

Mestrado Integrado em Engenharia Informática

# Guião 2

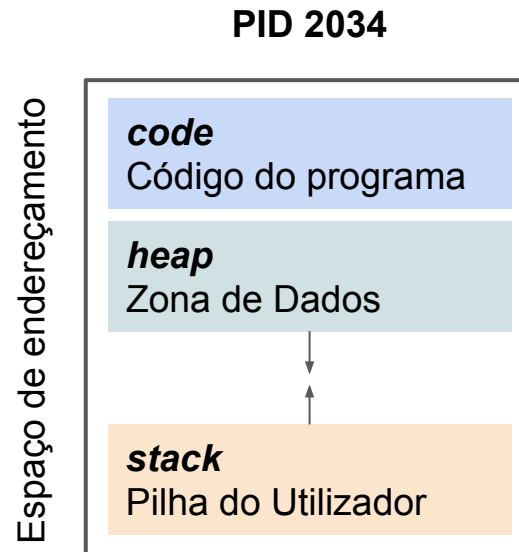
## **Gestão de Processos**

Sistemas Operativos 2018/2019

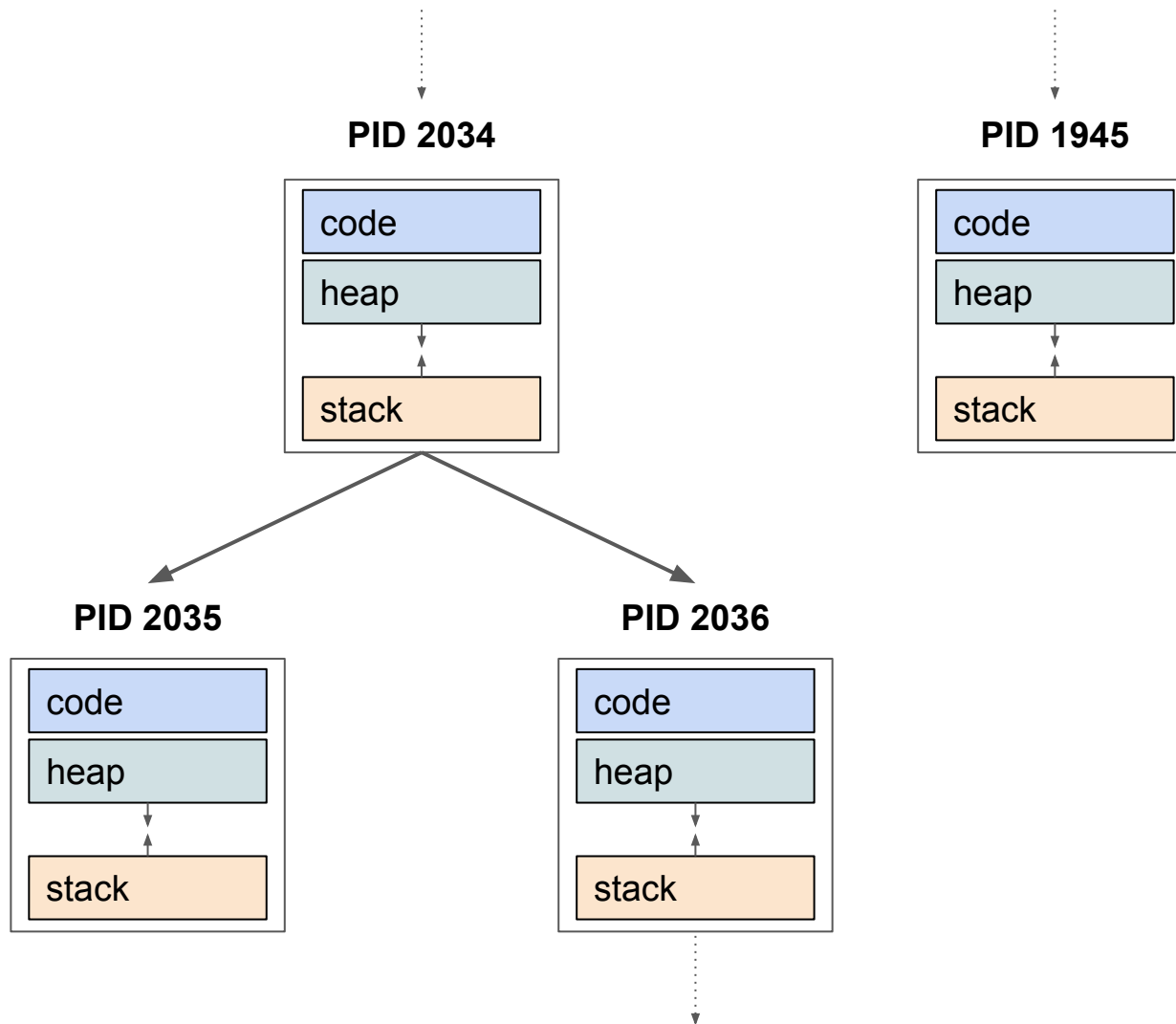
Universidade do Minho

# Processo

- Um processo tem associado a si um espaço de endereçamento constituído por: Código do programa, *Heap* e *Stack*.
- Cada processo é identificado por um valor inteiro atribuído aquando da sua criação — *PID* (*Process Identifier*).

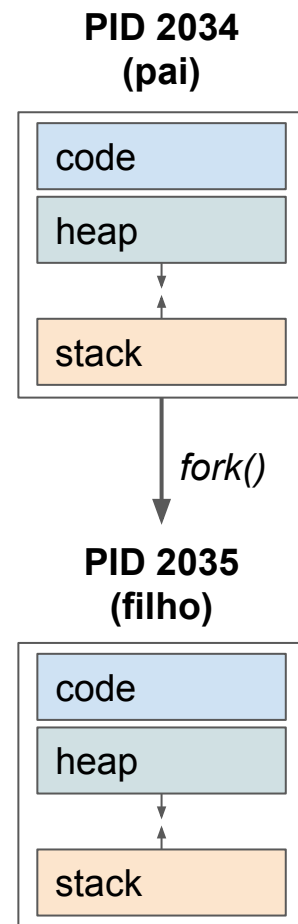


