Operating Systems

(Sistemas Operativos)

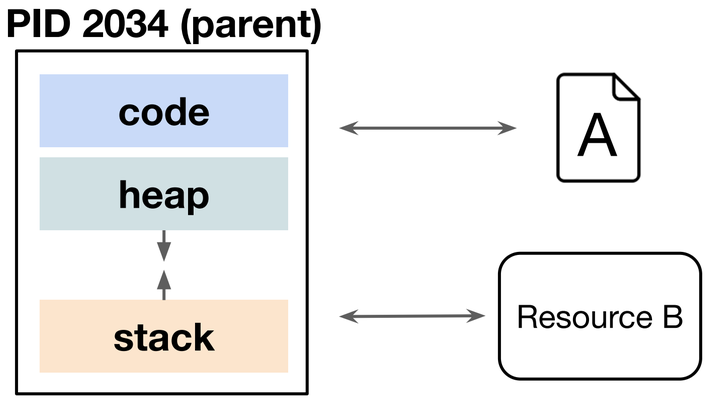
Guide 3: Exec

University of Minho

2024 - 2025

Process API

Fork recap

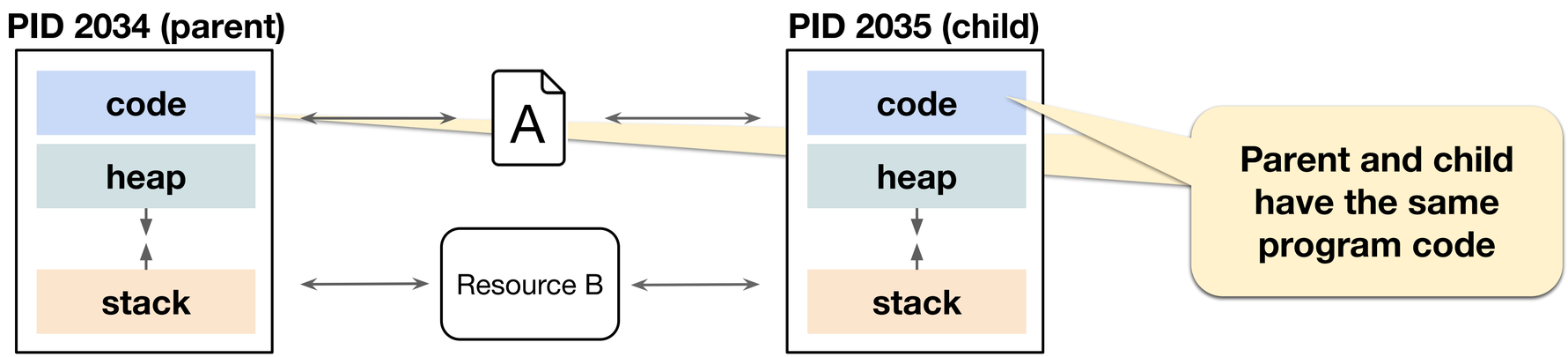



```
int main() {
  pid_t pid;

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

Process API

Fork recap




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

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

parent enters here

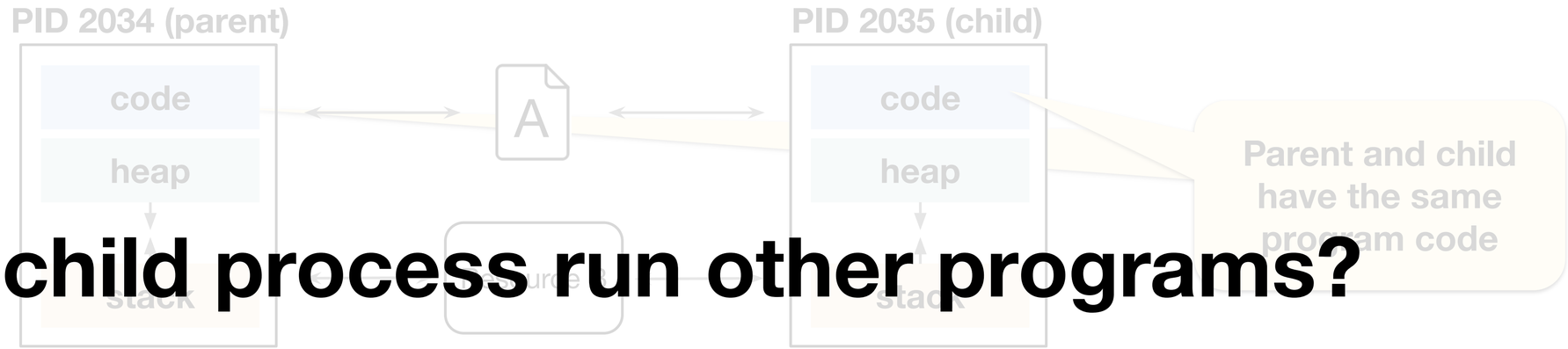
child enters here

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

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

Process API

Fork recap



Can the child process run other programs?

