Processos

Aula 1 - Chamadas ao sistema

José Pedro Oliveira
(jpo@di.uminho.pt)

Grupo de Sistemas Distribuídos
Departamento de Informática
Escola de Engenharia
Universidade do Minho

Sistemas Operativos I
2006-2007

Conteúdo

Programas e Processos

Programa

Um programa é um ficheiro executável que reside num directório de um disco. Um programa é carregado para memória e executado pelo kernel como resultado de uma das seis funções `exec`.

Processo

Uma instância em execução de um programa é designada por **processo**. Cada processo tem um identificador numérico único designado por ***process ID***. Este identificador é um número inteiro não negativo.

Conteúdo

Chamadas ao sistema

Chamadas ao sistema (*system calls*)

- fork - create a child process
- _exit - terminate the current process
- getpid - get current process identification
- getppid - get parent process identification
- wait - wait for process termination
- waitpid - wait for process termination

Chamada ao sistema: fork

Sumário

Criar um processo filho

Synopsis

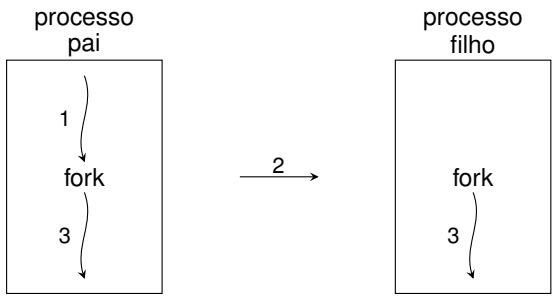
```
#include <sys/types.h>
#include <unistd.h>

pid_t fork(void);
```

Valor de retorno

Esta chamada ao sistema retorna duas vezes: uma no contexto do processo original (*pai*) e outra no contexto do novo processo (*filho*). Em caso de erro só retorna uma vez (valor -1 no contexto do processo pai).

Fork: uma invocação, dois retornos



Chamada ao sistema: fork - exemplo 1

Exemplo 1

```
1  #include <stdio.h>
2  #include <unistd.h>
3
4  int main(void)
5  {
6      printf("Antes\n");
7
8      fork();
9
10     printf("Depois\n");
11
12     return 0;
13 }
```

Chamada ao sistema: fork - exemplo 2

Exemplo 2

```
1  #include <stdio.h>
2  #include <sys/types.h>          /* pid_t */
3  #include <unistd.h>
4
5  int main(void)
6  {
7      pid_t p;
8
9      p = fork();
10
11     printf("p = %d\n", p);
12
13     return 0;
14 }
```


Chamada ao sistema: fork - valor de retorno

Valor de retorno

- -1 - Insucesso: o valor -1 é retornado no contexto do processo pai e nenhum processo filho é criado.
- 0 - Sucesso: o valor 0 é retornado no contexto do processo filho (novo processo).
- > 0 - Sucesso: o PID do processo filho é retornado no contexto do processo pai (processo original).

Chamada ao sistema: fork - exemplo 3

Exemplo 3 (extracto)

```
1      p = fork();
2
3      if (p == -1) {
4
5          /* Erro */
6
7      } else if (p == 0) {
8
9          /* Filho */
10
11     } else if (p > 0) {
12
13         /* Pai */
14
15     }
```


Chamada ao sistema: fork - exemplo 4

Exemplo 4 (extracto)

```
1      p = fork();
2
3      switch (p) {
4      case -1 :
5          /* Erro */
6          break;
7      case 0:
8          /* Codigo a executar no processo filho */
9          break;
10     default:
11         /* Codigo a executar no processo pai */
12         break;
13     }
```

Chamada ao sistema: _exit

Sumário

Finalizar processo corrente

Synopsis

```
#include <unistd.h>

void _exit(int status);
```

Valor de retorno

Esta chamada ao sistema termina imediatamente o processo invocador (não retorna). O valor `status` (0..255) é retornado ao processo pai, que o pode recolher através da chamada ao sistema **wait**.

