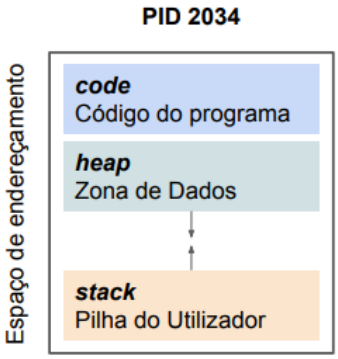
Anotações - Guião 2

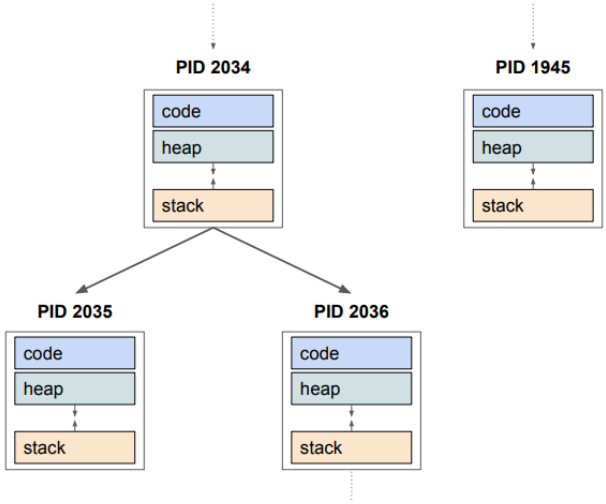
Data: 05/03/2023
Tags: #SoftwareEngineering #SO #C
Slides: slides_guiao2.pdf

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 (difere no seu PID, cópia idêntica do espaço de endereçamento)
- Ambos os processos procedem a sua execução concorrentemente

Chamada ao sistema

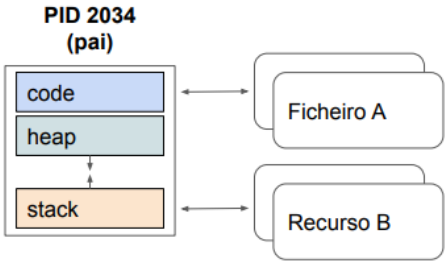
```
pid_t fork(void);
```

> cria um processo-filho a partir do processo atual.

Retorna, em caso de sucesso:

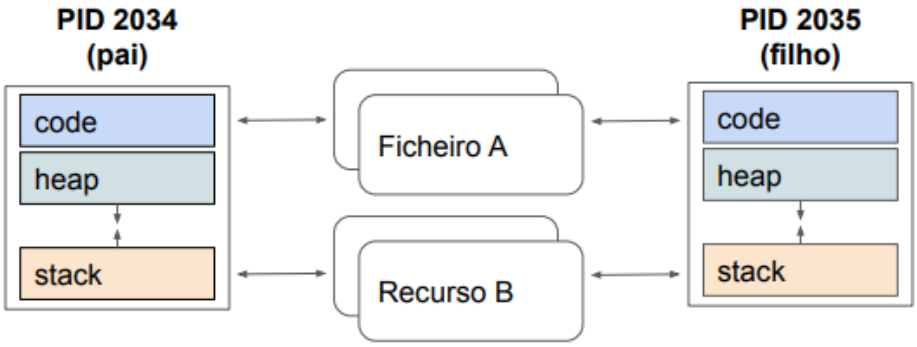
- o identificador do processo (PID) do processo-filho ao processo-pai;
- o valor 0 ao processo filho.
  Retorna -1 em caso de erro.

Exemplo




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

  if ((pid = fork()) == 0) {
    // código processo-filho
  } else {
    // código processo-pai
  }
}
```



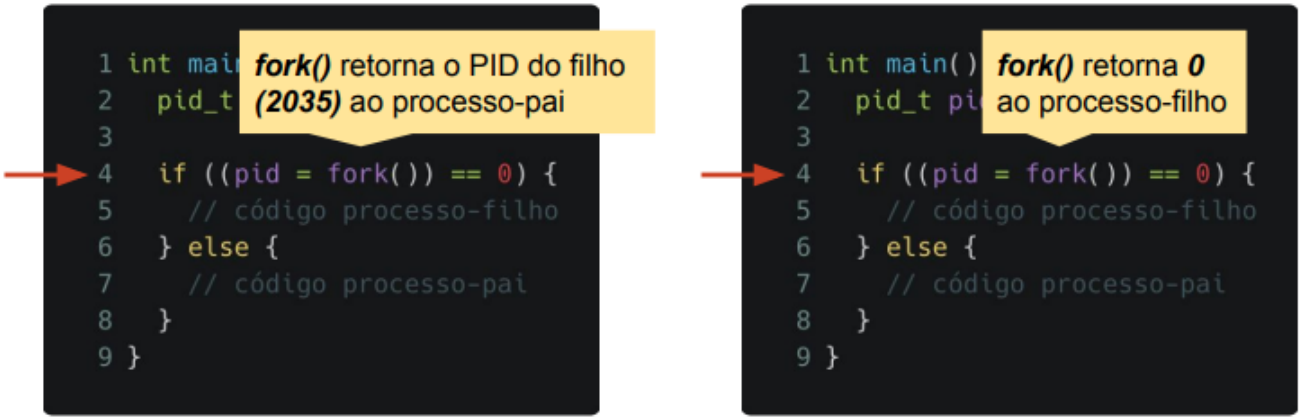

