

Processos

Hierarquia de Processos

Criação de Processos no Unix

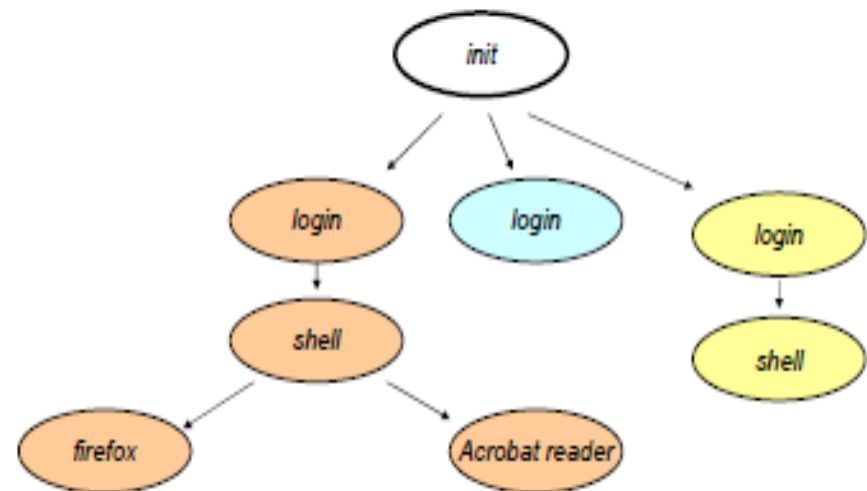
Hierarquia de Processos: Windows x Unix

- Windows

- Não há hierarquia
- Sem relação entre processo pai-filho
- Todos os processos são criados iguais

Unix

Existe hierarquia entre processo criador (pai) e criado (filho)



Processos no Unix

- Processo
 - programa em execução
 - ocupa espaço no gerenciador de processos
- Como identificar os processos ?
 - Process identification – PID
 - SO garante que enquanto o processo estiver em execução seu PID será único
 - Parent Process Identification - PPID

Identificação de processos

`pid_t getpid(void)`

Retorna o PID do processo

`pid_t getppid(void)`

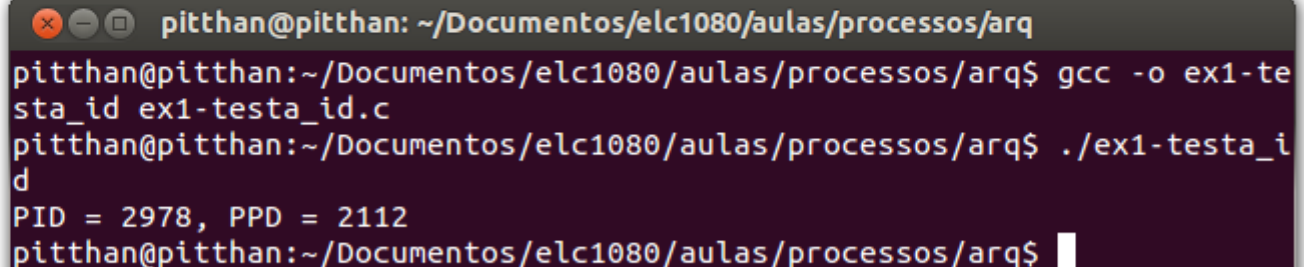
Retorna o PID do criador do processo (PAI)

Exemplo de teste do ID dos processos (ex1-testa-id.c)

ex1-testa_id.c x

```
/* arquivo ex1-testa_id */  
#include <stdio.h>  
#include <unistd.h>  
#include <stdlib.h>
```

```
int main() {  
    pid_t idPai;  
    pid_t idProcesso;  
    idPai = getppid();  
    idProcesso = getpid();  
  
    printf("PID = %d, PPD = %d\n", idProcesso, idPai);  
  
    exit(0);  
}
```



