Tratamento de erros

A linguagem C não é muito eloquente ao informar erros de execução. Contudo, existem mecanismos para detectar e informar sobre erros de execução, que são apresentados nesta página.

errno, perror e strerror

As chamadas de sistema e funções em C informam erros de execução retornando um valor inteiro negativo (como faz o `printf`) ou um ponteiro nulo (como faz o `malloc`) e ajustando uma variável predefinida chamada `errno` para indicar mais precisamente o erro ocorrido.

Os códigos de erro indicados pela variável `errno` são valores inteiros positivos que correspondem a macros e *strings* definidas na biblioteca C (no arquivo `errno.h` e outros). Em Linux, esses códigos, macros e strings podem ser consultados através do comando `errno` no terminal:

```
$ errno -l
EPERM 1 Operation not permitted
ENOENT 2 No such file or directory
ESRCH 3 No such process
EINTR 4 Interrupted system call
EIO 5 Input/output error
ENXIO 6 No such device or address
E2BIG 7 Argument list too long
ENOEXEC 8 Exec format error
EBADF 9 Bad file descriptor
ECHILD 10 No child processes
EAGAIN 11 Resource temporarily unavailable
ENOMEM 12 Cannot allocate memory
EACCES 13 Permission denied
... (+100 linhas)
```

Dessa forma, o erro ocorrido em uma chamada de função pode ser tratado de forma mais precisa pelo programa, como mostra este exemplo:

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

...

FILE* file ;

// abre arquivo de dados
file = fopen ("results.dat", "r+") ;

// erro na abertura do arquivo
if (! file)
{
  switch (errno)
  {
    case EPERM   : printf ("Operação não permitida\n") ; break ;
    case ENOENT  : printf ("Arquivo não encontrado\n") ; break ;
    case EACCESS : printf ("Permissão negada\n") ;       break ;
    default:
      printf ("Outro erro\n") ;
```

```
  }
  exit (1) ;
}
```

Para facilitar a geração de mensagens de erro, a função `perror (char *msg)` imprime na saída de erro (`stderr`) a string `msg` seguida da *string* de descrição do erro encontrado:

```
// abre arquivo de dados
file = fopen ("results.dat", "r+") ;

// erro na abertura do arquivo
if (! file)
{
  perror ("Erro no fopen") ;
  exit (1) ;
}
```

Caso o arquivo `results.dat` não exista, o código acima irá gerar esta saída:

```
Erro no fopen: No such file or directory
```

Para produzir mensagens de erro mais elaboradas, pode-se usar a função `strerror (int num)`, que retorna uma *string* com a descrição do erro cujo código é `num`.

O valor de `errno` (e portanto a saída de `perror`) se refere à última função chamada e pode mudar após a próxima chamada de função. Por isso, é importante testar `errno` imediatamente após o retorno da função, antes de chamar outra função.

Funções exit, abort e assert

A execução de um programa em C pode ser finalização pela conclusão/retorno da função `main` ou por chamadas a funções específicas como `exit`, `abort` ou `assert`, discutidas nesta seção.

Quando um programa em C encerra, ele devolve um status de saída (*exit status*) ao processo que o lançou, que pode ser o terminal/shell, um script ou outro programa. No terminal ou em scripts, o status de saída de um programa que acabou de encerrar pode ser consultado através da variável de ambiente `$?`.

A função `exit (int status)` encerra o programa e devolve o valor `status & 0xFF` ao processo que o lançou. Ao encerrar, todos os arquivos abertos são fechados e sincronizados, para evitar a perda de dados. Os *sockets* de rede, áreas de memória dinâmica e outros recursos alocados também são liberados.

Exemplo:

```
// abre arquivo de dados
file = fopen ("results.dat", "r+") ;

// erro na abertura do arquivo
if (! file)
{
  perror ("Erro no fopen") ;
  exit (2) ;
}
```

O status de saída do programa pode ser consultado no shell:

```
$ programa
Erro no fopen: No such file or directory
$ echo $?
2
```

Pode-se usar a função `atexit` para definir uma ou mais funções a executar automaticamente quando um programa encerra por `exit` ou por retorno da função `main`:

atexit.c

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

// mensagem de saída
void msg_saida ()
{
  printf ("O programa encerrou\n") ;
}

int main ()
{
  // registra função a executar no "exit"
  atexit (msg_saida) ;

  printf ("O programa iniciou\n") ;
}
```

Pode-se encerrar de forma mais “abrupta” um programa através da função `abort ()`, que não sincroniza os dados dos arquivos abertos pelo programa, não executa as funções definidas por `atexit` e retorna um status de saída não-nulo ao processo-pai.

Por fim, `assert` é uma função que verifica uma asserção (condição) e encerra o programa se ela for falsa (nula). Ao encerrar, uma mensagem de erro indicando a localização da asserção violada é gerada, o que facilita localizar problemas.

```c
#include <assert.h>

int main()
{
  ...
  assert (i >= 0) ;   // só continua se i >= 0
  ...
}
```

Se a condição `i >= 0` for falsa ao ser avaliada por `assert`, o programa será interrompido e a seguinte mensagem será gerada, indicando o arquivo (`program.c`), a linha (6), a função (`main`) e a asserção violada:

```
a.out: program.c:6: main: Assertion `i >= 0' failed.
```