Operating Systems

(Sistemas Operativos)

Guide 2: Fork

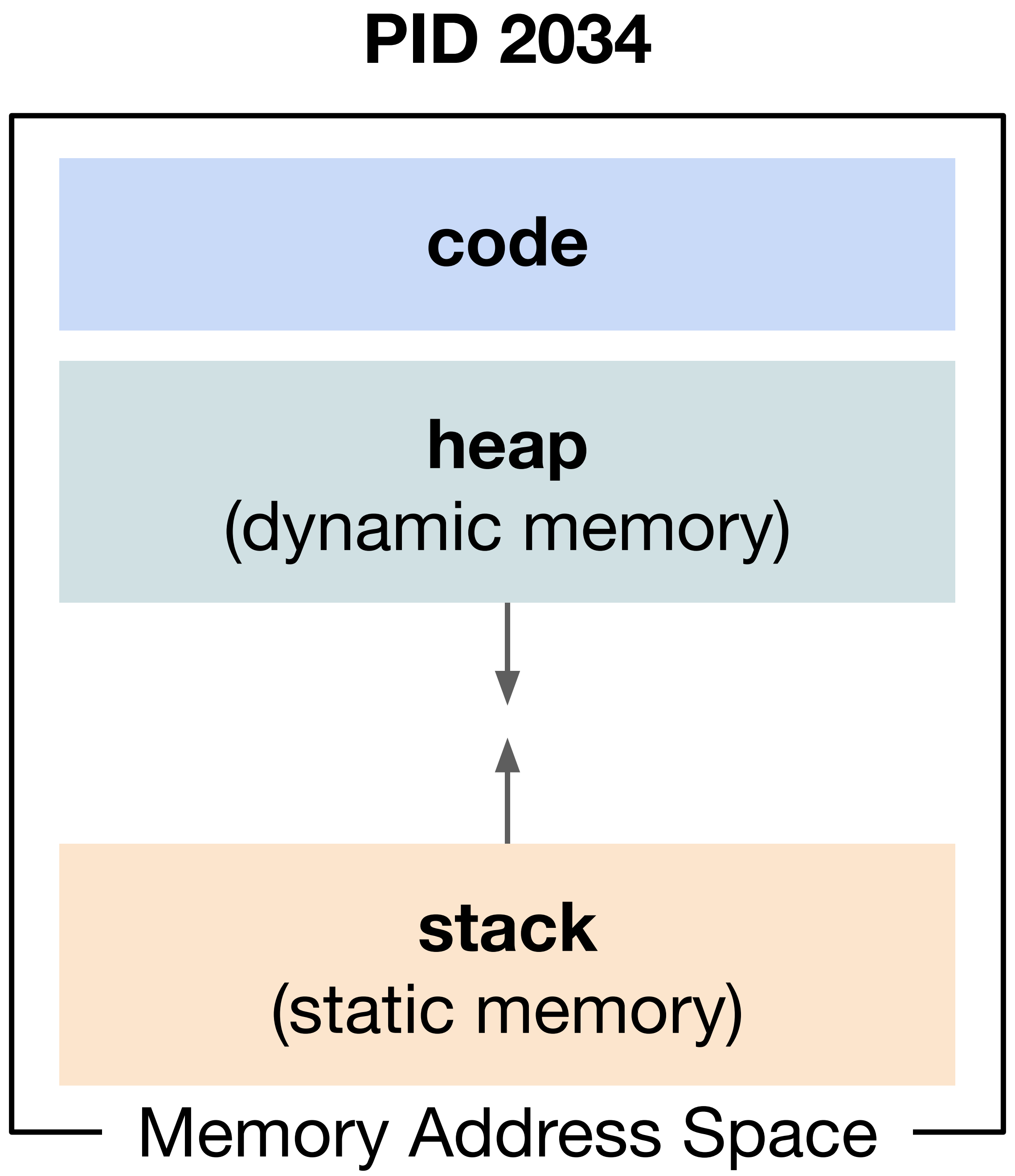
University of Minho

2024 - 2025

Process API

Memory address space

A **process**, identified by a **process identifier (PID)**, has access to its own **memory address space**



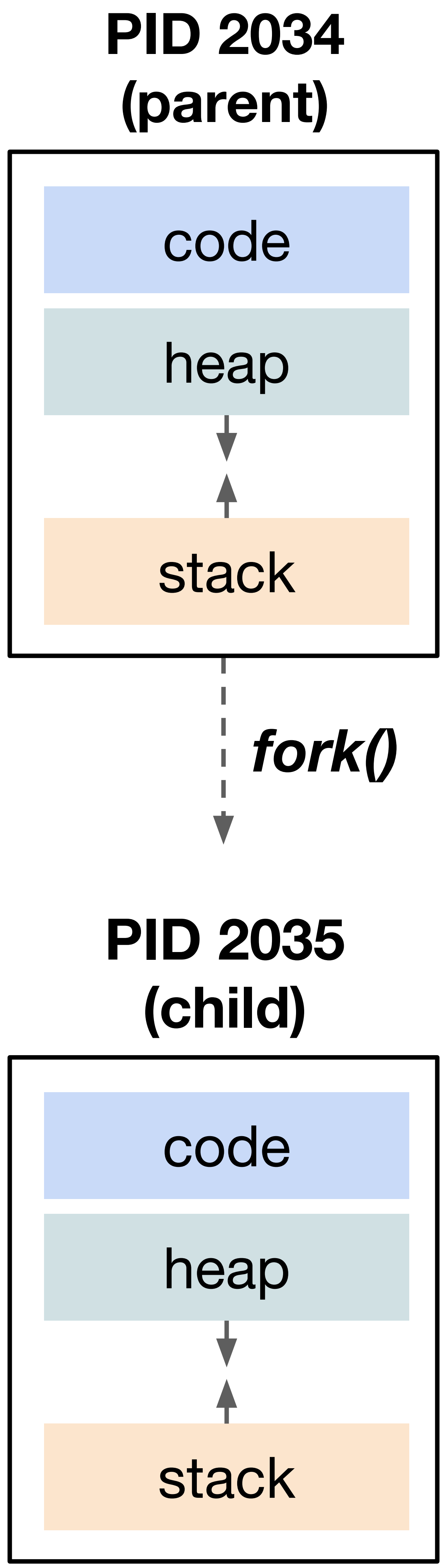
*simplified representation of an address space (*e.g.*, not including the static data segment)

