

LEIC/LETI – 2015/16 - 1º Exame de Sistemas Operativos**6 de Janeiro de 2016****Responda no enunciado, apenas no espaço fornecido. Identifique todas as folhas.****Duração: 2h30****Grupo I [5 Val]**

Considere os seguintes programas.

<pre>Prog1.c main () { 1 int pid; 2 int status; 3 pid = fork (); 4 if (pid == 0) { 5 execl ("./prog2", "prog2", 0); 6 /* ... */ 7 } else if (pid > 0) { 8 sleep(N); 9 wait(&status); 10 } else { 11 /* ... */ 12 } }</pre>	<pre>Prog2.c main () { 13 for (i=0; i<900000000; i++); 14 exit(5); }</pre>
---	---

1. O *prog1* pode chegar às linhas 6 e 10 em situações de erro. Para cada linha, descreva uma possível razão de erro associada:

a) [0,5v] Para *prog1* chegar à linha 6.

b) [0,5v] Para *prog1* chegar à linha 10.

2. [1,5v] Executou-se o programa *prog1*. Enquanto este se executava, usou-se o comando *ps* em diferentes momentos para listar os processos no sistema. Para cada resultado do *ps*, indique em que linha de código se encontra cada processo. No caso de existirem múltiplas respostas possíveis, indique apenas uma delas. Seja claro na sua resposta: indique o nº da linha e se o processo está prestes a executar/a está a executar/acabou de a executar.

```
a) UID  PID  PPID  CMD
     1000 5359 5357  prog1
     1000 5388 5359  prog1
```

Processo pai:

Processo filho:

```
b) UID  PID  PPID  CMD
     1000 5359 5357  prog1
     1000 5388 5359  prog2
```

Processo pai:

Processo filho:

```
c) UID  PID  PPID  CMD
     1000 5359 5357  prog1
     1000 5388 5359  prog2 <defunct>
```

Processo pai:

Processo filho:

3. [0,6v] Depois do retorno da função wait (no prog1), é possível que a variável status contenha um valor diferente de 5? Justifique.

4. [0,6v] Indique quais as linhas de ambos os programas resultam *sempre* numa comutação de processos. Justifique.

5. [0,7v] Assuma que não existem outros processos no sistema. Em Linux, pode o processo filho perder o processador enquanto executa o ciclo da linha 13 de *prog2*? Justifique relacionando com a política de escalonamento adotada pelo Linux.

6. [0,6v] Considere dois processos, X e Y, que executam o programa *prog2* apresentado acima numa máquina Linux com 1 CPU apenas. O processo X tem uma prioridade base de 5, enquanto que o processo Y tem prioridade base 50. Enquanto Y detém o processador e executa o ciclo da linha 13, alguma vez o processo X receberá tempo de execução? Justifique relacionando com o algoritmo de escalonamento do Linux.

Grupo II [4 Val]

Considere a seguinte solução para o problema dos produtores-consumidores.

<pre>int buf[N]; int prodptr=0, consptr=0; trinco_t trinco; semaforo_t pode_prod = criar_semaforo(N), pode_cons = criar_semaforo(0); void colocarItem(int item) { esperar(pode_prod); fechar(trinco); buf[prodptr] = item; prodptr = (prodptr+1) % N; abrir(trinco); assinalar(pode_cons); }</pre>	<pre>int obterItem(){ int item; esperar(pode_cons); fechar(trinco); item = buf[consptr]; consptr = (consptr+1) % N; abrir(trinco); assinalar(pode_prod); return item; }</pre>
--	---

1. Indique que comportamentos incorretos poderiam ocorrer caso aplicasse as seguintes alterações ao programa acima. Justifique cada resposta dando um exemplo de uma execução ilustrativa da falha.

a) [0,7v] Caso se omitisse o trinco; ou seja, se eliminassem as linhas `fechar(trinco)` e `abrir(trinco)`.

b) [0,7v] Caso o semáforo `pode_cons` fosse inicializado a N unidades (em vez de 0 unidades).

c) [0,7v] Caso as linhas `“esperar(pode_prod);fechar(trinco);”` trocassem de ordem, passando para `“fechar(trinco);esperar(pode_prod);”`.

2. [1,9v] Pretende-se estender a solução acima para suportar 2 filas, cada uma associada a um nível de prioridade: a fila normal e a fila de urgência.

Neste novo modelo, quando um produtor pretende colocar uma mensagem numa das filas e essa correspondente está cheia, o produtor fica bloqueado. Um consumidor deve esperar (bloqueando-se) até que haja pelo menos uma mensagem em qualquer uma das filas. Assim que houver, o consumidor deve retirar a mensagem da fila mais prioritária que tenha pelo menos uma mensagem.

Complete o seguinte programa com a sincronização necessária para implementar esta nova solução. Declare os mecanismos de sincronização que considerar necessários para a solução.

<pre>int bufNormal[N], bufUrgente[N]; int prodptrNormal=0, consptrNormal=0, prodptrUrgente=0, consptrUrgente=0; int numItensUrgentes = 0; void colocarItem(int item, bool urgente) { if (urgente) { bufUrgente[prodptrUrgente] = item; prodptrUrgente = (prodptrUrgente+1) % N; numItensUrgentes --; } else { bufNormal[prodptrNormal] = item; prodptrNormal = (prodptrNormal+1) % N; } }</pre>	<pre>int obterItem(){ int item; if (numItensUrgentes > 0) { item = bufUrgente[consptrUrgente]; consptrUrgente = (consptrUrgente+1) % N; numItensUrgentes --; } else { item = bufNormal[consptrNormal]; consptrNormal = (consptrNormal+1) % N; } return item; }</pre>
--	---

Grupo III [4 Val]

