Project #1:
Week 3

Principles of Operating Systems (LAB)

COP 4610/CGS 5765

Overview

- Pipelining
- Processes
  - fork()
  - execv()
- Environment Variables

Clarification in Description

- **`cd "$OLDPWD" && pwd`**
  - Do not have to implement **‘`&&`’**
  - OLDPWD can be stored as an environment variable, but it is not necessary since there is not requirement to allow user to display them

pipe()

- Interprocess data channel

```
#include <unistd.h>
int pipe(int fildes[2]);
```

http://pubs.opengroup.org/onlinepubs/9699919799/functions/pipe.html

Pipe

- **`ls | grep`**
- fork() x2
- create a pipe
- redirect fd=1 of process to be ls to write end of pipe
- redirect fd = 0 of process to be grep to read end of pipe
- execv grep and ls binaries

Pipe

int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 stdin
1 stdout
2 stderr

shell

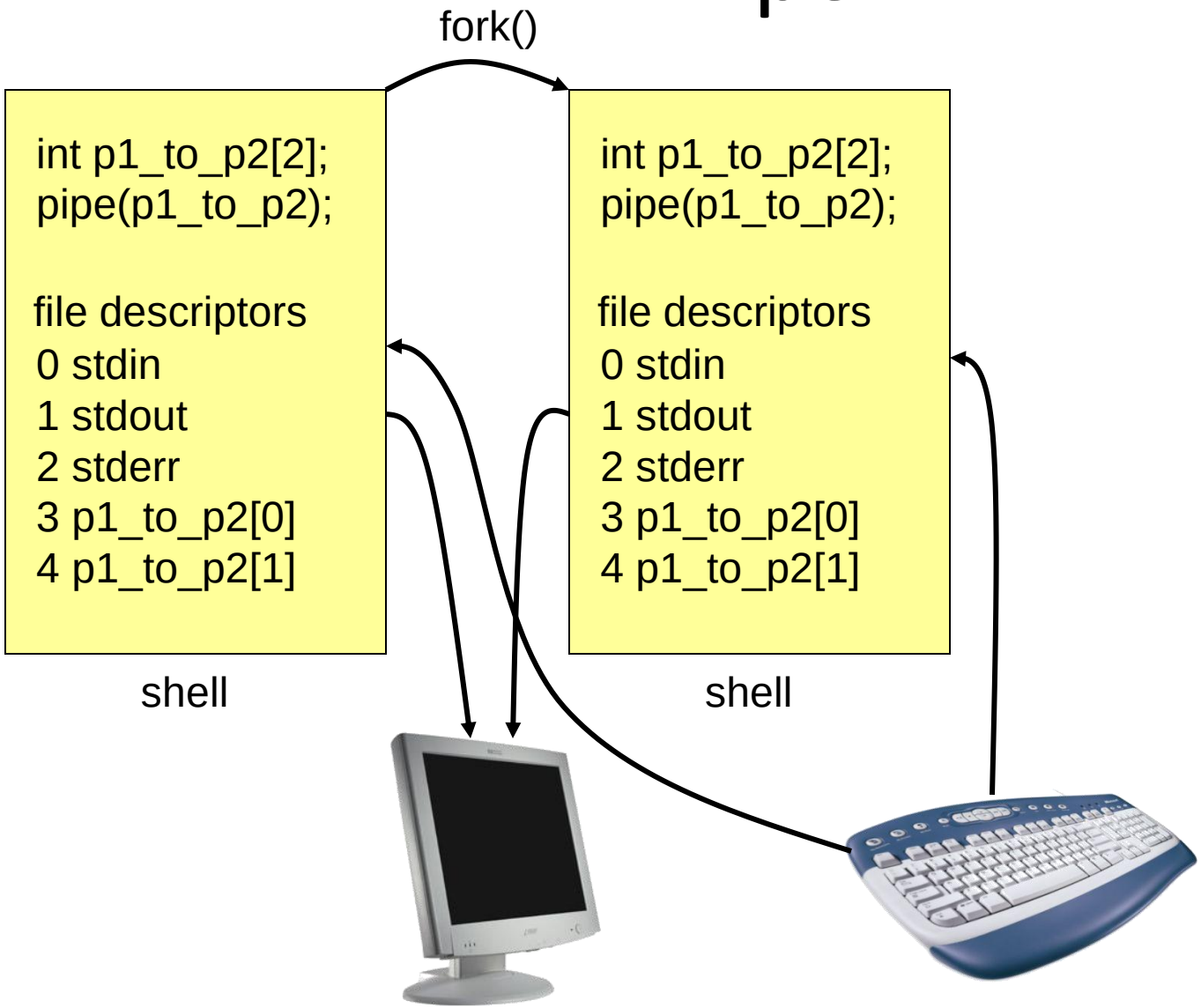
Pipe

int p1_to_p2[2];
pipe(p1_to_p2);