Process API

Creating a process

#include <unistd.h>

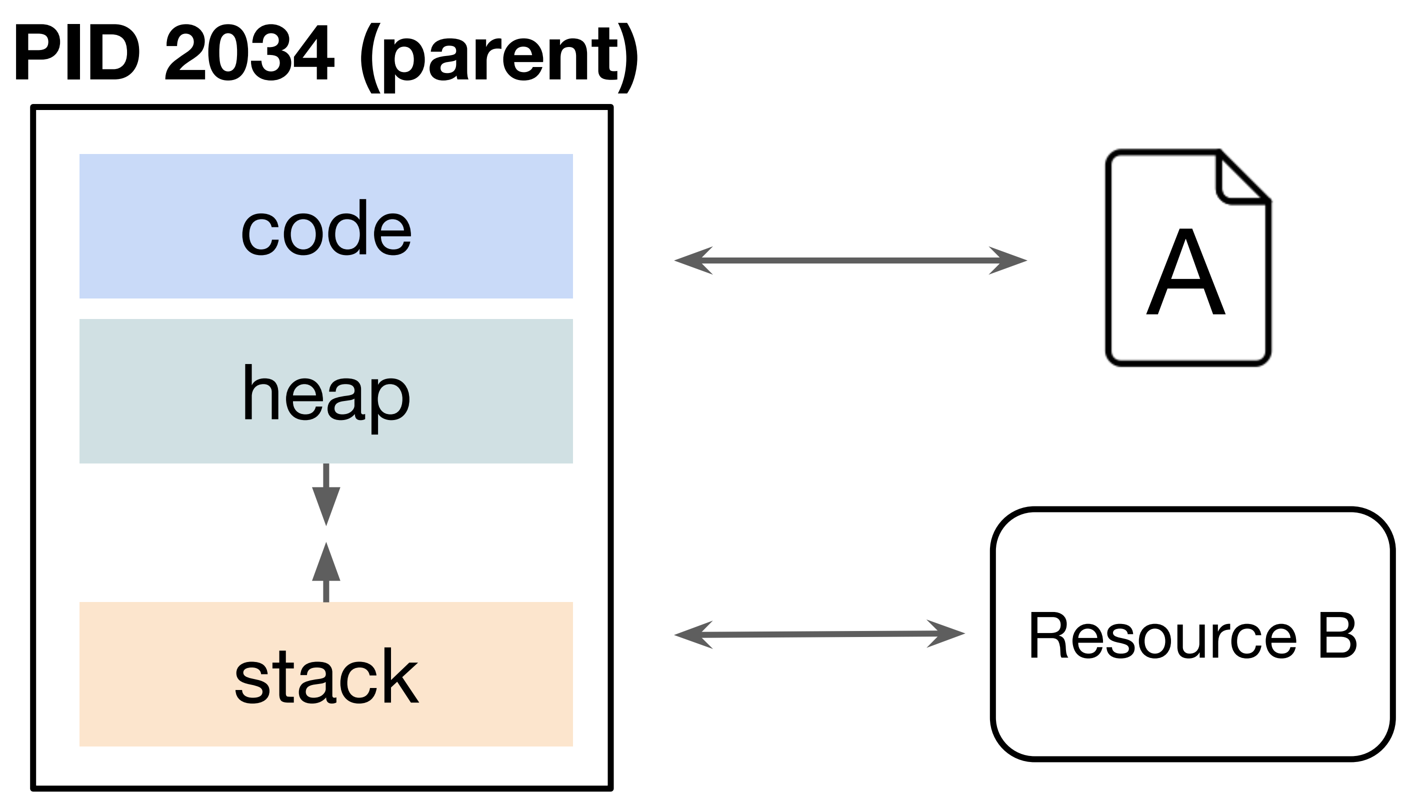
- *pid_t* ***fork****(void)*
  - Returns:
    - the **PID** of the **child-process** **<u>to the parent process</u>**
    - **0 <u>to the child-process</u>**
    - **-1 on error**



For more information: *$ man 2 fork*

Process API

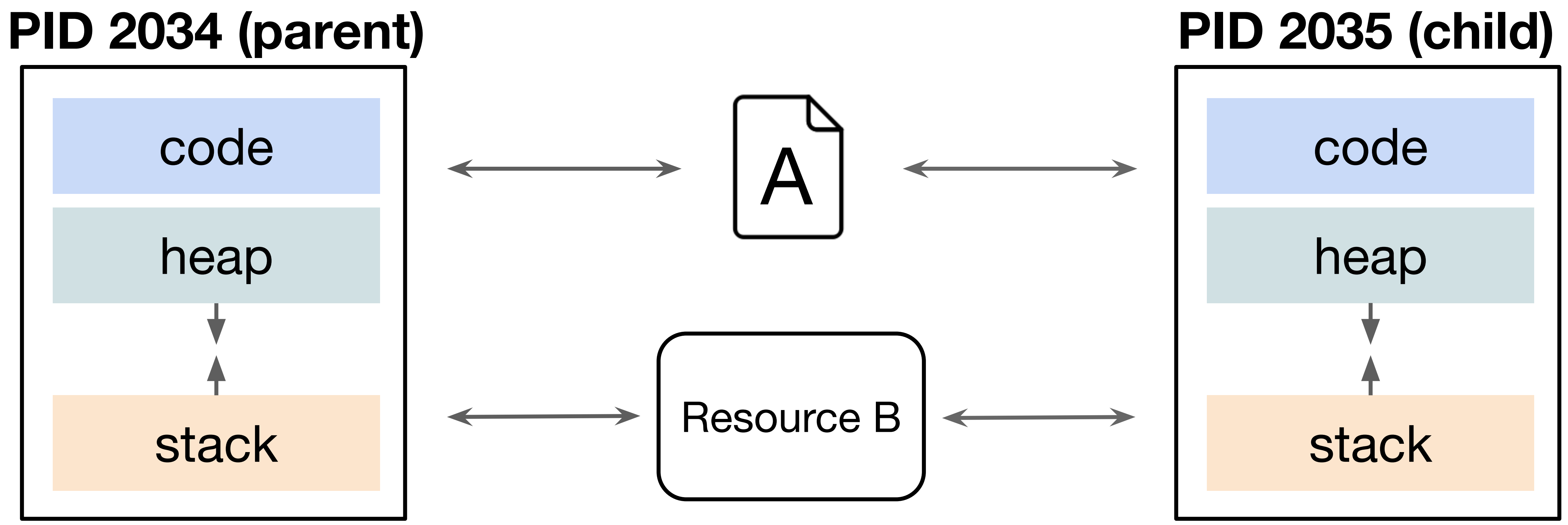
Example: creating a process



```
int main() {
  pid_t pid;

  if ((pid == fork()) == 0) {
    // child-process code
  } else {
    // parent-process code
  }
}
```

