TRABALHO PRÁTICO Nº 1

Ferramenta para estimar a utilização de espaço em disco

Sumário

Pretende-se desenvolver uma ferramenta para sumariar a utilização de espaço em disco de um ficheiro ou diretório. No caso de um diretório, a informação a disponibilizar deve contemplar ficheiros e subdiretórios que, eventualmente, nele estejam contidos.

Objetivos

Familiarizar os estudantes com a programação de sistema, em ambiente Unix/Linux, envolvendo a criação de processos, a comunicação entre estes usando sinais e *pipes* (sem nome), e, a obtenção de informação relativa ao sistema de ficheiros.

Descrição Geral

A ferramenta a desenvolver (**`simpledu`**) deve ter como referência o comando **`du`** - *estimate file space usage*, o qual apresenta informação sobre o espaço em disco utilizado por ficheiros e diretórios.

O espaço em disco ocupado por um ficheiro (ou diretório) pode não corresponder exatamente à quantidade de informação que o ficheiro (ou diretório) contém: a alocação em disco é efetuada em *blocos* de tamanho mínimo, pelo que, por exemplo, um ficheiro de 10 *bytes* pode ocupar 4096 *bytes* em disco. Infelizmente, há também grande confusão relativamente à informação prestada por diversas ferramentas Unix/Linux a este respeito:

- o tamanho dos blocos de disco aumentou ao longo do tempo (desde 512 *bytes* há mais de 30 anos até aos atualmente muito utilizados 4096 *bytes*), mas as funções **`stat()`**, **`fstat()`** e **`lstat()`** continuam a reportar o número de blocos (**`st_blocks`**) como tendo 512 *bytes*;
- por omissão, muitas ferramentas apresentam valores cuja unidade é um bloco de 1024 *bytes* (que corresponde a blocos internos ao sistema operativo Linux, mas não a blocos do disco);
- os múltiplos das unidades podem ser de base 1000 (unidades S.I., prefixo decimal) ou de base 1024 (unidades ISO/IEC, prefixo binário); por exemplo, 1 MB devia representar 1000000 *bytes* e nunca 1048576 *bytes* (1 MiB[*]).

Por isso, na ferramenta proposta para este trabalho, deve tentar-se reproduzir a informação apresentada pelo comando **`du`** correntemente instalado. Por omissão, o comando **`du`**:

- apresenta o espaço ocupado em número de blocos de 1024 *bytes*;
- apenas lista diretórios;
- não segue *links* simbólicos,
- contabiliza uma única vez cada ficheiro;
- apresenta de forma cumulativa o tamanho de subdiretórios e ficheiros incluídos;
- não restringe os níveis de profundidade na estrutura de diretórios.

Esta forma de operação deve ser reproduzida pela ferramenta **`simpledu`**, mas considerando apenas

[*] Os prefixos binários Ki (*kibi; 2^{10}*), Mi (*mebi; 2^{20}*), Gi (*gibi; 2^{30}*), Ti (*tebi*), Pi (*pebi*), Ei (*exbi*), Zi (*zebi*) e Yi (*yobi*) são análogos aos prefixos decimais K (kilo; 10^3), M (mega; 10^6), G (giga; 10^9), T (tera), P (peta), E (exa), Z (zetta) e Y (yotta).

um subconjunto das opções de invocação disponibilizadas pelo comando **`du`**:

- **`-a, --all`** – a informação exibida diz respeito também a ficheiros;
- **`-b, --bytes`** – apresenta o número real de *bytes* de dados (ficheiros) ou alocados (diretórios);
- **`-B, --block-size=SIZE`** – define o tamanho (*bytes*) do bloco para efeitos de representação;
- **`-l, --count-links`** – contabiliza múltiplas vezes o mesmo ficheiro;
- **`-L, --dereference`** – segue *links* simbólicos;
- **`-S, --separate-dirs`** – a informação exibida não inclui o tamanho dos subdiretórios;
- **`--max-depth=N`** – limita a informação exibida a N (0,1, …) níveis de profundidade de diretórios.