# Hierarquia de processos



# Criação de processos

- O processo-pai invoca a chamada ao sistema **fork** para criar um processo-filho.
- O processo-filho é um duplicado do processo-pai.
  - O processo-filho difere do processo-pai no seu PID
  - Cópia idêntica do espaço de endereçamento
  - Registos de CPU
  - Recursos abertos pelo processo-pai (ficheiros, ...)
- Ambos os processos procedem a sua execução concorrentemente.
- Processos-filho podem criar outros processos.



# Criação de processos — Chamada ao sistema

```
pid_t fork(void);
```

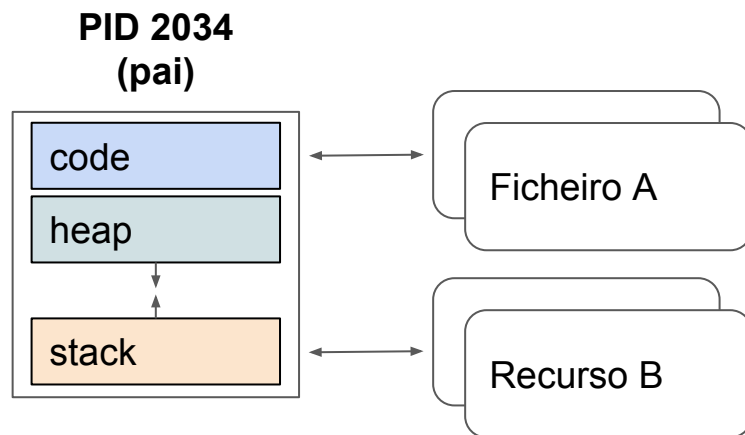
Cria um processo-filho a partir do processo atual.

