

# INE5412 Sistemas Operacionais I

---

Processos : chamadas de sistema

# Chamada de sistema EXEC

---

- Resposta : chamada de sistema exec invoca outros programas.

try\_exec.c

```
int main(void)
{
    puts("before execl ...");

    // invoke "/bin/lis"
    execl("/bin/lis", "/bin/lis", NULL);

    fprintf(stderr, "Command not found\n");
    return 0;
}
```

before execl ...

List of arguments


# Chamada de sistema EXEC

---

- “**/bin/ls**” é executado.

try\_exec.c

```
int main(void)
{
    puts("before execl ...");

    // invoke "/bin/ls"
     execl("/bin/ls", "/bin/ls", NULL);

    fprintf(stderr, "Command not found\n");
    return 0;
}
```

before execl ...  
try\_exec try\_exec.c

—

# Chamada de sistema EXEC

---

- O processo é **terminado** depois de uma invocação bem sucedida de `execl()`.

`try_exec.c`

```
int main(void)
{
    puts("before execl ...");

    // invoke "/bin/ls"
    execl("/bin/ls", "/bin/ls", NULL);

    fprintf(stderr, "Command not found\n");
    return 0;
}
```

before execl ...  
exec exec.c

—

# Chamada de sistema EXEC

---

- Se a invocação de `execl()` **falhar**, a execução do processo atual continua.

`try_exec.c`

```
int main(void)
{
    puts("before execl ...");

    // invoke "/bin/ls"
    execl("/bin/abc", "/bin/abc", NULL);

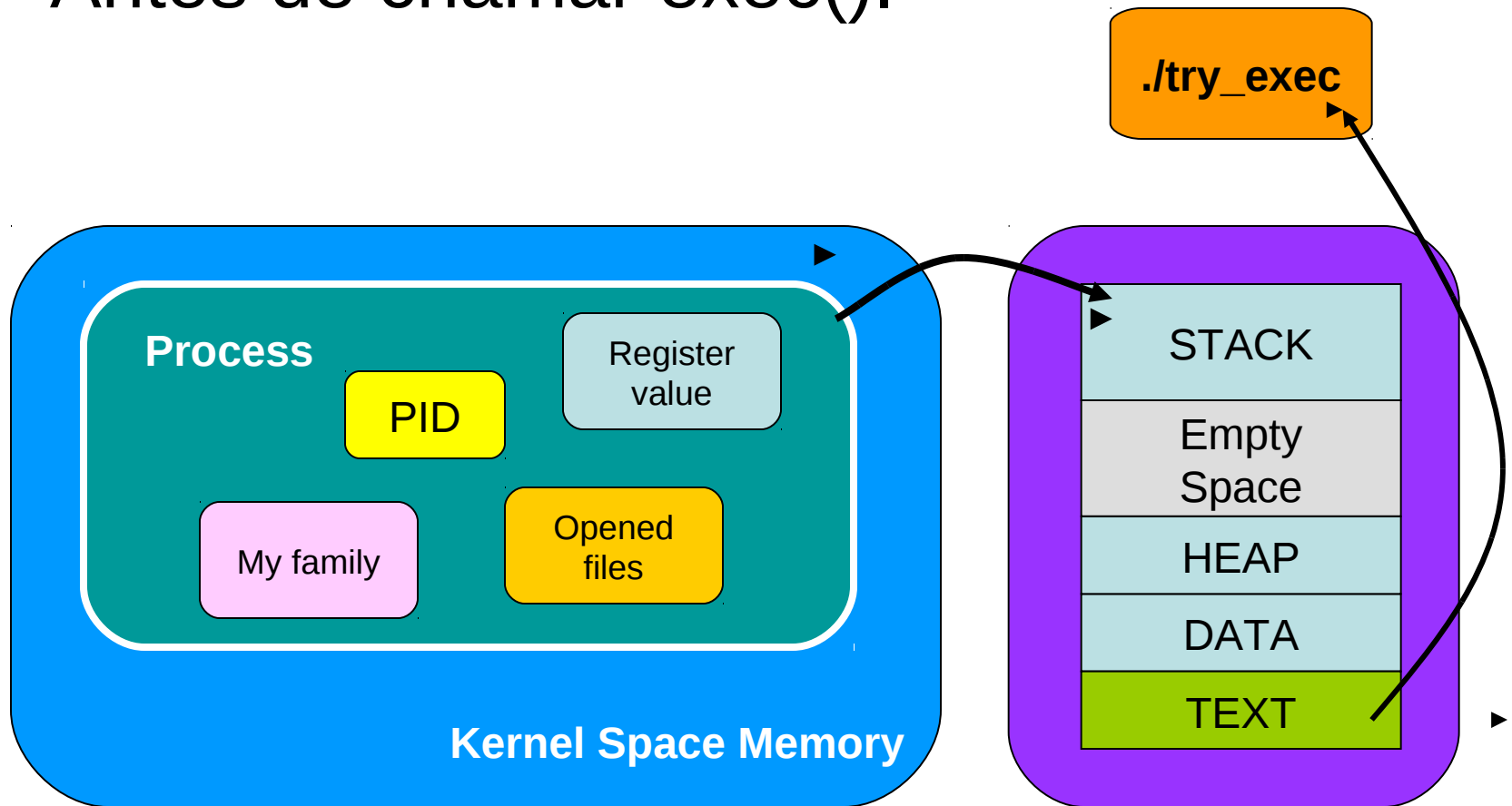
    fprintf(stderr, "Command not found\n");
    return 0;
}
```

