Processos

Aula 3 - Família de funções exec

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

Família de funções **exec**

Protótipos

```
1  #include <unistd.h>
2
3  extern char **environ;
4
5  int execl( const char *path, const char *arg, ...);
6  int execlp( const char *file, const char *arg, ...);
7  int execle( const char *path, const char *arg , ...,
8              char * const envp[]);
9
10 int execv( const char *path, char *const argv[]);
11 int execvp( const char *file, char *const argv[]);
12 int execve ( const char *path, char *const argv[],
13               char *const envp[])
```

Família de funções **exec**

Descrição

Quando um processo invoca uma das funções **exec** é completamente substituído por um novo programa que começa a sua execução a partir da função **main**. O identificador do processo não é alterado pela durante a execução da função exec devido ao facto de nenhum novo processo ser criado; a função exec substitui o processo corrente - os seus segmentos *text*, *data*, *heap* e *stack* - por um novo programa carregado de disco.

Valor de retorno

Sucesso - não retorna

Insucesso - retorna o valor **-1** e preenche a variável **errno** com o código específico do erro

Família de funções **exec**

Sufixos

- l - lista de argumentos (terminada com NULL)
- v - argumentos num array de strings (terminado com NULL)
- p - procura executável nos directórios definidos na variável de ambiente PATH (echo $PATH)
- e - variáveis de ambiente num array de strings (terminado com NULL)

Função **execl**

Exemplo 1 - execl

```
1  #include <stdio.h>
2  #include <unistd.h>
3
4  int main(void)
5  {
6      printf("Antes\n");
7
8      execl("./echoargv", "echoargv", "arg1",
9            "arg2", NULL);
10
11     printf("Depois\n");
12
13     return 0;
14 }
```

Função **execl**

Exemplo 2 - o pid é preservado

```
1  #include <stdio.h>
2  #include <unistd.h>
3
4  int main(void)
5  {
6      printf("Processo original: %d\n", getpid());
7
8      execl("./echopid", "echopid", NULL);
9
10     fprintf(stderr, "Erro\n");
11
12     return 0;
13 }
```

Função **execv**

Exemplo 3 - execv

```
1  #include <unistd.h>
2
3  int main(void)
4  {
5      char *args[] = {
6          "echoargv", "arg1", "arg2", NULL
7      };
8
9      execv("./echoargv", args);
10
11     return 0;
12 }
```

Exec

Exemplo 4

```
1  #include <stdio.h>
2  #include <unistd.h>
3
4  int main(void)
5  {
6      printf("Antes\n");
7
8      execl("/caminho/nao/existente/echoargv",
9            "echoargv", "arg1", "arg2", NULL);
10
11     printf("Depois\n");
12
13     return 0;
14 }
```

Exercícios

Executar os seguintes comandos

1. `ls -l /etc`
2. `ls -l /etc/s*.conf`

Expansão das *wildcards*

Quem é que realiza a expansão das wildcards?

Resolução do exercício 1

Exercício 1 - execlp

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4
5  int main(void)
6  {
7      execlp("ls", "ls", "-l", "/etc", NULL);
8
9      perror("execlp");
10
11     return EXIT_FAILURE;
12 }
```

Resolução do exercício 1

Exercício 1 - execvp

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4
5  char *args[] = { "ls", "-l", "/etc", NULL };
6
7  int main(void)
8  {
9      execvp( args[0], args );
10
11     perror("execvp");
12
13     return EXIT_FAILURE;
14 }
```

Resolução do exercício 2

Exercício 2 - A não expansão de *wildcards*

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

int main(void)
{
    execlp("ls", "ls", "-l", "/etc/s*.conf", NULL);

    perror("execlp");

    return EXIT_FAILURE;
}
```

Tratamento de erro

Exercício 1

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>              /* errno */
#include <string.h>             /* strerror() */

int main(void)
{
    execl("/caminho/nao/existente/echoargv",
        "echoargv", "arg1", "arg2", NULL);

    fprintf(stderr, "Errno = %d: %s\n", errno, strerror(errno));
    fprintf(stderr, "Errno = %d: %s\n", ENOENT, strerror(ENOENT));

    return EXIT_FAILURE;
}
```

Tratamento de erro

Exercício 2

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

int main(void)
{

    execl("/caminho/nao/existente/echoargv",
        "echoargv", "arg1", "arg2", NULL);

    perror("execl");

    return EXIT_FAILURE;
}
```

Uso típico

Extracto de código

```c
char *args[] = { "echoargv", "arg1", "arg2", NULL };

p = fork();

if (p == -1) {
    /* erro */
} else if (p == 0) {
    execv("./echoargv", args);
    perror("execv");
    exit(EXIT_FAILURE);
} else {
    /* ... */
    wait(NULL);
    /* ... */
}
```

Conteúdo

Imprimir identificador do processo

Exemplo 1 - echopid.c

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

int main(void)
{

    printf("Eu sou o processo %d\n", getpid());

    return 0;
}
```