Process API

Example: creating a process



fork() returns **2035 to the parent-process**

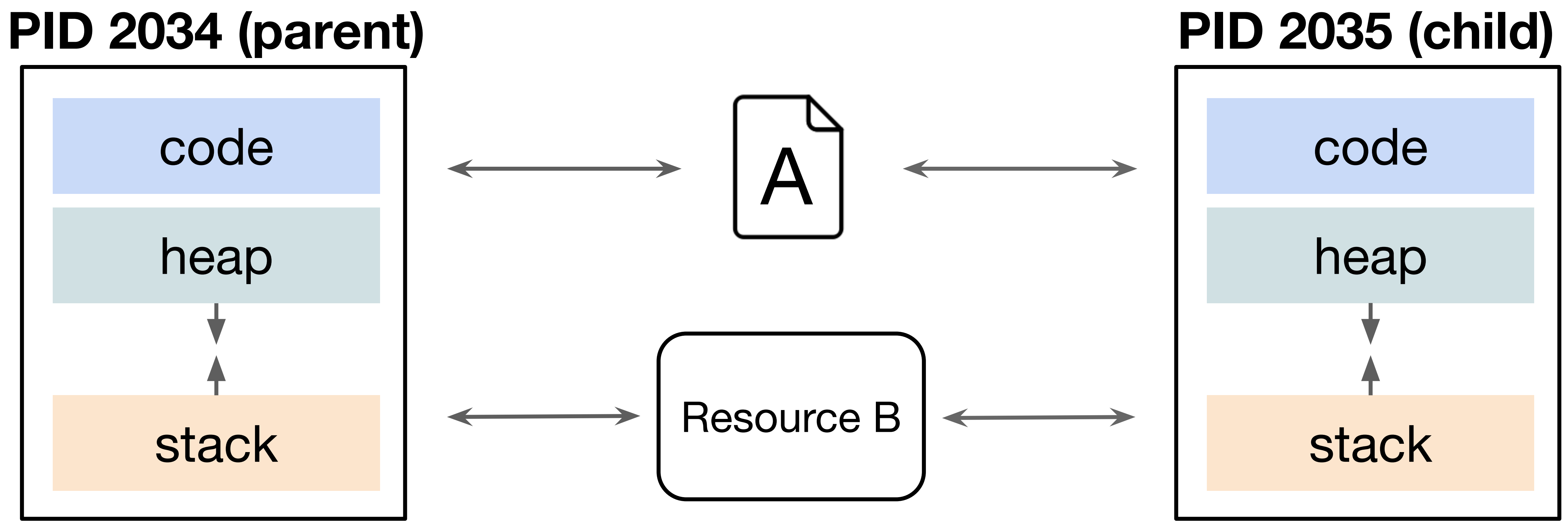
fork() returns **0 to the child-process**

```
int main() {
  pid_t pid;

  if ((pid == fork()) == 0) {
    // child-process code
  } else {
    // parent-process code
  }
}
```

```
int main() {
  pid_t pid;

  if ((pid == fork()) == 0) {
    // child-process code
  } else {
    // parent-process code
  }
}
```

Process API

Example: creating a process



parent enters here

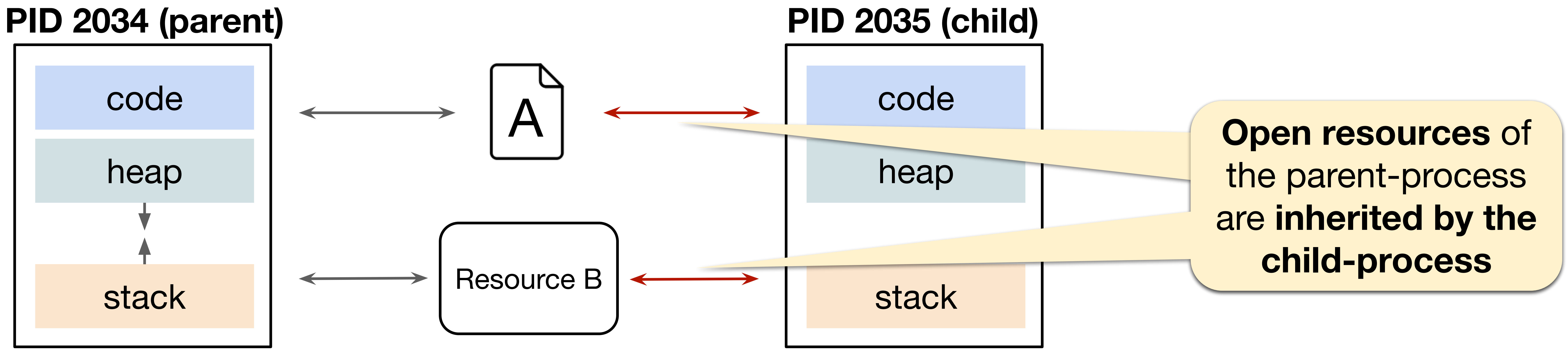
```c
int main() {
  pid_t pid;

  if ((pid == fork()) == 0) {
    // child-process code
  } else {
    // parent-process code
  }
}
```

child enters here

```c
int main() {
  pid_t pid;

  if ((pid == fork()) == 0) {
    // child-process code
  } else {
    // parent-process code
  }
}
```

Process API

Example: creating a process



Open resources of the parent-process are **inherited by the child-process**

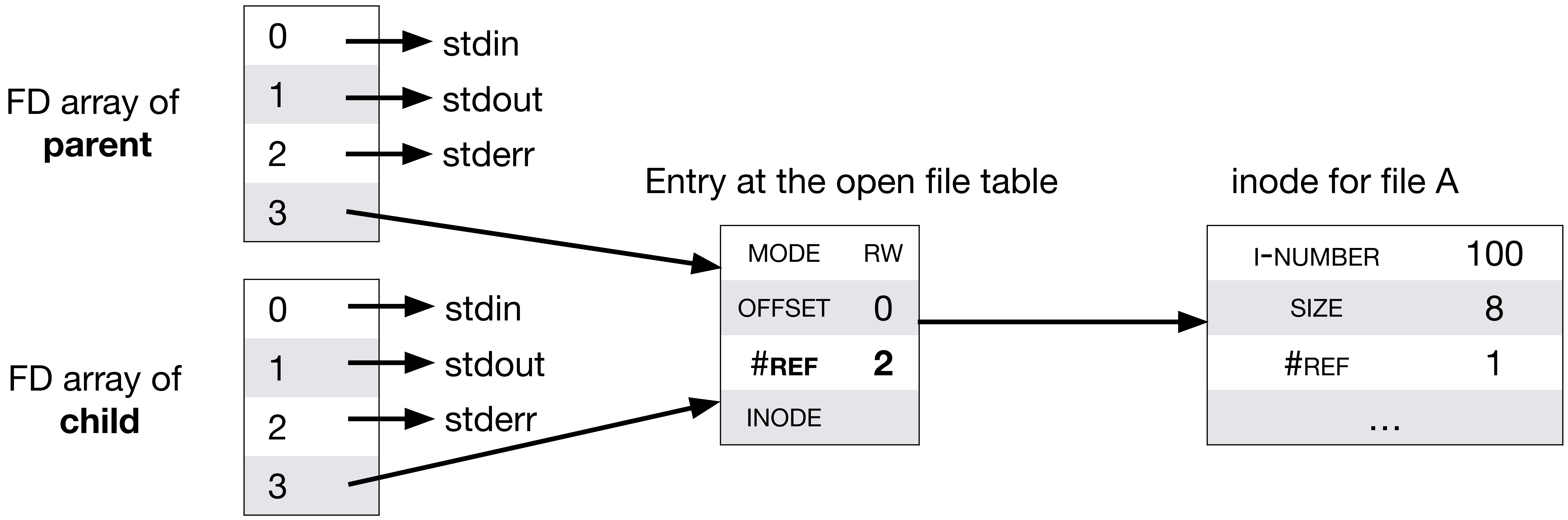
```c
int main() {
  pid_t pid;

  if ((pid == fork()) == 0) {
    // child-process code
  } else {
    // parent-process code
  }
}
```

```c
int main() {
  pid_t pid;

  if ((pid == fork()) == 0) {
    // child-process code
  } else {
    // parent-process code
  }
}
```

File System Interface

Shared open file table entries with **fork**

- Parent and child **share the open file table entry**
- **Be careful:** reads, writes, and seeks may update the **offset field concurrently**!



