



Universidade Federal de Uberlândia
Faculdade de Computação
GSI018 – Sistemas Operacionais



Processos em Linux

Prof. Marcelo Zanchetta do Nascimento

Sumário

- Processo
- Identificação
- Layout da memória
- Fork()
- Vfork()
- Sinais
- Leituras Sugeridas

Processo - Identificação

- Os atributos básicos de um processo filho são seus PID e seus processos pai PPID;
- Um processo que cria um novo processo é conhecido como processo pai;
- O kernel limita ID's de processos sendo menor ou igual a 32.767;
- O pai de qualquer processo pode ser encontrado no diretório /proc/ <PID>/. Em que <PID> refere ao número do processo.
 - **Exemplo:** `cat /proc/1/status`

Processo - Identificação

- As funções que permite um processo obter PID e PPID são **getpid** e **getppid**;
- Declarado em conjunto com a biblioteca;
 - **pid_t getpid(void);**
 - **pid_t getppid(void);**
- O **getpid** retorna o PID de seu chamador (o processo);
- O **getppid** retorna o PPID de seu chamador, o qual é o PID de seu pai.

Processo - Identificação

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(void)
{
    printf("PID = %d\n", getpid());
    printf("PPID = %d\n", getppid());
    exit(EXIT_SUCCESS);
}
```

Figura 1: imprime PID e PPID

Processo - Identificação

Attribute	Type	Function
Process ID	<code>pid_t</code>	<code>getpid(void);</code>
Parent Process ID	<code>pid_t</code>	<code>getppid(void);</code>
Real User ID	<code>uid_t</code>	<code>getuid(void);</code>
