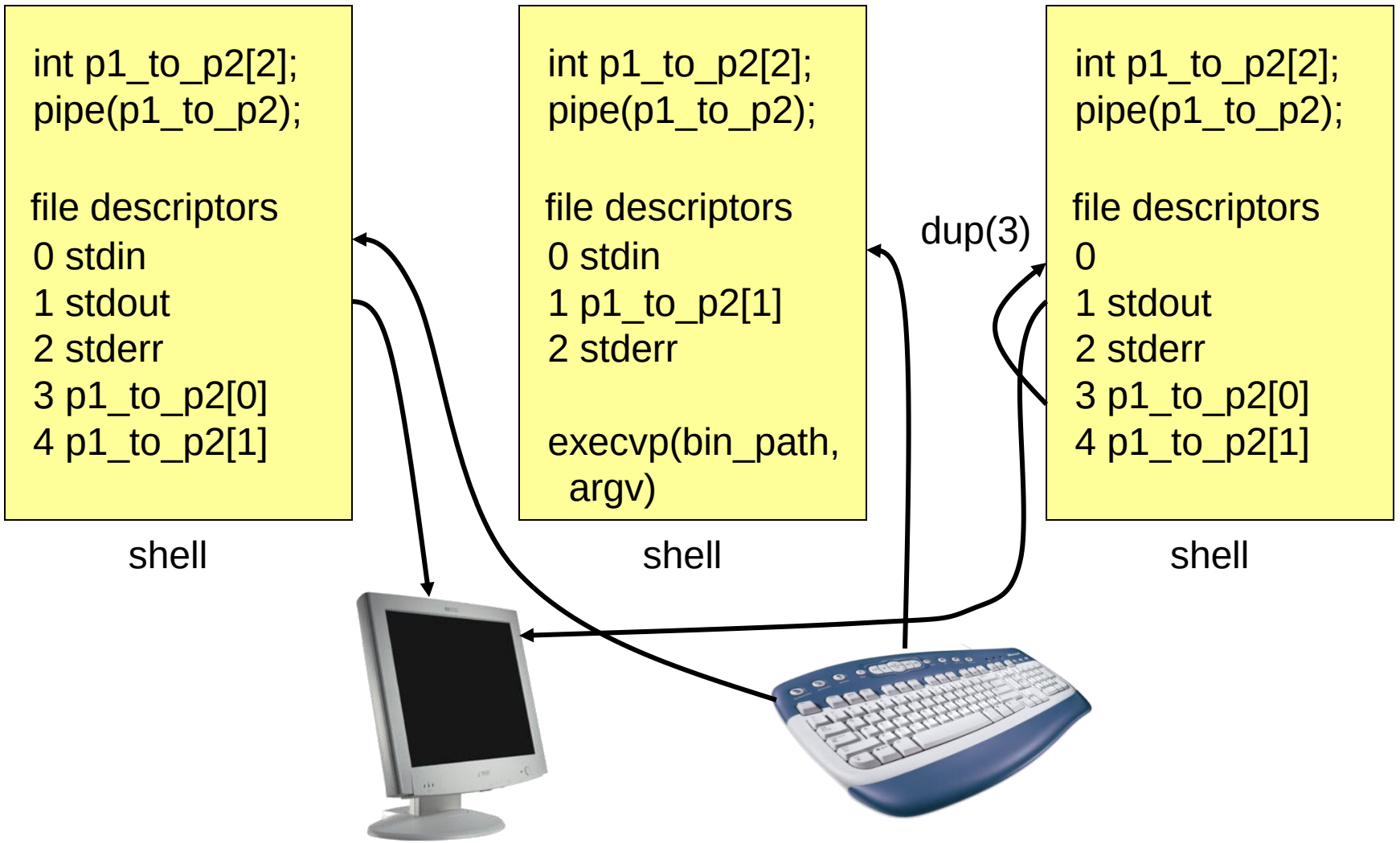
file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]