Retorna, em caso de sucesso:

- **0 identificador de processo (PID)** do processo-filho ao processo-pai
- O valor **0** ao processo-filho

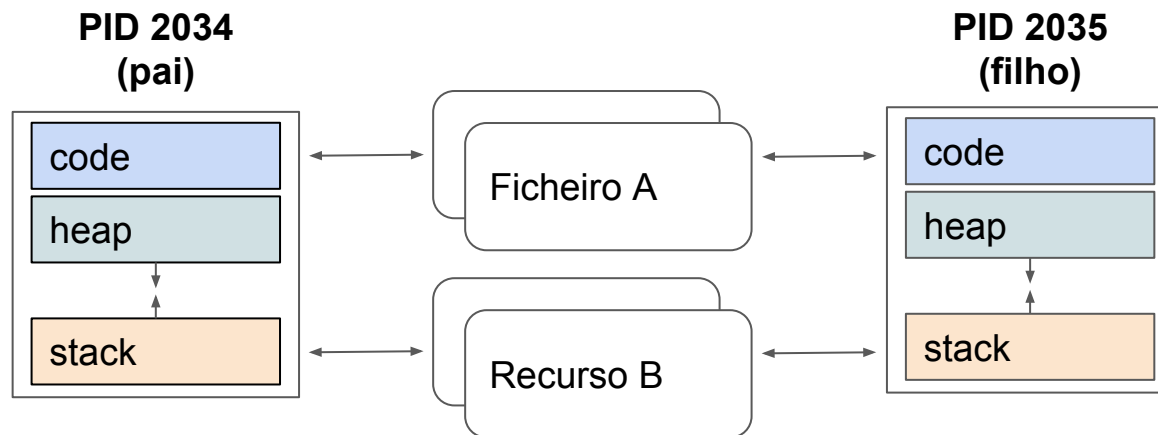
Retorna **-1** em caso de erro

# Criação de processos - Exemplo



```
1 int main() {  
2     pid_t pid;  
3  
4     if ((pid = fork()) == 0) {  
5         // código processo-filho  
6     } else {  
7         // código processo-pai  
8     }  
9 }
```

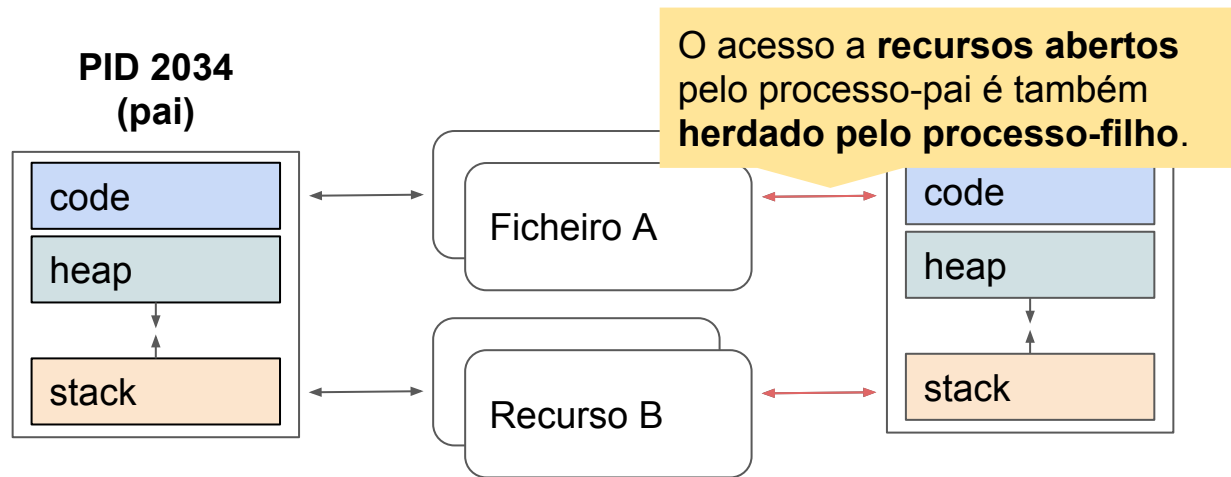
# Criação de processos - Exemplo



```
1 int main() {  
2     pid_t pid;  
3  
4     if ((pid = fork()) == 0) {  
5         // código processo-filho  
6     } else {  
7         // código processo-pai  
8     }  
9 }
```

```
1 int main() {  
2     pid_t pid;  
3  
4     if ((pid = fork()) == 0) {  
5         // código processo-filho  
6     } else {  
7         // código processo-pai  
8     }  
9 }
```