```c
int main() {
  pid_t pid;

  if ((pid = fork()) == 0) {
    // código processo-filho
  } else {
    // código processo-pai
  }
}
```

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

  if ((pid = fork()) == 0) {
    // código processo-filho
  } else {
    // código processo-pai
  }
}
```

O acesso a recursos abertos pelo processo-pai é também herdado pelo processo-filho.



Portanto, ambos executan o codigo concorrentemente, mas recebem diferentes outputs do `fork()`.

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**.

Chamada ao sistema

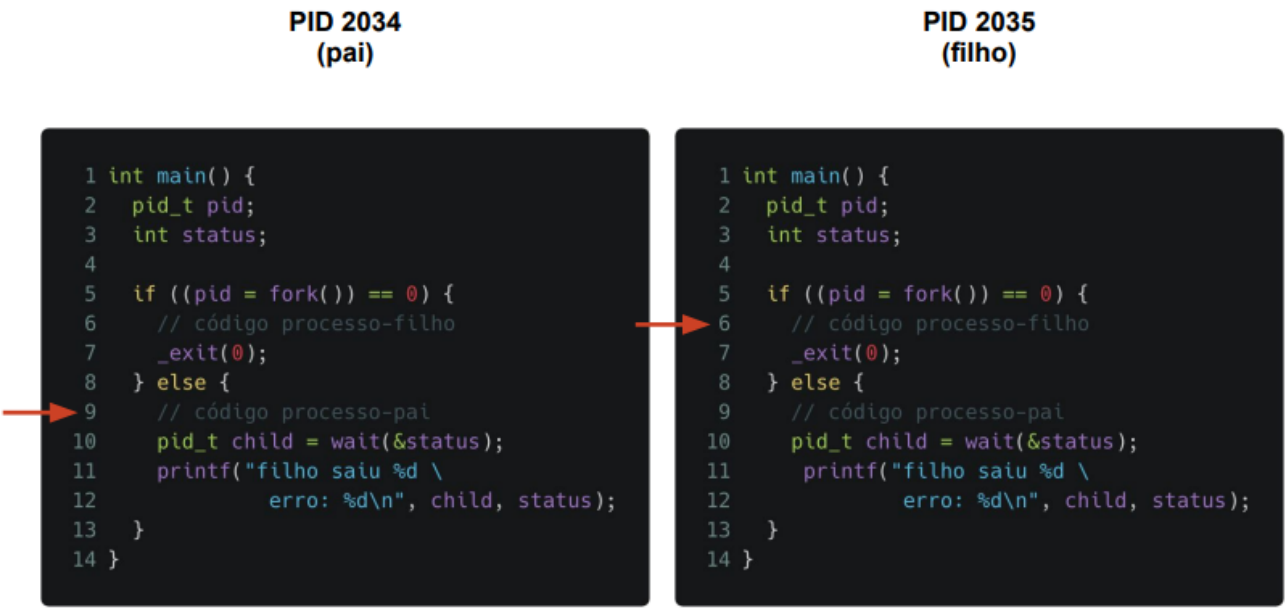
```
pid_t wait(int *status);
```

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

Retorna em caso de sucesso o PID do processo-filho que terminou e o valor do apontador *status* é atualizado com o código de terminação do processo-filho.