```
int main() {
  pid_t pid;

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

parent enters here

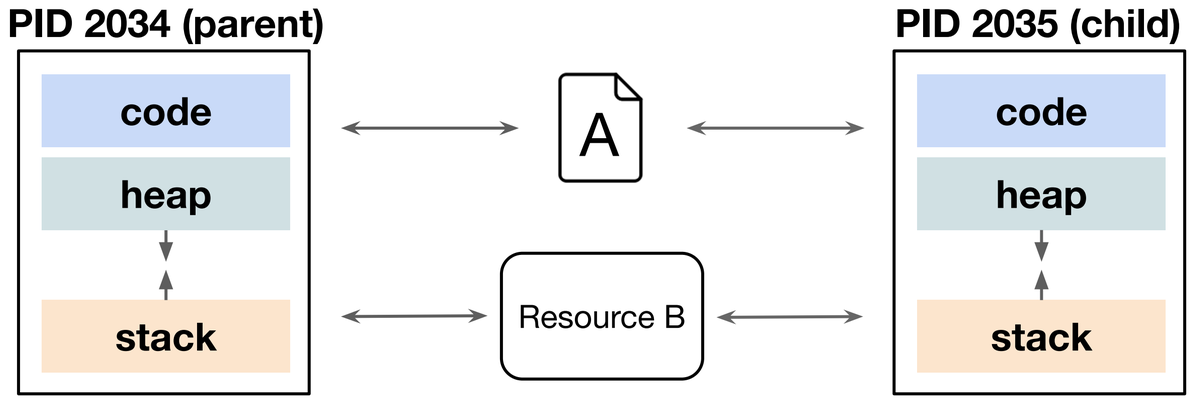
child enters here

```
int main() {
  pid_t pid;

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

Process API

Fork + Exec




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

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

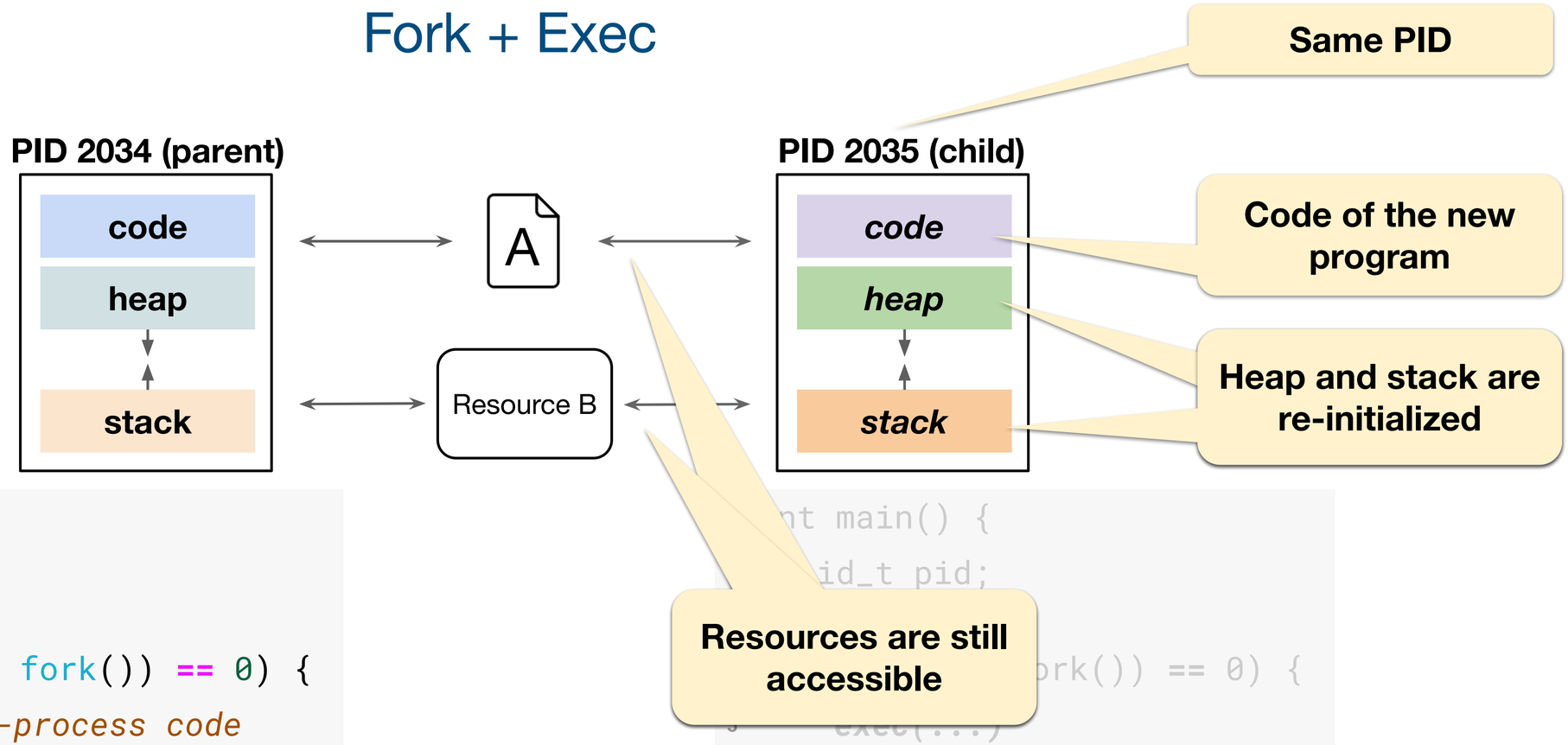
```c
int main() {
  pid_t pid;