Não se pretende que seja mantido um registo dos ficheiros ou diretórios já processados (por exemplo, para evitar a sua contabilização por múltiplas vezes ou para impedir a existência de ciclos através de *links* simbólicos), por isso, deve ser assumido que a opção **`-l`** (ou **`--count-links`**) é sempre usada.

A estrutura do programa **`simpledu`** é deixada ao critério de cada grupo, tendo somente de obedecer a alguns requisitos funcionais e arquiteturais que serão apresentados em detalhe mais à frente; nomeadamente, cada processo deve analisar somente um diretório e é responsável por criar um processo por cada um dos subdiretórios que, eventualmente, nele possam estar contidos.

INFORMAÇÃO SUPLEMENTAR

O tamanho dos blocos usados num dado disco, disponível no campo **`st_blksize`** da estrutura **`stat`** (**`struct stat`**), pode ser também obtido usando o comando

```
$ stat -f fich_dir
```

onde **`fich_dir`** é um qualquer ficheiro ou diretório do disco.

De notar que a informação fornecida é mais vasta do que o tamanho dos blocos de um dado disco e que o comando tem o mesmo nome que a chamada ao sistema que disponibiliza informação sobre um dado ficheiro ou diretório (confrontar **`man 1 stat`** e **`man 2 stat`**).

Exemplos de Invocação

Os exemplos apresentados consideram a estrutura de ficheiros e diretórios apresentada através do comando **`tree .`** O ficheiro **`symlnk`** é um *link* simbólico.

```
$ tree .
.
├── D1
│   ├── Da
│   │   └── F3.BIN
│   ├── Db
│   │   ├── archive.zip
│   │   ├── Dx
│   │   │   ├── doc.pdf
│   │   │   ├── output.log
│   │   │   └── symlnk -> ../../../D2
│   │   └── pkg.deb
│   ├── dep.d
│   └── src.c
├── D2
│   └── F2.BIN
└── f.txt
```

Os exemplos que se seguem pretendem fornecer uma breve amostra dos resultados esperados quando as diversas opções de invocação estão ativas; a opção de contagem por múltiplas vezes de um mesmo ficheiro encontra-se sempre ativa [**`-l`**]. A análise de comportamento do comando **`du`** para outros conjuntos de opções de invocação, incluindo argumentos inválidos, repetidos ou incompatíveis, é deixada ao cuidado de cada grupo mediante recurso ao manual ou a exemplos próprios.

$ du -l .	du -l . -b	du -l . -b -S	du -l . -b -S --max-depth=2
8 ./D2 8 ./D1/Da 56 ./D1/Db/Dx 2736 ./D1/Db 2756 ./D1 2772 .	6144 ./D2 4352 ./D1/Da 52709 ./D1/Db/Dx 2796030 ./D1/Db 2804515 ./D1 2815681 .	6144 ./D2 4352 ./D1/Da 52709 ./D1/Db/Dx 2743321 ./D1/Db 4133 ./D1 5022 .	6144 ./D2 4352 ./D1/Da 2743321 ./D1/Db 4133 ./D1 5022 .

$ du -l . -B 1	du -l . -a	du -l . -a -L
8192 ./D2 8192 ./D1/Da 57344 ./D1/Db/Dx 2801664 ./D1/Db 2822144 ./D1 2838528 .	4 ./D2/F2.BIN 8 ./D2 4 ./f.txt 4 ./D1/dep.d 4 ./D1/Da/F3.BIN 8 ./D1/Da 268 ./D1/Db/archive.zip 0 ./D1/Db/Dx/symlnk 20 ./D1/Db/Dx/output.log 32 ./D1/Db/Dx/doc.pdf 56 ./D1/Db/Dx 2408 ./D1/Db/pkg.deb 2736 ./D1/Db 4 ./D1/src.c 2756 ./D1 2772 .	4 ./D2/F2.BIN 8 ./D2 4 ./f.txt 4 ./D1/dep.d 4 ./D1/Da/F3.BIN 8 ./D1/Da 268 ./D1/Db/archive.zip 4 ./D1/Db/Dx/symlnk/F2.BIN 8 ./D1/Db/Dx/symlnk 20 ./D1/Db/Dx/output.log 32 ./D1/Db/Dx/doc.pdf 64 ./D1/Db/Dx 2408 ./D1/Db/pkg.deb 2744 ./D1/Db 4 ./D1/src.c 2764 ./D1 2780 .