before execl ...  
Command not found

# O que acontece?

---

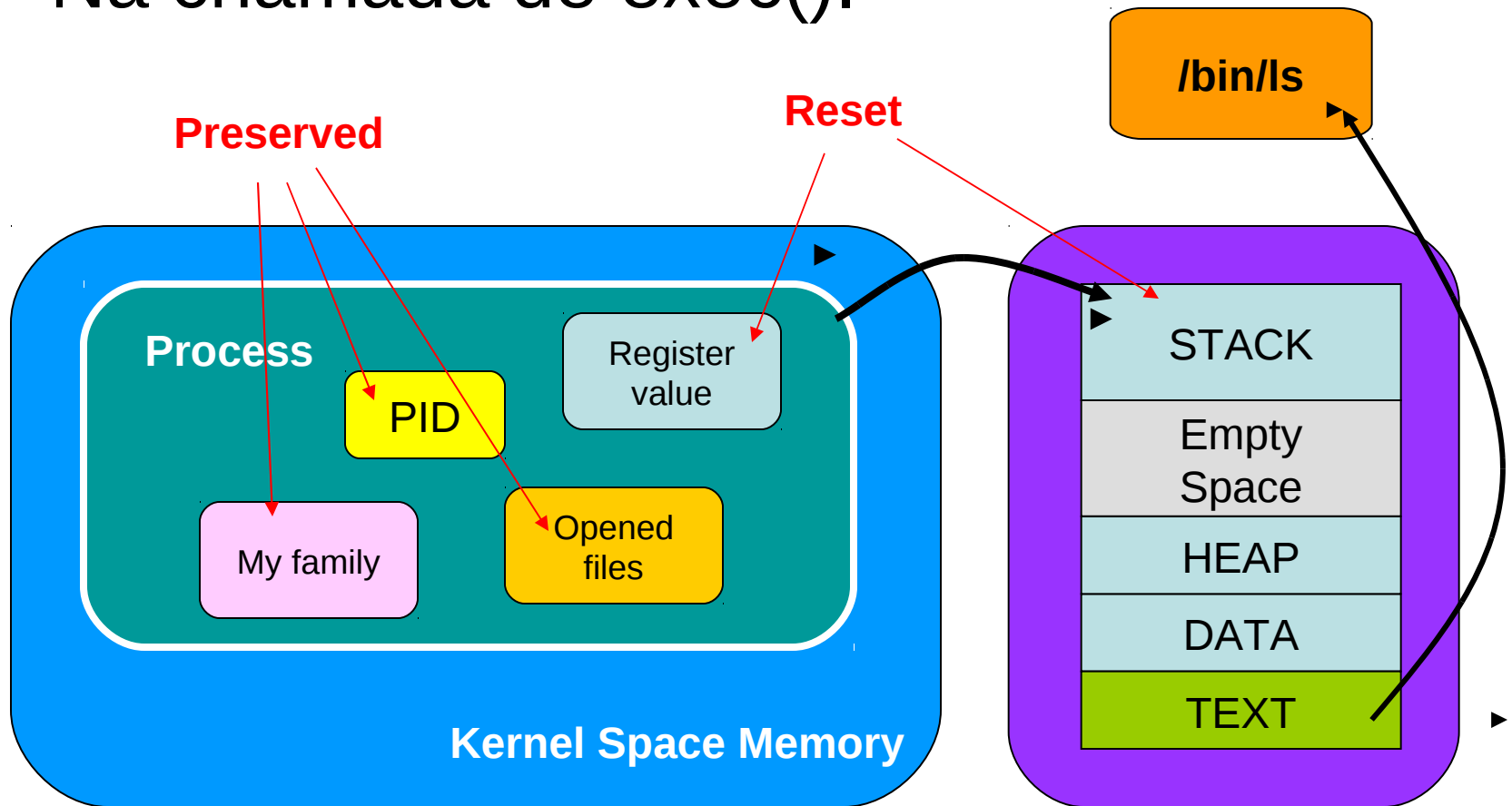
- Antes de chamar `exec()`.



# O que acontece?

- Na chamada de `exec()`.

Change program code



# O que acontece?

---

- A chamada exec modifica a **imagem do processo** que está invocando a função .
  - Os segmentos pilha, heap, dados, código são **sobre-escritos** pelo novo programa.
  - Todos os valores de registradores, incluindo PC, são **resetados**.
  - Partes do processo não são modificadas.
    - PID, ou seja., o processo ainda é o mesmo;
    - Relacionamentos do processo ;
    - Lista de arquivos abertos.

Quer dizer que o processo esta executando um programa diferente.

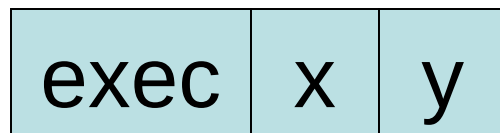
- É possível perceber a diferença entre processo e programa?
- Porque o processo continua executando o **program chamador** quando a chamada exec falha?



# Família EXEC

---

- Formas de chamar exec
  - execl, execlp, execl, execv, execvp, and execve.
- Convention de chamada:



para **x**,

- **l** – fornece uma **list** de argumentos, terminada por NULL;
- **v** – fornece um **vector** (array) de argumentos.

