Licenciatura em
Engenharia Informática
UC de Programação Avançada
2º ano – Engenharia Informática
Regime Diurno
Ano letivo 2023/2024 1ºSemestre

Prova Prática – Exame Normal – Diurno – Enunciado F

2024.01.18 / 14h30 Prova com consulta **Duração:** 2h30m

Nome completo: _____

N.º de estudante: _____ **Regime:** [ ] Diurno [ ] Pós-laboral

IMPORTANTE

É expressamente proibido o recurso à Internet durante a prova. Qualquer utilização não autorizada da Internet leva à anulação da prova e à participação da situação às autoridades competentes. O mesmo sucede com outros tipos de tentativa de fraude.

- **Antes de iniciar a prova:**
  - Execute os seguintes comandos:

    `cd; mkdir -p ~/ENormal/R_NUMERO/`

    (em que **R** deve ser substituído pela letra **D** se for do regime diurno e **N** se for aluno do regime pós-laboral e **NUMERO** deve ser substituído pelo seu número ESTG);
  - Para garantir que o seu diretório de trabalho seja o correto, faça:

    `cd ~/ENormal/R_NUMERO/`
- **Após ter terminado a prova:**
  - Deverá proceder à criação de um arquivo TAR, fazendo uso do seguinte comando:

    `cd ~/ENormal/R_NUMERO/; tar cvf ENormal_YYYYMMDD_R_NUMERO.tar *`

    (em que YYYYMMDD corresponde à data corrente (e.g., 20240118) e R_NUMERO obedece ao formato acima indicado);
  - Verifique que o arquivo ".tar" que criou não está vazio, através da execução de:

    `tar tvf ENormal_YYYYMMDD_R_NUMERO.tar`
  - Entregue o arquivo ".tar" através da plataforma moodle, no espaço reservado para o efeito. Em caso de dúvidas, pergunte ao professor;
  - Informe o professor para este validar a receção dos seus ficheiros.

Pergunta 1 [10 valores]

(Escreva as suas respostas a esta pergunta no diretório **"~/ENormal/R_NUMERO/Pergunta1"**. Deve indicar o seu nome completo e número de estudante IPLeiria no ficheiro **README.txt** a ser criado no diretório)
NOTA 1: não é permitida a chamada a comandos externos através da função **system** ou de outra com funcionalidade similar.
NOTA 2: a solução deve ser implementada com recurso aos ficheiros existentes no arquivo **EmptyProject-Template.zip**.
NOTA 3: código entregue que **não compile através da linha de comando "make"** (sem identificadores adicionais) e do respetivo makefile na máquina virtual da UC leva à atribuição da **classificação de 0 (zero) valores** à resposta.

Recorrendo à linguagem C, implemente a aplicação `process_data`, cujo propósito é o de processar dados em paralelo usando várias *threads*. A aplicação deve aplicar uma fórmula a *data-size* valores, simulando dessa forma um processamento de dados possivelmente demorado. Para tornar a simulação realista, antes de fazer cada cálculo, a *thread* deverá bloquear durante um número de milissegundos aleatório que deverá ser um múltiplo de 100ms e não superior a 1000ms (100ms, …, 900ms, 1000ms). O programa deve calcular a fórmula (n * 234323) % 345 para todos os valores de n de 0 até **`data-size - 1`**. O trabalho deverá ser dividido pelas *threads* da forma mais eficiente. Enquanto houver valores de n para calcular as *threads* deverão estar a processar um dos valores (ou bloqueados na espera que simula um processamento demorado). Tanto quanto possível nenhuma *thread* deverá impedir o progresso de outras *threads*. Após terminado o cálculo de todos os valores, o programa deverá apresentar os valores calculados.

A aplicação `process_data` deve suportar o seguinte parâmetro da linha de comandos, devendo os mesmos serem implementados com recurso ao utilitário `gengetopt`:
`--thread-count/t <int>`: número de threads (obrigatório).
`--data-size/d <int>`: número de valores a calcular (obrigatório).

A aplicação deverá ser configurada para responder ao sinal SIGINT mostrando uma mensagem no terminal indicando o progresso do processamento. Nesta mensagem deve constar o número de valores já processados, ou seja, cujo valor final é já conhecido quando o sinal chega, assim como o total de valores a processar, bem como a respetiva percentagem. O sinal SIGINT pode ser enviado ao processo introduzindo CTRL-C no teclado enquanto o processo está a correr no terminal.

Sugestões: 1) poderá usar a função `usleep` para bloquear as threads simulando a computação demorada. A função recebe a duração da espera em microsegundos (us). 1 000 000 us = 1000 ms = 1 s. 2**)** Para os valores aleatórios poderá usar a função `rand_r`, tal como é exemplificado no código abaixo.

```
unsigned int rand_state = time(NULL) ^ getpid() ^ pthread_self();
int value1 = rand_r(&rand_state); // gera um valor aleatório
int value2 = rand_r(&rand_state); // gera outro valor aleatório
```

Considere os seguintes exemplos de execução.