FD array of **parent**

0	stdin
1	stdout
2	stderr
3	

FD array of **child**

0	stdin
1	stdout
2	stderr
3	

Entry at the open file table

MODE	RW
OFFSET	0
#REF	**2**
INODE	

inode for file A

I-NUMBER	100
SIZE	8
#REF	1
…	

Process API

Terminating processes: **child's perspective**

#include <unistd.h>

- *void **_exit**(int status)*
  - **status:** status of the current process when exiting
    - 0: the process exited normally

For more information: *$ man 2 exit*

Process API

Terminating processes: **parent's perspective**

#include <sys/wait.h>

- *pid_t* ***wait****(int *status)*
  - **status:** memory address where termination information of the child-process is written to
  - Returns: the **PID** of the **terminated child-process**

For more information: *$ man 2 wait*

#include <sys/wait.h>

- **WIFEXITED***(status)*
  - Returns: 1 if the child-process exited normally
- **WEXITSTATUS***(status)*
  - employed **only if WIFEXITED returned 1**
  - Returns: the **exit status of the child –** the least significant 8 bits of *status* **specified when the child exited**

Process API

Terminating processes: **parent's perspective**

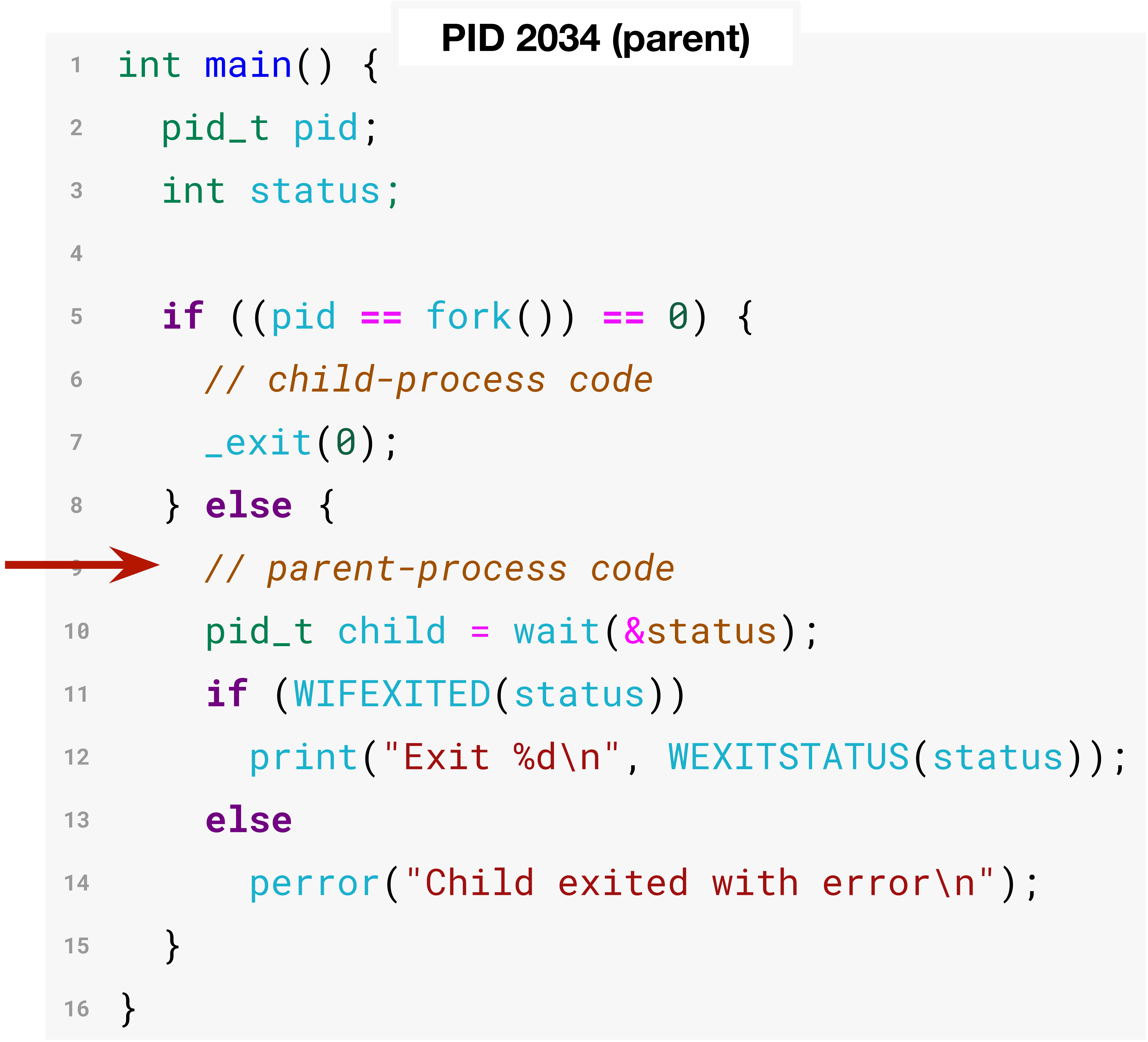
#include <sys/wait.h>

- *pid_t **waitpid**(pid_t pid, int * wstatus, int options)*
  - **pid:**
    - **> 0:** wait for the **child process whose PID is *pid***
    - check wait's man page for other wait behaviours that one can specify with ***pid***
  - **wstatus:** memory address where termination information of the child-process is written to
  - **options:** extra arguments that change waitpid's default behavior