A terminal window with a dark background and light text. The title bar shows the window name 'pitthan@pitthan: ~/Documentos/elc1080/aulas/processos/arq'. The terminal content shows the compilation and execution of the program: 'gcc -o ex1-testa_id ex1-testa_id.c', followed by './ex1-testa_id', which outputs 'PID = 2978, PPD = 2112'. The prompt returns to the shell.

```
pitthan@pitthan: ~/Documentos/elc1080/aulas/processos/arq  
pitthan@pitthan:~/Documentos/elc1080/aulas/processos/arq$ gcc -o ex1-testa_id ex1-testa_id.c  
pitthan@pitthan:~/Documentos/elc1080/aulas/processos/arq$ ./ex1-testa_id  
PID = 2978, PPD = 2112  
pitthan@pitthan:~/Documentos/elc1080/aulas/processos/arq$
```

Criação de processo – ações

- Atribui PID
- Aloca espaço p/ processo
- Inicializa PCB
- Prepara ligações apropriadas
 - Ex.: coloca na lista encadeada que implementa a fila de escalonamento
- Cria/expandi outras estruturas de dados
 - Ex.: mantém um arquivo de contabilidade

Criação de processo – implementação (1)

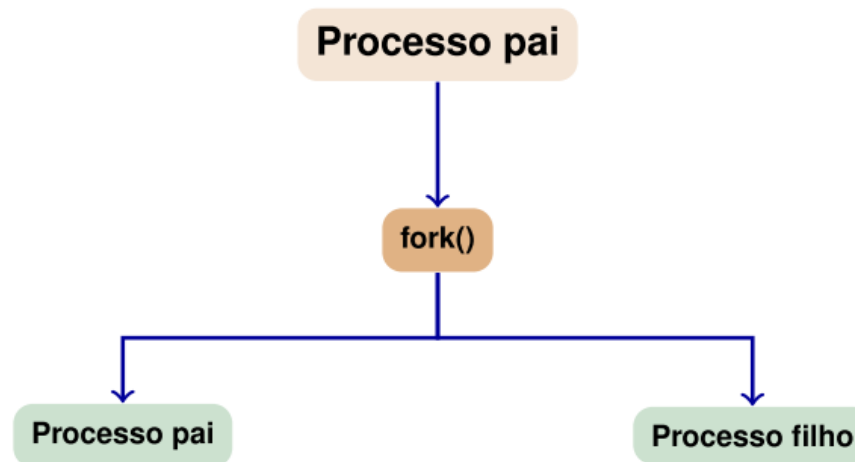
- Processo pai cria processos filhos, que podem criar outros processos, formando uma árvore de processos
- Recursos compartilhados
 - Pai e filho compartilham todos recursos
 - Filho compartilha parte dos recursos do pai
 - Pai e filho não compartilham recursos

Criação de processo – implementação (2)

- Execução
 - Pai e filho executam de forma concorrente
 - Pai espera filho terminar
- Espaço de endereços
 - Filho é duplicata do pai
 - Filho carrega um programa

Criação de processo – Unix

- Chamada de sistema *fork()*
 - cria um processo filho que herda:
 - cópia idêntica de variáveis e memória do pai
 - cópia idêntica de todos os registradores



Chamada de sistema *fork()*

- *fork()* é invocada uma vez, no processo pai, mas retorna 2 vezes, uma no pai e outra no filho
- processo filho é uma cópia do processo pai
 - áreas do processo pai são duplicadas (código, dados, pilha, memória dinâmica)
- processo filho (assim como o pai) continua a executar as instruções seguintes à chamada *fork()*
- em geral, não se sabe quem continua a executar imediatamente após uma chamada *fork()* (se é o pai ou o filho) - depende do algoritmo de escalonamento

Copy-on-write (COW)

- Como alternativa à ineficiência, no Linux, o `fork()` é implementado usando a técnica *copy-on-write* (COW)
 - atrasa ou evita a cópia dos dados
 - ao invés de copiar o espaço de endereçamento do processo pai, ambos compartilham uma única cópia somente de leitura
 - se ocorre uma escrita, é feita uma duplicação e cada processo recebe uma cópia
 - logo, a duplicação é feita apenas quando necessário, economizando tempo e espaço
 - O único overhead do `fork()` é a duplicação da tabela de páginas do processo pai e a criação de um novo PID para o filho
-

