Universidade Federal do Rio de Janeiro
Disciplina: Tópicos Especiais em Sistemas de Computação
Trabalho 1
Apresentação em Laboratório: Em 25/04/2024

Exercício Prático #1

Escreva, em C, a função **`isopen`**, cujo protótipo é:

```
int isopen (int fd);
```

A função recebe como parâmetro um número inteiro e deve verificar se o descritor de arquivo correspondente a esse número está aberto, retornando 1 se estiver ou 0 se não estiver.

Para testar se um dado descritor está aberto, você deverá utilizar alguma chamada ao sistema relativa à manipulação de arquivos, analisando seu valor de retorno e, eventualmente, o código de erro informado na variável **`errno`**. Escolha a chamada ao sistema menos onerosa possível e que não produza efeitos colaterais indesejáveis.

Codifique a função **`isopen`** e transcreva o programa seguinte, que a utiliza, no arquivo **`nopen.c`**:

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

extern int isopen (int fd);

int main (void)
{
      int  nopen, fd;

      for (nopen = fd = 0; fd < getdtablesize (); fd++)
      {
            if (isopen (fd))
                  nopen++;
      }

      printf ("Existem %d descritores abertos\n", nopen);

      return (0);

}     /* end main */
```

(A chamada ao sistema **`getdtablesize()`** retorna o número máximo de descritores que um processo pode manter abertos simultaneamente).

Sugestão: antes de escrever a função, leia a página do manual relativa ao uso da variável **`errno`**: **`man errno`**. Observe que, quando uma chamada ao sistema retorna *com sucesso*, o valor desta variável *não é alterado*.

Não se esqueça de monitorar o seu programa para que se possa saber o que está acontecendo internamente.

Exercício Prático #2

O percurso de um diretório é um procedimento no qual informações sobre as diversas entradas desse diretório são disponibilizadas a um programa.

A existência de funções específicas para percorrer diretórios (**`opendir`**, **`readdir`** e **`closedir`**) deve-se ao fato de diferentes tipos de sistemas de arquivos serem suportados, cada um deles possuindo um formato próprio de diretório. Assim, antes de serem divulgadas ao programa que as solicita, as informações sobre as diversas entradas integrantes de um diretório precisam ser compatibilizadas pelo núcleo do sistema e reunidas na **`struct dirent`**.

Para nossos propósitos, são relevantes apenas duas informações contidas em **`<dirent.h>`**:

- A constante **`NAME_MAX`** (em algumas plataformas **`MAXNAMLEN`**), que limita o comprimento do nome de uma entrada;
- O membro **`d_name`** da **`struct dirent`** – um vetor de **`NAME_MAX + 1`** caracteres – onde nomes de entradas são retornados.

O percurso inicia-se com a chamada a **`opendir`**, que recebe o caminho para o diretório e retorna um descritor, do tipo opaco **`DIR *`**. Se o retorno for **`NULL`**, teremos em **`errno`** a causa do erro; do contrário, o descritor retornado (um endereço válido) será usado na manipulação subseqüente.

A varredura do diretório dá-se em uma malha (*loop*) na qual a função **`readdir`** é sucessivamente invocada. Ela recebe como parâmetro o descritor do diretório (retornado por **`opendir`**) e retorna o endereço de uma **`struct dirent`**, preenchida com as informações sobre a entrada corrente. O campo **`d_name`** armazena o nome da entrada na forma de uma cadeia (portanto, *com* o terminador **`'\0'`** ao final). A função **`readdir`** retorna **`NULL`** quando não há mais entradas a informar. Observe que `"."` e `".."` são também retornadas, em geral (*mas não garantidamente*) nas duas primeiras invocações a **`readdir`**.

A função **`closedir`** encerra a manipulação do diretório.

A seguir, o percurso de um diretório é implementado na função **`walk_dir`**.

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

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>

int walk_dir (const char *path, void (*func) (const char *))
{
     DIR           *dirp;
     struct dirent *dp;
     char          *p, *full_path;
     int           len;

     /* abre o diretório */

     if ((dirp = opendir (path)) == NULL)
          return (-1);

     len = strlen (path);

     /* aloca uma área na qual, garantidamente, o caminho caberá */

     if ((full_path = malloc (len + NAME_MAX + 2)) == NULL)
     {
          closedir (dirp);
          return (-1);
     }

     /* copia o prefixo e acrescenta a '/' ao final */

     memcpy (full_path, path, len);

      p = full_path + len; *p++ = '/';     /* deixa "p" no lugar certo! */

     while ((dp = readdir (dirp)) != NULL)
     {
          /* ignora as entradas "." e ".." */

          if (strcmp (dp->d_name, ".") == 0 || strcmp (dp->d_name, "..") == 0)
               continue;

          strcpy (p, dp->d_name);

          /* "full_path" armazena o caminho */

          (*func) (full_path);
     }

     free (full_path);

     closedir (dirp);

     return (0);

}    /* end walk_dir */
```

Sua tarefa é utilizar **`walk_dir`** para construir o programa **`icount`**, que informa a quantidade de INODEs de um determinado tipo em cada um dos diretórios cujos caminhos são dados como argumentos:

`icount [-rdlbc] [<dir> ...]`

onde os modificadores **`rdlbc`** têm os seguintes significados:

- **`-r:`** arquivo regular (**`S_IFREG`**)
- **`-d:`** diretório (**`S_IFDIR`**)
- **`-l:`** elo simbólico (**`S_IFLNK`**)
- **`-b:`** dispositivo estruturado (**`S_IFBLK`**)
- **`-c:`** dispositivo não-estruturado **`(S_IFCHR`**)
- **`<dir>:`** caminho para diretório

Os modificadores são *mutuamente exclusivos*: no máximo um deles pode ser mencionado em uma invocação do programa. Na ausência de modificadores, deve ser assumido **`-r`**. Na ausência de caminhos para diretórios, deve ser assumido o diretório corrente. Utilize a função **`getopt`** da biblioteca padrão para obter e tratar educadamente os modificadores do programa.

Exemplos:

`icount /etc`	conta arquivos regulares em **`"/etc"`**
`icount -d /home`	conta subdiretórios em **`"/home"`**
`icount -b /dev`	conta dispositivos estruturados em **`"/dev"`**
`icount`	conta arquivos regulares em **`"."`**

Não se esqueça de monitorar o seu programa para que se possa saber o que está acontecendo internamente.