For more information: *$ man 2 wait*

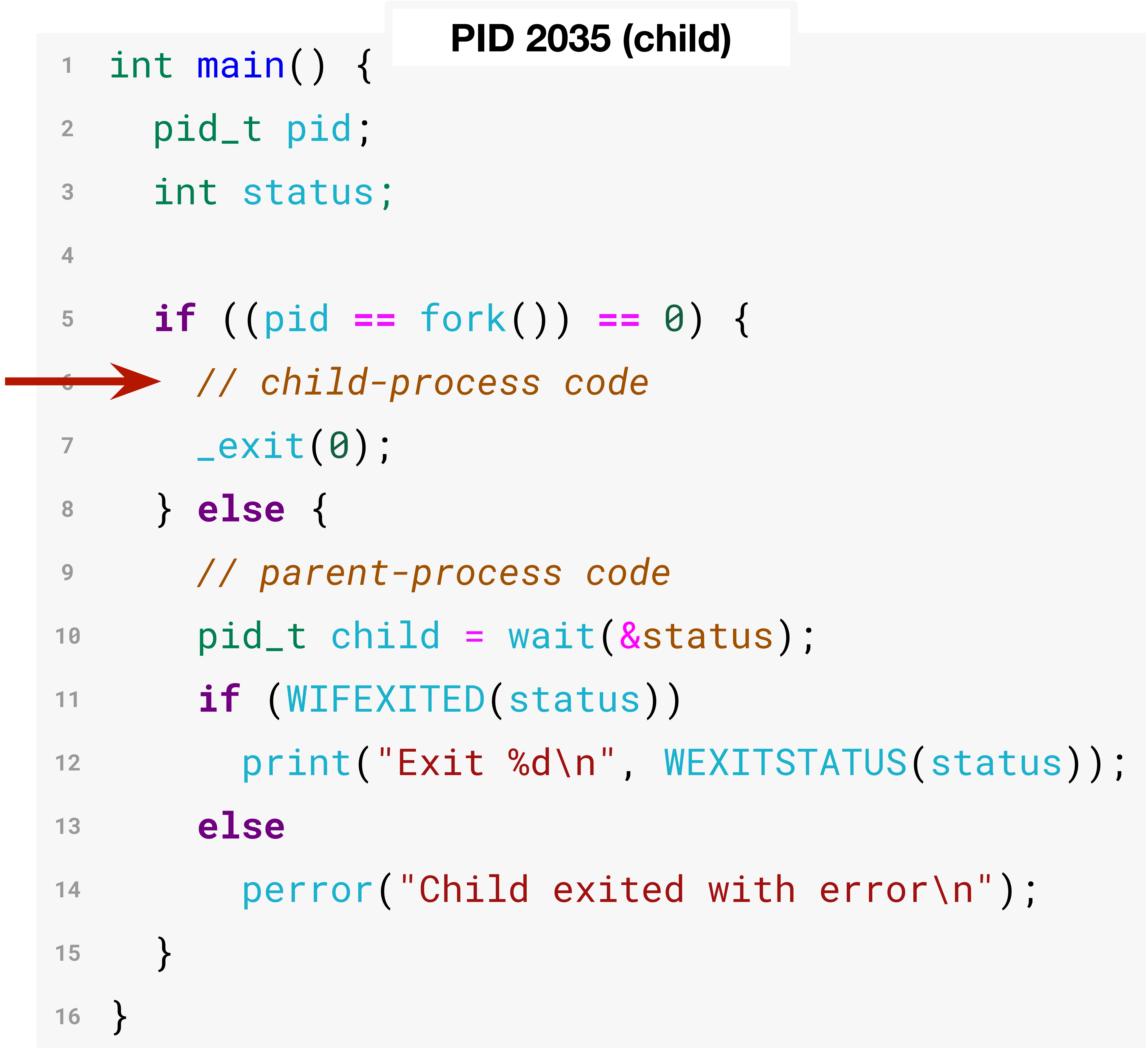
Process API

Example: Terminating a Process



PID 2034 (parent)

```c
int main() {
  pid_t pid;
  int status;

  if ((pid == fork()) == 0) {
    // child-process code
    _exit(0);
  } else {
    // parent-process code
    pid_t child = wait(&status);
    if (WIFEXITED(status))
      print("Exit %d\n", WEXITSTATUS(status));
    else
      perror("Child exited with error\n");
  }
}
```

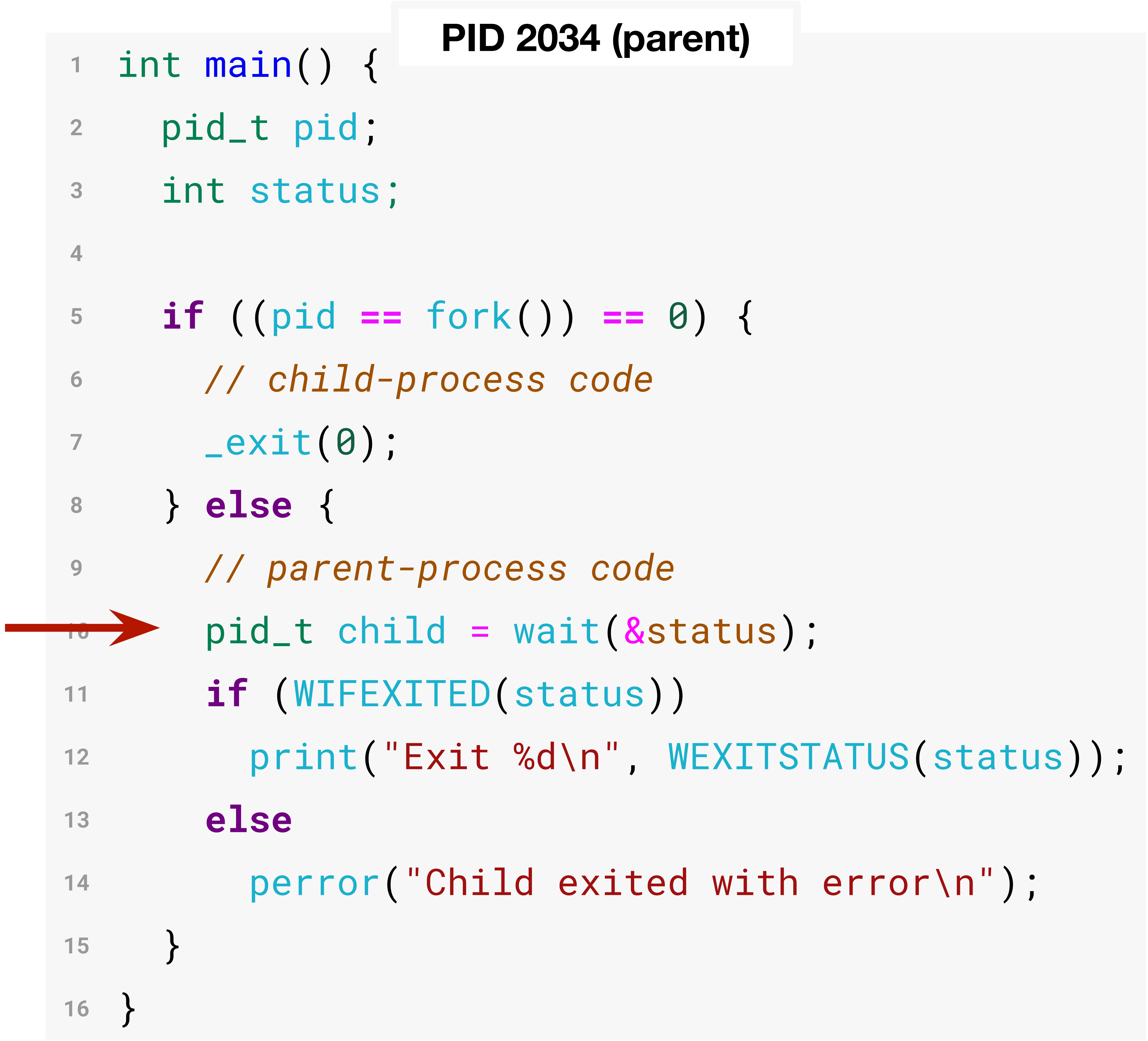


PID 2035 (child)

```c
int main() {
  pid_t pid;
  int status;

  if ((pid == fork()) == 0) {
    // child-process code
    _exit(0);
  } else {
    // parent-process code
    pid_t child = wait(&status);
    if (WIFEXITED(status))
      print("Exit %d\n", WEXITSTATUS(status));
    else
      perror("Child exited with error\n");
  }
}
```