Finalização de um processo

Finalização de um processo

- Finalização normal
  - Retorno a partir de `main`
  - Invocar `exit`
  - Invocar `_exit` ou `_Exit`
  - Retorno da última thread
  - Invocar `pthread_exit` a partir da última thread
- Finalização anormal
  - Invocar `abort`
  - Recepção de um sinal
  - Resposta da última thread a um pedido de cancelamento

Finalização de um processo

Notas

Quando a chamada ao sistema **_exit** é invocada, o processo corrente é terminado imediatamente:

- todos os descritores de ficheiros abertos são fechados
- todos os processos filhos passam a ter como pai o processo **init** (pid = 1)
- o sinal **SIGCHLD** é enviado ao processo pai

Chamada ao sistema _exit vs função exit

Sempre que possível utilizar a função **exit** da biblioteca de C em detrimento da chamada ao sistema **_exit**. Informação adicional nas páginas man: `man 2 _exit` e `man 3 exit`.

Exemplos de métodos de terminação

Exemplo

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4
5  void f1(void) {
6      ...
7      exit(1);              /* Normal */
8  }
9
10 void f2(void) {
11     ...
12     abort();              /* Anormal */
13 }
14
15 int main(void) {
16     f1();
17     f2();
18     return 0;             /* Normal */
19 }
```

Chamadas ao sistema: getpid, getppid

Sumário

A chamada ao sistema **getpid** permite obter o identificador de processo (PID) do processo corrente. A chamada so sistema **getppid** permite obter o PID do pai do processo corrente.

Synopsis

```
#include <sys/types.h>
#include <unistd.h>

pid_t getpid(void);
pid_t getppid(void);
```

Chamadas ao sistema: getpid/getppid - exemplo 1

Exemplo 1

```
1  #include <stdio.h>
2  #include <sys/types.h>                /* pid_t */
3  #include <unistd.h>
4
5  int main(void)
6  {
7      pid_t p = fork();
8
9      printf("p = %5d  pid = %5d  ppid = %5d\n",
10             p, getpid(), getppid());
11
12     return 0;
13 }
```

Chamadas ao sistema: getpid/getppid - exemplo 1

Resultado 1

```
p =     0  pid =  4284  ppid =  4283
p =  4284  pid =  4283  ppid =  2849
```

Resultado 2 - processo pai termina antes da invocação de getppid() no processo filho

```
p =  4307  pid =  4306  ppid =  2849
p =     0  pid =  4307  ppid =     1
```

Adopção de processos

Quando um processo fica orfão é imediatamente adoptado pelo processo init (processo com o PID 1).

Chamadas ao sistema: getpid/getppid - exemplo 2

Exemplo 2

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/types.h>          /* pid_t */
5
6  int main(void)
7  {
8      pid_t p = fork();
9
10     if (p == -1) {              /* Erro */
11
12         perror("fork"); exit(EXIT_FAILURE);
13
14     } else if (p == 0) {        /* Filho */
15
16         printf("Filho: p = %5d  pid = %5d  ppid = %5d\n",
17                p, getpid(), getppid());
18
19     } else {                    /* Pai ( p > 0 )*/
20
21         printf("Pai  : p = %5d  pid = %5d  ppid = %5d\n",
22                p, getpid(), getppid());
23
24     }
25
26     return 0;
27 }
```

Chamadas ao sistema: wait e waitpid

Sumário

Estas chamadas ao sistema permitem esperar pela mudança de estado de um processo filho e obter informação sobre essa mudança.