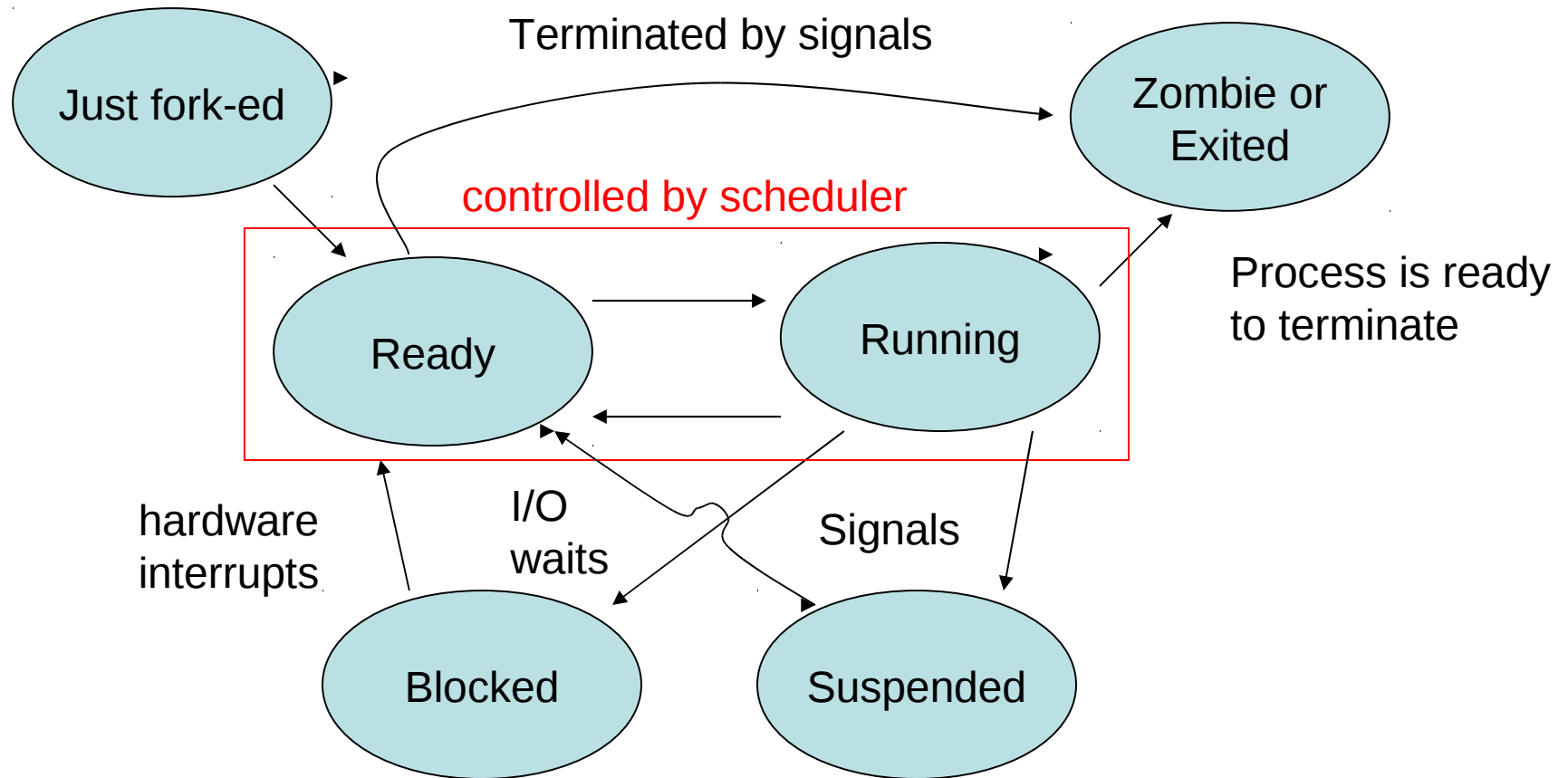
para **y**,

- **P** – variável de ambiente **PATH**\* será buscada;
- e** – um vetor de **variáveis de ambiente**\* deve ser fornecida

\* notes suplementares.