# Criação de processos - Exemplo

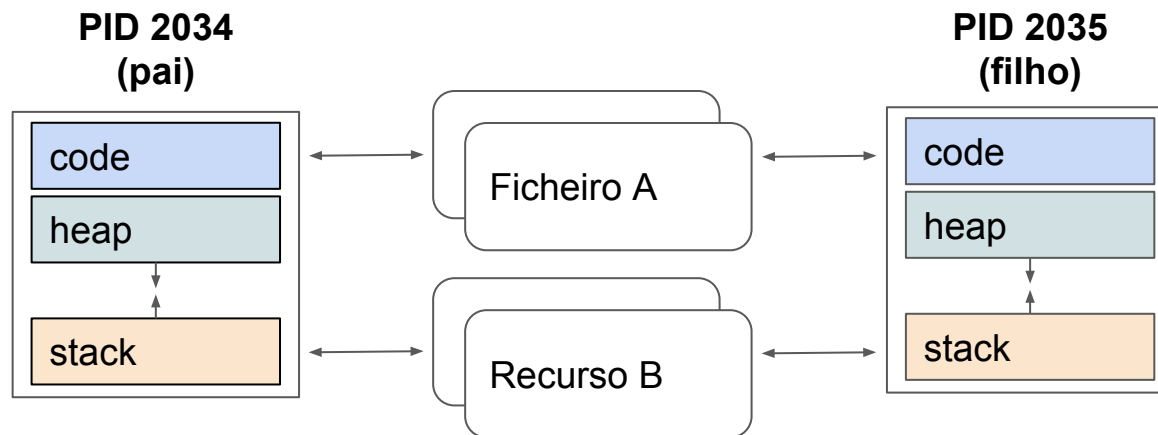


```
1 int main() {
2   pid_t pid;
3
4   if ((pid = fork()) == 0) {
5     // código processo-filho
6   } else {
7     // código processo-pai
8   }
9 }
```

```
1 int main() {
2   pid_t pid;
3
4   if ((pid = fork()) == 0) {
5     // código processo-filho
6   } else {
7     // código processo-pai
8   }
9 }
```



# Criação de processos - Exemplo



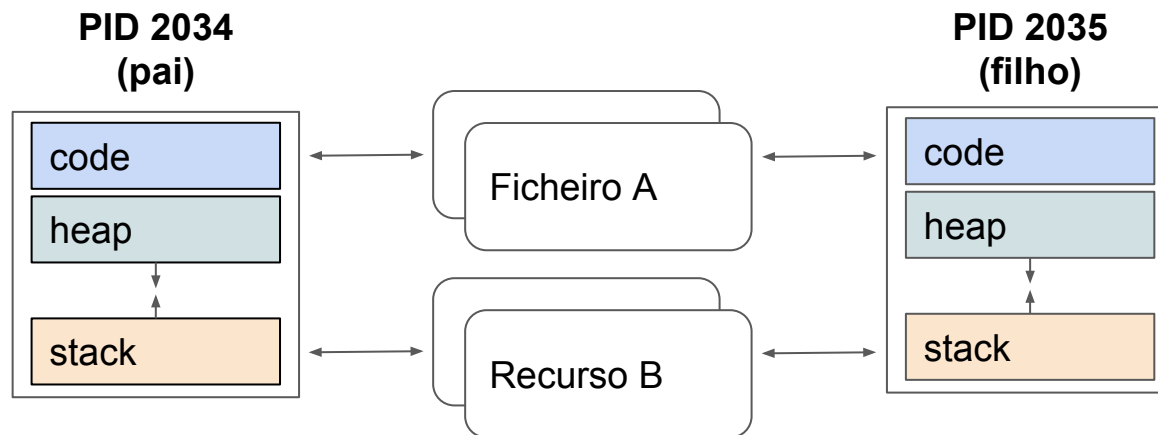
```
1 int main()
2   pid_t pid;
3
4   if ((pid = fork()) == 0) {
5       // código processo-filho
6   } else {
7       // código processo-pai
8   }
9 }
```