Synopsis

```
#include <sys/types.h>
#include <sys/wait.h>

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

Chamadas ao sistema: wait e waitpid

wait

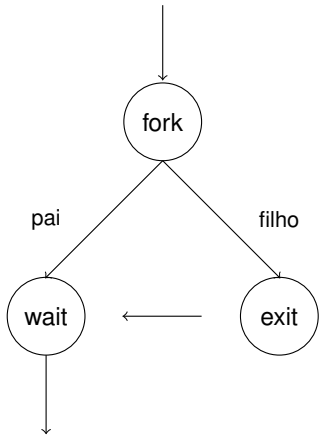
A chamada ao sistema **wait** suspende o processo corrente até que um dos seus processos filhos termine.

waitpid

A chamada ao sistema **waitpid** suspende o processo corrente até que o filho especificado mude de estado. Por omissão só espera que o filho em questão termine mas este comportamento pode ser alterado através do argumento *options*. Mudanças de estado:

- o processo filho terminou
- o processo filho foi congelado (stopped)
- o processo filho foi descongelado (resumed)

Fork, Wait e Exit



Chamada ao sistema: wait - exemplo 1

Exemplo 1 (extracto)

```
1     p = fork();
2
3     if (p == -1) {
4         perror("fork"); exit(EXIT_FAILURE);
5
6     } else if (p == 0) {
7         printf("Filho\n"); sleep(3);
8
9     } else if (p > 0) {
10        printf("Pai\n");
11        wait(NULL); /* Esperar que o filho termine */
12        printf("Fim\n");
13
14    }
```

Chamada ao sistema: wait - exemplo 1

Exemplo 1

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

int main(void)
{
    pid_t p = fork();

    if (p == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    } else if (p == 0) {    /* Filho */
        printf("Filho\n");
        sleep(3);
    } else if (p > 0) {
        printf("Pai\n");
        wait(NULL);         /* Esperar que o filho termine */
        printf("Fim\n");
    }

    return 0;
}
```


Chamadas ao sistema: wait, waitpid - macros (1/2)

Relação entre **wait()** e **waitpid()**

```
wait(&status) == waitpid(-1, &status, 0)
```

Macros

WIFEXITED(status) - permite determinar se o processo filho terminou normalmente.

WEXITSTATUS(status) - permite obter o código de saída do processo filho (argumento da função `exit`). Esta macro só deve ser invocada se WIFEXITED returnar um valor verdadeiro.

Chamadas ao sistema: wait, waitpid - macros (2/2)

Macros

WIFSIGNALED(status) - permite determinar se o processo filho terminou devido a um sinal não interceptado.

WTERMSIG(status) - permite obter o número do sinal que provocou a finalização do processo filho. Esta macro só deve ser invocada se WIFSIGNALED returnar um valor verdadeiro.

WCOREDUMP(status) - permite saber se foi gerado um ficheiro core.

WIFSTOPPED(status) - permite determinar se o processo filho que provocou o retorno se encontra congelado (*stopped*).

WSTOPSIG(status) - permite obter o número do sinal que provocou o congelamento do processo filho. Esta macro só deve ser invocada se WIFSTOPPED returnar um valor verdadeiro.

WIFCONTINUED(status) - permite determinar se o processo filho foi descongelado (*resumed*). Só aplicável para em kernels 2.6.10 ou mais recentes.

Chamada ao sistema: wait - exemplo 2

Extracto de código (processo pai)

```c
q = wait(&status);

if (q == -1) {
    /* Erro */
} else if (q > 0) {
    /* q -> pid do processo que terminou */

    if (WIFEXITED(status)) {
        /* Processo q terminou normalmente */
        /* Codigo de saida = WEXITSTATUS(status) */
    } else {
        /* Processo q terminou anormalmente */
    }
}
```

Conteúdo

Referências

Bibliografia

- **Advanced Programming in the UNIX Environment, 2nd ed.**
  W. Richard Steven, Stephen A. Rago
  `http://www.apuebook.com/`
  - Capítulo 1 - UNIX System Overview
  - Capítulo 7 - Process Environment
  - Capítulo 8 - Process Control
- **Advanced Programming in the UNIX Environment**
  W. Richard Steven
  `http://www.kohala.com/start/apue.html`
- **Linux Programming by Example: The Fundamentals**
  Arnold Robbins
  `http://authors.phptr.com/robbins/`