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

exec() makes the process **run the specified executable**

Process API

Fork + Exec




```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) {
    exec(...)
  } else {
    // parent-process code
  }
}
```

Process API

Exec family

#include <unistd.h>

- *int **execl**(const char *pathname, const char *arg0, …, NULL)*
- *int **execlp**(const char *file, const char *arg0, …, NULL);*
- *int **execv**(const char *pathname, char *const argv[]);*
- *int **execvp**(const char *file, char *const argv[]);*

…

Obs:

1. These functions **only return on error**.
2. By convention, ***argv0*** (or *argv[0]*) **is the name of the program being executed**
3. The list of **arguments** must be **terminated by a null pointer.**

For more information: *$ man 3 exec*

Exec suffixes:

- **l:** program arguments passed **as arguments** to the exec function
- **v:** program arguments passed **as a vector (array)** to the exec function
- **p:** search for the program in the ***PATH*** **variable**
  - **otherwise, provide the program path (not simply the name)**

Process API

Example: execlp

```c
int main() {
  pid_t pid;
  if ((pid = fork()) == 0) {
    printf("Child (pid: %d)\n", getpid()):
    execlp("wc", "wc", "exec.c", NULL);
    printf("I'm Here Child (pid: %d)\n", getpid()
    _exit(1);
  } else {
    wait(NULL);
    printf("Parent of %d (pid: %d)\n", pid, getpid());
  }
  return 0;
}
```

execlp **receives 'wc' arguments**

Process API

Example: execlp

execlp searches for the given filename in the *PATH* (which usually has */usr/bin*)

argv0 is the **program name**

```
int main() {
  pid_t pid;
  if ((pid = fork()) == 0) {
    printf("Child (pid: %d)\n", getpid());
    execlp("wc", "wc", "exec.c", NULL);
    printf("I'm Here Child (pid: %d)\n", getpid()
    _exit(1);
  } else {
    wait(NULL);
    printf("Parent of %d (pid: %d)\n", pid, getpid());
  }
  return 0;
}
```

Process API

Example: execl

without the suffix '*p*', execl **needs the path to 'wc'**

check: *$ which wc*

```c
int main() {
  pid_t pid;
  if ((pid = fork()) == 0) {
    printf("Child (pid: %d)\n", getpid());
    execl("/usr/bin/wc", "wc", "exec.c", NULL);
    printf("I'm Here Child (pid: %d)\n", getpid())
    _exit(1);
  } else {
    wait(NULL);
    printf("Parent of %d (pid: %d)\n", pid, getpid());
  }
  return 0;
}
```

Process API

Example: execvp

```
1  int main() {
2    pid_t pid;
3    if ((pid = fork()) == 0) {
4      printf("Child (pid: %d)\n", getpid());
5      char* args[3];
6      args[0] = strdup("wc");
7      args[1] = strdup("exec.c");
8      args[2] = NULL;
9      execvp(args[0], args);
10     printf("I'm Here Child (pid: %d)\n", getpid())
11     _exit(1);
12   } else {
13     wait(NULL);
14     printf("Parent of %d (pid: %d)\n", pid, getpid());
15   }
16   return 0;
17 }
```

execvp receives array of arguments

Process API

Example: execvp

```c
int main() {
  pid_t pid;
  if ((pid = fork()) == 0) {
    printf("Child (pid: %d)\n", getpid());
    char* args[3];
    args[0] = strdup("wc");
    args[1] = strdup("exec.c");
    args[2] = NULL;
    execvp(args[0], args);
    printf("I'm Here Child (pid: %d)\n", getpid())
    _exit(1);
  } else {
    wait(NULL);
    printf("Parent of %d (pid: %d)\n", pid, getpid());
  }
  return 0;
}
```