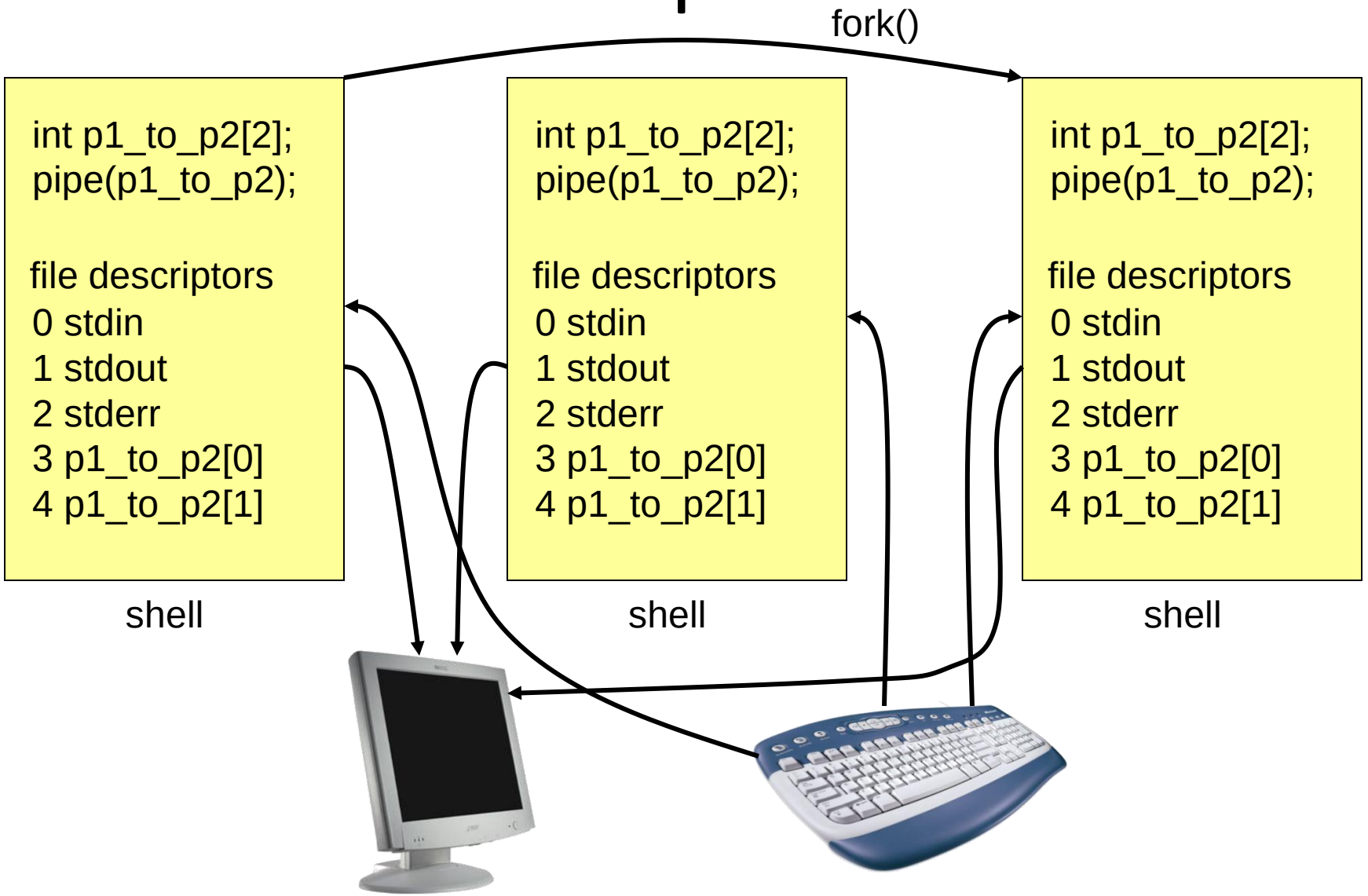
file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]

