Funções

Podemos usar algumas funções que já conhecemos (como `printf`, `malloc`, `free`, etc), então não as descreverei aqui.

Você provavelmente não usará todas essas funções, mas pelo menos você tem um lugar onde pode facilmente encontrar links para as páginas do manual. E para alguns, um exemplo de como usá-los.

lerlinha()

```
char *readline (const char *prompt);
```

A `readline()` função lê uma linha do terminal e a retorna, usando `prompt` como um prompt. Se nenhum prompt for dado como parâmetro, nenhum prompt será mostrado no terminal. A linha retornada é alocada com `malloc` e temos que liberá-la nós mesmos.

```
> lerlinha()
```

Você pode encontrar mais informações `readline()` aqui .

rl_limpar_histórico()

```
void rl_clear_history(void);
```

A `rl_clear_line()` função limpa a lista de histórico excluindo todas as entradas. A `rl_clear_line()` função libera dados que a `readline` biblioteca salva na lista de histórico.

rl_em_nova_linha()

```
int rl_on_new_line(void);
```

A `rl_on_new_line()` função informa à rotina de atualização que avançamos para uma nova linha vazia, geralmente usada após a saída de uma linha.

rl_substituir_linha()

rl_reexibição()

```
int rl_redisplay(void);
```

Altere `rl_redisplay()` o que é exibido na tela para refletir o conteúdo atual de `rl_line_buffer`.

adicionar_histórico()

```
void add_history(char *s);
```

A `add_history()` função salva a linha passada como parâmetro no histórico para que ela possa ser recuperada posteriormente no terminal (como pressionar a seta para cima no bash).

obtercwd()

```
#include <unistd.h>
char *getcwd(char *buf, size_t size);
```

O `getcwd()` retorna uma string terminada em nulo contendo o nome do caminho absoluto que é o diretório de trabalho atual do processo de chamada. O nome do caminho é retornado como o resultado da função e por meio do argumento `buf`.

> Exemplo getcwd()

Você pode encontrar mais informações `getcwd()` aqui .

chdir()

```
#include <unistd.h>
int chdir(const char *path);
```

`chdir()` altera o diretório de trabalho atual do processo de chamada para o diretório especificado em `path`.

> Exemplo de chdir()

Você pode encontrar mais informações `chdir()` aqui.

stat() e lstat() e fstat()

```
#include <sys/stat.h>
int stat(const char *restrict pathname, struct stat *restrict statbuf);
int lstat(const char *restrict pathname, struct stat *restrict statbuf);
int fstat(int fd, struct stat *statbuf);
```

Essas funções retornam informações sobre um arquivo na estrutura apontada por `statbuf`.

Você pode encontrar informações mais detalhadas sobre essas funções aqui.

abrirdir()

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *name);
```

A `opendir()` função abre um fluxo de diretório correspondente ao nome do diretório e retorna um ponteiro para o fluxo de diretório. O fluxo é posicionado na primeira entrada do diretório.

Você pode encontrar mais informações sobre a `opendir` função aqui.

leiadir()

```
#include <dirent.h>
struct dirent *readdir(DIR *dirp);
```

A `readdir()` função retorna um ponteiro para uma `dirent` estrutura que representa a próxima entrada de diretório no fluxo de diretório apontado por `dirp`. Ela retorna `NULL` ao atingir o fim do fluxo de diretório ou se um erro ocorreu.

Você pode encontrar mais informações `readdir` aqui.

fechadoir()

```
#include <sys/types.h>
#include <dirent.h>
int closedir(DIR *dirp);
```

A `closedir()` função fecha o fluxo de diretório associado a `dirp`. Uma chamada bem-sucedida para `closedir()` também fecha o descritor de arquivo subjacente associado a `dirp`. O descritor de fluxo de diretório `dirp` não fica disponível após essa chamada.

Você pode encontrar mais informações `closedir` aqui .

erro de entrada()

```
#include <string.h>
char *strerror(int errnum);
```

A `strerror()` função retorna um ponteiro para uma string que descreve o código de erro passado no argumento errnum. Esta string não deve ser modificada pelo aplicativo, mas pode ser modificada por uma chamada subsequente para `strerror()` ou `strerror_l()`. Nenhuma outra função de biblioteca, incluindo `perror()`, modificará esta string.

Você pode encontrar mais informações `strerror` aqui .

erro()

```
#include <stdio.h>
void perror(const char *s);
```

A `perror()` função produz uma mensagem de erro padrão descrevendo o último erro encontrado durante uma chamada para uma função de sistema ou biblioteca.

Você pode encontrar mais informações `perror` aqui .

é seguro()

```
#include <unistd.h>
int isatty(int fd);
```

A `isatty` função testa se `fd` é um terminal.

Você pode encontrar mais informações `isatty` aqui .

nome_do_tty()

```
#include <unistd.h>
char **ttyname(int fd);
```

A `ttyname()` função retorna um ponteiro para o caminho terminado em nulo do dispositivo terminal que está aberto no descritor de arquivo `fd` ou `NULL` em caso de erro.

Você pode encontrar mais informações `ttyname()` aqui .

slot tty()

```
#include <unistd.h>
int ttyslot(void);
```

Esta é uma função legada com alguma história de fundo. Você pode ler tudo sobre ela e como ela funciona aqui .

ioctl()

```
#include <sys/ioctl.h>
int ioctl(int fd, unsigned long request, ...);
```

A `ioctl()` chamada do sistema manipula os parâmetros de dispositivo subjacentes de um arquivo especial. Você pode encontrar informações mais detalhadas aqui .

obterv()

```
#include <stdlib.h>
char *getenv(const char *name);
```

A `getenv()` função pesquisa a lista de ambientes para encontrar o nome da variável de ambiente e retorna um ponteiro para a string de valor correspondente.

Você pode encontrar mais informações `getenv()` aqui .

tcsetattr()

```
#include <termios.h>
int tcsetattr(int fildes, int optional_actions, const struct *termios_p);
```

A `tcsetattr()` função deve definir os parâmetros associados ao terminal referenciado pelo descritor de arquivo aberto `fildes` a partir da `termios` estrutura referenciada por `termios_p` conforme descrito aqui .

tcgetattr()

```
#include <termios.h>
int tcgetattr(int fildes, struct termios *termios_p);
```

A `tcgetattr()` função deve obter os parâmetros associados ao terminal referenciado por `fildes` e armazená-los na `termios` estrutura referenciada por `termios_p` .

Você pode encontrar informações mais detalhadas aqui .

objetivo()

```
#include <curses.h>
#include <term.h>
int tgetent(char *bp, const char *name);
int tgetflag(char *id);
int tgetnum(char *id);
char *tgetstr(char *id, char **area);
char *tgoto(const char *cap, int col, int row);
int tputs(const char *str, int affcnt, int (*putc)(int));
```

Essas rotinas são incluídas como um auxílio de conversão para programas que usam a `termcap` biblioteca. Você pode encontrar mais informações sobre todas elas aqui .

Anterior
Entenda o Minishell

Próximo
Construindo a coisa

Última atualização6 meses atrás