# Process Life Cycle

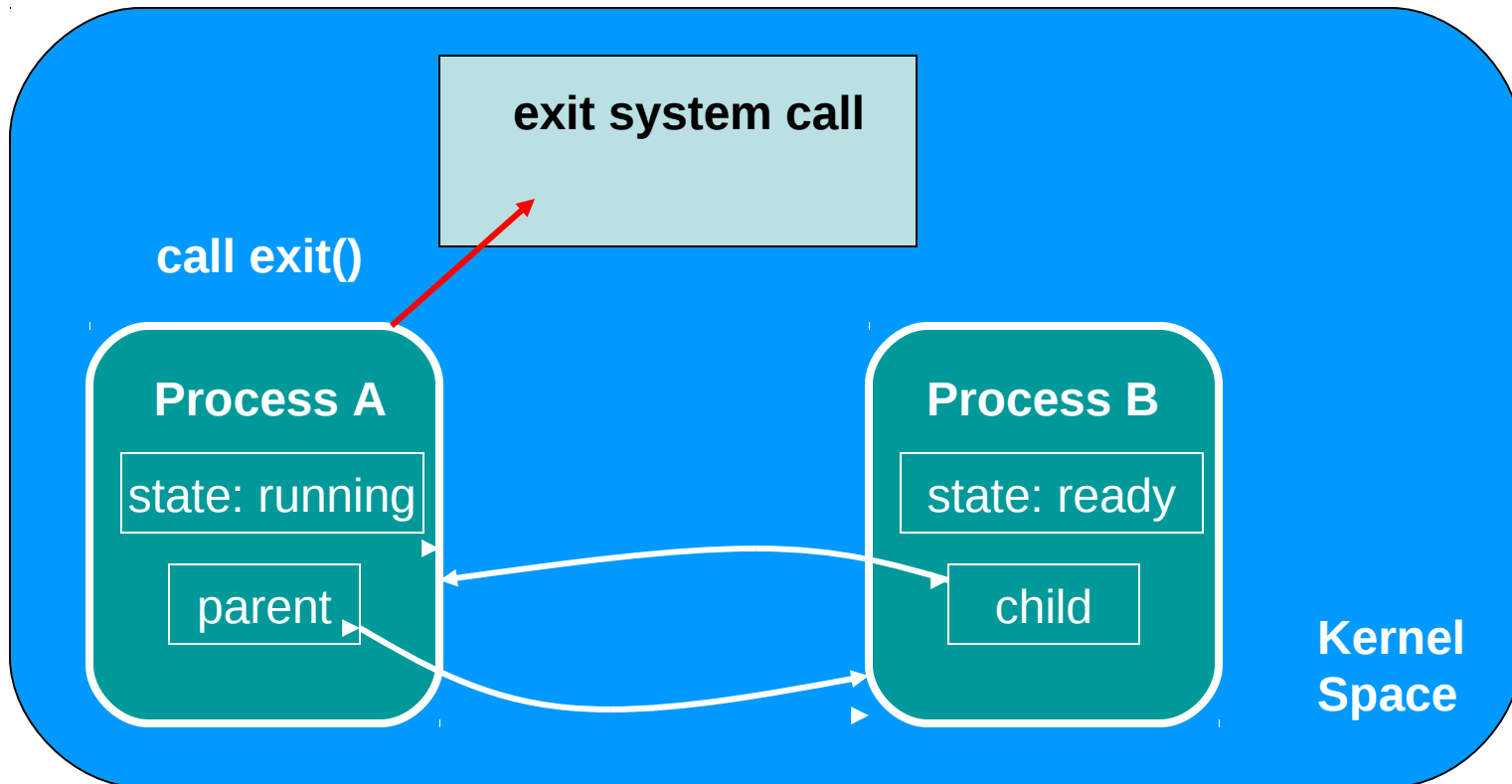
---



# Estado: Zombie



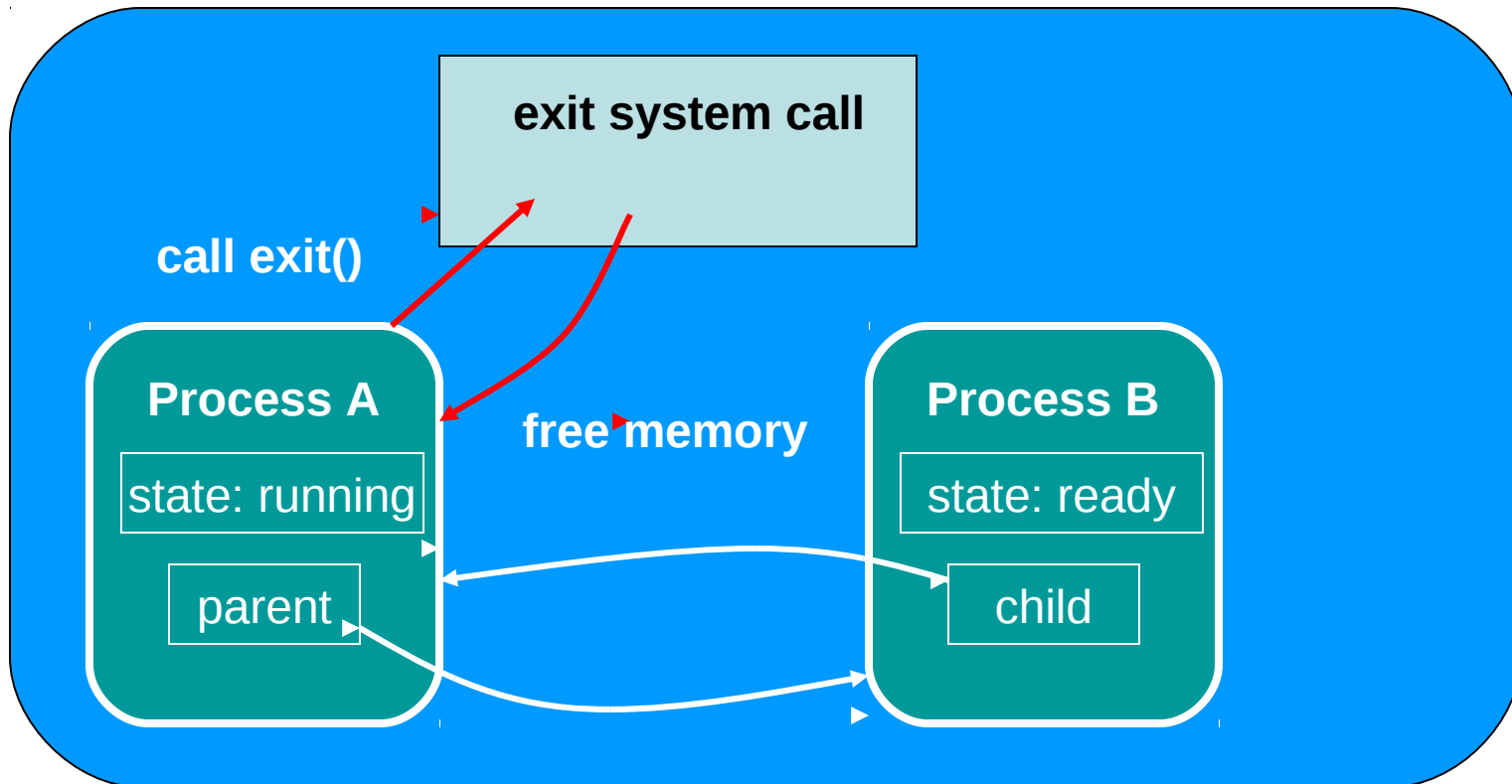
- O processo termina quando a chamada de sistema `exit()` é invocada.



