Ficha 8 – Meta-informação e diretórios

Tópicos abordados:

– Acesso à meta-informação sobre ficheiros (*stat*)
– Leitura do conteúdo de diretórios (*opendir*/*readdir*/*closedir*)
– Exercícios

Duração prevista: 1 aula

1 Acesso à meta-informação sobre ficheiros

A informação mais detalhada respeitante a cada entrada do sistema de ficheiros (ficheiro, diretório, ligação, etc.) é denominada "meta-informação". O acesso a essa informação faz-se através da chamada ao sistema `stat` (`man 2 stat`). O protótipo da função `stat` é o seguinte:

```
int stat(const char *path, struct stat *buf);
```

A função `stat` preenche uma estrutura do tipo `struct stat` com os *metadados* da entrada especificada através do parâmetro `path`. Em caso de erro (e.g., a entrada especificada por `path` não existe), é devolvido o valor -1, com o código de erro apropriado a ser atribuído à variável `errno`.

A estrutura `struct stat` possui diversos campos relativos aos *metadados* de uma entrada que são preenchidos após a chamada com sucesso da função `stat`. A descrição detalhada de cada um dos campos encontra-se na página de manual (`man 2 stat`). A seguir encontram-se listados os campos preenchidos pela função `stat`. Desde a versão

2.6 do *kernel* do Linux que os campos referentes a *timestamps* passaram a incluir a precisão de nano-segundos:

```
struct stat {
    dev_t     st_dev;      /* ID of device containing file */
    ino_t     st_ino;      /* inode number */
    mode_t    st_mode;     /* protection */
    nlink_t   st_nlink;    /* number of hard links */
    uid_t     st_uid;      /* user ID of owner */
    gid_t     st_gid;      /* group ID of owner */
    dev_t     st_rdev;     /* device ID (if special file) */
    off_t     st_size;     /* total size, in bytes */
    blksize_t st_blksize;  /* blocksize for filesystem I/O */
    blkcnt_t  st_blocks;   /* number of 512B blocks allocated */

/* Since Linux 2.6, the kernel supports nanosecond
   precision for the following timestamp fields. */

    struct timespec st_atim;  /* time of last access */
    struct timespec st_mtim;  /* time of last modification */
    struct timespec st_ctim;  /* time of last status change */

   #define st_atime st_atim.tv_sec  /* Backward compatibility */
   #define st_mtime st_mtim.tv_sec
   #define st_ctime st_ctim.tv_sec
};
```

Algumas versões de UNIX podem ter campos adicionais na `struct stat`, sendo que o uso desses campos extra num programa limita necessariamente a portabilidade do código.

Exemplo:

```
#define ERR_ARGS 1
#define ERR_SYS_CALL 2

int main(int argc, char const *argv[]) {
    if (argc > 2) {
        fprintf(stderr, "Usage: %s [path]\n", argv[0]);
        return ERR_ARGS;
    }
    const char *path = argc == 2 ? argv[1] : ".";

    struct stat metadata;
    int status = stat(path, &metadata);

    if (status == -1) {
        fprintf(stderr,"stat() failed for entry: %s (%s)\n",
           path, strerror(errno));
        return ERR_SYS_CALL;
    }
    print_metadata(&metadata);
    return 0;
}
```

```
const char *mode_to_str(mode_t mode) {
    if (S_ISREG(mode)) {
        return "File";
    }

    if (S_ISDIR(mode)) {
        return "Dir";
    }

    if (S_ISLNK(mode)) {
        return "Link";
    }

    return "Other";
}

void print_metadata(const struct stat *metadata_ptr){
    printf("st_dev = %u.%u\n",
       major(metadata_ptr->st_dev),
       minor(metadata_ptr->st_dev));
    printf("st_ino = %ld\n", metadata_ptr->st_ino);
    printf("st_mode = %x (%s)\n",
       metadata_ptr->st_mode,
       mode_to_str(metadata_ptr->st_mode));
    printf("st_nlink (number of hardlinks) = %zu\n",
       metadata_ptr->st_nlink);
    printf("st_uid = %d\n", metadata_ptr->st_uid);
    printf("st_gid = %d\n", metadata_ptr->st_gid);
    printf("st_rdev = %d\n", (int)metadata_ptr->st_rdev);
    printf("st_size (bytes) = %zu\n", metadata_ptr->st_size);
    printf("st_blksize = %zu\n", metadata_ptr->st_blksize);
    printf("st_blocks = %zu (512B each)\n",
       metadata_ptr->st_blocks);

    /* ctime returns a string terminated with a newline */
    char buffer[128];
    printf("st_atime (last accessed) = %s",
       ctime_r(&(metadata_ptr->st_atime), buffer));
    printf("st_mtime (last modified) = %s",
        ctime_r(&(metadata_ptr->st_mtime), buffer));
    printf("st_ctime (last changed) = %s",
        ctime_r(&(metadata_ptr->st_ctime), buffer));
}
```