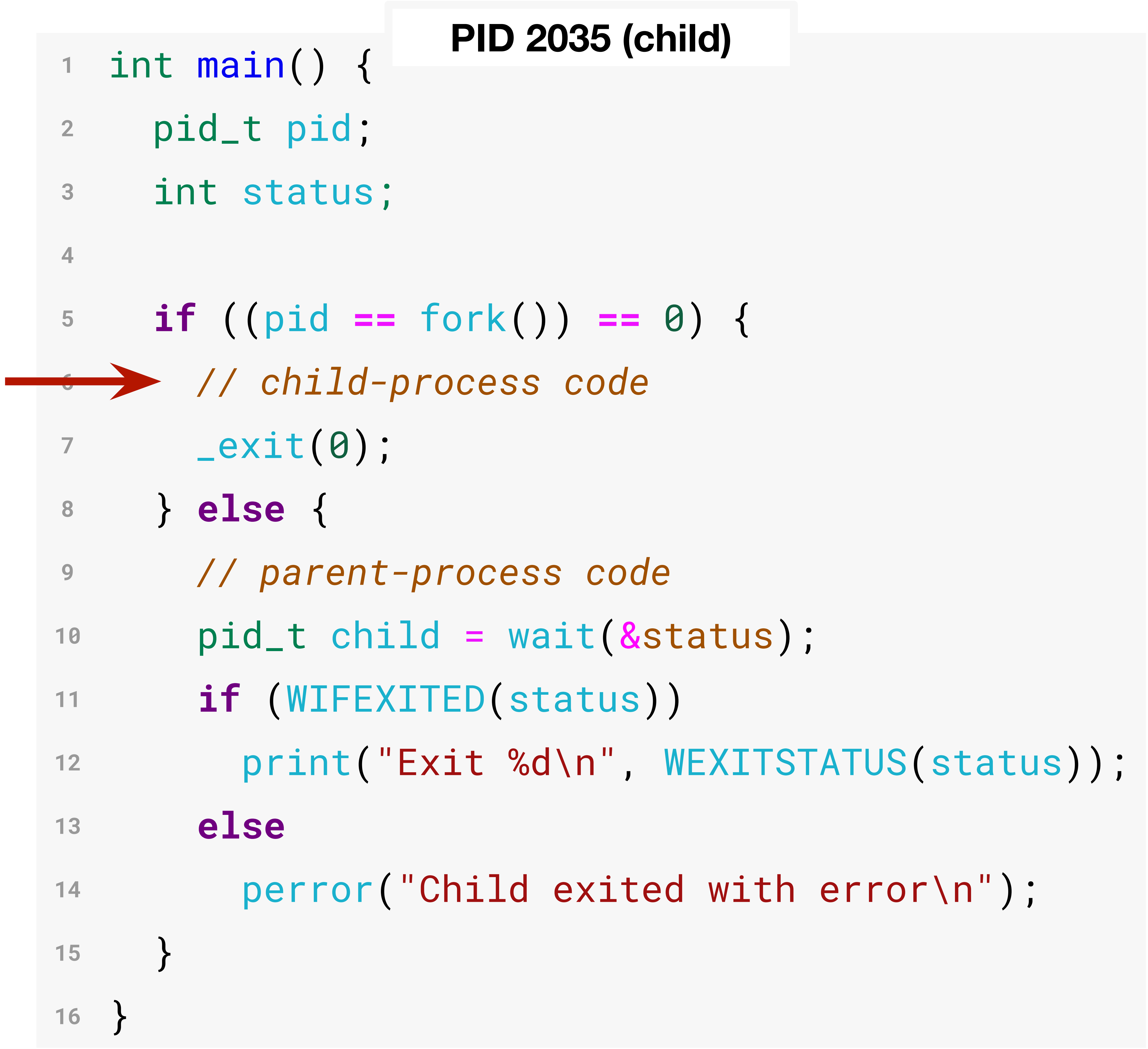
Process API

Example: Terminating a Process



PID 2034 (parent)

```c
int main() {
  pid_t pid;
  int status;

  if ((pid == fork()) == 0) {
    // child-process code
    _exit(0);
  } else {
    // parent-process code
    pid_t child = wait(&status);
    if (WIFEXITED(status))
      print("Exit %d\n", WEXITSTATUS(status));
    else
      perror("Child exited with error\n");
  }
}
```



PID 2035 (child)

```c
int main() {
  pid_t pid;
  int status;

  if ((pid == fork()) == 0) {
    // child-process code
    _exit(0);
  } else {
    // parent-process code
    pid_t child = wait(&status);
    if (WIFEXITED(status))
      print("Exit %d\n", WEXITSTATUS(status));
    else
      perror("Child exited with error\n");
  }
}
```

Process API

Example: Terminating a Process

PID 2034 (parent)

```
int main() {
  pid_t pid;
  int status;

  if ((pid == fork(
    // child-proces
    _exit(0);
  } else {
    // parent-process  ode
    pid_t child = wait(&status);
    if (WIFEXITED(status))
      print("Exit %d\n", WEXITSTATUS(status));
    else
      perror("Child exited with error\n");
  }
}
```