Requisitos Funcionais

A ferramenta **simpledu**, como exemplificado, deve disponibilizar informação relativa à utilização de disco por parte de ficheiros e diretórios, considerando sempre que a opção **-l** (ou **--count-links**) está ativa. A linha de comandos genérica, sem restrições relativamente à ordem dos argumentos, será:

simpledu -l [path] [-a] [-b] [-B size] [-L] [-S] [--max-depth=N]

Apesar de ser referida apenas uma alternativa por opção de invocação, caso exista uma segunda, a implementação da ferramenta deve suportar ambas (por exemplo, **-a** ou **--all**). Por questões de simplicidade, não é requerido que múltiplas opções possam ser especificadas usando um único argumento (por exemplo, **-LabS**).

O processo-pai (primeiro a ser executado) deve sempre aguardar pela terminação de todos os processos-filho antes de terminar a sua execução (ver requisitos arquiteturais).

Apresentação de resultados

Os resultados devem ser apresentados replicando o formato de saída do comando **du**: existe um carácter de tabulação entre o número de *bytes* ou blocos e o caminho para o ficheiro ou diretório.

Geração de registos de execução

Para facilitar a análise, desenvolvimento e avaliação da ferramenta **simpledu**, devem ser registados num ficheiro os seguintes eventos: criação e terminação de processos, envio e receção de sinais e de dados através de *pipes* (ver requisitos arquiteturais), e, os ficheiros e diretórios analisados por cada processo. A variável de ambiente LOG_FILENAME, quando definida pelo utilizador, determina o nome do ficheiro de registo dos processos e que deve ser criado se ainda não existir.

Qualquer um dos processos participantes na operação do programa acede ao ficheiro, acrescentando-lhe informação, linha a linha, no seguinte formato:

instant – pid – action – info

- **`instant`** é o instante de tempo imediatamente anterior ao registo, medido em milissegundos e com 2 casas decimais, tendo como referência o instante em que o programa começou a executar;
- **`pid`** é o identificador do processo que faz o registo da linha, com espaço fixo para 8 algarismos;
- **`action`** é a descrição do tipo de evento: **`CREATE`**, **`EXIT`**, **`RECV_SIGNAL`**, **`SEND_SIGNAL`**, **`RECV_PIPE`**, **`SEND_PIPE`** e **`ENTRY`**;
- **`info`** é a informação adicional para cada uma das ações:
  - **`CREATE`** – os argumentos da linha de comandos;
  - **`EXIT`** – o código de saída (*exit status*);
  - **`RECV_SIGNAL`** – o sinal recebido (por exemplo, SIGINT);
  - **`SEND_SIGNAL`** – o sinal enviado seguido do **`pid`** do processo a quem se destina;
  - **`RECV_PIPE`** – a mensagem enviada;
  - **`SEND_PIPE`** – a mensagem recebida;
  - **`ENTRY`** – número de *bytes* (ou blocos) seguido do caminho.

Funcionalidades Adicionais

Interrupção pelo utilizador

Estando a ferramenta **`simpledu`** em execução, quando for enviado o sinal **`SIGINT`** (**`CTRL+C`**), todo o programa deve suspender a sua operação, incluindo todos processos associados, e ser apresentada uma mensagem de confirmação ao utilizador para a terminação do mesmo. Caso o utilizador confirme a intenção de terminar o programa, devem ser terminadas quaisquer operações que possam estar pendentes antes da saída; caso o utilizador pretenda continuar a execução do programa, as operações devem ser retomadas imediatamente.