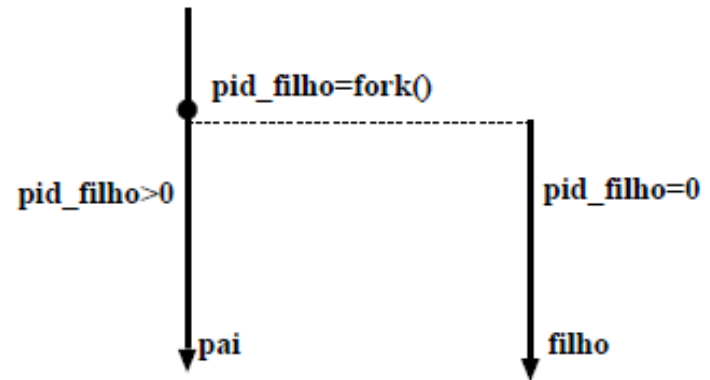
Sintaxe da chamada de sistema *fork()*

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

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

Retorna:

- 0, para o processo filho
- PID do filho, para o processo pai
- -1, se houve erro e o serviço não foi executado



Exemplo de fork com if (ex2-fork.c)

```
ex2-fork.c x
/* arquivo ex2-fork */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
    pid_t idProcesso;

    idProcesso = fork();

    if (idProcesso < 0) { // erro no fork
        fprintf(stderr, "fork falhou\n");
        exit(-1);
    }
    else if (idProcesso == 0) // filho
        printf("sou o FILHO, meu id = %d, meu pai eh %d\n", getpid(), getppid());
    else // pai
        printf("sou o PAI, meu id = %d, meu pai eh %d\n", getpid(), getppid());

    exit(0);
}
```

```
pitthan@pitthan: ~/Documentos/elc1080/aulas/processos/arq
pitthan@pitthan:~/Documentos/elc1080/aulas/processos/arq$ gcc -o ex2-fork ex2-fork.c
pitthan@pitthan:~/Documentos/elc1080/aulas/processos/arq$ ./ex2-fork
sou o PAI, meu id = 3105, meu pai eh 2112
sou o FILHO, meu id = 3106, meu pai eh 3105
pitthan@pitthan:~/Documentos/elc1080/aulas/processos/arq$
```