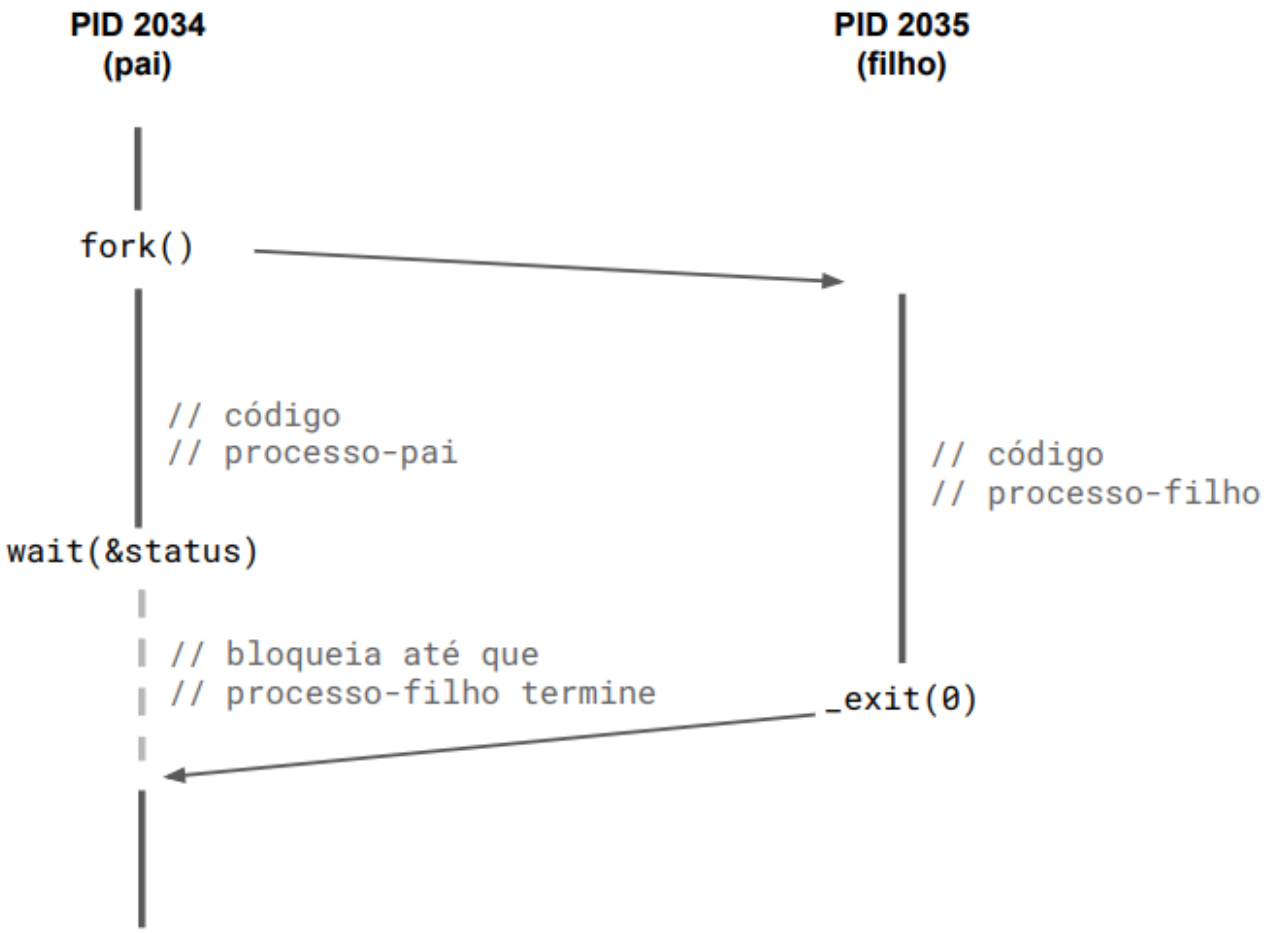
> O processo-pai pode verificar se o processo-filho terminou sem erros através da macro WIFEXITED(status). O código de terminação é representado por apenas 8 bits e pode ser obtido com WEXITSTATUS(status).

Exemplo



-> `wait()` bloqueia o processo-pai até um processo-filho terminar. Retorna PID do processo-filho que terminou;
-> `_exit()` termina o processo atual com código passado por argumento;

-> A variável *status* é atualizada com o código passado na chamada da função `_exit()`.
-> O processo-pai continua a execução após a terminação do processo-filho.

