Insper

Sistemas Hardware-Software

Carregamento de Programas

Engenharia

Fabio Lubacheski
Maciel Vidal
Igor Montagner

Criação de processos

Criamos processos usando a chamada de sistema *fork*

```
pid_t fork();
```

O fork cria um clone do processo atual e retorna duas vezes

No processo original (pai) `fork` retorna o pid do filho

O pid do pai é obtido chamando

```
pid_t getpid();
```

No processo filho `fork` retorna o valor 0.
O pid do filho é obtido usando

```
pid_t getpid();
```

O pid do pai pode ser obtido usando a chamada

```
pid_t getppid();
```

Valor de retorno

- Um processo pode esperar pelo fim de outro processo filho usando as funções

```
pid_t wait(int *wstatus);
pid_t waitpid(pid_t pid, int *wstatus, int options);
```

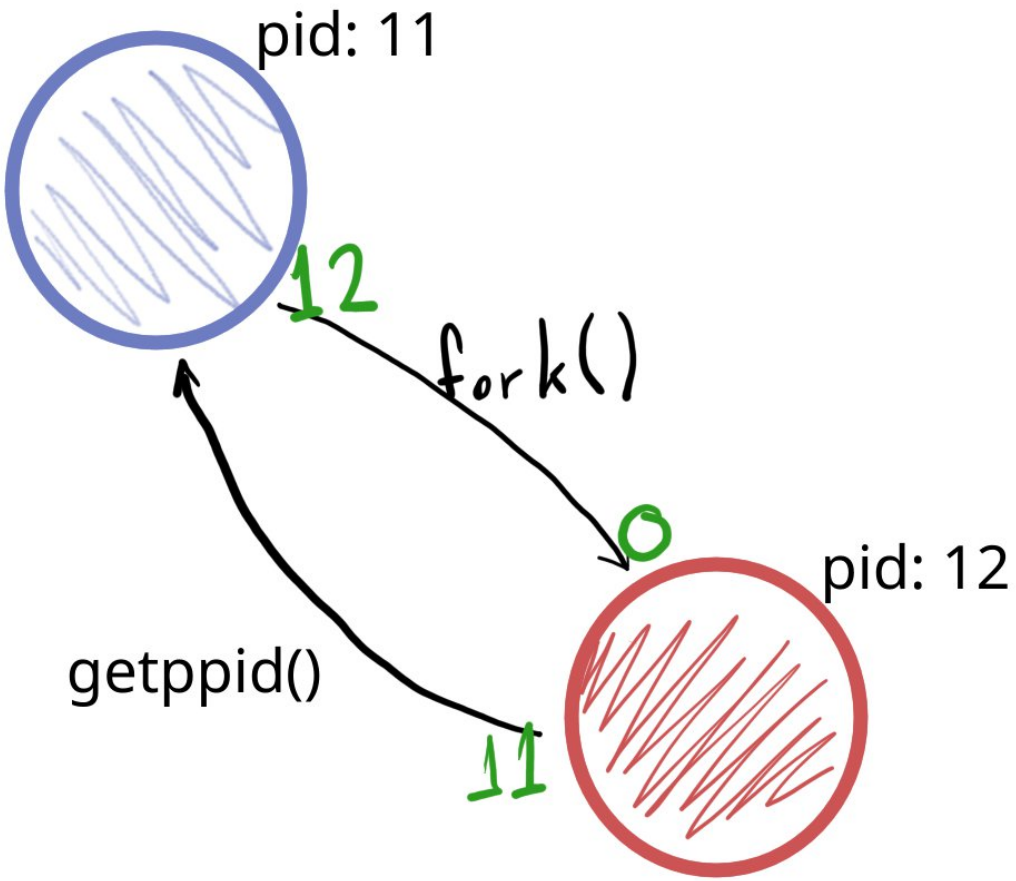
- A primeira espera qualquer um dos filhos, enquanto a segunda espera um filho (ou grupo de filhos) específico.
- Ambas bloqueiam até que um processo filho termine e retornam o pid do processo que acabou de terminar.
- O valor de retorno do processo é retornado via o ponteiro `wstatus`.