**fork() retorna o PID do filho (2035) ao processo-pai**

```
1 int main()
2   pid_t pid;
3
4   if ((pid = fork()) == 0) {
5       // código processo-filho
6   } else {
7       // código processo-pai
8   }
9 }
```

**fork() retorna 0 ao processo-filho**

# Criação de processos - Exemplo



```
1 int main() {  
2   pid_t pid;  
3  
4   if ((pid = fork()) == 0) {  
5     // código processo-filho  
6   } else {  
7     // código processo-pai  
8   }  
9 }
```

```
1 int main() {  
2   pid_t pid;  
3  
4   if ((pid = fork()) == 0) {  
5     // código processo-filho  
6   } else {  
7     // código processo-pai  
8   }  
9 }
```

# Terminação de processos

- O processo-filho termina a sua execução através da invocação da função **\_exit**.
- O processo-pai pode aguardar que os processos-filho terminem através da chamada ao sistema **wait/waitpid**.
- O processo-pai pode aguardar por um processo-filho em particular usando a chamada ao sistema **waitpid**.

# Terminação de processos – Chamada ao sistema

```
pid_t wait(int *status);
```

Bloqueia a execução do processo até um processo-filho terminar.

Retorna, em caso de sucesso:

- **O identificador de processo (PID)** do processo-filho que terminou
- O valor do apontador ***status*** é atualizado com o código de terminação do processo-filho.

O processo-pai pode verificar se o processo-filho terminou sem erros através da macro `WIFEXITED(status)`. O código de terminação é representado por apenas 8 bits e pode ser obtido com `WEXITSTATUS(status)`.

# Terminação de processos - Exemplo

**PID 2034**  
(pai)

```
1 int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid = fork()) == 0) {
6         // código processo-filho
7         _exit(0);
8     } else {
9         // código processo-pai
10        pid_t child = wait(&status);
11        printf("filho saiu %d \
12                erro: %d\n", child, status);
13    }
14 }
```

**PID 2035**  
(filho)

```
1 int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid = fork()) == 0) {
6         // código processo-filho
7         _exit(0);
8     } else {
9         // código processo-pai
10        pid_t child = wait(&status);
11        printf("filho saiu %d \
12                erro: %d\n", child, status);
13    }
14 }
```

# Terminação de processos - Exemplo

PID 2034  
(pai)

```
1 int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid = fork()) == 0) {
6         // código processo-filho
7         _exit(0);
8     } else {
9         // código processo-pai
10        pid_t child = wait(&status);
11        printf("filho saiu %d \
12              erro: %d\n", child, status);
13    }
14 }
```

PID 2035  
(filho)

```
1 int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid = fork()) == 0) {
6         // código processo-filho
7         _exit(0);
8     } else {
9         // código processo-pai
10        pid_t child = wait(&status);
11        printf("filho saiu %d \
12              erro: %d\n", child, status);
13    }
14 }
```

# Terminação de processos - Exemplo

PID 2034  
(pai)

PID 2035  
(filho)

```
1 int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid = fork()) == 0) {
6         // código processo-filho
7         _exit(0);
8     } else {
9         // código processo-pai
10        pid_t child = wait(&status);
11        printf("filho saiu %d \
12              erro: %d\n", child, status);
13    }
14 }
```

**wait()** bloqueia o processo-pai até um processo-filho terminar. Retorna PID do processo-filho que terminou.

```
1 int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid = fork()) == 0) {
6         // código processo-filho
7         _exit(0);
8     } else {
9         // código processo-pai
10        pid_t child = wait(&status);
11        printf("filho saiu %d \
12              erro: %d\n", child, status);
13    }
14 }
```