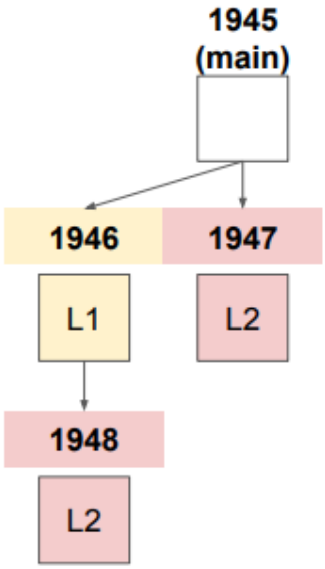


Notas

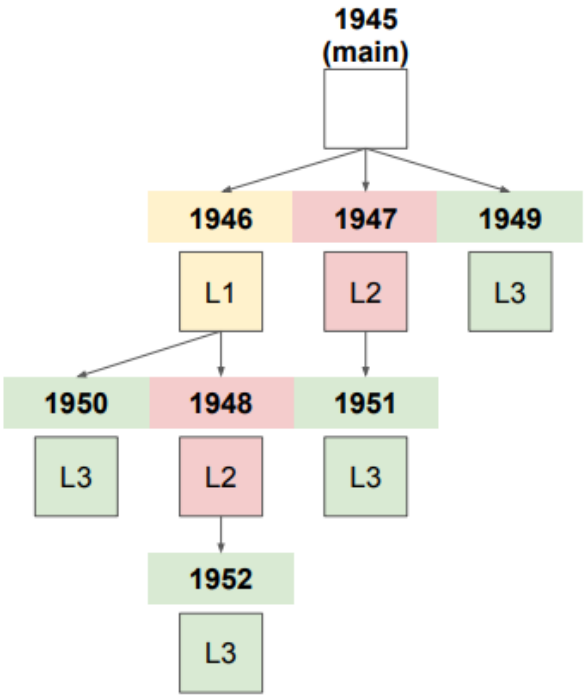
- Se o processo-pai terminar antes do processo-filho, o processo-filho torna-se órfão (neste caso, o processo-filho é adotado pelo processo **init**, cujo identificador de processo é **1**).
- Um processo diz-se no estado *zombie* se este terminou e o seu processo-pai ainda não recolhei a correspondente informação (usado 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.

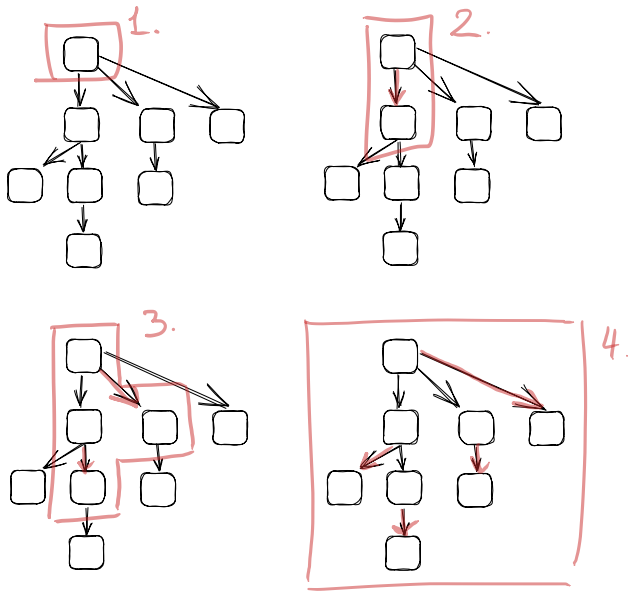
Exemplos de fork

```c
int main() {
  fork();
  // Level 1
  fork();
  // Level 2
}
```




```c
int main() {
  fork();
  // Level 1
  fork();
  // Level 2
  fork();
  // Level 3
}
```





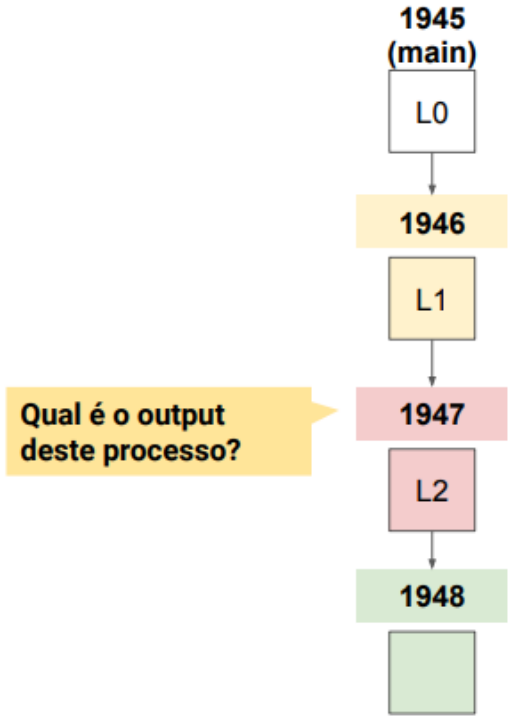

```c
int main() {
  printf("L0\n");

  if(fork() == 0) {
    printf("L1\n");

    if(fork() == 0) {
      println("L2\n");
      fork();
      println("Forked\n");
    }

  }

  printf("Bye\n");
}
```



Exemplo

```c
    for (size_t i = 0; i < ROWS; i++) {
    if ((pid = fork()) == 0) {
        for (size_t j = 0; j < COLUMNS; j++) {
            if (matrix[i][j] == num) {
                _exit(1);
            }
        }
        _exit(0);
    }
}

int ocurrences = 0;
    int status;
    for(size_t i = 0; i < ROWS; i++) {
            int pid = wait(&status);
```

```
                if(WEXITSTATUS(status) == 1)
                        ocurrences++;
        }
```

> Procura um número numa matriz