

Enunciado: Utilitário ngrep (efB 2024-25)

Universidade Aberta

Escreva um programa multitarefa em linguagem C99 padrão e segundo a norma POSIX, de nome `ngrep.c`, que simule o comando `grep` com a opção `-n` para encontrar linhas num ficheiro de texto que contenham uma determinada “string”. O programa deverá gerar dados de saída similares aos obtidos com o seguinte comando `grep` dado na linha de comandos:

```
$ grep -n “string” ficheiro.txt
```

Para efetuar este processamento, o programa deve cumprir as seguintes especificações,

- O programa é constituído por uma tarefa principal (main) e mais nt tarefas, designadas tarefas trabalhadoras, num total de $nt+1$ tarefas.

- O programa `ngrep` lê da entrada padrão (stdin) três parâmetros numa única linha com a seguinte sequência:

```
“string” ficheiro.txt nt
```

- `string` (apresentada entre aspas porque pode conter espaços) é a sequência de caracteres que deve ser procurada.

- `ficheiro.txt` é o nome do ficheiro onde é feita a procura. A dimensão do ficheiro em bytes será designada por L .

- nt é o nº de tarefas trabalhadoras, com $1 \leq nt \leq 9$. Internamente ao programa, cada tarefa trabalhadora é numerada de 0 a $nt-1$. No caso particular de se verificar $L < nt$, deve considerar-se $nt=1$.

- O programa `ngrep` deve testar se o número de parâmetros constantes na linha da entrada padrão é correto e se os seus valores são válidos. Em caso de erro o programa deve emitir uma mensagem “Parâmetros inválidos” e terminar.

- No início do programa após ler os parâmetros de entrada, a tarefa principal deve imprimir a mensagem “Procurar linhas com o texto “XXX” no ficheiro XXX com XXX tarefas” (substituir valores).

- Cada tarefa trabalhadora irá efetuar a procura numa parte do ficheiro com dimensão aproximadamente igual. Para tal, a tarefa i é responsável pelas linhas inteiras que **começam** inclusive entre o byte $(L*i)/nt$ e o byte $(L*(i+1))/nt-1$. Existem casos particulares que requerem atenção: (i) A primeira linha começa no byte inicial; (ii) A última linha ultrapassa o byte final; (iii) Não existem linhas a começar no intervalo de bytes.

- Cada tarefa deve ter o seu próprio apontador tipo FILE* de modo a poder aceder independentemente à sua parte do ficheiro.
- Existe um contador global que cada tarefa deve incrementar com o nº de linhas que encontrou depois de processar a sua parte do ficheiro.
- Após todas as tarefas terminarem, a tarefa principal deve imprimir um relatório com a mensagem “Encontradas XXX linhas” na primeira linha, seguida da lista ordenada de todas as linhas encontradas imprimidas com o formato tipo C “(%d) %d:%s” indicando respetivamente o nº de tarefa entre parêntesis, o nº de linha original no ficheiro e o texto da linha separados por dois pontos (formato igual ao comando grep -n).
- Para a tarefa principal imprimir o relatório, cada tarefa trabalhadora deve retornar para a tarefa principal informação sobre: o nº de linhas analisadas; o nº de linhas encontradas; o nº (relativo) da linha na parte do ficheiro que lhe corresponde e o respetivo texto da linha. Toda esta informação deve ser memorizada com memória dinâmica (uso de malloc) e cabe à tarefa principal depois de usada a informação libertá-la.
- O programa só deve utilizar variáveis globais que contenham informação específica comum necessária a todas as tarefas. Justifique no relatório as variáveis utilizadas.
- Indique no relatório os troços de código correspondentes a regiões críticas do programa e justifique a sua existência/necessidade.
- Pondere quais as funções da biblioteca pthread que vai utilizar no programa e consulte as respetivas man pages para se informar dos detalhes de funcionamento de cada uma. Pondere também cuidadosamente quais os recursos e as estruturas de dados manipuladas pelas tarefas e que requeiram exclusão mútua no seu acesso para o bom funcionamento do programa.
- Este e-fólio baseia-se nos conhecimentos adquiridos no capítulo 2 do livro recomendado MOS3e sobre tarefas e nos conteúdos do LAB3. Não é permitido utilizar elementos da biblioteca pthread significativamente mais avançados que os descritos no LAB3, como por exemplo variáveis de condição, R/W locks, etc, assim como semáforos e outros elementos de sincronização.

Bom trabalho!