# Terminação de processos - Exemplo

PID 2034  
(pai)

```
1 int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid = fork()) == 0) {
6         // código processo-filho
7         _exit(0);
8     } else {
9         // código processo-pai
10        pid_t child = wait(&status);
11        printf("filho saiu %d \
12              erro: %d\n", child, status);
13    }
14 }
```

PID 2035  
(filho)

```
1 int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid = fork()) == 0) {
6         // código processo-filho
7         _exit(0);
8     } else {
9         // código processo-pai
10        pid_t child = wait(&status);
11        printf("filho saiu %d \
12              erro: %d\n", child, status);
13    }
14 }
```



# Terminação de processos - Exemplo

PID 2034  
(pai)

```
1 int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid = fork()) == 0) {
6         // código processo-filho
7         _exit(0);
8     } else {
9         // código processo-pai
10        pid_t child = wait(&status);
11        printf("filho saiu %d \
12              erro: %d\n", child, status);
13    }
14 }
```

PID 2035  
(filho)

```
1 int main() {
2     pid_t pid;
3     _exit(0);
4     // código processo-filho
5     _exit(0);
6 } else {
7     // código processo-pai
8     pid_t child = wait(&status);
9     printf("filho saiu %d \
10           erro: %d\n", child, status);
11 }
12 }
```

**\_exit()** termina o processo atual com código passado por argumento.

# Terminação de processos - Exemplo

PID 2034  
(pai)

```
1 int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid = fork()) == 0) {
6         // código processo-filho
7         _exit(0);
8     } else {
9         // código processo-pai
10        pid_t child = wait(&status);
11        printf("filho saiu %d \
12              erro: %d\n", child, status);
13    }
14 }
```

A variável **status** é atualizada com o código passado na chamada da função **\_exit()**.

PID 2035  
(filho)

```
1 int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid = fork()) == 0) {
6         // código processo-filho
7         _exit(0);
8     } else {
9         // código processo-pai
10        pid_t child = wait(&status);
11        printf("filho saiu %d \
12              erro: %d\n", child, status);
13    }
14 }
```