# Estado: Zombie



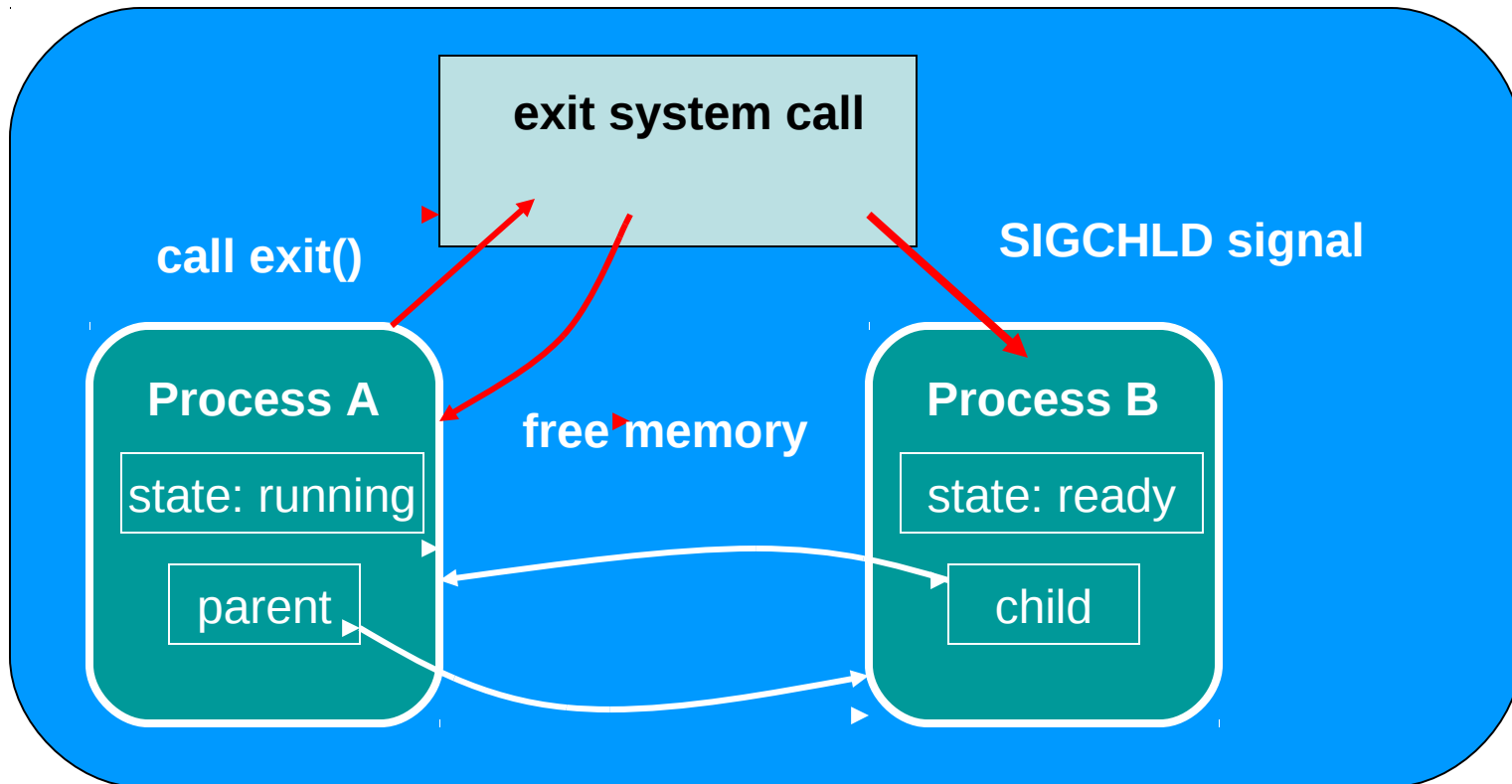
- Recursos alocados para o processo serão liberados na chamada `exit()`.



# Estado: Zombie



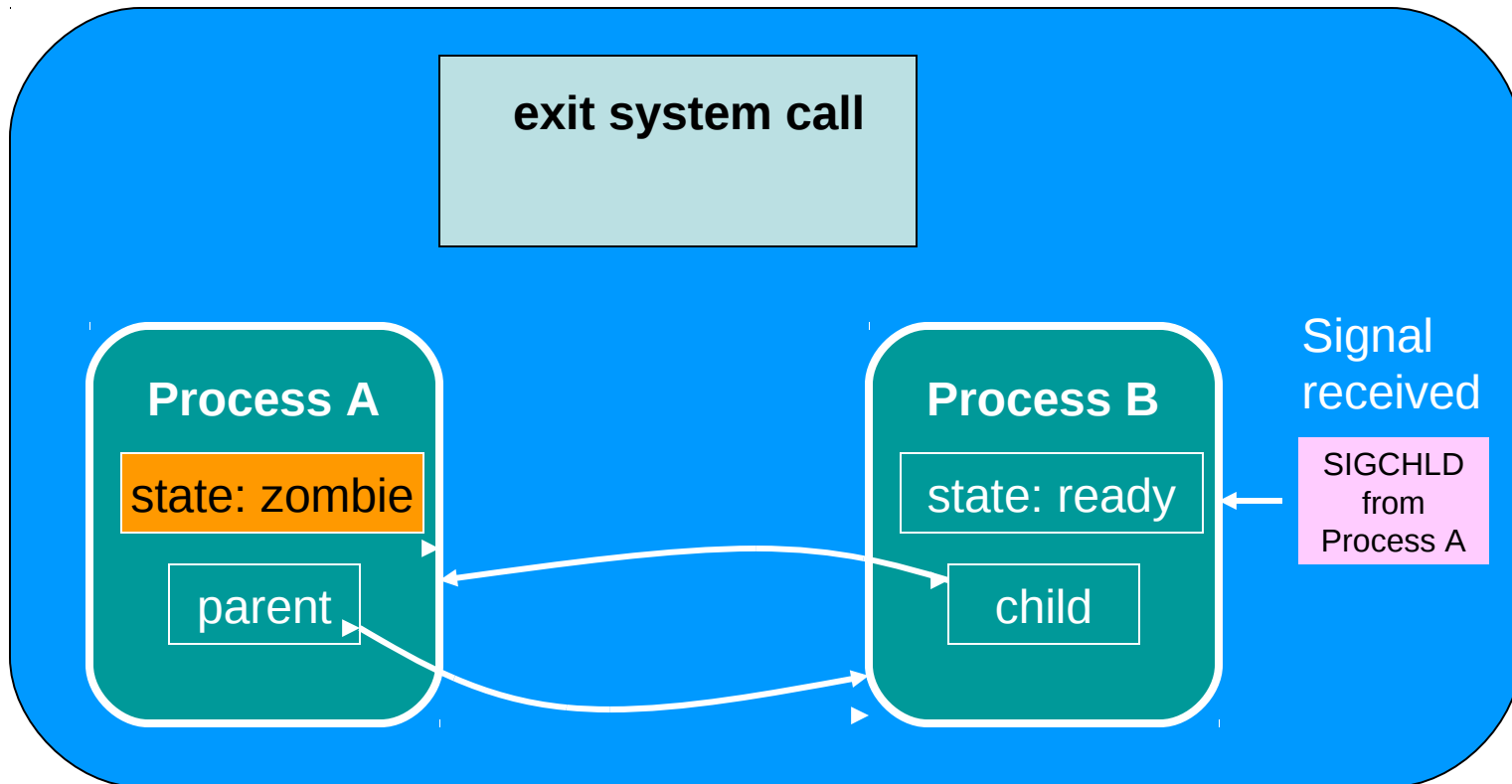
- O pai do processo é notificado pelo sinal SIGCHLD.



