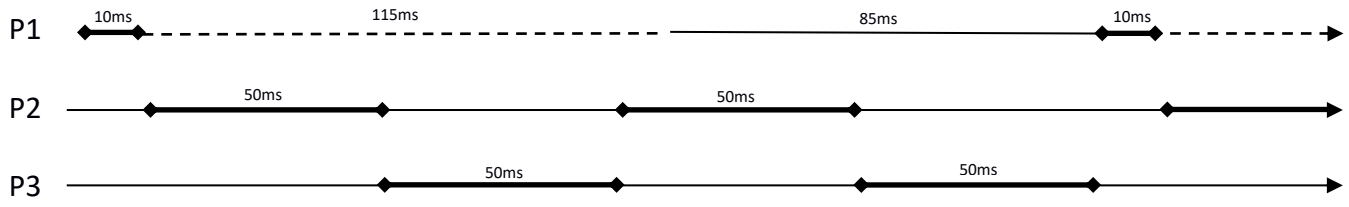
Número: Nome:

LEIC/LETI – 2019/20 – Repescagem do 2º Teste - 04 de fevereiro de 2020

- **Duração: 1h30m**
- **Responda no enunciado, apenas no espaço fornecido. Identifique todas as folhas.**
- **Nas perguntas de escolha múltipla, uma resposta errada desconta 1/3 da cotação.**

Grupo I [7 Val]

1. Considere um sistema operativo com escalonador *round-robin*, numa máquina com um único CPU *single-core.* Num dado instante, existem 3 processos ativos (P1, P2 e P3). Monitorizou-se o estado destes processos ao longo de um dado período de tempo e observou-se a história descrita na figura abaixo.



Legenda: ◆——◆ Em execução | —— Executável | ·- - - - Bloqueado

a) [1v] Marcando diretamente sobre a figura acima, indique os momentos em que o CPU esteve garantidamente em modo núcleo.

b) [1v] O processo P1 é um processo interativo, que se bloqueia à espera de comandos do utilizador. No período monitorizado, o utilizador de P1 inseriu um comando e, ao fim de algum tempo, P1 imprimiu (no *stdout*) a resposta a esse comando e voltou a bloquear-se.
Com base na informação disponível na figura acima, qual o tempo de resposta sentido pelo utilizador de P1?

c) [0,7v] Sempre que receberam execução, os processos P2 e P3 esgotaram o seu *time-slice*. No fim de cada *time*-slice, o que é que permite ao despacho (do núcleo) atuar para trocar de um processo para outro?

A. Cada processo mede o tempo que passa e, assim que esgota o seu *time-slice,* faz uma chamada sistema para libertar o CPU.
B. O núcleo está sempre em execução num *core* dedicado.
C. Uma interrupção do temporizador *(timer interrupt)*.
D. Aparecimento de um processo mais prioritário.

d) [1v] Assuma que se recompilava este sistema operativo de forma a que o *time-slice* passava a ter metade da duração, 25 ms. Indique uma vantagem e uma desvantagem dessa alteração.

2. Correndo os mesmos 3 processos num sistema Linux, verificou-se que o tempo médio de resposta de P1 melhora consideravelmente.
   a) [0,8v] Indique um aspeto do escalonador do Linux que contribua fortemente para essa melhoria.

   b) [0,8v] No período ilustrado na figura acima, qual a principal alteração que ocorreria caso se mudasse para o escalonador do Linux? Assuma que, no início do período monitorizado, P1 era o processo mais prioritário.

3. [1v] Considere o seguinte excerto do programa de um dado processo, P1.

```
int sockfd = socket (AF_INET, SOCK_STREAM, 0);
connect(sockfd,&serv_addr,sizeof(serv_addr)) < 0) {…}
read(sockfd,buffer,255);
```

Neste excerto, P1 bloqueia-se, por vezes durante vários segundos, até receber novas mensagens de outros processos em máquinas remotas, recebidas através da placa de rede na máquina de P1.

Quando P1 chama *read*, ocorre sempre uma chamada sistema, sendo o núcleo responsável por implementar a funcionalidade descrita na frase acima.
Seria possível uma implementação alternativa de função *read* que conseguisse assegurar a funcionalidade acima exclusicamente em modo utilizador, sem exigir qualquer intervenção do núcleo? Responda justificando de forma objectiva, referindo partes da frase acima para apoiar a sua justificação.

4. [0,7v] Um Unix, comparando um processo a correr com UID=alice e outro processo a correr com UID=root:
   A. O primeiro corre o seu programa em modo utilizador, enquanto que o segundo corre sempre em modo núcleo.
   B. O primeiro usa memória virtual, enquanto que o segundo usa endereçamento real.
   C. Há certas chamadas sistema que o núcleo só permite ao segundo processo, não ao primeiro.
   D. O primeiro tem sempre prioridade menor que o segundo.