# Terminação de processos - Exemplo

**PID 2034**  
**(pai)**

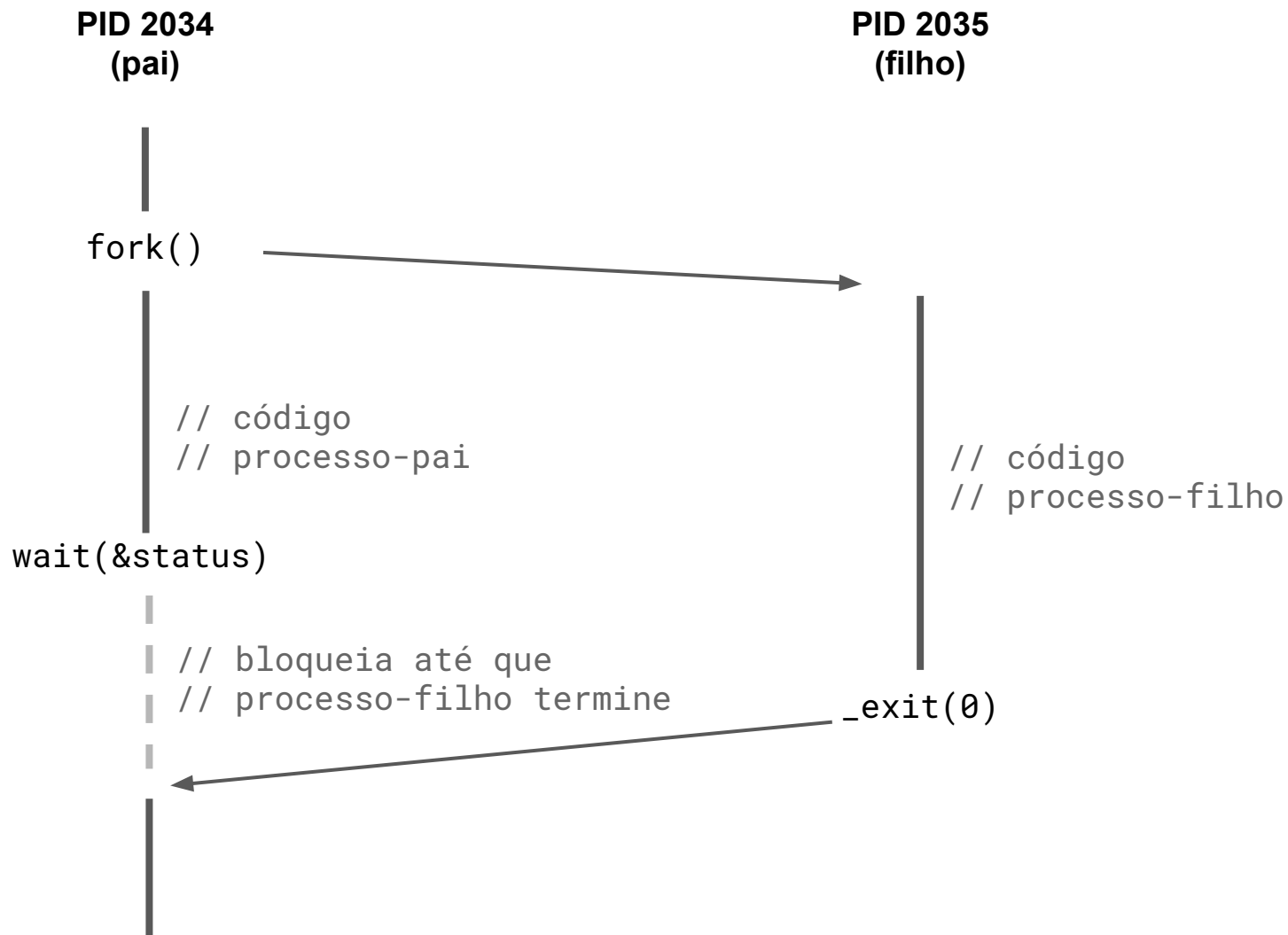
```
1 int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid = fork()) == 0) {
6         // código processo-filho
7         _exit(0);
8     } else {
9         // código processo-pai
10        pid_t child = wait(&status);
11        printf("filho saiu %d \
12              erro: %d\n", child, status);
13    }
14 }
```

O processo pai continua a execução após a terminação do processo-filho.

**PID 2035**  
**(filho)**

```
1 int main() {
2     pid_t pid;
3     int status;
4
5     if ((pid = fork()) == 0) {
6         // código processo-filho
7         _exit(0);
8     } else {
9         // código processo-pai
10        pid_t child = wait(&status);
11        printf("filho saiu %d \
12              erro: %d\n", child, status);
13    }
14 }
```

# Terminação de processos - Exemplo

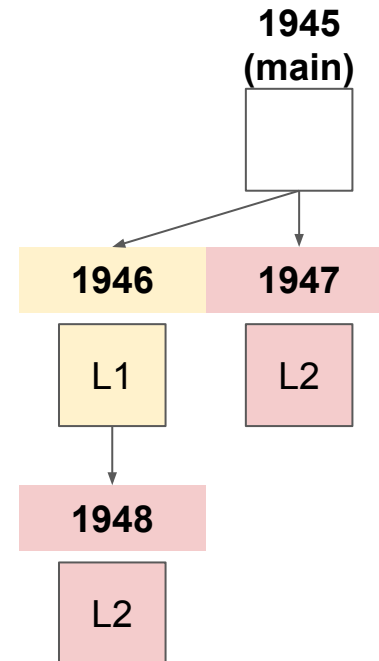


# Notas

- Se o processo-pai terminar antes do processo-filho, o processo-filho torna-se órfão.
  - Neste caso, o processo-filho é adotado pelo processo **init**, cujo identificador de processo é **1**.
- Um processo diz-se no estado *zombie* se este terminou e o seu processo-pai ainda não o recolheu a correspondente informação (usando **wait/waitpid**).
- As chamadas ao sistema **getpid** e **getppid** podem ser usadas para retornar o PID do processo atual e o PID do processo-pai, respetivamente.