shell

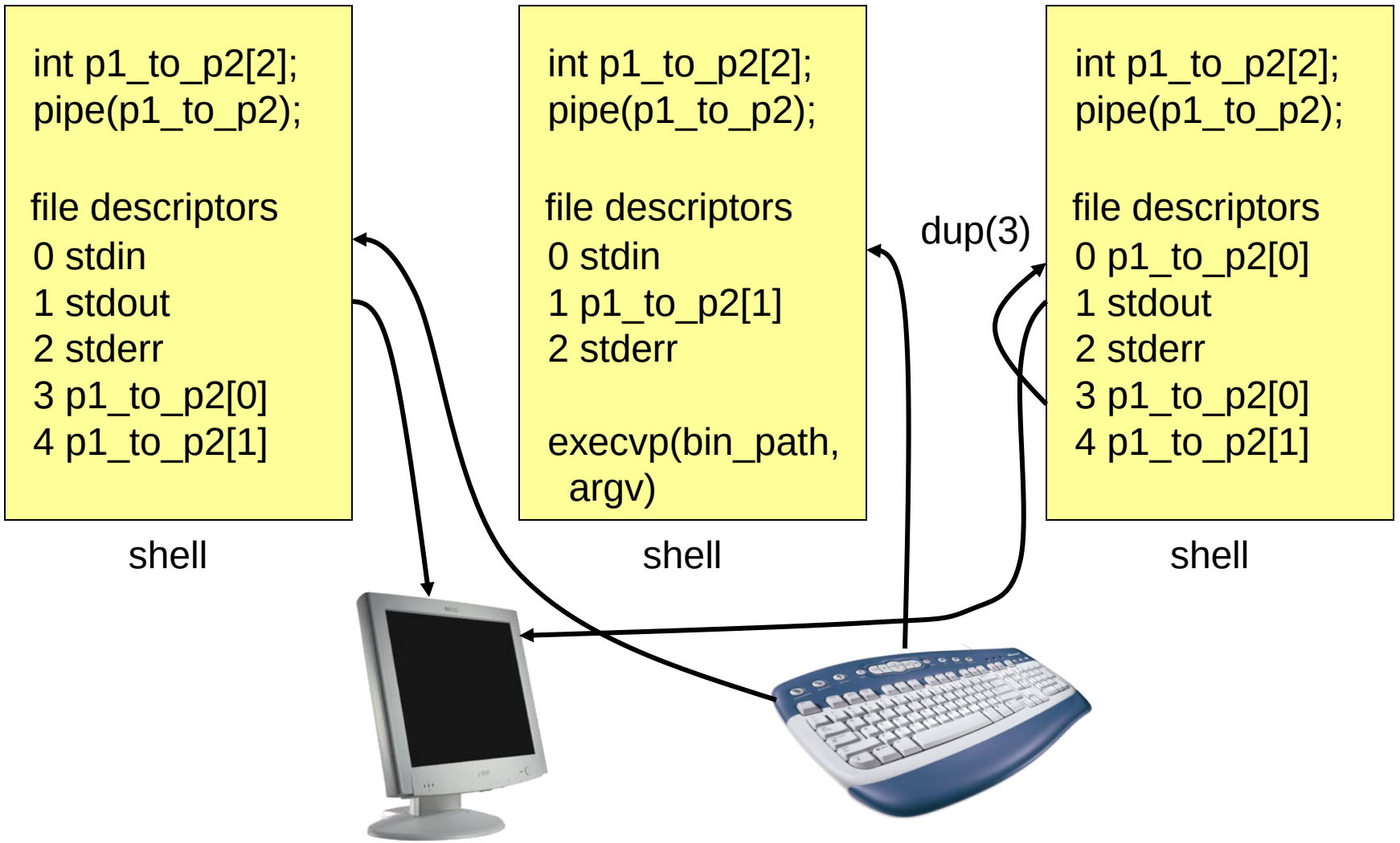
Pipe


int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]
shell
int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0 stdin
1 p1_to_p2[1]
2 stderr
execvp(bin_path,
argv)
shell
int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]
shell

Pipe


int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]
shell
int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0 stdin
1 p1_to_p2[1]
2 stderr
execvp(bin_path,
argv)
shell
dup(3)
int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]
shell

Pipe


int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]
shell
int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0 stdin
1 p1_to_p2[1]
2 stderr
execvp(bin_path,
argv)
shell
dup(3)
int p1_to_p2[2];
pipe(p1_to_p2);
file descriptors
0 p1_to_p2[0]
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]
shell

Pipe

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]

shell

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 p1_to_p2[1]
2 stderr

execvp(bin_path,
argv)

shell

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 p1_to_p2[0]
1 stdout
2 stderr
3 ~~p1_to_p2[0]~~
4 ~~p1_to_p2[1]~~

shell

Pipe

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]

shell

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 p1_to_p2[1]
2 stderr

execvp(bin_path,
argv)

shell

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 p1_to_p2[0]
1 stdout
2 stderr

shell

Pipe

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]

shell

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 p1_to_p2[1]
2 stderr

execvp(bin_path,
argv)

shell

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 p1_to_p2[0]
1 stdout
2 stderr

execvp(bin_path,
argv)

shell

Pipe

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 stdout
2 stderr
3 ~~p1_to_p2[0]~~
4 ~~p1_to_p2[1]~~

shell

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 p1_to_p2[1]
2 stderr

execvp(bin_path,
argv)

shell

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 p1_to_p2[0]
1 stdout
2 stderr

execvp(bin_path,
argv)

shell

Pipe

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 stdout
2 stderr

shell

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 p1_to_p2[1]
2 stderr

execvp(bin_path,
argv)

shell

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 p1_to_p2[0]
1 stdout
2 stderr

execvp(bin_path,
argv)

shell

Pipe

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 stdout
2 stderr

waitpid(…)
waitpid(…)

shell

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 p1_to_p2[1]
2 stderr

execvp(bin_path,
argv)

shell

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 p1_to_p2[0]
1 stdout
2 stderr

execvp(bin_path,
argv)

shell

Takeaways

- Operating System Objects and APIs Used by a Fully Functional Shell
  - Session, Process Group, Controlling Terminal
  - File Descriptors: open, close, pipe, dup, read, write
  - State Machine Management: fork, execve, exit, wait, signal, kill, setpgid, getpgid, …
- As you dive deeper into the project, your sense of mystery around the operating system will gradually fade …