U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA E COMPUTAÇÃO | 2º ANO
EIC0027| SISTEMAS OPERATIVOS | 2015/2016 – 2º SEMESTRE

EXAME - ÉPOCA NORMAL | 2016-06-22

Com consulta. Duração: 2h

Nome do estudante: _____ **Código:** _____

1. [8.0 valores = 1.0 + 1.0 + 1.5 + 1.0 + 1.0 + 1.0 + 1.5]

a) [1.0] Muitos dos "mecanismos" dos sistemas operativos modernos não poderiam ser implementados sem suporte adequado do *hardware*. Identifique o suporte de *hardware* necessário para a implementação dos "mecanismos" abaixo indicados. Explique numa frase.
(**nota**: se for necessário mais do que um tipo de suporte de *hardware*, identifique apenas um)

• proteção do sistema operativo	
• escalonamento do processador	
• semáforos	

b) [1.0] Os processos dos utilizadores podem ser classificados como sendo *CPU-bound* ou *I/O-bound*. Que informação pode usar o sistema operativo para distinguir os processos *CPU-bound* dos *I/O-bound*?

Em geral, os algoritmos de escalonamento do processador tendem a penalizar um destes tipos de processos. Qual? Porquê? Como pode ser feita essa penalização?

c) [1.5] Um parque de estacionamento tem capacidade para **N** viaturas. Cada viatura ocupa um lugar individual. As viaturas que pretendem estacionar no parque têm de passar um semáforo que só está verde quando há lugares no parque; nesse caso as viaturas podem entrar. Após entrar no parque, cada viatura deve aguardar que a viatura anterior já tenha escolhido um lugar livre para estacionar.
Usando semáforos, escreva o código de um processo "viatura" que simule este comportamento. Considere que dispõe das primitivas que operam sobre semáforos: **init(sem,value)**, **wait(sem)** e **signal(sem)**.

d) [1.0] Considere o código e o resultado de uma sua execução apresentados ao lado.

Como explica que os endereços da variável **x** sejam os mesmos em ambos os processos mas os valores de **x** sejam diferentes?

Será expectável que os resultados surjam sempre pela ordem apresentada no exemplo ao lado? Justifique.

```
#include ...
int main()
{
   int x;
   if (fork() > 0) {
      x = 1;
      printf("&x = %p; x = %d\n", &x, x);
      wait(NULL);
   }
   else {
      x = 2;
      printf("&x = %p; x = %d\n", &x, x);
   }
   return 0;
}
----------------------------------------
RESULTADO DE UMA EXECUÇÃO:
   &x = 0x7fff62860754; x = 1
   &x = 0x7fff62860754; x = 2
```

e) [1.0] Em geral, os sistemas operativos não se preocupam com a existência de *deadlocks* nos processos dos utilizadores. Porquê?

Qual a solução que os utilizadores dispõem para prevenir a existência de *deadlocks* nos seus processos?

f) [1.0] Tendo em conta os dados ao lado, referentes aos processos em execução num sistema, algum dos processos poderá ter entrado em *thrashing*? Justifique.

Processo	Utilização do *CPU*	Utilização do disco de paginação
P1	70%	5%
P2	5%	85%
P3	10%	8%

Será aconselhável aumentar o grau de multiprogramação do sistema? Justifique.

g) [1.5] Apresenta-se ao lado o resultado da execução do comando **`ls –laiR`** sobre um determinado diretório. Recorda-se que a 1ª coluna da listagem representa o *i-node*.

- Sabendo que o diretório *home* do utilizador é **`/home/user1/`**, qual o *path* do diretório sobre o qual foi executado o comando?

- Identifique o tipo das entradas **a**, **b**, e **g** que surgem na listagem e indique as permissões de acesso de **b**.

a-

b-

g-

permissões de acesso de **b**:

```
user1@ubuntu:~/SOPE$ ls -laiR
.:
total 24
157050 drwxrwxr-x  4 user1 user1 4096 Jun  8 22:18 .
934234 drwxrwxr-x 23 user1 user1 4096 Jun  8 22:17 ..
157051 drwxrwxr-x  2 user1 user1 4096 Jun  8 22:17 a
157052 -rw-rw-r--  1 user1 user1    7 Jun  8 22:18 b
157053 -rw-rw-r--  2 user1 user1   31 Jun  8 22:19 c
157054 drwxrwxr-x  2 user1 user1 4096 Jun  8 22:20 d

./a:
total 8
157051 drwxrwxr-x  2 user1 user1 4096 Jun  8 22:17 .
157050 drwxrwxr-x  4 user1 user1 4096 Jun  8 22:18 ..

./d:
total 16
157054 drwxrwxr-x  2 user1 user1 4096 Jun  8 22:20 .
157050 drwxrwxr-x  4 user1 user1 4096 Jun  8 22:18 ..
157053 -rw-rw-r--  2 user1 user1   31 Jun  8 22:19 e
157055 -rw-rw-r--  1 user1 user1   31 Jun  8 22:19 f
157056 prw-rw-r--  1 user1 user1    0 Jun  8 22:20 g
user1@ubuntu:~/SOPE$
```

- Escreva o comando da *shell* que dá permissão de execução de **b**, ao seu dono, mantendo as restantes permissões.

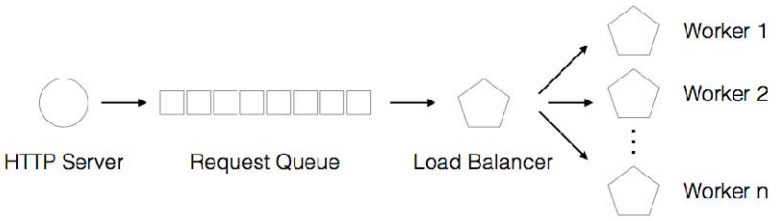
- Diga se algumas das entradas (**a** … **g**) têm um conteúdo seguramente igual. Justifique a resposta.