# Estado: Zombie



- O pai do processo é notificado pelo sinal SIGCHLD.



# Estado: Zombie

---

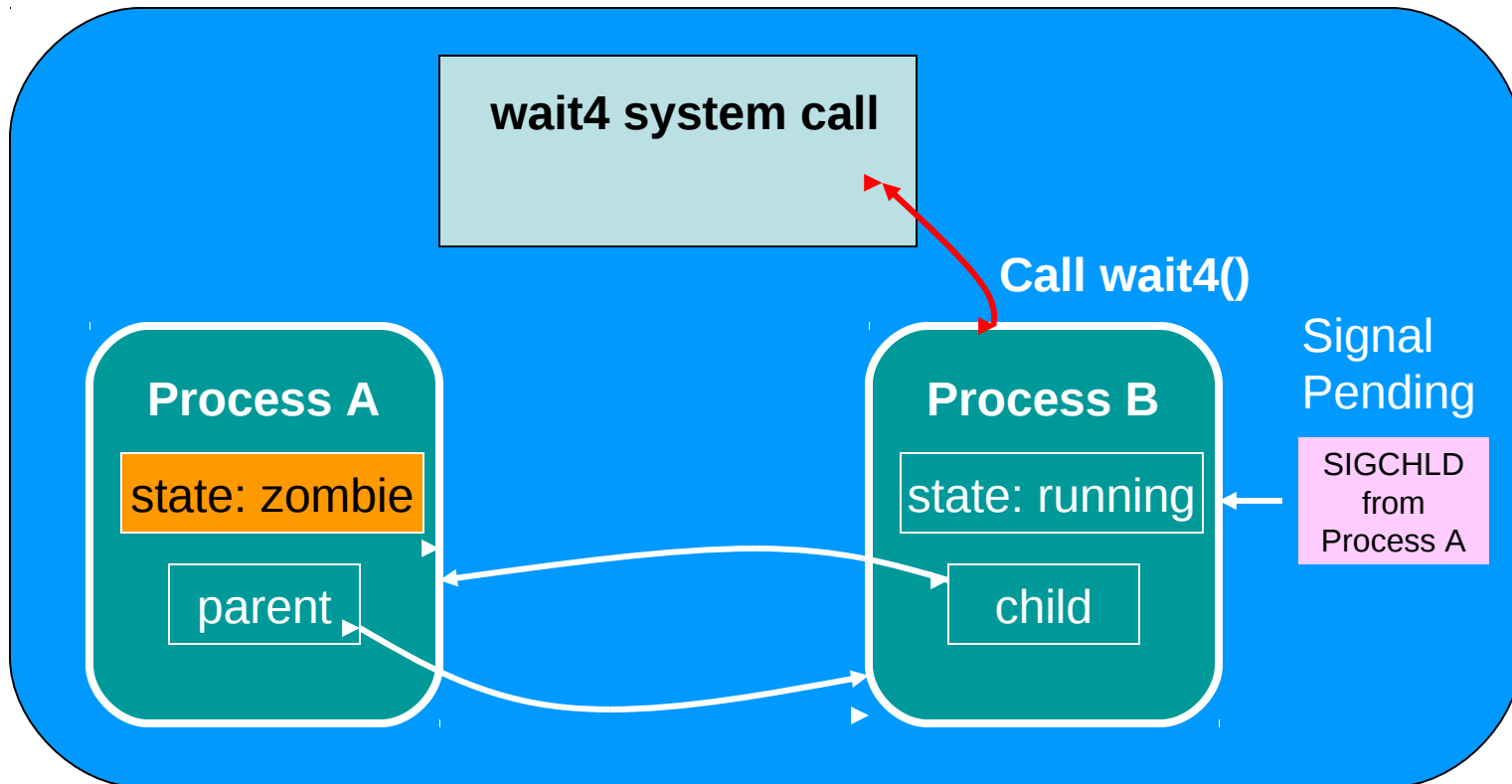


- Estado Zombie
  - Quando um processo termina, toda memória alocada, como o heap e arquivos abertos , são removidos.
  - A estrutura descritora, `task_struct`, do processo continua no kernel.
  - O processo é chamado de `zombie`.
  - Processo Zombie ainda consome alguns recursos do kernel e devem ser evitados.

# Zombie: tratamento



- Para tratar processos zombie, o pai invoca chamada de sistema `wait` ou `waitpid`, que, resulta na invocação de `wait4()`.