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]

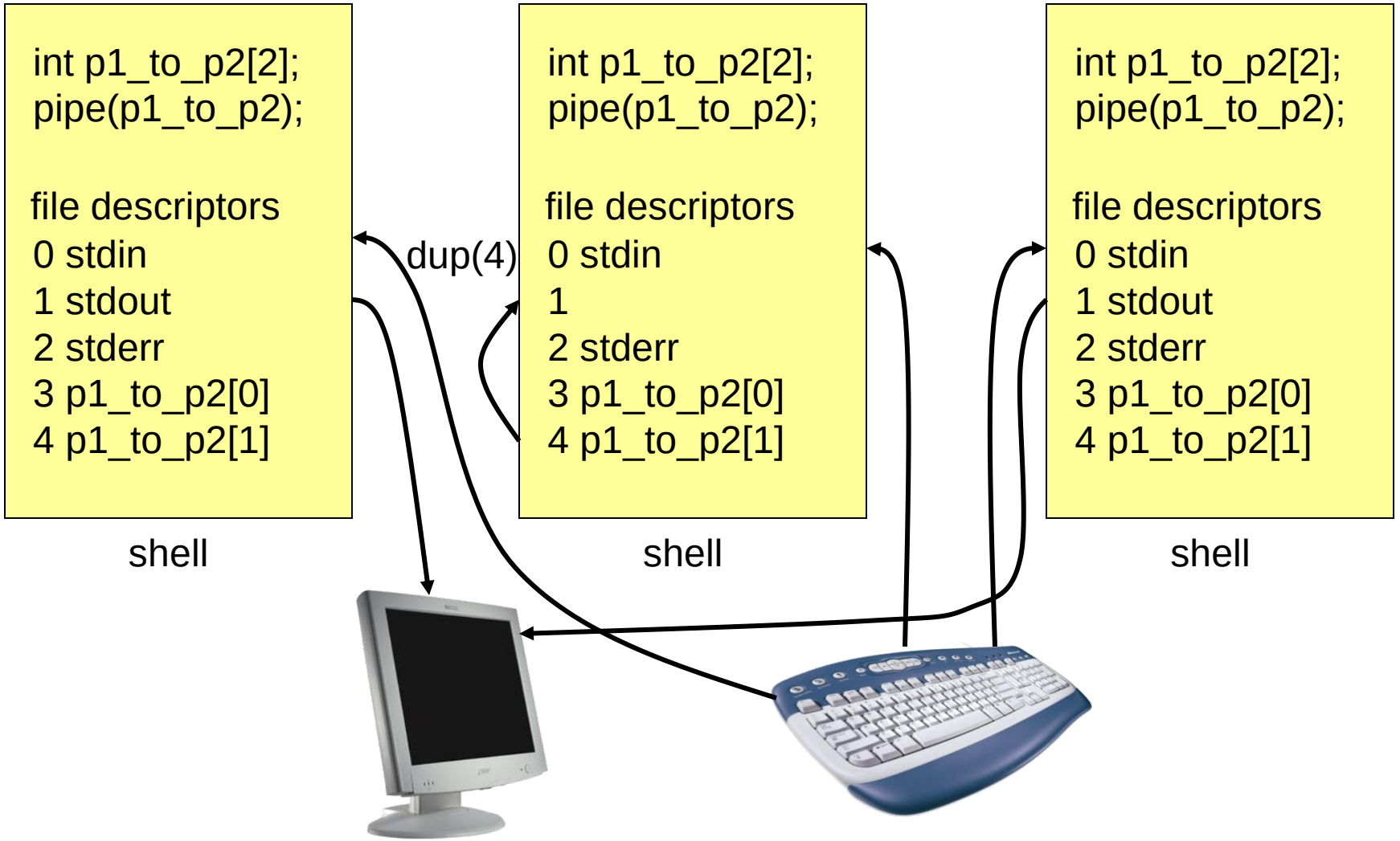
shell

int p1_to_p2[2];
pipe(p1_to_p2);