Correção

A chamada wait

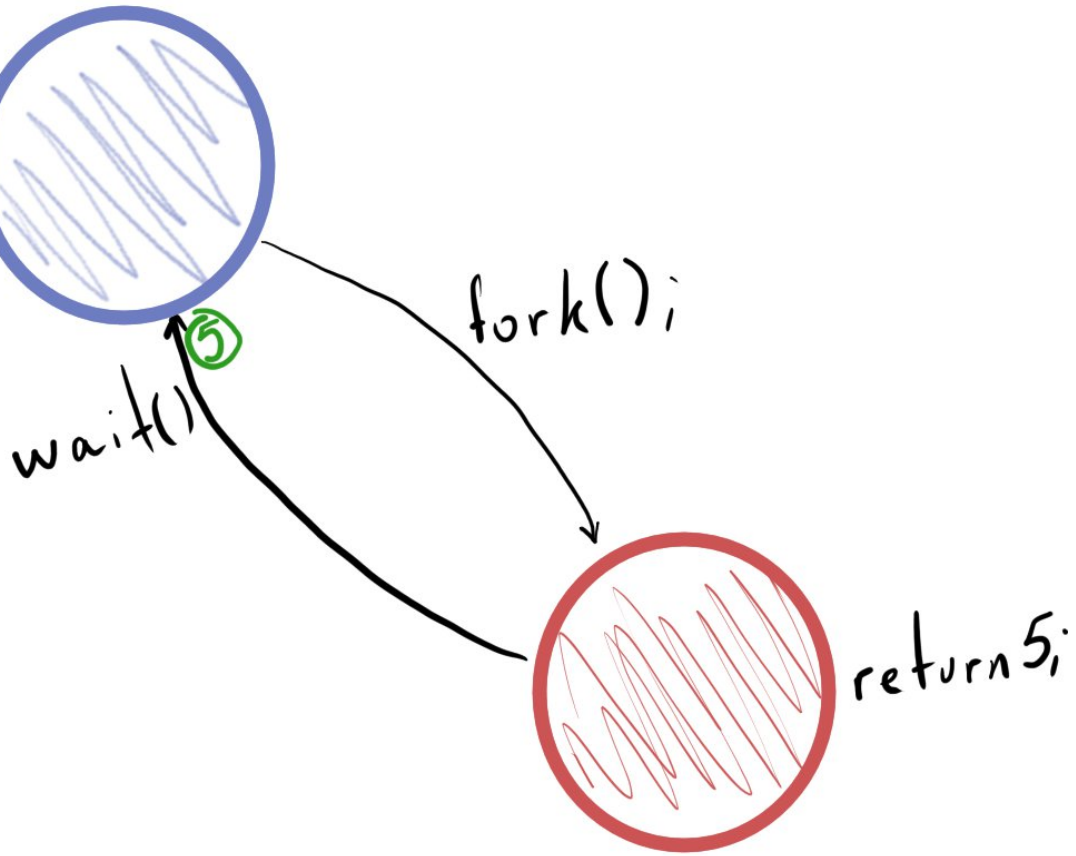
1. Criação de processos
2. Identificação de término de processos
3. Utilização do manual para dúvidas sobre as chamadas

Parentesco de processos



Insper

Parentesco de processos – II



Testando o entendimento do fork()

Quais das seguintes saídas são possíveis no exemplo abaixo ?

```c
int main() {
    printf("L0\n");
    if(fork() == 0) {
        printf("L1\n");
        if(fork() == 0) {
            printf("L2\n");
        }
    }
    printf("Bye\n");
}
```

	Seq 1:	Seq 2:
	L0 L1 Bye Bye Bye L2	L0 Bye L1 L2 Bye Bye
A.	Não	Não
B.	Não	Sim
C.	Sim	Não
D.	Sim	Sim
E.	Não sei :(...	

Atividade prática

Argumentos: main(int argc, char *argv[]) (20 minutos)

1. Recepção de argumentos por programas
2. Conversão de strings para inteiros

A chamada execvp

```
int execvp(const char *file, char *const argv[]);
```

A chamada **execvp** faz duas coisas:

1. Carrega um programa na memória dentro do contexto do processo atual
2. Inicia esse programa, preenchendo os argumentos do main

O programa que estava em execução antes do `execvp` é completamente destruído.

Exemplo de uso - execvp

```c
#include <unistd.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    char prog[] = "ls";
    // a lista de argumentos sempre começa com o nome do
    // programa e termina com NULL
    char *args[] = {"ls", "-l", "-a", NULL};

    execvp(prog, args);
    printf("Fim do exec!\n");

    return 0;
}
```

Exemplo de uso - execvp

```c
#include <unistd.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    char prog[] = "ls";
    // a lista de argumentos sempre começa com o nome do
    // programa e termina com NULL
    char *args[] = {"ls", "-l", "-a", NULL};

    execvp(prog, args);
    printf("Fim do exec!\n");

    return 0;
}
```

Essa linha só roda se o **execvp** falhar!

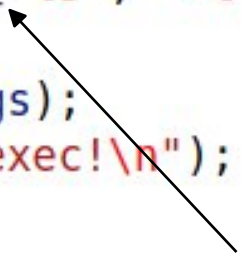
Exemplo de uso - execvp

```c
#include <unistd.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    char prog[] = "ls";
    // a lista de argumentos sempre começa com o nome do
    // programa e termina com NULL
    char *args[] = {"ls", "-l", "-a", NULL};

    execvp(prog, args);
    printf("Fim do exec!\n");

    return 0;
}
```



Argumento `char *argv[]` do main!
Também seta `argc = 3`, pois tem 3 strings!