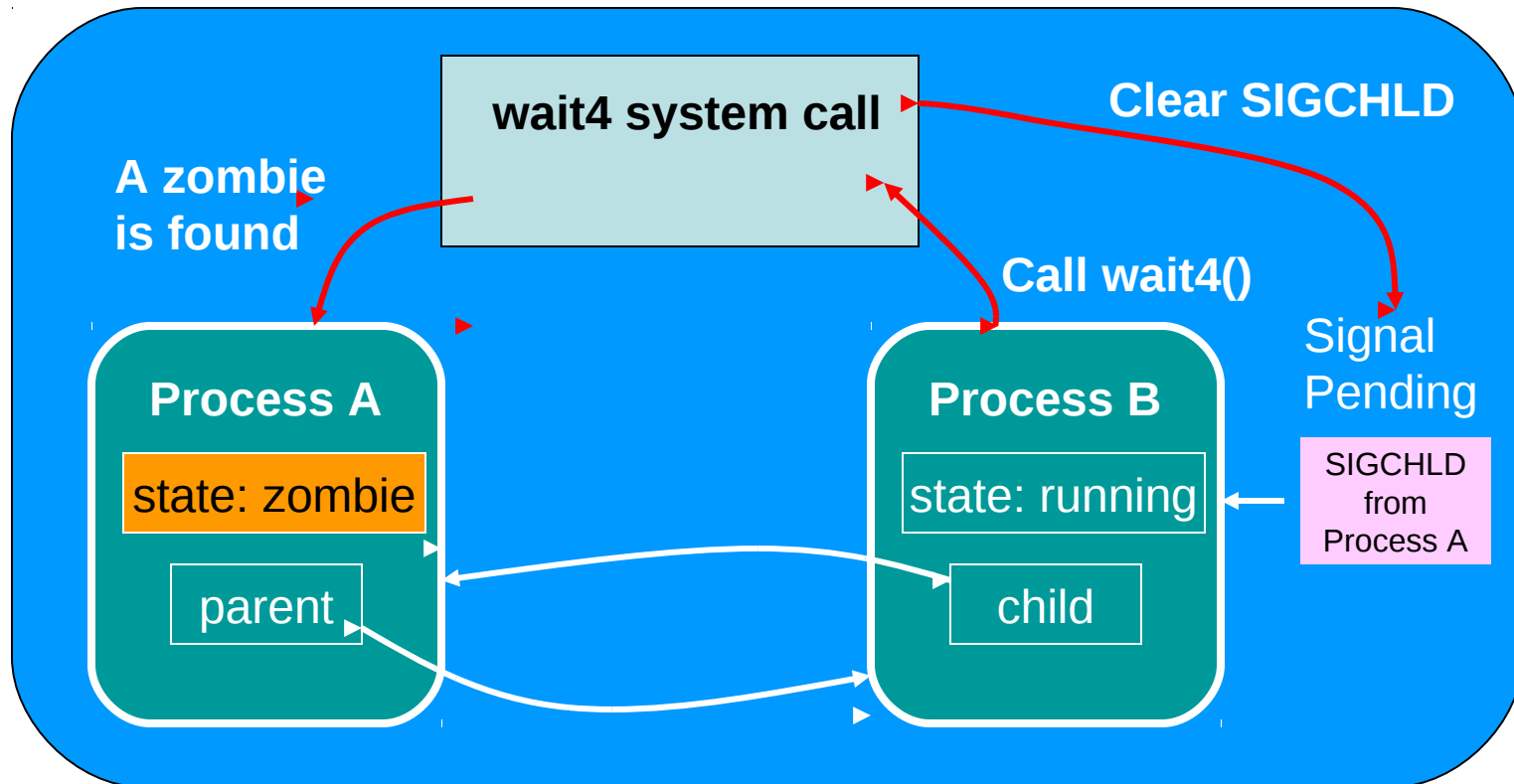




# Zombie: tratamento



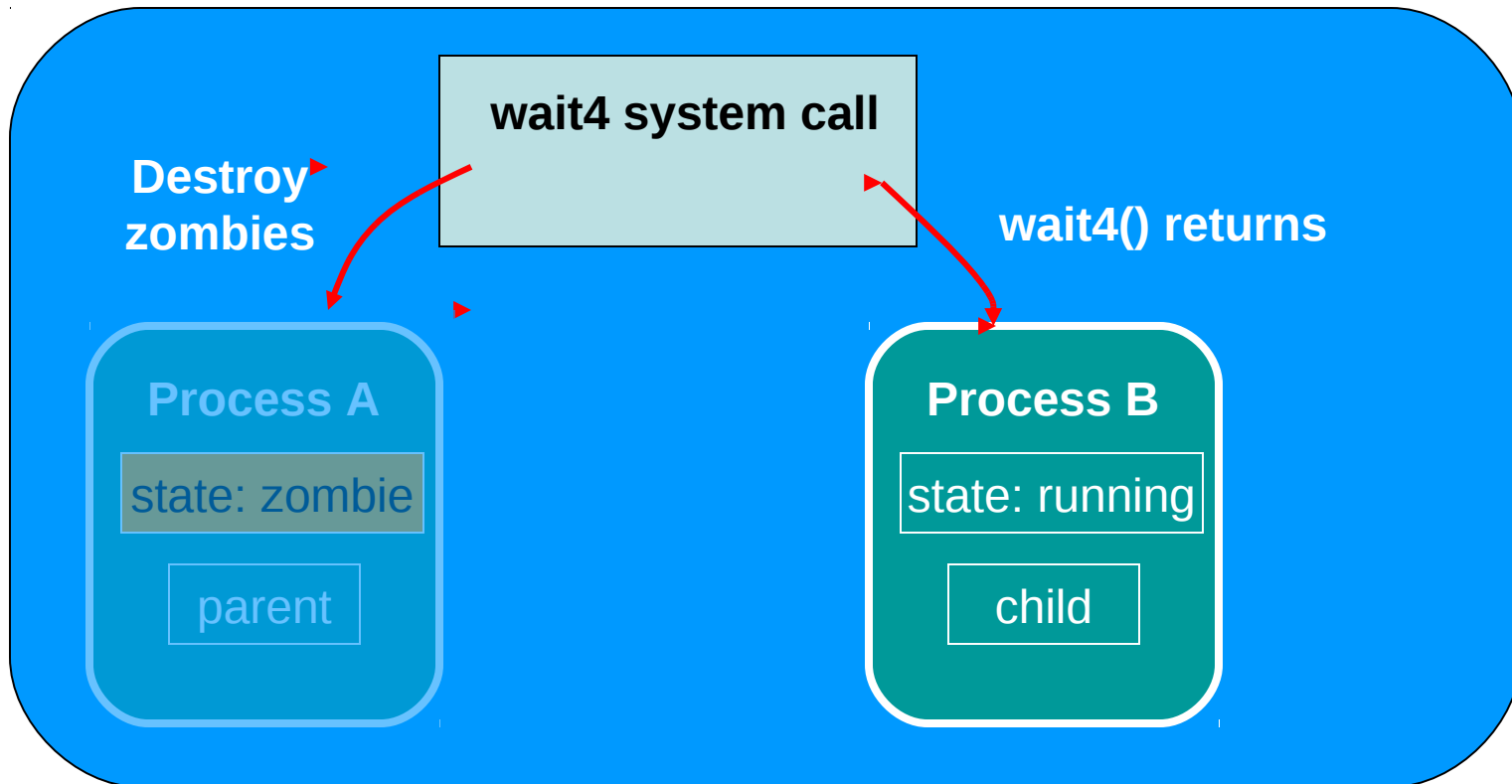
- `wait4()` procura filhos terminados do processo, limpando o correspondente sinal `SIGCHLD`.