Grupo II [6,4 Val]

1. Considere uma arquitetura paginada de 24 bits e páginas de 256 Bytes. Num dado instante, a tabela de páginas (de 1 nível) do processo em execução é a seguinte:

Página virtual	P	Prot	R	M	Base	CoW
0x0000	1	R	0	0	0x8000	1
0x0001	1	R	0	0	0x8001	1
0x0002	1	R	1	0	0xA004	0
0x0003	0	R	0	0	0xB000	0
0x0004	1	R	1	0	0xA005	0
0x0005	1	R	1	0	0x0010	0
0x0006	1	R	1	0	0x0F12	0
0x0007	1	R	1	0	0x0220	0

a) [3v] O processo executou (em sequência) os acessos apresentados nas alíneas seguintes. Considere que, ao longo desta sequência de acessos, sempre que foi preciso alocar novas páginas em memória primária, foram usadas as seguintes *page frames:* 0xAAAA00, 0xAAAB00, 0xAAAC00, 0xAAAD00 e seguintes.
Para cada acesso, responda às questões em cada alínea.

1) Leitura de 0x0000F0

Ocorreu exceção? Qual?	Componente(s) envolvidas?	Endereço real?
	[UGM] [Núcleo]	

2) Leitura de 0x0000F4

Ocorreu exceção? Qual?	Componente(s) envolvidas	Endereço real
	[UGM] [Núcleo]	

3) Leitura de 0x000480

Ocorreu exceção? Qual?	Componente(s) envolvidas	Endereço real
	[UGM] [Núcleo]	

4) Leitura de 0x000312

Ocorreu exceção? Qual?	Componente(s) envolvidas	Endereço real
	[UGM] [Núcleo]	

5) Escrita em 0x000108

Ocorreu exceção? Qual?	Componente(s) envolvidas	Endereço real
	[UGM] [Núcleo]	

6) Escrita em 0x000240

Ocorreu exceção? Qual?	Componente(s) envolvidas	Endereço real
	[UGM] [Núcleo]	

2. [0,7v] Assuma que, no início da sequência de acessos anterior, a TLB estava vazia. Indique qual/quais dos acessos enumerados acima (1 a 6) garantidamente beneficiaram da TLB (ou seja, resultaram num *TLB hit*).

3. [2v] Considere os bits R e M da tabela de páginas. Para cada transição do seu valor, indique qual a situação que causa essa transição e qual a componente (UGM ou núcleo) que escreve esse novo valor.
    a) Bit R mudar para valor 1.

    b) Bit R mudar para valor 0.

    c) Bit M mudar para valor 1.

    d) Bit M mudar para valor 0.

4. [0,7v] O registo BTP (base da tabela de páginas) da unidade de gestão de memória tinha o valor 0x240B00. Em que momento é que este registo mudará de valor?
    A. Muda a cada acesso a um endereço virtual.
    B. Muda quando houver uma troca de contexto para um outro processo.
    C. Muda quando houver uma troca de contexto para uma outra tarefa do mesmo processo.
    D. Muda quando a máquina for reiniciada.

Grupo III [6,6 Val]

1. Considere o seguinte excerto de código, que lê o conteúdo de um ficheiro mantido numa partição Ext-3.

```
int fd = open("foo.txt", O_RDONLY);
read(fd,buffer,255);
```

Ambas as funções do sistema de ficheiros que este programa chama precisam de consultar o *i-node* do ficheiro para completarem.

a) [1v] Que campos do i-node são garantidamente consultados por cada função? Justifique sucintamente.

Função open:

Função read:

b) [1v] Cada função encontra o *i-node* do ficheiro em causa de formas diferentes. Descreva-as sucintamente.

Função open:

Função read:

c) [1v] O i-node é mantido na tabela de i-nodes num disco magnético. O tempo de acesso a essa tabela é muito elevado, dado o fraco desempenho do disco. Que mecanismo do sistema de ficheiros permite minimizar esse custo?