Exemplo 1 – execução com 2 threads para calcular 10 valores

```
$./process-data -t 2 -d 10
processing...
Processing finished.
0: 0
1: 68
2: 136
3: 204
4: 272
5: 340
6: 63
7: 131
8: 199
9: 267
```

Exemplo 2 – Uso do CTRL-C para obter informação sobre o progresso da aplicação

```
$ ./process-data -t 2 -d 10
processing...
^C4 / 10 (40.0%)
^C7 / 10 (70.0%)
^C9 / 10 (90.0%)
Processing finished.
0: 0
1: 68
2: 136
3: 204
4: 272
5: 340
6: 63
7: 131
8: 199
9: 267
```

Exemplo 3 – Informação sobre uso

```
$./process-data -h
Usage: process-data [OPTION]...
process data in parallel
  -h, --help              Print help and exit
  -V, --version           Print version and exit
  -t, --thread-count=INT  Number of threads
  -d, --data-size=INT     Number of elements to process
```

Pergunta 2 [10 valores]

(Escreva as suas respostas a esta pergunta no diretório **`"~/ENormal/R_NUMERO/Pergunta2"`**. Deve indicar o seu nome completo e número de estudante IPLeiria no ficheiro **README.txt** a ser criado no diretório)
NOTA 1: não é permitida a chamada a comandos externos através da função *system* ou de outra com funcionalidade similar.
NOTA 2: a solução deve ser implementada com recurso aos ficheiros do diretório **EmptyProject-client-server-template.zip**
NOTA 3: código entregue que **não compile através da linha de comando "make"** (sem identificadores adicionais) e do respetivo makefile na máquina virtual da UC leva à atribuição da **classificação de 0 (zero) valores** à resposta.

Recorrendo à linguagem C, pretende-se que implemente o serviço cliente/servidor UDP **compl2decimal**, cujo propósito é o de devolver a representação decimal de um valor representado em complementos de 2. Para o efeito, a aplicação cliente deve receber como parâmetro um valor binário (8 bits) em complemento de 2 (formato *string*, por exemplo, "11111011"). O cliente deve enviar a *string* com o valor que se pretende converter através de um único datagrama UDP para o servidor. O servidor, por sua vez, deve devolver ao cliente uma *string* com a representação em formato decimal -- bem como os detalhes do processo num formato específico, conforme exemplificado abaixo.

Para obter o valor de um número negativo representado em complemento de dois deve-se, primeiramente, converter a *string* para decimal; depois devem seguir-se os seguintes passos: **1)** inverter todos os bits; **2)** somar 1 ao resultado; **3)** colocar o sinal como negativo.
Por exemplo, caso seja especificado no cliente, via linha de comando, a string 11111011, o servidor deverá devolver a string seguidamente mostrada, em que "numAbs" representa o valor absoluto do número:
`"comp2: 11111011 | comp2Inv: 00000100 | numAbs: 5 | num: -5"`

A aplicação servidora **compl2decimal_s** deve disponibilizar o seguinte parâmetro da linha de comandos:
-p/--port <int>: porto de escuta da aplicação que deve estar compreendido entre 1024 e 65535. Caso não seja especificado este parâmetro deve ser considerado o valor 9999.

A aplicação cliente **compl2decimal_c** deve implementar os seguintes parâmetros da linha de comandos:
-p/--port <int>: porto de escuta da aplicação servidora. O valor indicado para o porto deve estar compreendido entre 1024 e 65535. Caso não seja especificado este parâmetro deve ser considerado o valor 9999.

-i/--ip <string>: endereço IP do servidor.

--bin <string>: valor de oito bits em formato string (e.g., "01110011") e para a qual se pretende a respetiva representação em decimal.

Nota: os parâmetros de ambas as aplicações devem ser processados com recurso à ferramenta gengetopt, devendo os valores passados pela linha de comandos ser validados.

Exemplos de execução
Exemplo #1

`./comp2decimal_s`	**`./compl2decimal_c --ip 127.0.0.1 --port 9999 --bin 11111011`**
[SERVER] Waiting in port 9999 [SERVER] Received 8 bytes ('11111011') [SERVER] Sent 58 bytes [SERVER] Waiting in port 9999	[CLIENT] Sending "11111011" to server 127.0.0.1/9999 (8 bytes) [CLIENT] Awaiting server response – Got it: ok. (58 bytes received) comp2: 11111011 | comp2Inv: 00000100 | numAbs: 5 | num: -5

Exemplo #2
./compl2decimal_c --ip 127.0.0.1 --port 1234 --bin -1101
ERROR: --bin must be an 8-bit binary number

Exemplo #3
./compl2decimal_c --ip 127.0.0.1 --port 1234 --bin 1906
ERROR: --bin must be an 8-bit binary number