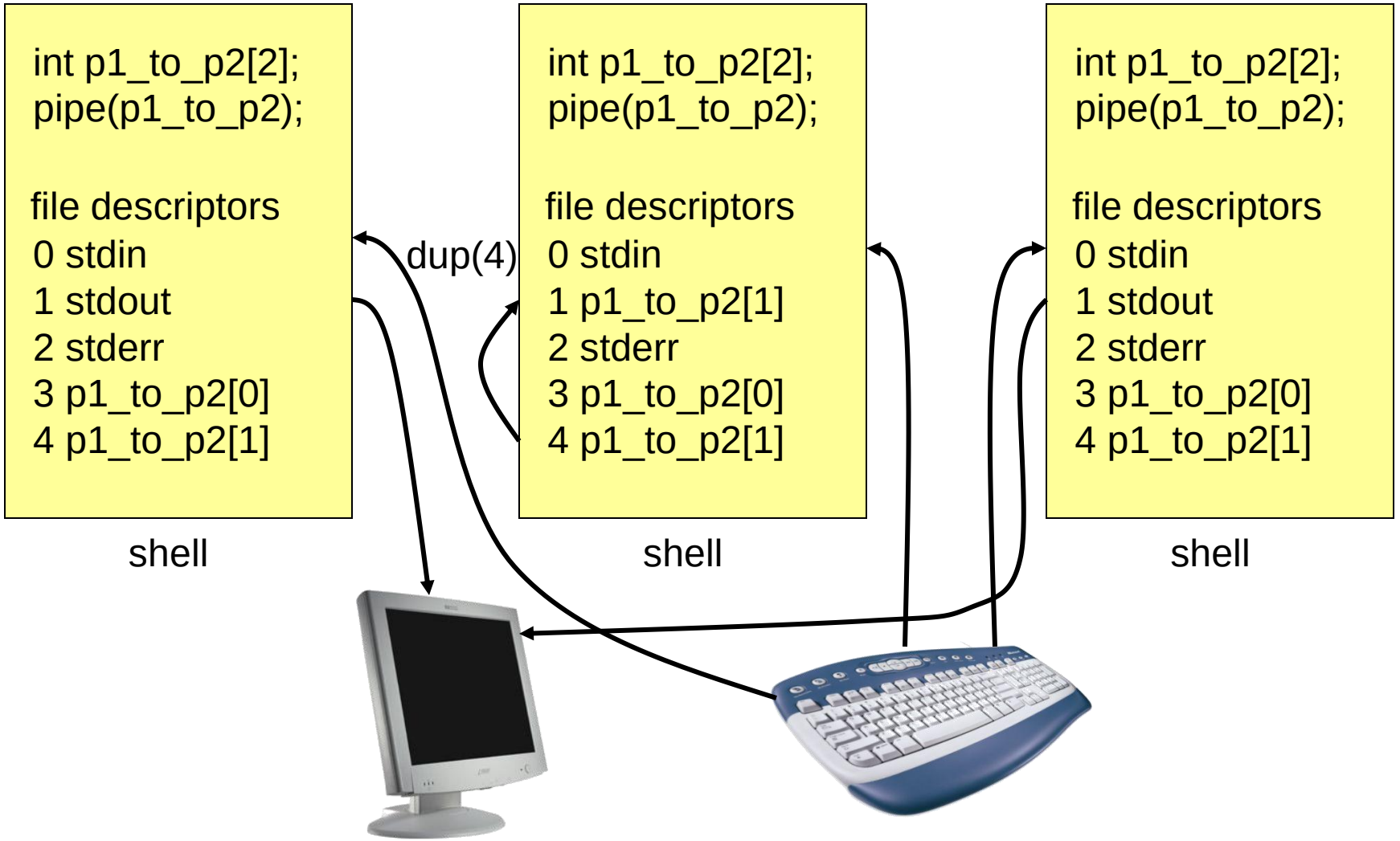
file descriptors
0 stdin
1 stdout
2 stderr
3 p1_to_p2[0]
4 p1_to_p2[1]

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)