Alterando o ex anterior (ex2-fork-altera.c)

```
ex2-fork-altera.c (~/Documentos/sisop/progs/processos) - gedit
Abrir Salvar Desfazer
ex2-fork-altera.c x
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(){
    pid_t idProcesso;
    idProcesso = fork();

    if (idProcesso < 0){           // erro no fork
        fprintf(stderr, "fork falhou\n");
        exit(-1);
    }
    else if (idProcesso == 0)     // filho
        printf("sou o FILHO, meu id = %d, meu pai eh %d\n\n", getpid(), getppid());
    else                          // pai
        printf("sou o PAI, meu id = %d, meu pai eh %d\n\n", getpid(), getppid());

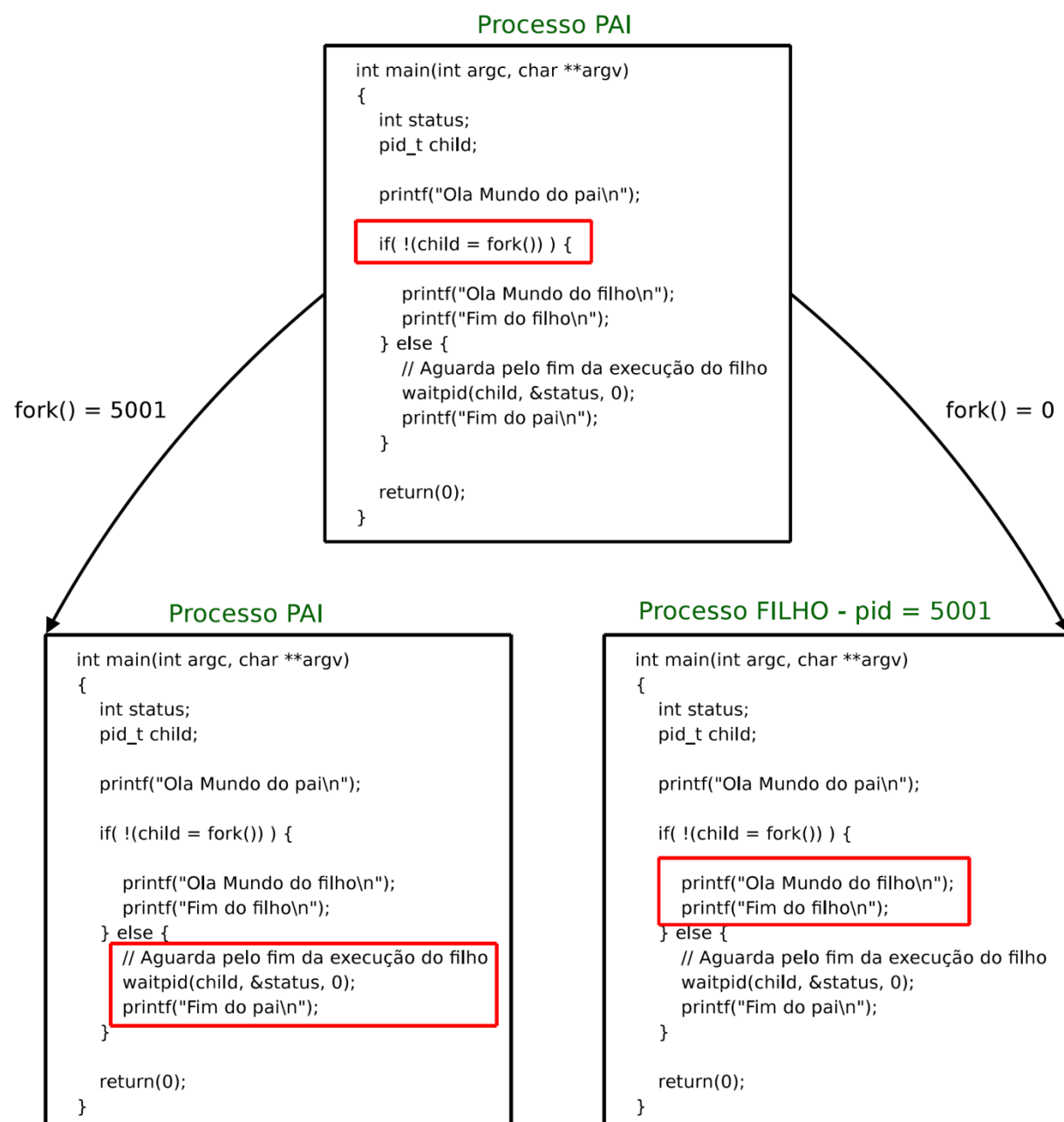
    printf("Print executado por ambos. Sou %d.\n", getpid());
    exit(0);
}
```

```
pitthan@pitthan: ~/Documentos/sisop/progs/processos
pitthan@pitthan:~/Documentos/sisop/progs/processos$ gcc -o ex2-fork-altera ex2-fork-altera.c
pitthan@pitthan:~/Documentos/sisop/progs/processos$ ./ex2-fork-altera
sou o PAI, meu id = 3307, meu pai eh 2284

Print executado por ambos. Sou 3307.
sou o FILHO, meu id = 3308, meu pai eh 3307

Print executado por ambos. Sou 3308.
pitthan@pitthan:~/Documentos/sisop/progs/processos$
```