Listagem 1 – Programa que mostra os metadados de uma entrada (por omissão é utilizada a diretoria corrente)

Lab 1

Compile o programa da listagem anterior e execute-o. Experimente passar diferentes tipos de ficheiros.

Exercício 1

Elabore, fazendo uso da linguagem C, a aplicação **show-type** que, recebendo um ou mais nomes de ficheiros/diretórios através da linha de comando, deve determinar, para cada um deles, se se trata de ficheiro ou de diretório. Considere o seguinte exemplo de execução:

```
./show-type file1 file2 dir1 file3
'file1': FILE
'file2': FILE
'dir1': DIR
'file3': FILE
```

Sugestão: campo `st_mode` da estrutura `struct stat`.

Exercício 2

Elabore, com recurso à linguagem C, a aplicação **show-time** que, recebendo como único parâmetro o nome da entrada do sistema de ficheiros a analisar (ficheiros, diretório, etc.), mostra a data/hora em que foi efetuada a última modificação. Considere o seguinte exemplo de execução:

```
./show-time   /tmp/tmp.txt
/tmp/tmp.txt: 2016-05-18 23:07:59.421163995 +0100
```

Sugestão: o campo `st_mtime` da estrutura `struct stat` e a função `localtime_r`.

2 Leitura do conteúdo de diretórios

O acesso ao conteúdo de um diretório é feito através da metodologia *opendir/readdir/closedir* (ver listagem 2). Concretamente, o diretório é aberto com a função `opendir`, sendo devolvido um descritor de diretório. Seguidamente, itera-se o conteúdo do diretório através da função `readdir`, indicando-se como parâmetro o descritor do diretório previamente obtido através da função `opendir`. Cada chamada à função `readdir` devolve uma entrada do diretório (ficheiro, subdiretório, *link*, etc.), de tipo `struct dirent*`, sendo que se deve chamar `readdir` dentro de um ciclo. O campo `d_name` desta estrutura corresponde ao nome da entrada em análise. Quando já tiverem sido percorridas todas as entradas do diretório, a função `readdir` devolve NULL.

Seguidamente, deve-se encerrar o diretório através da chamada à função `closedir`.

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

DIR *opendir(const char *name);

struct dirent *readdir(DIR *dirp);
int readdir_r(DIR *dirp, struct dirent *entry,
      struct dirent **result);

int closedir(DIR *dirp);
```

Listagem 2 – protótipos das funções opendir, readdir, closedir

Exercício 3

Elabore, com recurso à linguagem C, a aplicação **list-cwd**. O programa deve mostrar os nomes de todas as entradas que existem no diretório no qual é executado o programa. Considere o seguinte exemplo de execução:

```
Entry: '.'
Entry: '..'
Entry: 'args.c'
Entry: 'args.ggo'
Entry: 'args.h'
Entry: 'args.o'
Entry: 'debug.c'
Entry: 'debug.h'
Entry: 'debug.o'
Entry: 'list-cwd'
Entry: 'main.c'
Entry: 'main.o'
Entry: 'makefile'
Entry: 'memory.c'
Entry: 'memory.h'
Entry: 'memory.o'
Iteration terminated. 16 entries found
```

Exercício 4

Com base no exercício anterior, elabore a aplicação **list-files-cwd** que deve mostrar somente as entradas do diretório corrente (".") que são ficheiros.

Sugestão: usar a função `stat` para determinar se uma entrada é um ficheiro.

3 Bibliografia

- "Files & Directories", Capítulo 8 dos apontamentos das aulas teórico-práticas, Patrício Domingues.

- "File and Directory Management", Chapter 8, Linux System Programming, Robert Love, 2nd Edition, O'Reilly, 2013

- "Travessia de uma árvore de diretórios usando recursividade", Patrício Domingues, Vítor Carreira, Carlos Grilo, Revista Programar, nº51, Dezembro 2015

(http://www.revista-programar.info/anuncios/revista-programar-no-51-dezembro-de-2015/)