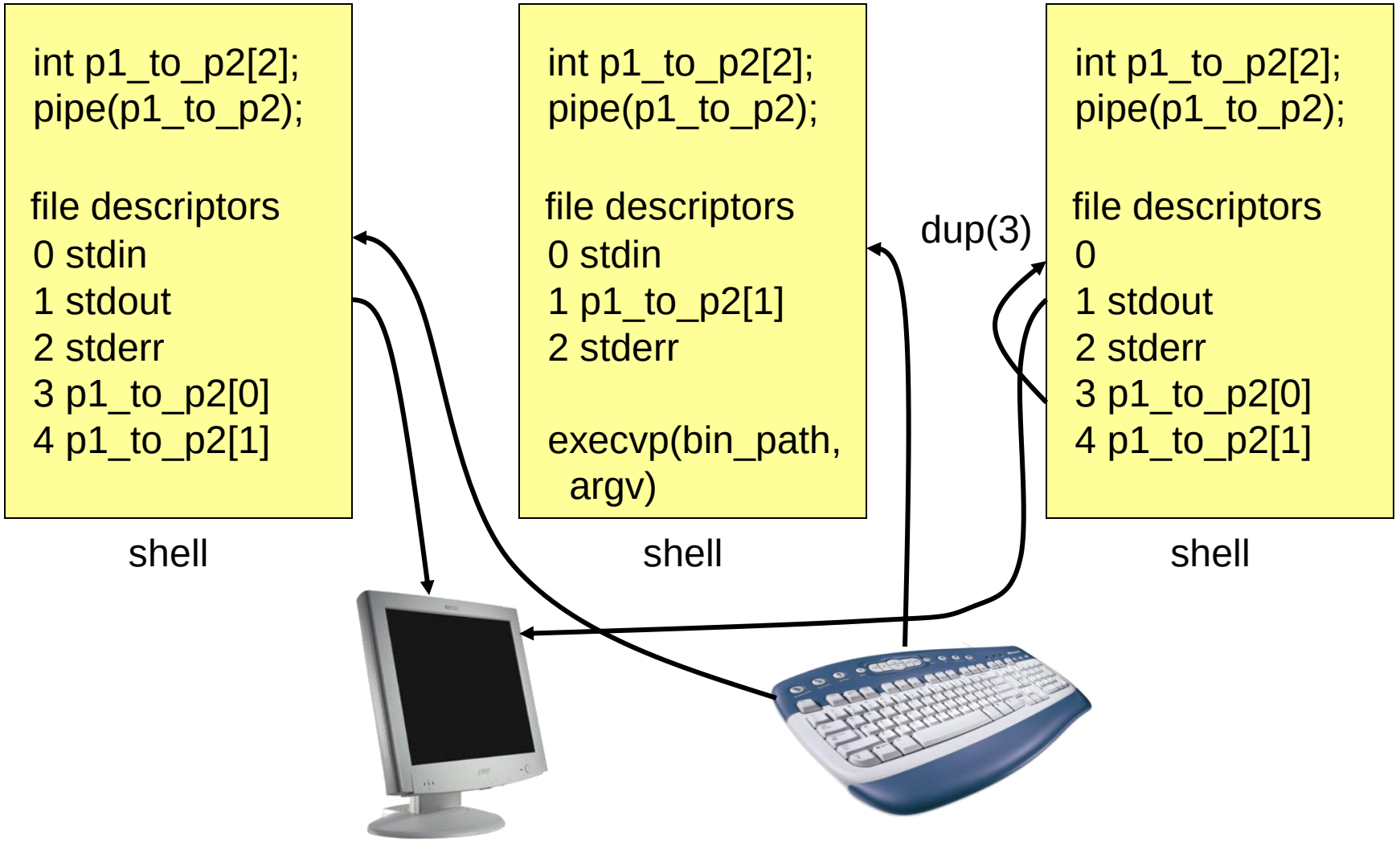
shell

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

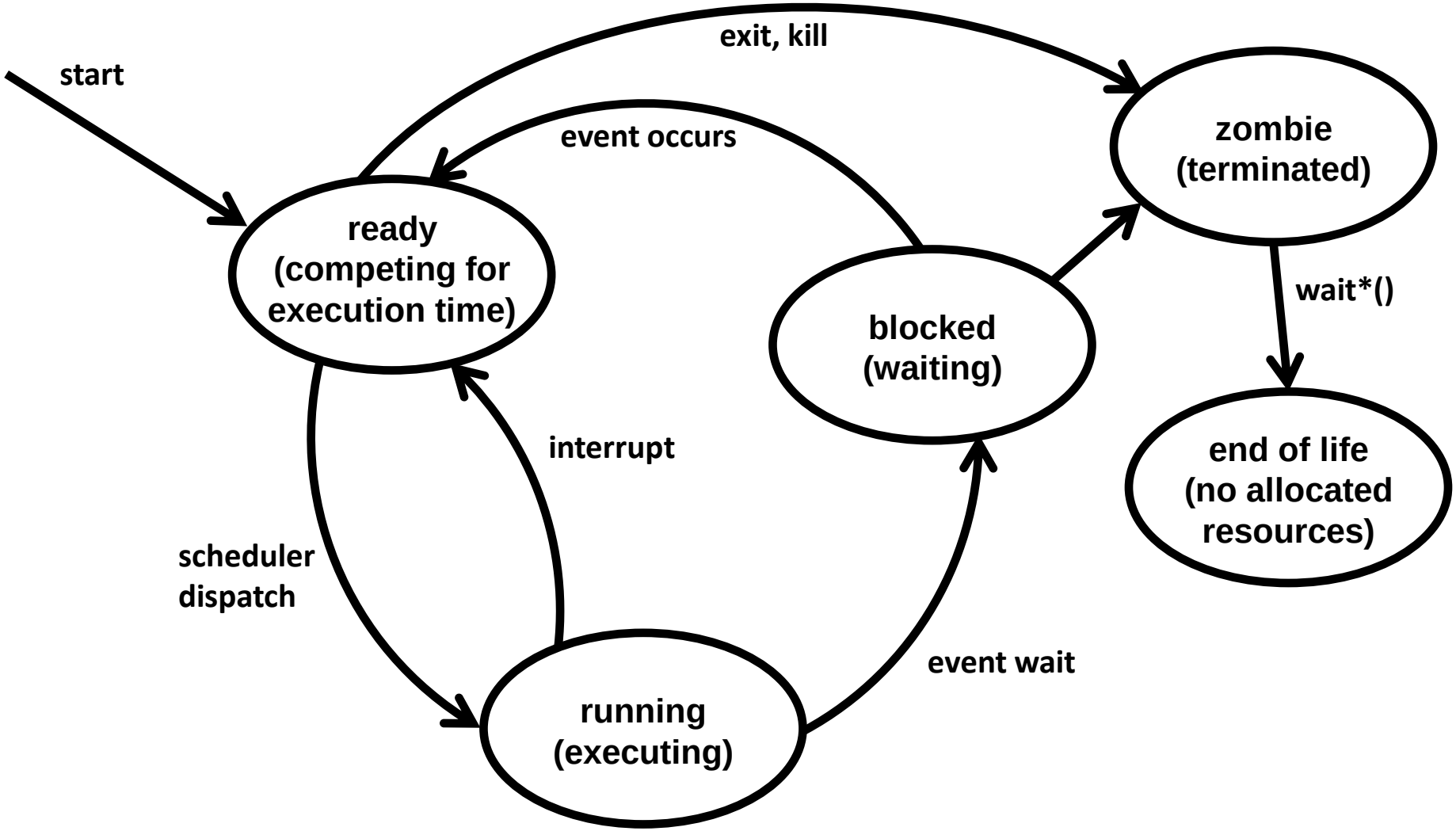
```
int p1_to_p2[2];
pipe(p1_to_p2);

file descriptors
0 p1_to_p2[0]
1 stdout
2 stderr

execvp(bin_path,
  argv)
```