1. Considere o seguinte excerto de um programa:

```
1 char *buffer = ...;
2 int f = open("/home/docs/so.txt", O_WRONLY);
3 write(f, buffer, strlen(buffer));
```

a. [0,7v] Considere a chamada à função `open`. A resolução do nome `"/home/docs/so.txt"` pode obrigar a consultar algum inode? Se sim, indique qual/quais? Se não, justifique.

b. A função `open` também verifica se o utilizador que executa o processo tem permissões para abrir o ficheiro da forma solicitada.

i) [0,5v] Em que estrutura de dados se encontra a informação sobre qual o utilizador que está a pedir para abrir o ficheiro?

ii) [0,5v] Em que estrutura de dados é que a função `open` descobre quais as operações permitidas sobre o ficheiro?

iii) [0,5v] Caso a verificação de permissões tenha sucesso, em que estrutura de dados é guardado o modo de acesso com que o ficheiro foi aberto?

2. [0,6v] Considere agora a seguinte variante do programa anterior.

```
1 char *buffer = ...;
2 FILE * f = fopen("/home/docs/so.txt", "w");
3 fwrite(buffer, 1, strlen(buffer), f);
4 fflush(f);
```

A função `fflush` é tipicamente muito mais demorada que a função `fwrite`. Porquê?

3. Considere um sistema operativo do tipo Linux e, em particular, as *caches* no sistema de ficheiros.
- a. [0,6v] Qual a cache que tem maior impacto nas operações efectuadas pela chamada sistema `open` ? Justifique a sua resposta focando a função da cache em causa e em que momento é preenchida com que informação.

- b. [0,6v] Considere agora a *cache de inodes*. Diga se concorda com a informação seguinte: “A cache de inodes pode conter *inodes* de ficheiros que estão fechados.” Diga se concorda. Justifique.

Grupo IV [3,5 Val]

Considere um sistema operativo do tipo Linux.

1. Assuma a existência de um processo P1 que dispõe de 3 descritores (i.e. 3 entradas na tabela de ficheiros abertos correspondendo assim a 3 canais abertos) sobre os quais pode invocar as seguintes chamadas sistema bloqueantes: `accept`, `read`, `recvfrom`. Considere que, para possibilitar a recepção de dados nos descritores em causa, tem à sua disposição 3 tipos de solução: 1) um processo para cada descritor, 2) uma tarefa para cada descritor, e 3) utilização da chamada sistema `select`.

- a. [0,6v] A solução 2 pode ser implementada com tarefas reais, pseudo-tarefas, ou ambas? Justifique.

- b. [0,5v] Na solução 3, como é que o programa de P1 indica ao sistema operativo quais os descritores nos quais P1 está interessado em receber dados? Refira de forma clara o argumento que permite indicar quais os descritores em causa.

- c. [0,6v] Na solução 1, considere agora que a chamada sistema `accept` é invocada, levando ao bloqueio de P1. Quando esta chamada sistema retornar (assumindo que se executa com sucesso), qual o número de descritores abertos por P1 (que correspondem a canais abertos) na sua tabela de ficheiros abertos? Justifique.

- d. [0,6v] Como compara as soluções em causa no que diz respeito ao número de comutações de contexto (*context switches*) assumindo que todos os descritores vão receber dados através dos canais respectivos? Justifique a sua resposta.

2. Considere um sistema do tipo Linux no qual existe um processo P1 que cria um processo filho P2.

- a. [0,6v] Tendo em conta o pseudo-código que se apresenta de seguida, apresente uma figura que ilustre o estado final das tabelas de ficheiros abertos de cada um dos processos. Assuma, que no início de cada programa, tanto P1 como P2 não têm nenhum outro ficheiro aberto além de *stdin*, *stdout* e *stderr*.

Processo P1:	Processo P2:
1 pipe (fd1)	
2 pipe (fd2)	
3 fork()	
4 if (P1) {	14 if (P2) {
5 close (1)	15 close (0)
6 dup (fd1[1])	16 dup (fd1[0])
7 close (0)	17 close (1)
8 dup (fd2[0])	18 dup (fd2[1])
9 close (fd1[0])	19 close (fd1[0])
10 close (fd1[1])	20 close (fd1[1])
11 close (fd2[0])	21 close (fd2[0])
12 close (fd2[1])	22 close (fd2[1])
13 }	23 }

- b. [0,6v] As quatro últimas instruções executadas por P1 têm algum impacto no seu funcionamento? Justifique.

Grupo V [3,5 Val]

1. Considere um sistema operativo do tipo Linux a correr sobre uma arquitetura paginada. Assuma que o computador em causa tem uma arquitectura paginada de memória virtual de 16 bits. Neste sistema, cada endereço virtual é composto em 6 bits (mais significativos) que indicam o nº de página e 10 bits (menos significativos) que indicam o deslocamento. Assuma que não existe TLB.

a. [0,5v] Qual a dimensão de uma página neste sistema? Justifique.

b. [0,5v] Quantas entradas (PTEs) pode ter, no máximo, a tabela de páginas de um processo ? Justifique.

2. Assuma agora que existem dois processos, P1 e P2, que partilham uma página física. P1 pode ler e escrever a página em causa, mas P2 pode apenas ler. Tenha em conta os componentes das PTEs correspondentes nas tabelas de página de ambos os processos que se indicam de seguida e, para cada um deles, diga se o seu conteúdo é: **“igual”** ou **“diferente”** ou **“na”** (caso não tenha informação que lhe permita responder). Não é preciso justificar.

a. [0,5v] BASE (endereço inicial da página física)

b. [0,5v] PROT (protecção)

c. [0,5v] P (presença)

d. [0,5v] R (página referenciada)

e. [0,5v] M (página modificada)