Nome do estudante: _____ **Código:** _____

<u>Nota</u>: não é necessário indicar os ficheiros de cabeçalho (.h)
nem o teste de sucesso das chamadas às bibliotecas ou sistema operativo

2. [6.0 valores = 1.0 + 0.5 + 1.5 + 2.0 + 1.0]

Considere a figura abaixo que sintetiza a arquitetura de um sistema *multi-threaded* do tipo *master-slave*, típico de um servidor de HTTP. Os pedidos chegam da rede e são colocados numa fila para processamento. Um *load balancer* vai lendo esses pedidos por ordem, e distribui-os para múltiplos *workers* (cada *worker* a correr numa *thread* independente).



Considere também o esqueleto de código ao lado que implementa esse servidor (admita que os **`#include`**'s necessários foram efetuados).

```
const int NUM_WORKERS = 4;

pthread_t workers[NUM_WORKERS];
pthread_t loadBalancer;

void* dequeueJob();
void* lbEntry(void* p);
void* workerEntry(void* p);
int setupLB();
int setupFifos();
int setupWorkers();
int teardownWorkers();
int teardownLB();
int teardownFifos();
void listenHTTPrequests();

int main(int argc, const char * argv[])
{
  if (setupFifos()) {
    printf("FIFOs setup failed\n");
    return 1;
  }

  if (setupWorkers()) {
    printf("Worker setup failed\n");
    return 2;
  }

  if (setupLB()) {
    printf("LB setup failed\n");
    return 3;
  }

  listenHTTPrequests();

  teardownWorkers();
  teardownLB();
  teardownFifos();

  return 0;
}
```

a) [1.0] Escreva a função **`setupFifos`**, que deve criar FIFOs com o nome "**`/tmp/myfifoX`**", onde **`X`** é um número inteiro indicando o identificador da FIFO. Admita que existe uma FIFO por *worker*, e que os seus identificadores são sequenciais, começando em 1.

b) [0.5] Escreva a função **`setupLB`** que cria uma *thread* para o *load balancer*, cuja função de entrada é **`lbEntry`**.

c) [1.5] Escreva a função **`setupWorkers`** que cria uma *thread* para cada *worker*, cuja função de entrada é **`workerEntry`**. A função **`workerEntry`** tem como parâmetro o identificador do *worker* que é um número inteiro sequencial, começando em 1.

d) [2.0] Escreva a função **`workerEntry`**, que deve **(1)** abrir a FIFO respetiva em modo de leitura, **(2)** esperar que seja enviada uma mensagem do tipo *C-string* (terminada por um caráter nulo), e **(3)** escrever essa mensagem para um ficheiro de nome **`workerX.log`**, onde **X** é o identificador do *worker*. Para efeitos de simplicidade, considere que cada *worker* executa um ciclo infinito de receção de mensagens.

e) [1.0] Escreva as funções **`teardownWorkers`** e **`teardownLB`** que devem assegurar uma boa terminação do servidor relativamente aos recursos a que estão associadas.

Nome do estudante: _____ **Código:** _____

<u>Nota</u>: não é necessário indicar os ficheiros de cabeçalho (.h)
nem o teste de sucesso das chamadas às bibliotecas ou sistema operativo

3. [6.0 valores = 1.0 + 1.0 + 0.5 + 2.0 + 0.5 + 1.0]

Dois *threads* num mesmo processo (**t1** e **t2**) precisam de utilizar uma mesma variável de condição. O segundo *thread* tem uma secção de código que só deve executar quando se verificar uma condição **C** (expressão booleana). Enquanto não se verificar essa condição deve permanecer bloqueado. Os elementos que podem mudar a avaliação da condição **C** são modificados no primeiro *thread*.

a) [1.0] Sabendo que o *thread* **t2** é lançado no código de **t1**, e que necessita de um parâmetro de tipo **`info_t`** cujo valor é produzido pela chamada à função **`info_t get_info()`**, escreva as linhas de código de **t1** que põem em execução o segundo *thread* (**t2**).	

b) [1.0] Escreva agora as linhas de código que criam e inicializam os itens necessários para a utilização da variável de condição.

c) [0.5] Em que local do código devem ser colocadas essas linhas? Justifique.

d) [2.0] O esquema interno dos dois *threads* é o que se mostra na figura ao lado. No bloco **A** de instruções pode mudar-se a avaliação da condição **C**. O bloco **B** de instruções só deve ser executado se se verificar a condição **C**. Enquanto não se verificar essa condição, **t2** deve esperar bloqueado.

Escreva nos dois *threads* o código que utiliza a variável de condição, indicando onde o colocaria no esquema, e de modo a obter o funcionamento indicado.

t1	**t2**
inicializações_1;	**inicializações_2;**
. . .	. . .
A;	**B;**
. . .	. . .
finalização_1;	**finalização_2;**

t1	**t2**

e) [0.5] Admita agora que há várias instâncias de **t2** à espera da condição **C** e que se pretende que todas elas executem **B** quando se verificar a condição **C**. Reescreva o código de **t1** que permite esse comportamento.

f) [1.0] Se em vez de dois *threads* fossem dois processos a utilizar a mesma variável de condição, indique se haveria ou não necessidade de criação de outros itens e de modificação da sua inicialização? Justifique.

FIM