Effective User ID	<code>uid_t</code>	<code>geteuid(void);</code>
Real Group ID	<code>gid_t</code>	<code>getgid(void);</code>
Effective Group ID	<code>gid_t</code>	<code>getegid(void);</code>

Figura 2: Atributos para os PIDs e PPIDs

Layout da Memória - Processo

- A memória alocada para cada processo é composta de partes;
- O segmento texto contém a linguagem de instruções de um programa executado pelo processo.
 - Esse segmento é apenas para leitura.
- O segmento de dados **inicializado** contém variáveis globais e locais que são explicitamente inicializadas;
- O segmento dados **não inicializado** contém as variáveis que não foram explicitamente inicializadas;

Layout da Memória - Processo

- A stack é um **segmento dinamicamente** crescente e reduzido que contém os quadros da pilha;
- A heap é uma área da qual a memória (para variáveis) pode **alocar dinamicamente** em tempo de execução.

Layout da Memória - Processo

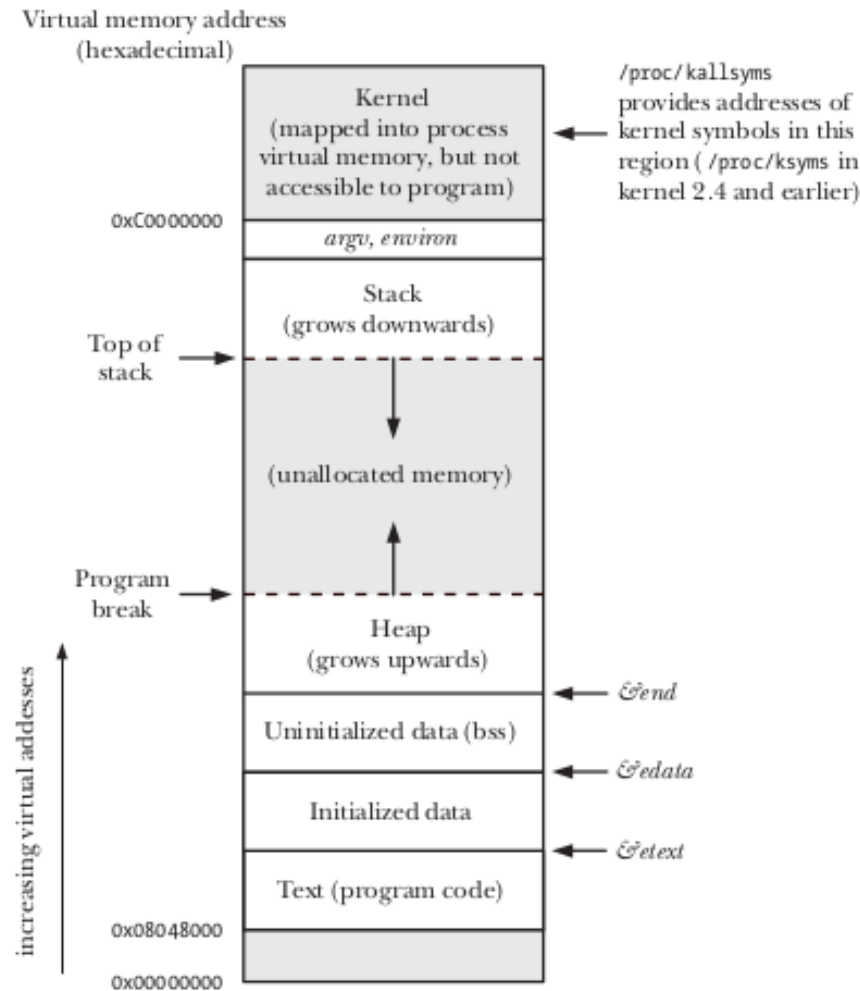


Figura 3: Espaço de alocação de um processo

Comandos `fork()` , `exit()` , `wait()` e `execve()`

- O sistema de chamada **`fork()`** permite, um processo, o pai criar um novo processo filho;
 - Isso é feito criando o novo processo filho um (quase) duplicado do pai: o filho obtém cópias da pilha, dados, heap e segmentos de texto;
- O comando **`exit (status)`** termina um processo em que as fontes (memória, descritores de arquivos abertos, etc) usadas pode ser realocação subsequentes pelo kernel;
- A chamada **`wait(&status)`** tem duas funções:
 - se o filho de um processo ainda não terminou pela chamada `exit()`, o `wait()` suspende execução até que um de seus filhos terminarem;
 - o status de finalização do filho é retornado no argumento de status do `wait ()`.

Comandos `fork()` , `exit()` , `wait()` e `execve()`

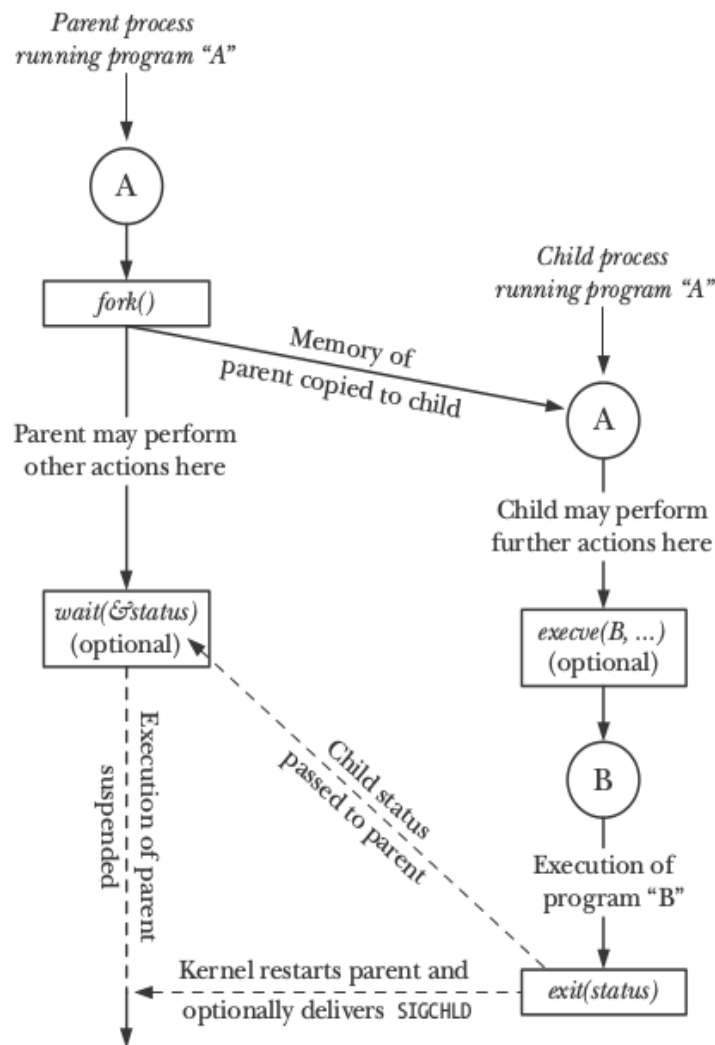


Figura 4: Fase em que ocorre a criação do filho

Comando Fork() - Copy on Write

- No Linux, o `fork()` usa a técnica chamada *copy-on-write* (COW);
- Essa técnica evita a cópia dos dados. Ao invés de copiar o espaço do processo pai, ambos compartilham uma única cópia somente leitura.
- Se uma escrita é feita, uma duplicação é feita e cada processo recebe uma cópia.
- A duplicação é feita apenas quando necessário, economizando tempo e espaço;
- O único *overhead* realmente necessário do `fork()` é a duplicação da **tabela de páginas** da memória do processo pai e a criação de um novo PID para o filho.

Comando Fork() - Copy on Write

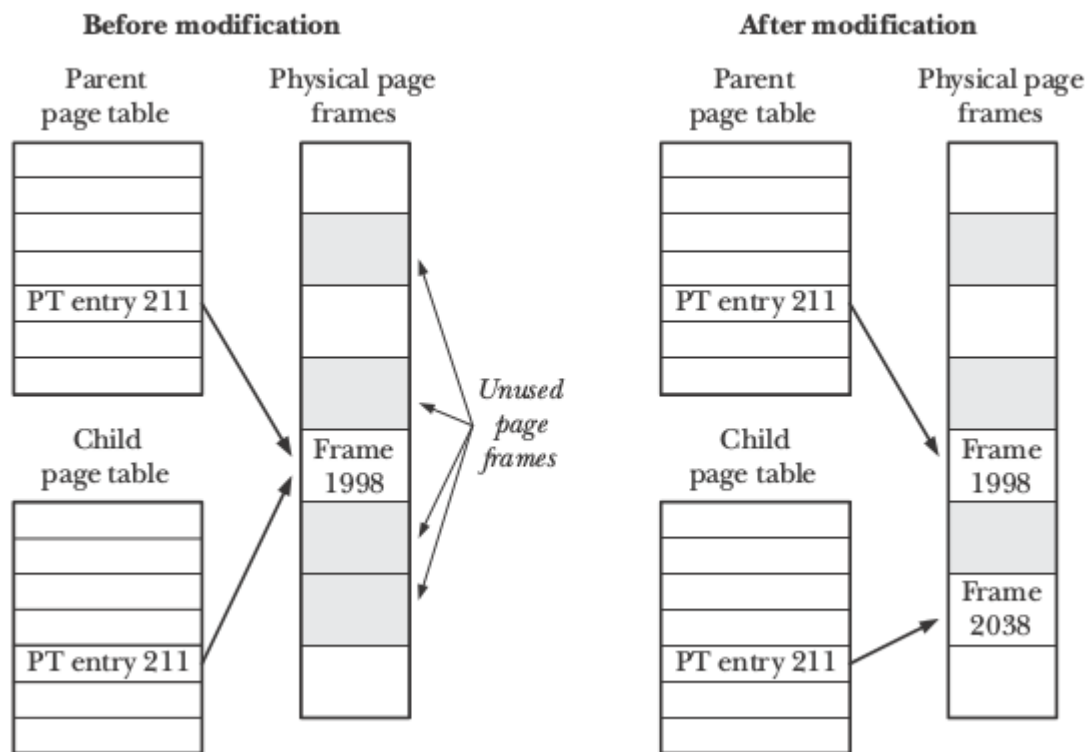


Figura 5: Espaço de memória do Pai e Filho.

Comando `vfork()`

- A chamada `vfork()` é mais eficiente que o `fork()` embora opere com uma semântica um pouco diferente;
- As principais diferenças estão:
 - Não há duplicação de páginas de memória virtual ou tabelas de páginas.
 - O processo filho compartilha memória até que execute uma chamada `exec()` ou `exit()` para finalizar;
 - A execução do processo pai é suspenso até o filho executar um `exec()` or `_exit()`.

vFork()

