Trabalho Prático 2

Data Publicação: 04-04-2024 **Data Conclusão: 05-05-2024**

Entrega: A entrega da resolução é realizada na página da disciplina no Moodle juntando, num único ficheiro compactado, o **código fonte**, o **Makefile**. Existirão aulas práticas para a realização parcial deste trabalho.

Objetivos: Familiarização com o ambiente UNIX/LINUX; Conceção de programas baseados no paradigma cliente/servidor utilizando *sockets* como mecanismo de comunicação entre processos; Conceção de programas concorrentes com base em múltiplas tarefas; Consolidação da programação ao nível de sistema.

Livro: A resolução deste trabalho pressupõe a utilização do livro R. Arpaci-Dusseau, A. Arpaci-Dusseau, Operating Systems: Three Easy Pieces, November, 2023 (Version 1.10).

I. Realize os seguintes exercícios

Durante a realização dos exercícios propostos utilize, no terminal, o comando `man` de forma a esclarecer dúvidas sobre as chamadas de sistema e as funções C, como por exemplo, quais os seus argumentos, os valores de retorno e como verificar as situações de erro.

Para a resolução de cada questão comece por criar uma pasta contendo os ficheiros C com a resolução do exercício e um ficheiro `Makefile` (Incluindo, entre outras, as regras **`all`**, e **`clean`**) que permita a compilação da solução.

Considere o tratamento de erros das chamadas de sistema utilizadas.

1. Realize um programa que apresente no standard de output as características sobre a arquitetura dos processadores da máquina, considerando os seguintes pontos:

   - A informação sobre o processador é obtida através da execução do programa `/usr/bin/lscpu`.
   - Toda a informação produzida pelo programa `lscpu` deve ser recolhida pelo seu programa e apresentada, no standard de output, em letras maiúsculas.
   - O programa deve usar apenas chamadas de sistema e recorrer ao redireccionamento de I/O (Não usar funções da biblioteca C, e.g, popen, system, etc).

2. No primeiro exercício do primeiro trabalho prático realizou o programa cpu_stress baseado em múltiplos processos. Realize uma nova versão deste programa baseado em múltiplas tarefas.

   O programa recebe pelos argumentos da linha de comando o número de tarefas (threads) a executarem-se em concorrência e o programa principal espera pela terminação de todas as tarefas.

II. Processamento estatístico de vetores

3. Na última questão do primeiro trabalho prático realizou um programa, baseado em múltiplos processos, para extrair de um vetor (`array`) de números inteiros o subvector com os elementos compreendidos num determinado intervalo de valores. Realize uma nova versão concorrente **baseada em múltiplas tarefas** e seguindo a mesma estratégia de divisão do processamento.

   A dimensão do vetor e o número de tarefas a utilizar devem ser recebidos pela linha de comandos. Os elementos do vetor podem ser iniciados da mesma forma usada no primeiro trabalho.

   A função deve respeitar a seguinte assinatura:

```
int vector_get_in_range_with_threads (int v[], int v_sz, int sv[],
                                      int min, int max, int n_threads);
```

   Compare, de forma justificada, os resultados obtidos entre as versões **sequencial**, **concorrente usando múltiplos processos** e a **concorrente através de múltiplas tarefas**.

4. Desenvolva um programa servidor que presta o serviço de estatística sobre vetores. O serviço consiste num processo servidor, a executar numa máquina UNIX acessível através de uma rede TCP/IP. Desenvolva, também, o respetivo programa cliente que se liga ao servidor, submete o vetor, o intervalo de valores para análise e recebe o subvetor resultado. Considere os seguintes pontos

   - Define um módulo de funções auxiliares para a instanciação de *sockets streams* no domínio Internet a serem usadas na realização do servidor e cliente, i.e.:

```
int tcp_socket_server_init   (int serverPort);
int tcp_socket_server_accept (int serverSocket);
int tcp_socket_client_init   (const char *host, int port);
```

   - O servidor para processar o vector deve utilizar a função desenvolvida na questão anterior.
   - O servidor deve ser estruturado de forma concorrente usando **uma tarefa dedicada a cada ligação com um cliente**.
   - O servidor deve, também, ficar disponível através de um *socket stream* no domínio UNIX. Desta forma os clientes podem-se ligar através de *sockets* no domínio que lhes for mais conveniente (O servidor é constituído apenas por um processo contendo múltiplas tarefas). Adicione ao módulo de funções auxiliares anterior as seguintes funções para a instanciação de *sockets streams* no domínio UNIX:

```
int un_socket_server_init   (const char *serverEndPoint);
int un_socket_server_accept (int serverSocket);
int un_socket_client_init   (const char *serverEndPoint);
```