shell

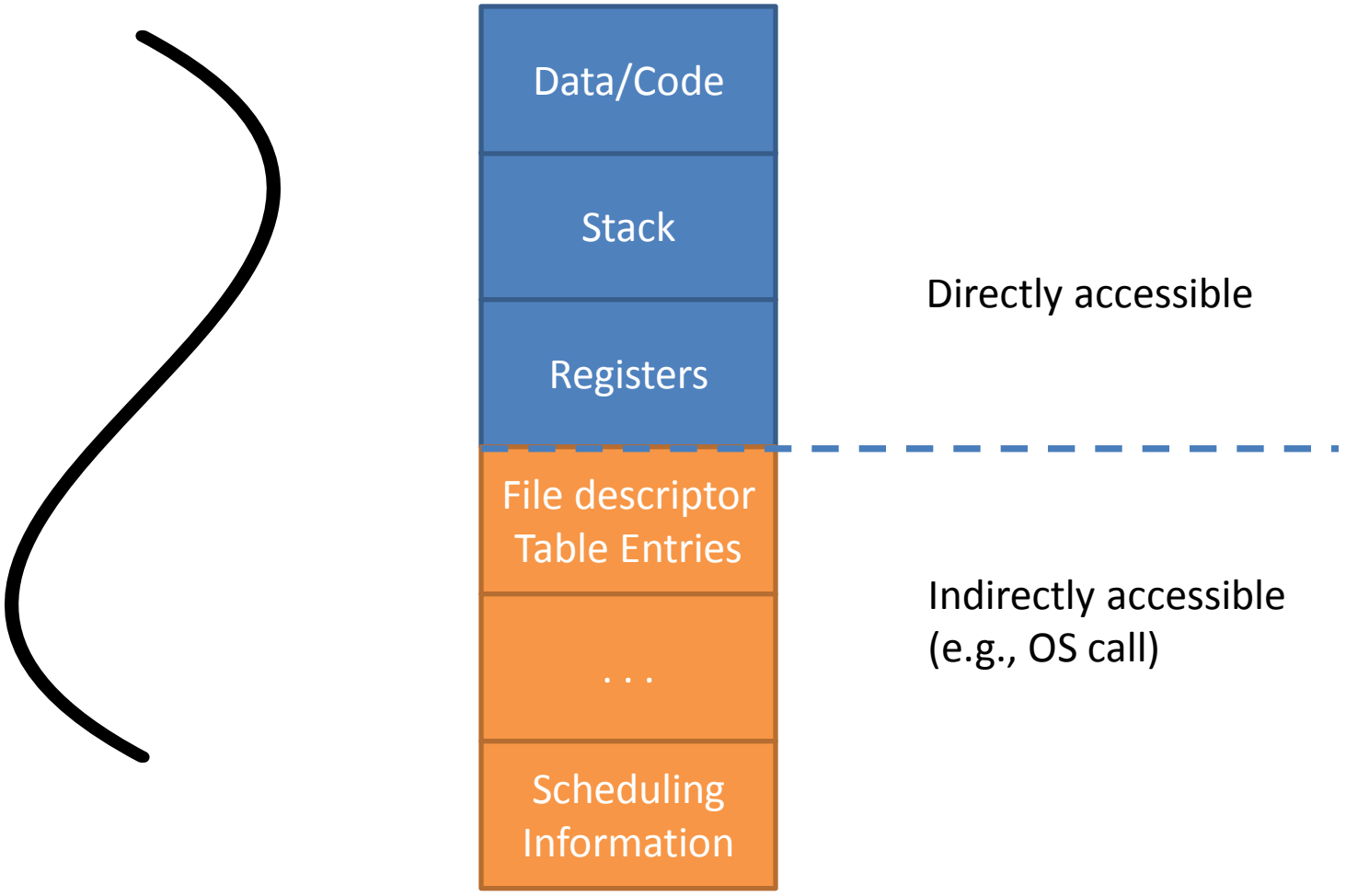
Process State Diagram



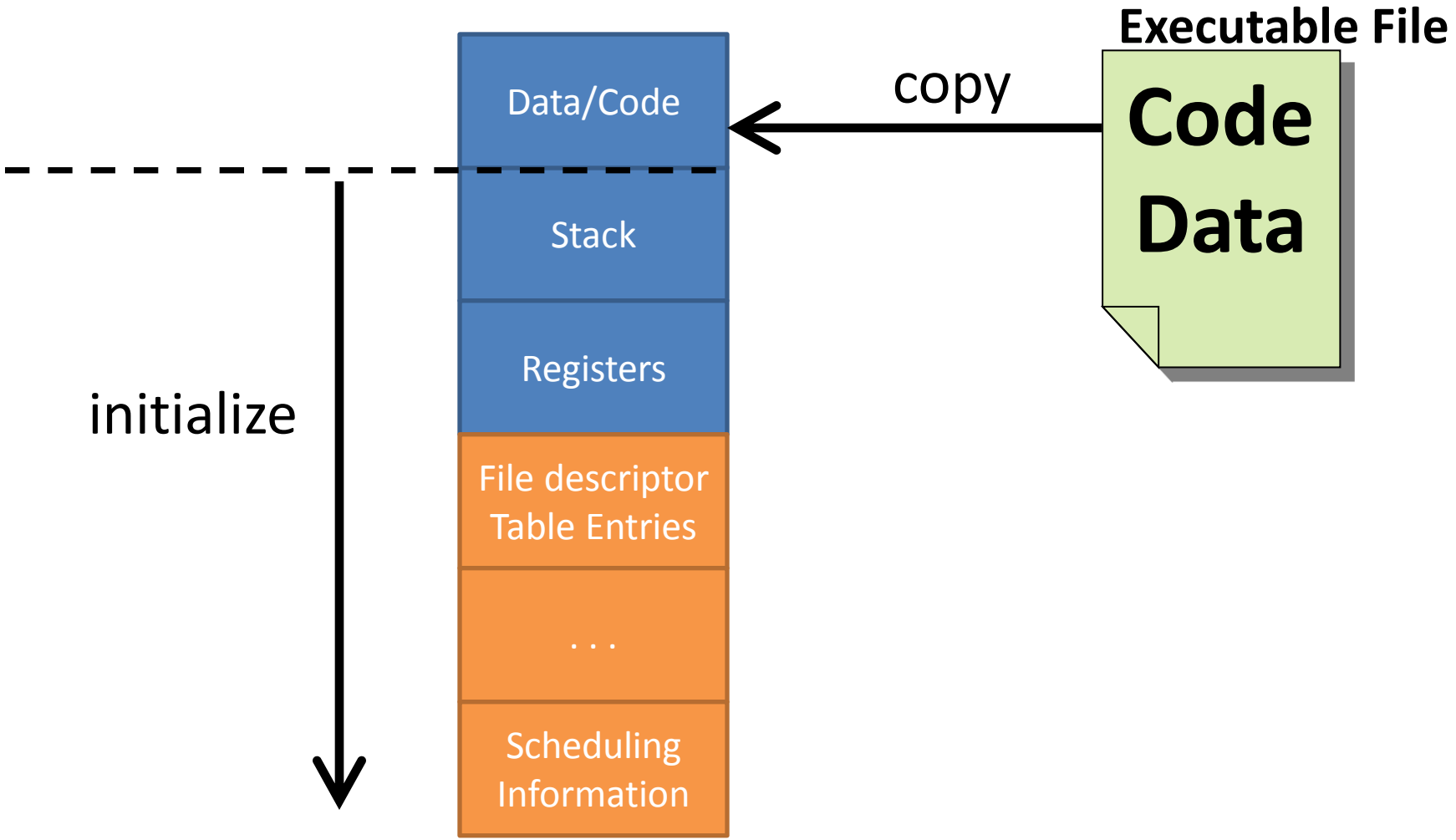
`fork()`

- How does the forked process differ?
  - http://pubs.opengroup.org/onlinepubs/9699919799/functions/fork.html

Process State



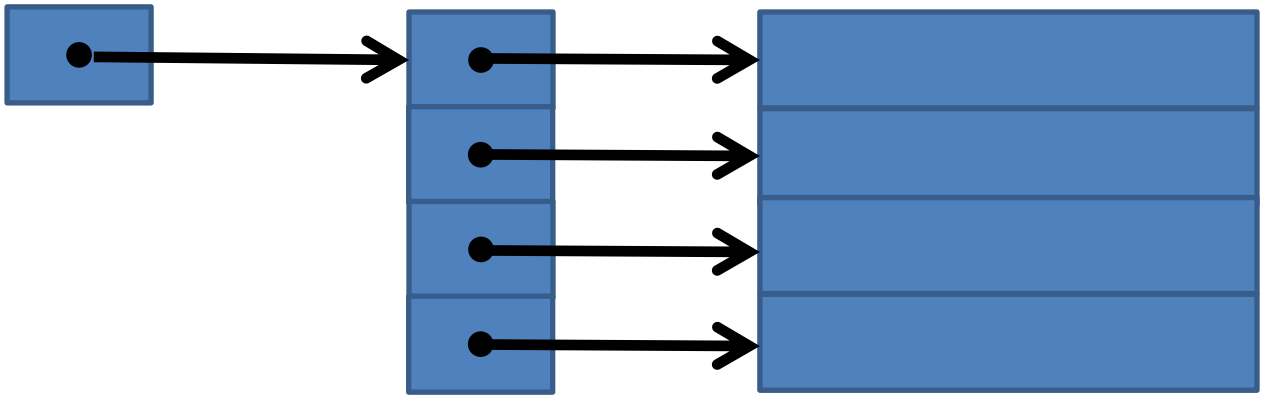
execv()


Executable File
Code
Data
copy
Data/Code
Stack
Registers
File descriptor
Table Entries
. . .
Scheduling
Information
initialize

Environment Variables

```
extern char **environ;
```

- pointer to an array of strings
- Required not to directly modify the pointers to which **`environ`** points



Parsing Strings

- **`strtok_r()`**
  - See code example on website

Process Groups

- See example

QUESTIONS