fork



Exemplo de fork com case (ex3-fork1.c)

```
ex3-fork1.c x
/* arquivo ex3-fork1.c */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
    pid_t idProcesso;
    printf("Iniciando o programa ...\n");

    idProcesso = fork();
    switch(idProcesso) {
        case -1: exit(1);

        case 0: printf("Sou o processo %d, meu pai eh %d\n", getpid(), getppid());
                break;

        default: printf("Sou o processo pai %d, meu pai eh %d e meu filho eh %d\n", getpid(), getppid(), idProcesso);
                 break;
    }
    exit(0);
}
```

```
pitthan@pitthan: ~/Documentos/elc1080/aulas/processos/arq
pitthan@pitthan:~/Documentos/elc1080/aulas/processos/arq$ gcc -o ex3-fork1 ex3-fork1.c
pitthan@pitthan:~/Documentos/elc1080/aulas/processos/arq$ ./ex3-fork1
Iniciando o programa ...
Sou o processo pai 3190, meu pai eh 2112 e meu filho eh 3191
Sou o processo 3191, meu pai eh 3190
pitthan@pitthan:~/Documentos/elc1080/aulas/processos/arq$
```


Exemplo de fork com sleep (ex4-fork-sleep.c)

```
ex4-fork-sleep.c x
/* arquivo ex4-fork-sleep.c */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
    pid_t idProcesso;
    int i;
    printf("Iniciando o programa ...\n");

    idProcesso = fork();
    switch(idProcesso) {
        case -1: exit(1);

        case 0: for(i=0; i<4; i++){
            printf("Sou o processo filho %d\n",
getpid());
            sleep(5);
        }
        break;

        default: for(i=0; i<4; i++){
            printf("Sou o processo pai %d\n",
getpid());
            sleep(5);
        }
        break;
    }
    printf("O processo com ID=%d terminou !!!\n", getpid());
    exit(0);
}
```

```
pitthan@pitthan: ~/Documentos/elc1080/aulas/processos/arq
pitthan@pitthan:~/Documentos/elc1080/aulas/processos/arq$ gcc
-o ex4-fork-sleep ex4-fork-sleep.c
pitthan@pitthan:~/Documentos/elc1080/aulas/processos/arq$ ./e
x4-fork-sleep &
[1] 2758
pitthan@pitthan:~/Documentos/elc1080/aulas/processos/arq$ Ini
ciando o programa ...
Sou o processo pai 2758
Sou o processo filho 2759
ps
  PID TTY          TIME CMD
 2638 pts/0        00:00:00 bash
 2758 pts/0        00:00:00 ex4-fork-sleep
 2759 pts/0        00:00:00 ex4-fork-sleep
 2760 pts/0        00:00:00 ps
pitthan@pitthan:~/Documentos/elc1080/aulas/processos/arq$ Sou
o processo pai 2758
Sou o processo filho 2759
Sou o processo pai 2758
Sou o processo filho 2759
Sou o processo filho 2759
Sou o processo pai 2758
O processo com ID=2758 terminou !!!
O processo com ID=2759 terminou !!!
```

Processos órfãos

- se um processo pai morre antes de seu filho, este último é “adotado” pelo processo init
 - kernel garante que todos filhos do processo terminado viram órfãos e são adotados pelo init (ppid vira 1)

Exemplo de processos “órfãos” (ex-orfao.c)

```
ex-orfao.c x
/* arquivo ex-orfao.c */

#include <stdio.h>

int main(){

    int pid;

    printf("Sou o processo original com PID=%d, meu pai eh %d\n", getpid(), getppid());
    pid=fork();           // duplica -pai e filho continuam daqui
    if (pid !=0) {        // processo pai
        printf("Sou o pai com PID=%d, meu pai eh %d\n", getpid(), getppid());
        printf("Criei um filho com PID = %d\n", pid);
    }
    else {               // processo filho
        sleep(5);        // garante que o pai termina antes
        printf("Sou o filho com PID=%d, meu pai eh %d\n", getpid(), getppid());
    }
    printf("Processo com PID=%d terminou\n", getpid()); // ambos executam
}
```

```
pitthan@pitthan: ~/Documentos/elc1080/aulas/processos/arq
pitthan@pitthan:~/Documentos/elc1080/aulas/processos/arq$ gcc -o ex-orfao ex-orfao.c
pitthan@pitthan:~/Documentos/elc1080/aulas/processos/arq$ ./ex-orfao
Sou o processo original com PID=2266, meu pai eh 2192
Sou o pai com PID=2266, meu pai eh 2192
Criei um filho com PID = 2267
Processo com PID=2266 terminou
pitthan@pitthan:~/Documentos/elc1080/aulas/processos/arq$ Sou o filho com PID=2267, meu
pai eh 1
Processo com PID=2267 terminou
```