The *wc* program executes on the child process

Process API

Example: execvp

```
int main() {
  pid_t pid;
  if ((pid = fork()) == 0) {
    printf("Child (pid: %d)\n", getpid());
    char* args[3];
    args[0] = strdup("wc");
    args[1] = strdup("exec.c");
    args[2] = NULL;
    execvp(args[0], args);
    printf("I'm Here Child (pid: %d)\n", getpid())
    _exit(1);
  } else {
    wait(NULL);
    printf("Parent of %d (pid: %d)\n", pid, getpid());
  }
  return 0;
}
```

Is "I'm Here Child …" **printed in this program?**

Process API

Example: execvp

```c
int main() {
  pid_t pid;
  if ((pid = fork()) == 0) {
    printf("Child (pid: %d)\n", getpid());
    char* args[3];
    args[0] = strdup("wc");
    args[1] = strdup("exec.c");
    args[2] = NULL;
    execvp(args[0], args);
    printf("I'm Here Child (pid: %d)\n", getpid())
    _exit(1);
  } else {
    wait(NULL);
    printf("Parent of %d (pid: %d)\n", pid, getpid());
  }
  return 0;
}
```

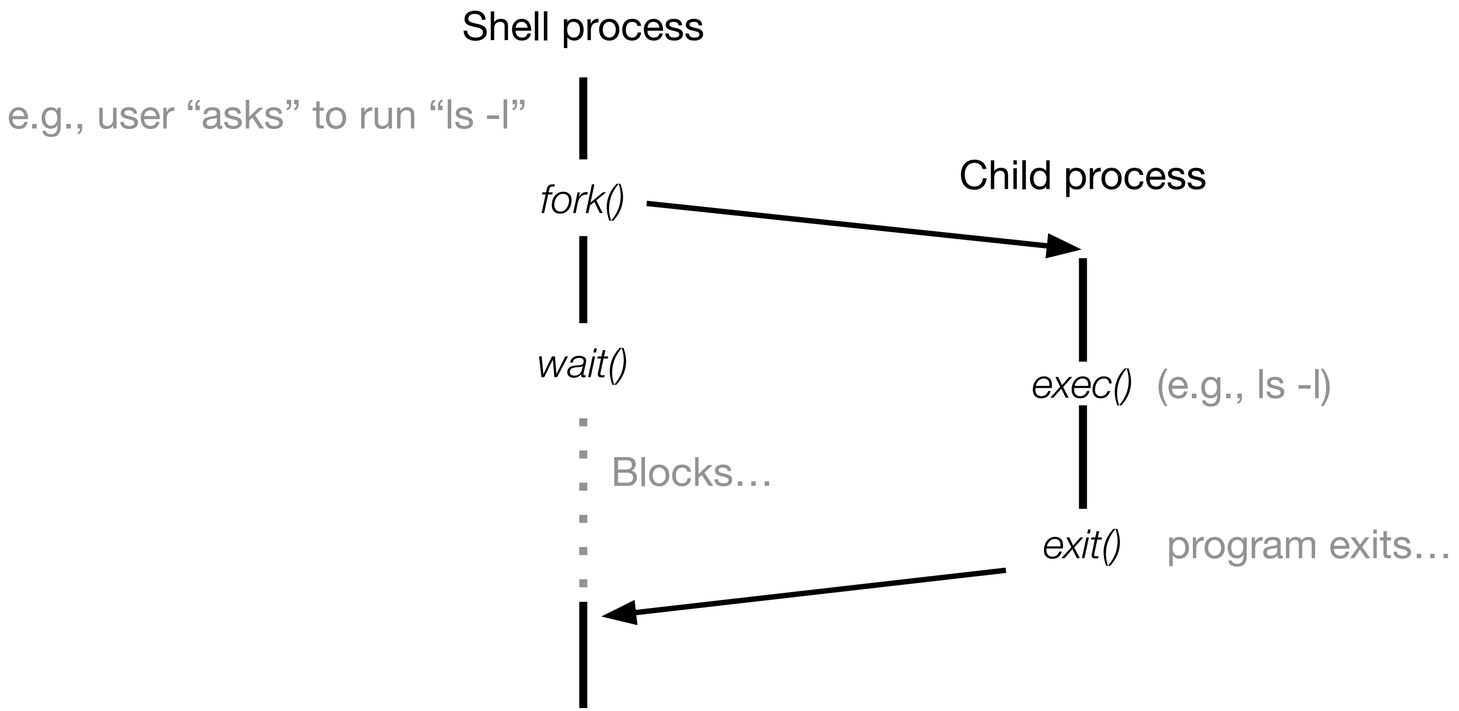
Output

```
prompt> ./p1
Child (pid: 77262)
    23    77    648 exec.c
Parent of 77262 (pid: 77261)
prompt>
```

Process API

A powerful combination

Question: How does shell (bash) execute your programs?



More Information

- **Chapters 4 and 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.