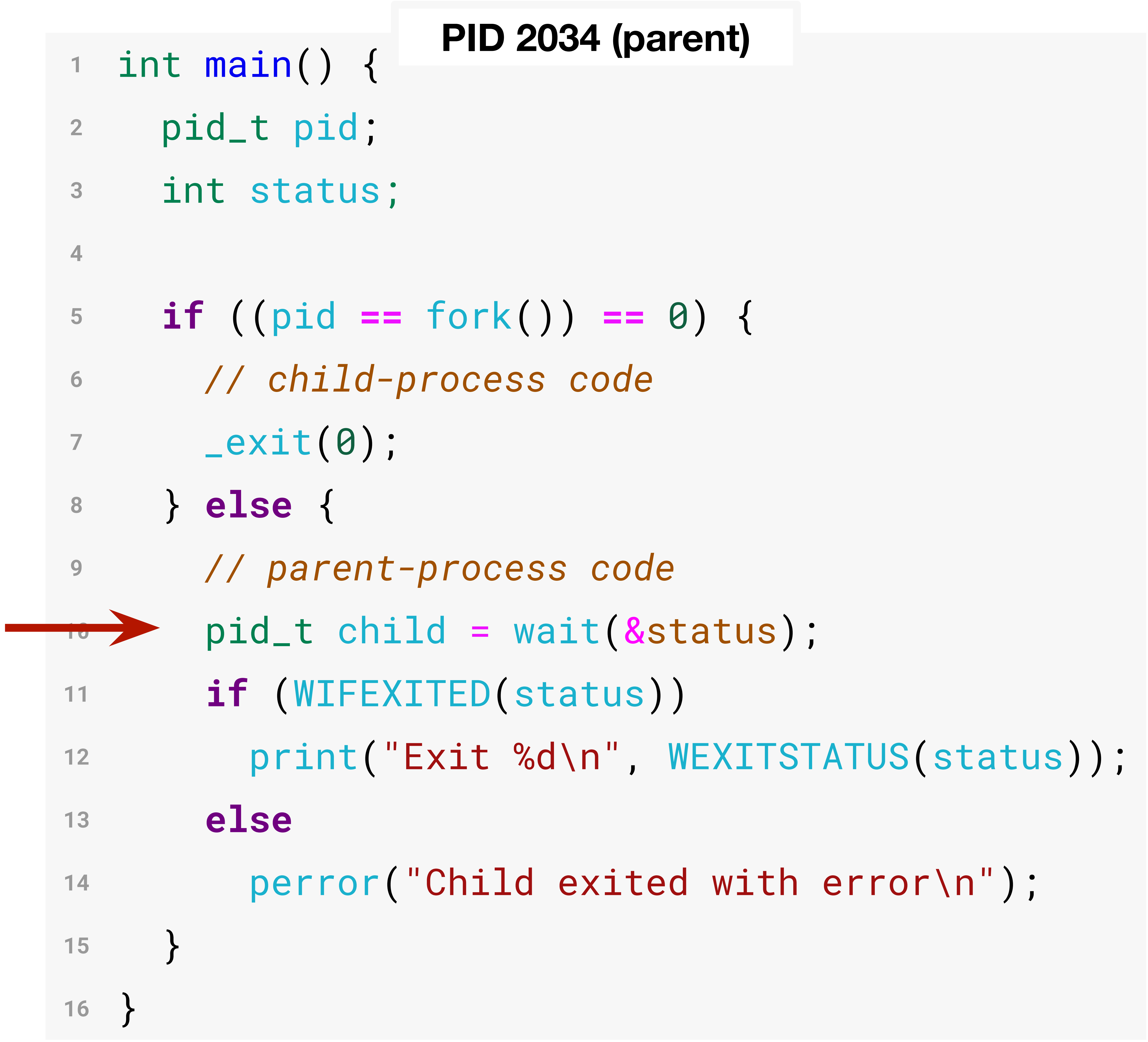
wait() blocks the parent until the child exits

PID 2035 (child)

```
int main() {
  pid_t pid;
  int status;

  if ((pid == fork()) == 0) {
    // child-process code
    _exit(0);
  } else {
    // parent-process code
    pid_t child = wait(&status);
    if (WIFEXITED(status))
      print("Exit %d\n", WEXITSTATUS(status));
    else
      perror("Child exited with error\n");
  }
}
```

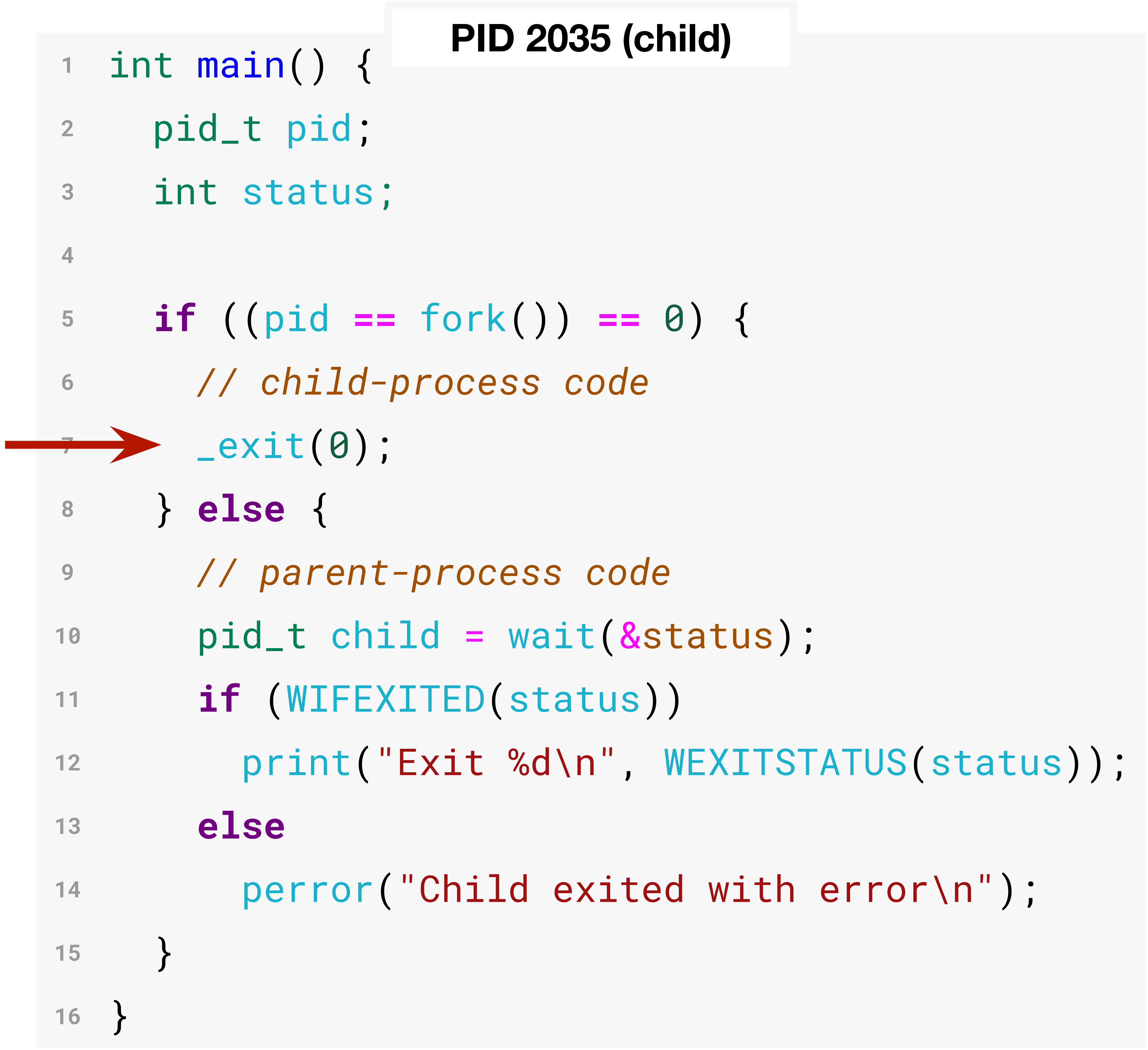
Process API

Example: Terminating a Process



PID 2034 (parent)

```c
int main() {
  pid_t pid;
  int status;

  if ((pid == fork()) == 0) {
    // child-process code
    _exit(0);
  } else {
    // parent-process code
    pid_t child = wait(&status);
    if (WIFEXITED(status))
      print("Exit %d\n", WEXITSTATUS(status));
    else
      perror("Child exited with error\n");
  }
}
```



PID 2035 (child)

```c
int main() {
  pid_t pid;
  int status;

  if ((pid == fork()) == 0) {
    // child-process code
    _exit(0);
  } else {
    // parent-process code
    pid_t child = wait(&status);
    if (WIFEXITED(status))
      print("Exit %d\n", WEXITSTATUS(status));
    else
      perror("Child exited with error\n");
  }
}
```