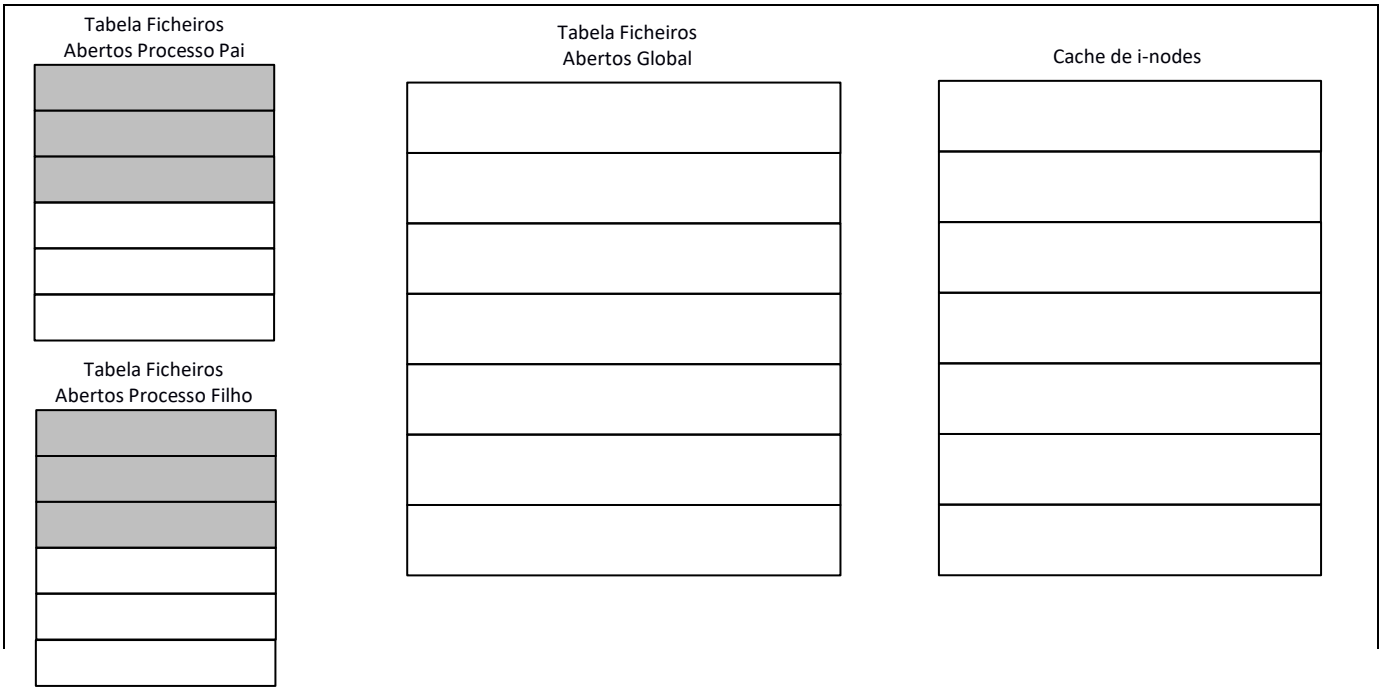
2. Num projeto de SO, pediu-se aos alunos que desenvolvessem um programa em que dois processos (pai e filho) carregam e processam um conjunto de estruturas de dados (de dimensão *DIM*) a partir de 2 ficheiros origem distintos. Como resultado, o conjunto de todas as estruturas processadas por cada processo deve ser colocado num único ficheiro destino.
   Abaixo apresenta-se uma possível solução (correta) para este projeto:

```
#define DIM 10

int main() {
  char buffer[DIM];
  int fsrc;
  int fdest = open("dest", O_APPEND);

  int p = fork();

  if (p > 0)
    fsrc = open("orig1", O_RDONLY);
  else
    fsrc = open("orig2", O_RDONLY);

  while (read(fsrc,buffer,DIM) > 0) {
    computeAndUpdate(buffer);
    write(fdest, buffer, DIM);
  }

  if (p >0) wait(NULL);
  exit(0);
}
```

a) [1v] Executou-se o programa acima num cenário em que o ficheiro *orig1* tinha 20 estruturas (de 10 bytes) no seu conteúdo, o ficheiro *orig2* tinha 30 estruturas (de 10 bytes) no seu conteúdo, e o ficheiro *dest* estava inicialmente vazio. Assuma que, num dado instante, ambos os processos completaram a sua execução do ciclo *while* mas cada processo ainda não terminou (chamando *exit*).

Na figura abaixo, preencha o conteúdo das principais estruturas mantidas pelo sistema de ficheiros em memória com a informação que este enunciado lhe permite saber.

Nota: na sua resposta, omita o preenchimento da informação sobre os ficheiros stdin, stdout e stderr de ambos os processos.



b) [1v] Um projeto submetido por um grupo de alunos tinha um programa idêntico, apenas com a diferença que a chamada a *open* para abrir o ficheiro *dest* tinha sido movida para após o *fork*:

```
[...]
int p = fork();
fdest = open("dest", O_APPEND);
[...]
```

Esta variante também é correta ou introduz um *bug*?
Se é igualmente correta, explique porque é que a mudança de linha não tem qualquer efeito.
Se não, explique de que forma é que o *bug* se manifesta, dando um exemplo.

3. Considere os dois excertos de código abaixo, que abrem diferentes ficheiros mantidos na **mesma partição física**.

```
//Programa 1
int main() {
  int f1 = open("/home/alice/so/f.txt", O_RDONLY);
  int f2 = open("/home/alice/so/g.txt", O_RDONLY);
  …
}

//Programa 2
int main() {
  int f1 = open("/home/alice/so/f.txt", O_RDONLY);
  int f2 = open("/tmp/alice/old-so/g.txt", O_RDONLY);
  …
}
```

Executou-se ambos os programas múltiplas vezes para se comparar o desempenho das linhas apresentadas acima. Em cada experiência, as *caches* do sistema de ficheiros foram limpas previamente.

a) [0,8v] Qual espera ter melhor desempenho?
   A. Programa 1
   B. Programa 2
   C. Ambos têm desempenho comparável.
   D. Não há resposta determinística.

b) [0,8v] Qual a *cache* do sistema de ficheiros que tem maior impacto no desempenho dos excertos apresentados acima?