Exemplo de uso - execvp e fork

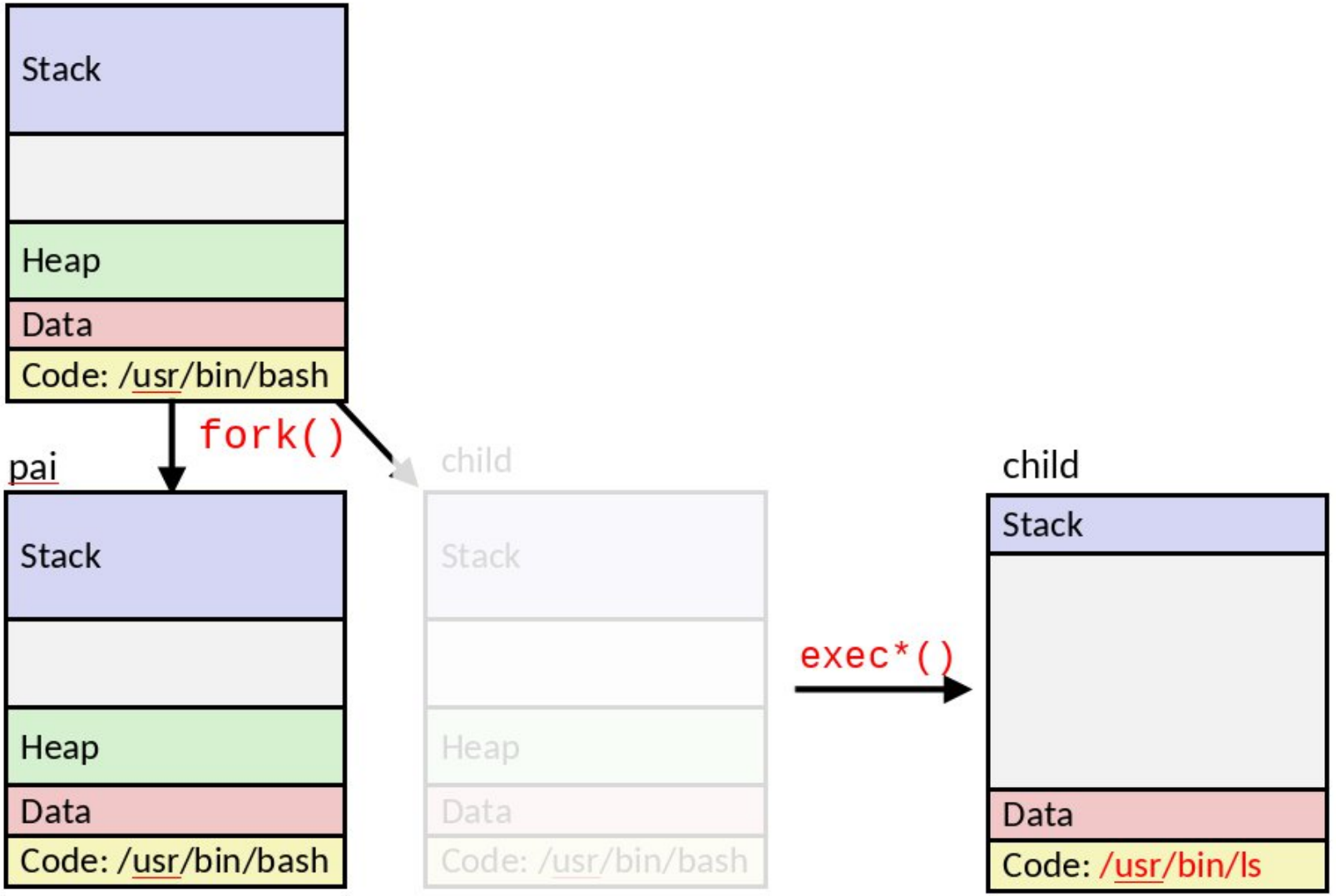
```c
int main() {
    if (fork() == 0) {
        char *args[] = {"ls", "-l", "-a", NULL};
        execvp(args[0], args);
        printf("fim filho\n");
        return 10;
    } else {
        int status;
        printf("Esperando o filho acabar!\n");
        wait(&status);
        if(WIFEXITED(status))
            printf("retorno = %d \n",WEXITSTATUS(status));
        printf("fim do pai\n");
    }

    return 0;
}
```

Esse trecho não é executado

O retorno obtido é da execução o comando ls

Exemplo de uso - execvp e fork



Atividade prática

A chamada exec (30 minutos)

1. Carregamento de programas
2. Passagem de argumentos
3. Coleta de resultados de um programa

Insper

www.insper.edu.br