Listar argumentos (1/2)

Exemplo 1 - echoargv.c

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    int i;

    for (i = 0; i < argc; i++) {
        printf("argv[%d] = %s\n", i, argv[i]);
    }

    return 0;
}
```

Listar argumentos (2/2)

Exemplo 2 - echoargv2.c

```c
#include <stdio.h>

int main(/*@unused@*/ int argc, char **argv)
{
    char **ptr;
    int i;

    for (ptr=argv, i=0; *ptr != NULL; ptr++, i++) {
        printf("argv[%d] = %s\n", i, *ptr);
    }

    return 0;
}
```

Listar variáveis de ambiente (1/2)

Exemplo 1 - echoenv.c

```
1  #include <stdio.h>
2
3  int main(int argc, char *argv[], char *envp[])
4  {
5      int i;
6
7      for (i=0; argv[i] != NULL; i++) {
8          printf("argv[%02d] > %s\n", i, argv[i]);
9      }
10
11     for (i=0; envp[i] != NULL; i++) {
12         printf("envp[%02d] > %s\n", i, envp[i]);
13     }
14     return 0;
15 }
```


Listar variáveis de ambiente (2/2)

Exemplo 2 - echoenv2.c

```
1  #include <stdio.h>
2
3  extern char **environ;
4
5  int main(void)
6  {
7      int i;
8
9      for (i=0; environ[i] != NULL; i++) {
10         printf("env[%02d] > %s\n", i, environ[i]);
11     }
12     return 0;
13 }
```


Conteúdo

Interpretadores de comandos

Interpretador de comandos

Programa que aceita comandos do teclado e os executa. A **bash** é um exemplo de um interpretador de comandos UNIX.

Funções de um interpretador de comandos

- fornecer um interface de linha de comando
- realizar redirecção de Entrada/Saída (I/O - Input/Output)
- realizar substituição de nome de ficheiros
- realizar substituição de variáveis
- fornecer uma linguagem de programação interpretada

Caracteres especiais

Caracteres especiais

white space	caracteres (espaços e tabs) usados para separar argumentos
newline	indica o fim de uma linha de comando
' " \	caracteres de citação; permitem alterar a maneira de como a shell interpreta caracteres especiais
&	no fim de um comando indica à shell para correr esse comando em *background*
< > >> ` |	caracteres de redirecção
* ? [ ] [^	substituição de caracteres em nomes de ficheiros
$	indica a presença de uma variável
;	usado para separar comandos numa mesma linha

Esqueleto de um mini-interpretador de comandos

Extracto de código

```
1  int  flag = 0;
2
3  while (!flag) {
4
5      printf("#%s#> ", getcwd(dir, DIRSIZE));
6
7      fgets(cmd, SIZE, stdin);     /* guarda o '\n' */
8
9      if (strncmp(cmd, "exit", 4) == 0) {
10         flag++;
11     } else {
12         system(cmd);
13     }
14 }
```

A implementar

Funcionalidades a implementar

Comandos externos:

- mysystem - função para invocar programas externos (em *foreground* e em *background*)

Comandos internos:

- echo
- pwd
- cd
- mkdir
- rmdir

Sugestões

Biblioteca **glibc**

- **strsep**, **strtok**, **strtok_r** - extração de tokens de strings
- **wordexp** - expansão de palavras similar à dos interpretadores de comandos

Bibliotecas **readline** e **history**

Permitem a leitura de linhas com possibilidade de edição e de invocação do histórico

Função **strsep**

Exemplo - strsep.c

```
1  int main(void)
2  {
3      char linha[] = "ls -l /tmp";
4      char delim[] = " ";
5
6      char *token, *ptr = linha;
7
8      while (ptr != NULL) {
9          token = strsep(&ptr, delim);
10         printf("%s\n", token);
11     }
12
13     return 0;
14 }
```


Função **wordexp**

Extracto de código

```
1  const char command[] = "ls -l ~/.ba*";
2  wordexp_t result;
3
4  switch ( wordexp(command, &result, 0) ) {
5      case 0:                    /* OK */
6          p = fork();
7          if (0 == p) {
8              execvp(result.we_wordv[0], result.we_wordv);
9              perror("exec");  exit(1);
10         } else if (p > 0) {
11             wait(NULL);
12         }
13         break;
14     default:
15         fprintf(stderr, "wordexp: erro.\n");
16         break;
17 }
18 wordfree(&result);
```


Conteúdo

Referências

Bibliografia

- **Advanced Programming in the UNIX Environment, 2nd ed.**
  W. Richard Steven, Stephen A. Rago
  http://www.apuebook.com/
  - Capítulo 7 - Process Environment
  - Capítulo 8 - Process Control
- **Linux Programming by Example: The Fundamentals**
  Arnold Robbins
  http://authors.phptr.com/robbins/
- **Secure Programming for Linux and Unix HOWTO**
  http://www.tldp.org/HOWTO/Secure-Programs-HOWTO/