Process API

Example: Terminating a Process

PID 2034 (parent)

```
int main() {
  pid_t pid;
  int status;

  if ((pid == fork()) == 0) {
    // child-process code
    _exit(0);
  } else {
    // parent-process code
    pid_t child = wait(&status);
    if (WIFEXITED(status))
      print("Exit %d\n", WEXITSTATUS(status));
    else
      perror("Child exited with error\n");
  }
}
```

PID 2035 (child)

```
int main() {
  pid
  int

  if ((pi   fork()) == 0) {
    // c  ld-process code
    _exit(0);
  } else {
    // parent-process code
    pid_t child = wait(&status);
    if (WIFEXITED(status))
      print("Exit %d\n", WEXITSTATUS(status));
    else
      perror("Child exited with error\n");
  }
}
```

_***exit()*** terminates the current process

Process API

Example: Terminating a Process

PID 2034 (parent)

```c
int main() {
  pid_t pid;
  int status;

  if ((pid == fork()) == (
    // child-process code
    _exit(0);
  } else {
    // parent-process code
    pid_t child = wait(&status);
    if (WIFEXITED(status))
      print("Exit %d\n", WEXITSTATUS(status));
    else
      perror("Child exited with error\n");
  }
}
```

The ***status*** passed by the child's _***exit()*** *is* ***forward to the parent***

PID 2035 (child)

```c
int main() {
  pid_t pid;
  int status;

  if ((pid == fork()) == 0) {
    // child-process code
    _exit(0);
  } else {
    // parent-process code
    pid_t child = wait(&status);
    if (WIFEXITED(status))
      print("Exit %d\n", WEXITSTATUS(status));
    else
      perror("Child exited with error\n");
  }
}
```

Process API

Example: Terminating a Process

PID 2034 (parent)

```
int main() {
  pid_t pid;
  int status;

  if ((pid == fork()) == 0) {
                    s code



    / parent-process code
    pid_t child = wait(&status);
    if (WIFEXITED(status))
      print("Exit %d\n", WEXITSTATUS(status));
    else
      perror("Child exited with error\n");
  }
}
```

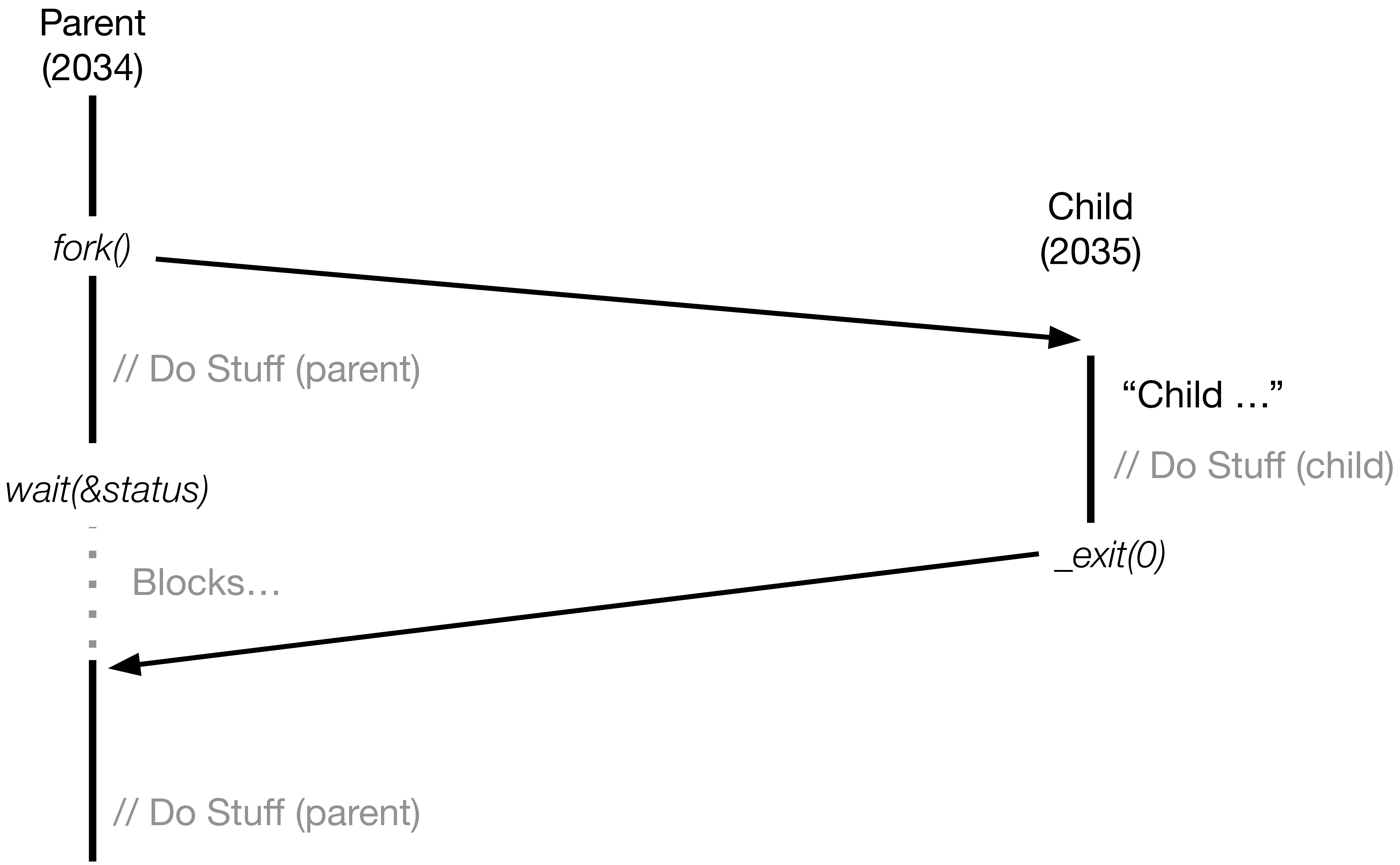
The **parent continues** its execution **after the child has terminated**

PID 2035 (child)

```
int main() {
  pid_t pid;
  int status;

  if ((pid == fork()) == 0) {
    // child-process code
    _exit(0);
  } else {
    // parent-process code
    pid_t child = wait(&status);
    if (WIFEXITED(status))
      print("Exit %d\n", WEXITSTATUS(status));
    else
      perror("Child exited with error\n");
  }
}
```

Process API

Example: Terminating a Process



Parent (2034)

fork()

// Do Stuff (parent)

wait(&status)

Blocks…

// Do Stuff (parent)

Child (2035)

“Child …”

// Do Stuff (child)

_exit(0)

More Information

- **Chapter 5** - Remzi H. Arpaci-Dusseau, Andrea C. Arpaci-Dusseau. **Operating Systems: Three Easy Pieces.** Arpaci-Dusseau Books, 2018.
- Avi Silberschatz, Peter Baer Galvin, Greg Gagne. **Operating System Concepts (10. ed)**. John Wiley & Sons, 2018.