Processos “zombies”

- um processo pode se terminar quando seu pai não está a sua espera
 - o processo filho se torna um processo “zombie”
 - identificação “defunct” ou “zombie” ao lado do nome do processo
 - segmentos de instruções e dados o sistema são automaticamente suprimidos com sua morte
 - Processo continua ocupando espaço na tabela de processos do kernel
 - quando seu fim é esperado, ele simplesmente desaparece ao fim de sua execução
-

Exemplo de processos “zombies” (ex-zombie.c)

```
Documentos 9 int main() {
10     int pid ;
11
12     printf("Sou o processo pai, PID = %d, e vou criar um filho.\n",getpid());
13     printf("%d entrando em um loop infinito\n", getpid());
14     pid = fork();
15
16     if(pid == -1) {
17         fprintf(stderr, "fork falhou\n");
18         exit(-1);
19     }
20     else if(pid == 0) { // filho
21         printf("Sou o filho, PID = %d, vou dormir um pouco\n",getpid());
22         sleep(10);
23         printf("Sou %d e acordei! Vou terminar agora. Ops, Virei um 'zumbi!!!\n", getpid());
24         exit(0);
25     }
26     else { // pai
27         for(;;);
28     }
29     exit(0);
30 }
```

Linha: 30 de 32 Coluna: 2 LINHA INS

ex-zombie.c UTF-8

```
pitthan@pitthan:~/Documentos/sisop/progs/processos$ ./ex-zombie &
[1] 4709
pitthan@pitthan:~/Documentos/sisop/progs/processos$ Sou o processo pai, PID = 4709, e vou criar um filho.
4709 entrando em um loop infinito
Sou o filho, PID = 4710, vou dormir um pouco
Sou 4710 e acordei! Vou terminar agora. Ops, Virei um zumbi!!!
ps
  PID TTY          TIME CMD
 2340 pts/2    00:00:01 bash
  4709 pts/2    00:00:14 ex-zombie
  4710 pts/2    00:00:00 ex-zombie <defunct>
  4711 pts/2    00:00:00 ps
pitthan@pitthan:~/Documentos/sisop/progs/processos$
```

Unix – Chamada de sistema *wait()* (1)

- Processo pai pode esperar o término de um processo filho através da função *wait*
 - A função *wait* retorna o status de retorno de qualquer processo filho que termine
 - um processo que invoque *wait* pode:
 - bloquear – se nenhum dos seus filhos tiver terminado
 - retornar imediatamente com o código de terminação de um filho caso o filho já tenha terminado
 - retornar um erro – se não tiver filhos

Unix – Chamada de sistema *wait()* (2)

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

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

Retorna:

- PID do processo que terminou
- -1, em caso de erro

Exemplo de wait (ex6-fork-wait.c)

```
ex6-fork-wait.c x
/* arquivo ex6-fork-wait.c */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>

int main() {
    pid_t idProcesso;
    int estado, cont=0;

    idProcesso = fork();
    if (idProcesso < 0)
        exit(errno);
    else if (idProcesso != 0) {    // pai
        printf("Processo o pai (%d) e vou esperar pelo
filho\n", getpid());
        wait(&estado);
        printf("Sou o pai (%d), esperei pelo filho %d\n",
getpid(), idProcesso);
    }
    else if (idProcesso == 0) {    // filho
        while (cont < 5) {
            printf("Sou o filho (%d), meu pai eh %d\n",
getpid(), getppid());
            sleep(2);
            cont++;
        }
        exit(1);
    }
    exit(0);
}
```

```
pitthan@pitthan: ~/Documentos/elc1080/aulas/processos/arq
pitthan@pitthan:~/Documentos/elc1080/aulas/processos/arq$ gcc -o e
x6-fork-wait ex6-fork-wait.c
pitthan@pitthan:~/Documentos/elc1080/aulas/processos/arq$ ./ex6-fo
rk-wait
Processo o pai (3033) e vou esperar pelo filho
Sou o filho (3034), meu pai eh 3033
Sou o filho (3034), meu pai eh 3033
Sou o filho (3034), meu pai eh 3033
Sou o filho (3034), meu pai eh 3033
Sou o filho (3034), meu pai eh 3033
Sou o pai (3033), esperei pelo filho 3034
pitthan@pitthan:~/Documentos/elc1080/aulas/processos/arq$
```

