Compactador de Arquivos Paralelo

Trabalho 1 de FSO 2019-2

- Data de Entrega 08/12/2019
- Este documento está sempre em autalização
  - Sat, 16 Nov 2019 10:13:01 -0300 - Divulgação na página

OBJETIVOS

O objetivo do trabalho é escrever um programa que dado um diretório origem faz a compactação recursiva de todos os arquivos pertecentes ao diretório origem e seus sub-diretórios. O algoritmo de compactação deve ser o utilizado pelo `bzip2`, você pode utilizar as funções da biblioteca `libbz2`. O que você não pode fazer é chamar o comando `bzip2` diretamente.

Supondo que executamos o comando: `bz2tar DIR DESTINO`

O programa deve ter resultado equivalente a seguinte sequência de comandos:

```
cp -ax DIR_ORIGEM DESTINO.bz2
cd DESTINO.bz2
find . -type f -exec bzip2 "{}" \;
cd ..
tar cf DESTINO.bz2.tar DESTINO.bz2
rm -rf DESTINO.bz2
```

- Verificando a corretude

```
cd DIR_ORIGEM
find . -type f -exec md5sum "{}" \; > /tmp/checksum
cd ..
tar xf DESTINO.bz2.tar
cd DESTINO.bz2
find . -type f -exec bunzip2 "{}" \;
md5sum -c /tmp/checksum
```

> OBSERVAÇÃO: Isto não é a mesma coisa de um tar.bz2

Onde `DIR_ORIGEM` é o diretório de origem dos dados, podendo conter um número indefinido de sub-diretórios e `DESTINO.bz2.tar` é o arquivo que vai conter o backup com arquivos compactado de `DIR_ORIGEM`.

OBSERVAÇÕES

a. você pode utilizar o aplicativo `tar`, preferência se comunicando com ele via `PIPE`.

    - veja `popen(2)`

b. você não precisa de fato fazer a cópia dos arquivos para `DIR_TMP`. Tudo pode ser feito em memória.

c. para percorrer a árvore de diretório você deve utilizar as funções relacionadas da libc (não vale chamar um `ls -lR`)

d. você não deve se preocupar com as permissões dos arquivos.

e. deve ser possível recuperar o backup que você utilizando os comandos `tar` e `bzip2`.

ARQUITETURA

Uma opção seria utilizar uma thread (linha de execucação) para cada arquivo a ser compactado. Entretanto para um número grande de arquivos rapidamente seriam esgotados os recurso do sistema.

A solução é implentar um pool of consumers, um conjunto de threads que executa a função de compactar os arquivos. O pool consome um buffer onde cada entrada possui uma estrutura mínima (pode ser necessários outros campos):

```
typedef struct {
  int infd;
  int outfd;
  char filename[PATH_MAX];
}
```

O *pool of producer* é responsável por produzir esse buffer, ou seja, percorrer o diretório de origem, abrir o arquivo de origem (infd) e criar o arquivo de destino (outfd) alimentando o buffer.

A medida que os arquivos compactados são gerados eles devem alimentar um `tar`.

AVALIAÇÃO

A avalição será feita conforme combinado em sala.

- Grupos de até 5 alunos
    - Pode fazer individualmente

O seu programa será avaliado dentro do sistema MOJ com diversas etapas, conforme abaixo

Distribuição da avaliação

As submissões estarão disponíveis a partir do dia 2 de dezembro.

- A,B
    - Teste de 1 e 2 cores
    - Diretório original de até 20MB
- C,D

    - 1 e 2 cores
    - 100MB
- E,F
    - 1 e 4 core
    - ??MB
- G,H
    - 2 e 4 cores
    - 500MB
- I
    - 4 e 8 cores
    - Kernel Linux 5.3.11
- J
    - Todos os testes anteriores em 1, 4 e 8 cores

Distribuição da Nota

- Cada item da seção anterior vale 10 pontos
- Trabalhos com 40 pontos ou menos terão a nota Zero

Defesa do Trabalho

O trabalho deverá ser defendido quando a diferença entre a nota do trabalho e a média das provas for maior que 40.

REFERÊNCIAS

Kay A. Robbins, S. Robbins, Unix Systems Programming: Communication, Concurrency, and Threads.

Advanced Linux Programming by Mark Mitchell, Jeffrey Oldham, and Alex Samuel, of CodeSourcery LLC