Funções

Podemos usar alguma função que já conhecemos (como , , , etc), então não vou descrevê-las aqui. `printf` `malloc` `free`

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

readline()

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

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

readline()

```c
#include <stdio.h>
#include <readline/readline.h>
#include <readline/history.h>

int main(void)
{
    char *rl;
    rl = readline("Prompt > ");
    printf("%s\n", rl);
    return (0);
}
```

Compilar este programa e executá-lo resultará no seguinte.

```
$> ./minishell
Prompt > Hi ! How are you ?
Hi ! How are you ?
$>
```

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

rl_clear_history()

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

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

rl_on_new_line()

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

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

rl_replace_line()

Não encontrei nenhuma informação sobre essa função.

rl_redisplay()

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

A alteração do que é exibido na tela para refletir o conteúdo atual do . `rl_redisplay()` `rl_line_buffer`

add_history()

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

A 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). `add_history()`

getcwd()

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

O retorna uma cadeia de caracteres 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 . `getcwd()` `buf`

Exemplo de getcwd()

```c
#include <unistd.h>
#include <stdio.h> // for printf

int main(void)
{
    char *pwd;

    pwd = getcwd(NULL, 0);
    printf("pwd: %s\n", pwd);
    return (0);
}
```

```
$> pwd: /Users/saeby/Documents/tmp
```

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

chdir()

```c
#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()

```c
#include <unistd.h>
#include <stdio.h> // for printf

int main(void)
{
    char *pwd;

    pwd = getcwd(NULL, 0);
    printf("pwd before chdir: %s\n", pwd);
    chdir("/Users/saeby/Documents/42/minishell");
    pwd = getcwd(NULL, 0);
    printf("pwd after chdir: %s\n", pwd);
    return (0);
}
```

```
$> pwd before chdir: /Users/saeby/Documents/tmp
$> pwd after chdir: /Users/saeby/Documents/42/minishell
```

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

stat() & lstat() & fstat()

```c
#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.

opendir()

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

A 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 no diretório. `opendir()`

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

readdir()

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

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

Você pode encontrar mais informações aqui. `readdir`

Fechado()

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

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

Você pode encontrar mais informações aqui. `closedir`

strerror()

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

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

Você pode encontrar mais informações aqui. `strerror`

perror()

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

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

Você pode encontrar mais informações aqui. `perror`

isatty()

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

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

Você pode encontrar mais informações aqui. `isatty`

ttyname()

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

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

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

ttyslot()

```
#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 chamada do sistema manipula os parâmetros subjacentes do dispositivo de arquivos especiais. Você pode encontrar informações mais detalhadas aqui. `ioctl()`

getenv()

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

A função pesquisa a lista de ambiente para localizar o nome da variável de ambiente e retorna um ponteiro para a cadeia de caracteres de valor correspondente. `getenv()`

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

tcsetattr()

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

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

tcgetattr()

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

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

Você pode encontrar informações mais detalhadas aqui.

tgetent()

```
#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 biblioteca. Você pode encontrar mais informações sobre todos eles aqui. `termcap`

Entenda o Minishell

Construindo a coisa

Última atualização há 6 meses