Requisitos Arquiteturais

Apesar da estrutura do programa ser deixada a cargo de cada grupo de trabalho, há um conjunto de requisitos arquiteturais que são exigidos. O programa deve:

- cada processo deve analisar somente um diretório e ser responsável por criar um processo por cada um dos subdiretórios que, eventualmente, nele possam estar contidos; os argumentos que lhe são passados devem ser ajustados para que o caminho (**`path`**) e o nível máximo de profundidade (**`--max-depth`**) reflitam o pretendido;
- O processo-pai (primeiro a ser executado) deve sempre aguardar pela terminação de todos os processos-filho antes de terminar a sua execução;
- usar o mesmo código (sem alterações) independentemente de ser ou não o processo principal;
- o tamanho total de cada um dos subdiretórios deve ser comunicado ao processo-pai através de um *pipe* (sem nome) criado para cada um dos processos-filho;
- o processo-pai deve enviar um sinal **`SIGSTOP`** a todos os processos-filho que estiverem em execução quando receber um sinal **`SIGINT`**; o envio do sinal seguinte depende da confirmação (**`SIGTERM`**) ou não (**`SIGCONT`**) por parte do utilizador relativamente à terminação do programa.

Plano de Trabalho

No sentido de tornar o desenvolvimento da ferramenta **`simpledu`** mais modular, é sugerido que o desenvolvimento seja feito de acordo com os seguintes passos:

1. Receber, processar e guardar os argumentos da linha de comandos e as variáveis de ambiente;
2. Adicionar as mensagens de registo à medida que forem implementadas novas funcionalidades e validar a correção de ambas;
3. Começar por escolher apenas ficheiros e apresentar a informação pretendida (*bytes* e blocos);

a. Considerar que a opção **`-L`** (ou **`--dereference`**) está ativa para não ser necessário distinguir links simbólicos de ficheiros regulares.

4. Fazer a distinção entre ficheiros e *links* simbólicos e apresentar resultados distintos em função da opção **`-L`**;
5. Considerar entradas que sejam diretórios, mas limitar a análise a um nível (**`--max-depth=1`**);
6. Criar um novo processo por subdiretório e tentar passar-lhe os argumentos corretos;
   a. Os argumentos serão os mesmos exceto o caminho (**`path/entry`**) e, eventualmente, o nível máximo de profundidade permitido (**`--max-depth=N-1`**);
   b. Assumir que a opção **`-S`** (ou **`--separate-dirs`**) está ativa para que não seja necessário considerar cumulativamente o tamanho dos subdiretórios.
7. Criar *pipes* para comunicar o tamanho de um dado subdiretório ao processo-pai e assim apresentar corretamente resultados cumulativos, incluindo para subdiretórios.

Notas sobre o desenvolvimento

- Os nomes do programa, variáveis de ambiente e dos argumentos da linha de comandos devem ser respeitados escrupulosamente. **O incumprimento pode conduzir à não avaliação do trabalho**, com as inerentes consequências;
- Tudo o que não estiver especificado, e que não venha a ser especificado numa nova versão do trabalho (a publicar, se necessário), deverá ser implementado pelos elementos do grupo de trabalho usando como referência o comando **`du`**.

Entrega do Trabalho

Deve ser enviado o ficheiro T**x**G**y**.zip com o trabalho desenvolvido até **10/04/2020, às 23h55**, em que **x** é o número da turma e **y** é o número do grupo. Esse ficheiro deve conter, na raiz, uma pasta com o mesmo nome e dentro da qual deve existir uma *Makefile* que permita compilar o código e gerar o executável **`simpledu`** através da invocação do comando **`make`**.

Avaliação

A avaliação funcional será feita de forma totalmente automática, usando o comando **`du`** como termo de comparação e baseando-se no código de saída (*exit status*) para determinar se argumentos inválidos são devidamente detetados. Os resultados da avaliação não são afetados pela ordem de apresentação das diferentes entradas, uma vez que serão ordenados antes da comparação.

Considerando um teste **`T`** definido por um conjunto de argumentos **`arg1`**, **`arg2`**, …, **`argN`**, o sucesso ou insucesso de um determinado teste será determinado pelo seguinte conjunto de comandos:

```
$ (./simpledu <arg1 args2 … argN> || echo $?) | sort -k2 > testeT_simpledu.txt
$ (du <arg1 args2 … argN> || echo $?) | sort -k2 > testeT_du.txt
$ diff -q testeT_simpledu.txt testeT_du.txt > /dev/null 2>&1 && echo OK || echo FAILED
O resultado será OK ou FAILED.
```

NOTA: os argumentos **`<arg1 args2 … argN>`**, obviamente, devem ser substituídos pelos argumentos pretendidos (por exemplo, **`../my_test_folder/ -l -S`**).