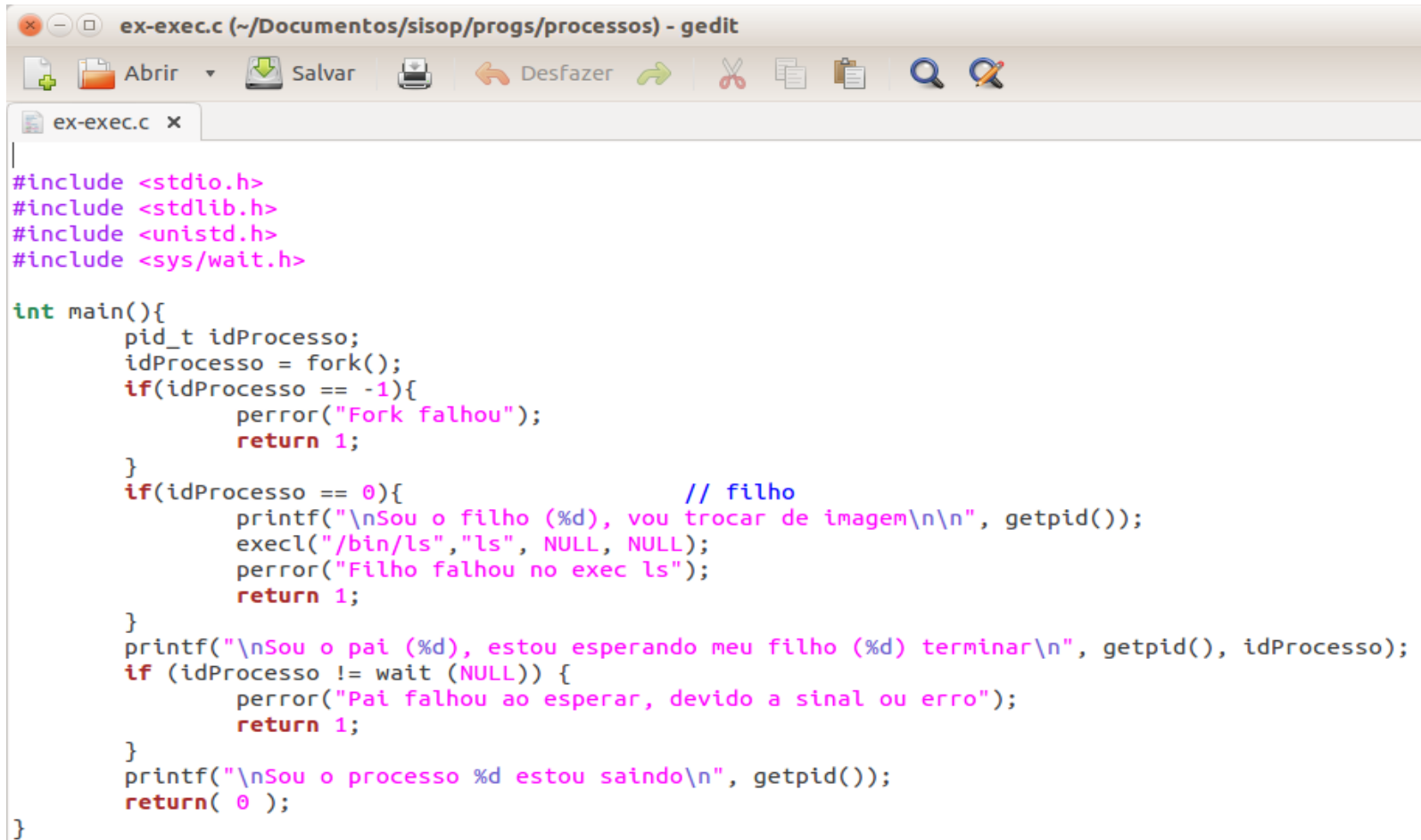

Execução de processos – Unix

- Chamada de sistema **exec***
 - após criado, o processo filho executa uma outra chamada de sistema (**exec***) para mudar sua imagem de memória (conteúdo do espaço de endereçamento) e executar um novo programa

Unix – Família `exec*`

- Família `exec*` - 6 primitivas, divididas em 2 grupos:
 - `exec()` - número de argumentos do programa lançado é conhecido
 - os argumentos são passados um a um, terminando com a string nula
 - `exec()`, `execle()` e `execlp()`
 - `execv()` – número de argumentos desconhecido
 - argumentos são passados num array de strings
 - `execv()`, `execve()` e `execvp()`
 - Em ambos os grupos, o 1º argumento deve ter o nome do arquivo executável
-

Exemplo de exec (ex-exec.c)



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main(){
    pid_t idProcesso;
    idProcesso = fork();
    if(idProcesso == -1){
        perror("Fork falhou");
        return 1;
    }
    if(idProcesso == 0){ // filho
        printf("\nSou o filho (%d), vou trocar de imagem\n\n", getpid());
        execl("/bin/ls", "ls", NULL, NULL);
        perror("Filho falhou no exec ls");
        return 1;
    }
    printf("\nSou o pai (%d), estou esperando meu filho (%d) terminar\n", getpid(), idProcesso);
    if (idProcesso != wait (NULL)) {
        perror("Pai falhou ao esperar, devido a sinal ou erro");
        return 1;
    }
    printf("\nSou o processo %d estou saindo\n", getpid());
    return( 0 );
}
```

Saída



Exemplo de exec: saída

```
pitthan@pitthan: ~/Documentos/sisop/progs/processos
pitthan@pitthan:~/Documentos/sisop/progs/processos$ gcc -o ex-exec ex-exec.c
pitthan@pitthan:~/Documentos/sisop/progs/processos$ ./ex-exec

Sou o pai (4176), estou esperando meu filho (4177) terminar

Sou o filho (4177), vou trocar de imagem

ex1-testa-id      ex2-fork-altera.c  ex3-fork1.c       ex-exec           ex-zombie
ex1-testa-id.c    ex2-fork-altera.c~ ex4-fork-sleep    ex-exec.c         ex-zombie.c
ex1-testa-id.c~   ex2-fork.c         ex4-fork-sleep.c  ex-exec.c~        old
ex2-fork          ex2-fork.c~        ex6-fork-wait     ex-orfao
ex2-fork-altera  ex3-fork1          ex6-fork-wait.c   ex-orfao.c

Sou o processo 4176 estou saindo
pitthan@pitthan:~/Documentos/sisop/progs/processos$
```

Terminação de Processos no Unix (1)

Término normal (voluntário):

a tarefa a ser executada é finalizada

- `exit()`

Término com erro (voluntário):

o processo em execução não pode ser finalizado

- Ex.: `gcc exemplo.c`, onde o arquivo `exemplo.c` não existe

Terminação de Processos no Unix (2)

Término com erro fatal (involuntário)

Erro causado por um bug no programa

- ex: divisão por 0, acesso à posição de memória inexistente ou não pertencente ao processo, execução de uma instrução ilegal, ...

Término causado por algum outro processo (involuntário)

- `kill()`

fork

Simulador:

fork, wait, exec com opção copy-on-write