   - Adicione ao servidor a capacidade de suportar o processamento do vetor através de **múltiplos processos ou múltiplas tarefas. Esta opção é definida quando se executa o processo servidor através duma opção passada na linha de argumentos, e.g.:**

     `./vector_stat_server -p` Servidor suporta o processamento do vetor com múltiplos processos

     `./vector_stat_server -t` Servidor suporta o processamento do vetor com múltiplas tarefas

   - A transferência do vetor entre os clientes e o servidor é independente da arquitetura das máquinas? Justifique. O suporte da comunicação implementada tem em consideração a característica dos *sockets streams* não preservarem a fronteira das mensagens? Justifique.
   - Crie um cliente que tenha a opção de realizar várias ligações simultâneas ao servidor. O número de ligações deve ser recebido por argumento da linha de comandos. Esta funcionalidade tem por objetivo testar o servidor aumentando a tensão (*stress*) na utilização do serviço prestado.

II. Questões de escolha múltipla

1. Indique, para cada uma das afirmações, se a utilização de múltiplas tarefas, num programa a executar-se num sistema operativo UNIX, é uma razão válida.

Para dividir o processamento por ações concorrentes de forma a maximizar a utilização de toda a capacidade de processamento do *hardware*.	
Para poder executar dois programas (ficheiros executáveis) diferentes em concorrência e de uma forma mais rápida.	
Para poder realizar operações I/O em simultâneo com outras operações num mesmo processo.	
Para ter a execução de dois troços de código concorrentes com espaços de endereçamento separados num mesmo processo.	

2. No contexto de programação em ambiente Linux indique se as seguintes afirmações são verdadeiras (V) ou falsas (F).

Os sockets do domínio UNIX e os fifos são identificados através de um ficheiro especial no sistema de ficheiros.	
O mecanismo de comunicação fifo (named pipe) apenas funciona entre processos com grau de parentesco.	
Nos sockets stream a função bind serve para associar o socket a uma tarefa	
Os sockets são representados ao nível do núcleo do sistema operativo como um tipo de ficheiro e podem ser usados para o redireccionamento de I/O e a receção e envio de dados realizados através das funções de read() e write().	

3. Considerando que a função `handleClient()` processa uma ligação com o cliente indique se as afirmações seguintes são verdadeiras (V) ou falsas (F)

`int main () {`
`  int s = tcp_serversocket_init(HOST, PORT);`
 	
`  while (1) {`
`  int ns = tcp_serversocket_accept(s);`
`  handle_client(ns);`
`  }`
`  return 0;`
`}`
	Servidor disponível através de um *socket* no domínio internet atendendo múltiplos clientes em sequência.	

	Servidor disponível através de um *socket* no domínio internet atendendo múltiplos clientes em concorrência.	
`void `thHandleCient (void *arg) {` ` int ns = *((int *)arg);` ` handle_client(ns);` ` return NULL;` `}`	Servidor disponível através de um *socket* no domínio internet atendendo múltiplos clientes em concorrência.	

`int main () {`
`  int s = tcp_serversocket_init(HOST, PORT);`
`  pthread_t th;`
`  while (1) {`
`  int ns = tcp_serversocket_accept(s);`
`  int *ps = malloc(sizeof(int));`
`  *ps = ns;`
`  pthread_create(&th, NULL,`
`  thHandleClient, ps);`
`  pthread_join(th);`
`  }`
`  return 0;`
`}`
	Servidor disponível através de um *socket* no domínio internet atendendo múltiplos clientes em sequência.	

Bibliografia de suporte

1) Bibliografia de suporte disponível na página comum do Moodle:
   a) Slides utilizados nas aulas.
   b) Exemplos fornecidos.
   c) Exemplos realizados nas aulas.
2) R. Arpaci-Dusseau, A. Arpaci-Dusseau, Operating Systems: Three Easy Pieces, November, 2023 (Vers 1.10) [Ch 1 - 6]. Available: http://pages.cs.wisc.edu/~remzi/OSTEP/. [Accessed: 19-02-2024].

Bom trabalho,
Diogo Cardoso, *Nuno Oliveira*