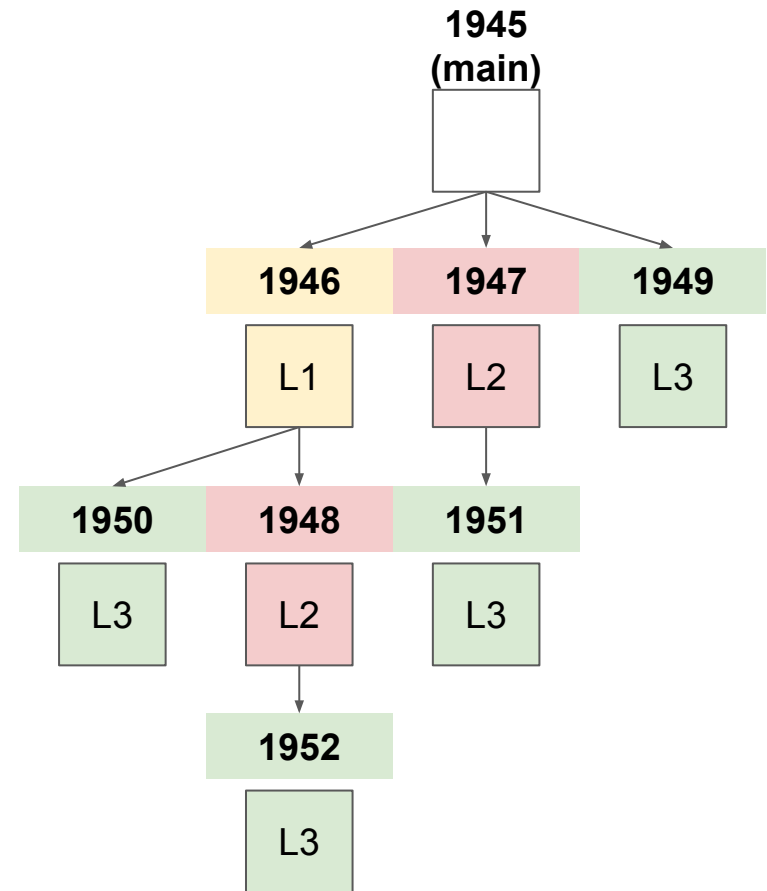
# Exemplos de *fork*

```
1 int main() {  
2     fork();  
3     // Level 1  
4     fork();  
5     // Level 2  
6 }
```



# Exemplos de *fork*

```
1 int main() {  
2   fork();  
3   // Level 1  
4   fork();  
5   // Level 2  
6   fork();  
7   // Level 3  
8 }
```



# Exemplos de *fork*

```
1 int main() {
2     printf("L0\n");
3
4     if(fork() == 0) {
5         printf("L1\n");
6
7         if(fork() == 0) {
8             println("L2\n");
9             fork();
10            println("Forked\n");
11        }
12    }
13
14    printf("Bye\n");
15 }
16 }
```

Qual é o output deste processo?

1945  
(main)

L0

1946

L1

1947

L2

1948



# Sumário

## **fork**

- Cria um processo duplicado do processo atual, incluindo o acesso a recursos abertos.
- Resulta em dois processos: pai e filho.
- Ambos continuam a execução do mesmo ponto.

## **\_exit**

- Termina o programa de forma ordeira, libertando recursos associados.
- Desbloqueia o processo-pai, caso este esteja bloqueado a aguardar que o processo-filho termine.

## **wait/waitpid**

- Usado pelo processo-pai para aguardar que os processos-filho terminem.

# Material de apoio

- José Alves Marques, Paulo Ferreira, Carlos Nuno da Cruz Ribeiro, Luís Veiga e Rodrigo Rodrigues, Sistemas Operativos., Oct 2012, FCA, ISBN 978-972-722-756-3.
- <http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/create.html>