# Zombie: tratamento



- `wait4()` libera a `task_struct` do zombie. Finalmente, a chamada `wait4()` retorna.



# wait() & waitpid()



- Eles esperam pelos processos filhos do processo para mudar de estado.
  - de pronto/executando para suspenso; ou
  - de pronto/executando para terminado, (zombie).
- wait() e waitpid() tem diferentes usos:
  - wait() retorna quando **qualquer um** dos filhos termina.
  - waitpid() retorna quando o **child especificado** termina.
  - Com argumentos, waitpid() retorna quando o filho especificado é **suspenso**, por sinais SIGTSTP (Ctrl + Z) and SIGSTOP.
- Quando um filho termina, o wait() e o waitpid() liberam a estrutura descritora do process filho.

# Fork + Exec + Wait



- Um simples shell.

```
int main(void) {
    char input[1024];
    while(1) {
        printf("[shell]$ ");
        fgets(input, 1024, stdin);
        input[strlen(input)-1] = '\0';

        if(fork() == 0) {
            execlp(input, input, NULL);
            fprintf(stderr, "command not found\n");
            exit(1);
        }
        else
            wait(NULL);
    }
}
```

[shell]\$ \_

Shell  
Process

# Fork + Exec + Wait



- `fork()` cria o filho do processo shell.

```
int main(void) {
    char input[1024];
    while(1) {
        printf("[shell]$ ");
        fgets(input, 1024, stdin);
        input[strlen(input)-1] = '\0';

        if(fork() == 0) {
            execlp(input, input, NULL);
            fprintf(stderr, "command not found\n");
            exit(1);
        }
        else
            wait(NULL);
    }
}
```

[shell]\$ ls

—

Shell  
Process

Shell  
Process

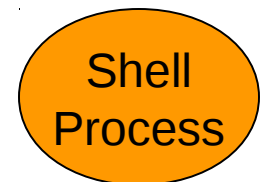
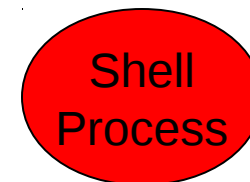
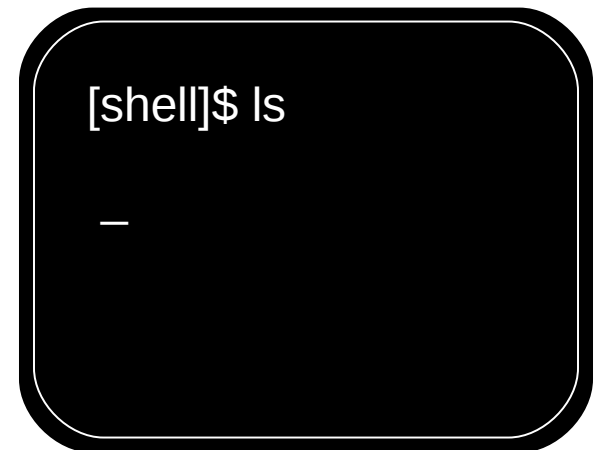
# Fork + Exec + Wait



- O pai é suspenso até que o filho termina.

```
int main(void) {
    char input[1024];
    while(1) {
        printf("[shell]$ ");
        fgets(input, 1024, stdin);
        input[strlen(input)-1] = '\0';

        if(fork() == 0) {
            execlp(input, input, NULL);
            fprintf(stderr, "command not found\n");
            exit(1);
        }
        else
            wait(NULL);
    }
}
```



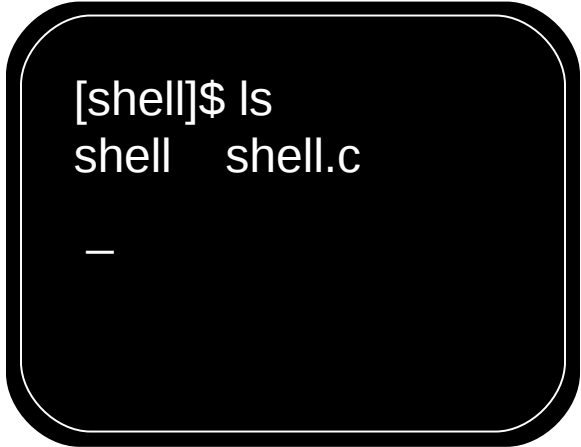
Suspended

# Fork + Exec + Wait



- O filho do processo shell torna-se o processo “ls”.

```
int main(void) {  
    char input[1024];  
    while(1) {  
        printf("[shell]$ ");  
        fgets(input, 1024, stdin);  
        input[strlen(input)-1] = '\0';  
  
        if(fork() == 0) {  
            → execvp(input, input, NULL);  
            fprintf(stderr, "command not found\n");  
            exit(1);  
        }  
        else  
            → wait(NULL);  
    }  
}
```



```
[shell]$ ls  
shell  shell.c  
  
—
```

Shell  
Process

“ls”  
Process

Suspended