```
int globvar = 33;

int main(void) {

    pid_t pid;

    if ((pid = vfork()) < 0) {
        printf("vfork error");
    } else if (pid == 0) {
        globvar++;

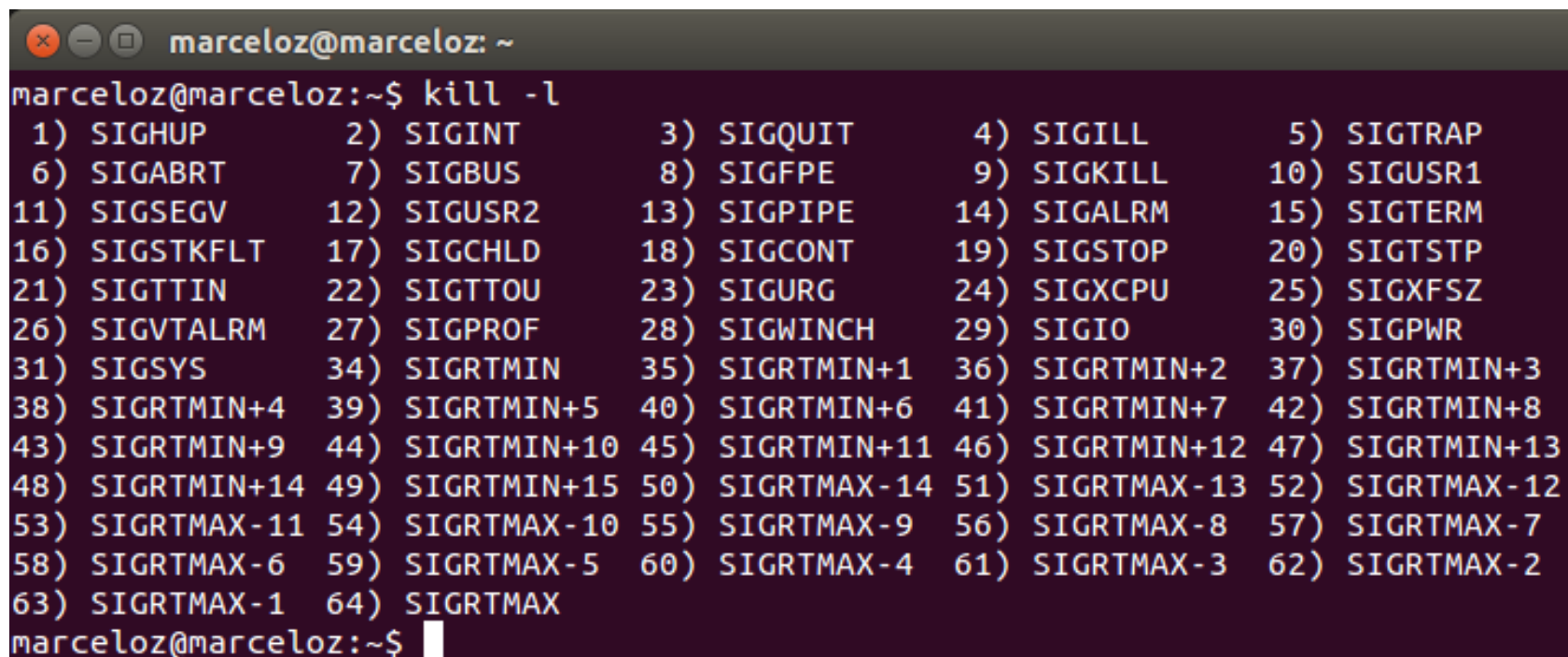
        exit(0);
    }
    printf("pid = %ld, glob = %d\n", getpid(), globvar);
    return(0);
}
```

Figura 6: Exemplo do uso do vfork

Sinais

- Os sinais são interrupções;
- Um sinal é uma notificação para um processo avisando sobre a ocorrência de um evento;
- O Sinal é análogo ao hardware de interrupção em que interrompe o fluxo normal de execução de um programa;
- Indicam a ocorrência de condições excepcionais:
 - Divisao por zero;
 - Acesso inválido à memória;
 - Interrupção do programa;
 - Término de um processo filho.

Sinais



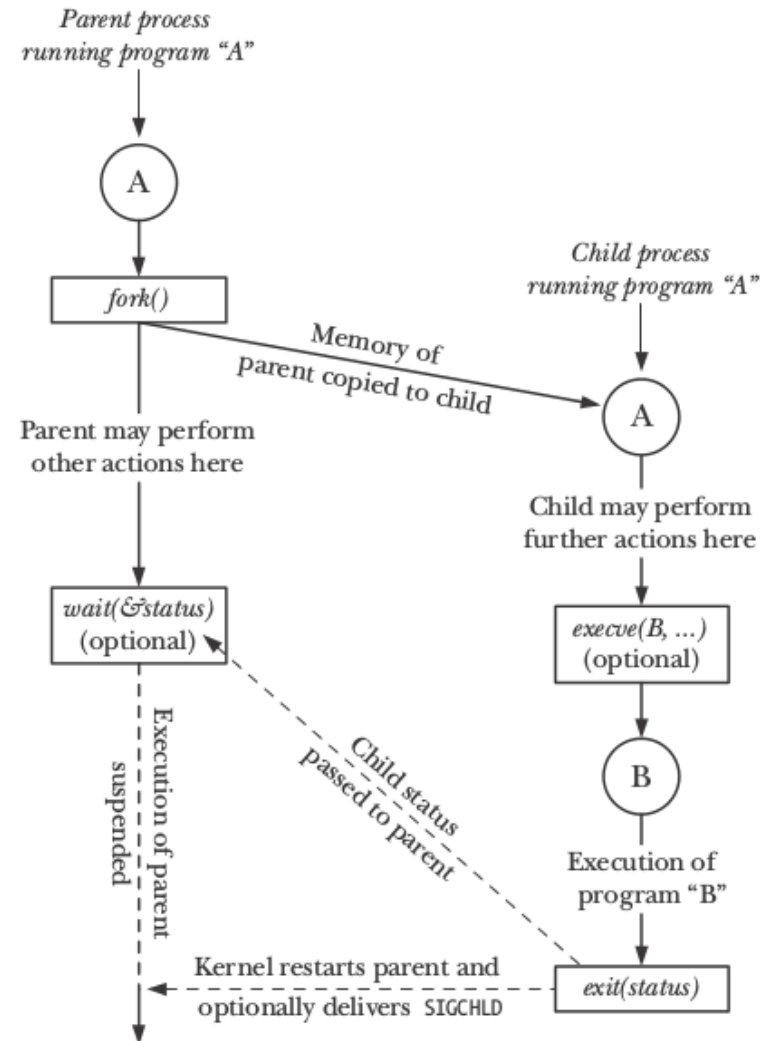
A terminal window titled 'marceloz@marceloz: ~' showing the command 'kill -l' and its output. The output is a list of 64 signal names, numbered 1 to 64, arranged in five columns. The signals include standard POSIX signals like SIGHUP, SIGINT, SIGQUIT, SIGILL, SIGTRAP, SIGABRT, SIGBUS, SIGFPE, SIGKILL, SIGUSR1, SIGSEGV, SIGUSR2, SIGPIPE, SIGALRM, SIGTERM, SIGSTKFLT, SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU, SIGURG, SIGXCPU, SIGXFSZ, SIGVTALRM, SIGPROF, SIGWINCH, SIGIO, SIGPWR, SIGSYS, SIGRTMIN, SIGRTMIN+1 through SIGRTMIN+15, SIGRTMAX-14 through SIGRTMAX-2, and SIGRTMAX.

```
marceloz@marceloz:~$ kill -l
 1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
 6) SIGABRT     7) SIGBUS      8) SIGFPE      9) SIGKILL     10) SIGUSR1
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE    14) SIGALRM    15) SIGTERM
16) SIGSTKFLT  17) SIGCHLD    18) SIGCONT    19) SIGSTOP    20) SIGTSTP
21) SIGTTIN    22) SIGTTOU    23) SIGURG     24) SIGXCPU    25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF    28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS     34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Figura 7: Códigos para o sinal kill

Sinais

Invocação de um sinal pode interromper o fluxo do programa principal em qualquer tempo;



Alarme

- Exemplo de sinal assíncrono:

```
unsigned int alarm(unsigned int seconds);
```

- Envia um sinal do SIGALRM para o processo após alguns segundos.

Alarme

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

```
int main (void) {
    char c;
```

Alarm (1); % aguarda 1 segundo e finaliza

```
printf("Tecle enter para terminar: ");
scanf("%c", &c);
```

```
return 0;
}
```

Figura 8: Exemplo de um sinal de alarme.

Leitura Sugerida

- Matthew & Stones, Beginning Linux Programming, 4th Ed, Wrox Press, 2008. ISBN: 9780470147627

