LEIC/LETI – 2016/17 - 1º Exame de Sistemas Operativos

11 de Janeiro de 2017

Responda no enunciado, apenas no espaço fornecido. Identifique todas as folhas.

Duração: 2h30

Grupo I [2,4 Val]

1. Considere o pseudo-código apresentado de seguida.

```
1. #include <stdlib.h>
2. #include <stdio.h>
3. #include <unistd.h>
4. #include <pthread.h>
5.
6. #define N 3
7. #define TAMANHO 10
8.
9. int buffer [N][TAMANHO];
10.  int nsomas;
```

```
1. void *xpto (int *linha) {
2.   int c, soma=0;
3.   int *b = linha;
4.
5.   printf ("xpto: tarefa comecou %lu\n",
   pthread_self());
6.   sleep(1);
7.   for (c=0;c<TAMANHO-1;c++){
8.     soma += b[c];
9.     nsomas++;
10.    }
11.    b[c]=soma;
12.    printf ("xpto: tarefa terminou
   %lu\n", pthread_self());
13.    return NULL;
14.  }
```

```
1. int main (void) {
2.   int i,j;
3.   pthread_t tid[N];
4.
5.   for (i=0; i< N; i++){
6.     if(pthread_create (&tid[i], 0, (void*)xpto,(void *) buffer[i])== 0)
   {
7.       printf ("Criada a tarefa %lu\n", tid[i]);
8.     }
9.     else {
10.            printf("Erro na criação da tarefa\n");
11.            exit(1);
12.          }
13.        }
14.        for (i=0; i<N; i++)
15.          pthread_join (tid[i], NULL);
16.        printf ("Terminaram todas as threads\n");
17.        exit(0);
18.      }
```

a. [0,8 val] Quantos fios de execução existem no máximo, no total, durante a execução do programa? Justifique a sua resposta dizendo como são criados e qual a função que cada um executa.

b. [0,8 val] Apresente um *output* possível; assuma que os identificadores das tarefas são impressos como inteiros entre 1 e 99, e que as tarefas são reais.

c. [0,8 val] Apresente um *output* possível (assuma que os identificadores das tarefas são impressos como inteiros entre 1 e 99, e que as tarefas são reais) supondo que a linha "`for (i=0; i<N; i++) pthread_join (tid[i], NULL);`" é apagada. Justifique a sua resposta.

Grupo II [3,8 Val]

Considere o seguinte problema:

- Num dado sistema, existem múltiplas tarefas emissoras que geram mensagens.
  - Após gerar uma mensagem, uma tarefa emissora chama a função *colocaMensagem* para que a nova mensagem seja colocada num buffer.
  - O buffer tem dimensão fixa, N, e é partilhado entre todas as tarefas.
  - Caso o buffer esteja cheio, *colocaMensagem* bloqueia até encontrar de novo espaço no buffer para colocar a mensagem.
- Existe uma tarefa receptora, que consome blocos de N mensagens de uma vez só.
  - Para tal, a tarefa receptora chama a função *recebeNMensagens*, que espera até o buffer partilhado estar cheio; quando tal condição se verifica, a função move as mensagens no buffer para um buffer auxiliar (passado por referência à função) e o buffer partilhado passa a estar vazio de novo.

1. [0,8 val] Considere a seguinte solução (incorreta) para o problema acima.

```
mensagem_t buffer[N]; //buffer de mensagens
int numMensagens = 0; //número de mensagens atualmente no buffer

void colocaMensagem(mensagem_t m) {

   while (numMensagens == N); //Espera enquanto o buffer estiver cheio

   buffer[numMensagens] = m;
   numMensagens ++;

}

recebeNMensagens (mensagem_t b[]) {

   while (numMensagens < N); //Espera enquanto o buffer não estiver cheio

   copia(buffer, b, N); //Copia as N mensagens do buffer partilhado
   numMensagens = 0;

}
```

Apresente um exemplo de uma execução que ilustre um dos erros da solução acima. Descreva o seu exemplo passo por passo.

2. [1,5 val] Proponha uma solução alternativa para o problema acima que seja correta e eficiente, recorrendo a trincos lógicos (mutexes) e semáforos.

3. [1,5 val] Apresente uma variante da solução acima, desta vez usando trincos lógicos e variáveis de condição.

Grupo III [3,2 Val]

1. Considere o escalonamento com *time-slices* fixas, em *round robin*.

a. [0,8 val] Indique 2 situações em que um processo que está em execução perde o processador para outro processo.

b. [0,8 val] Em cada uma das situações acima, a comutação para outro processo é feita pelo núcleo. Para cada uma das 2 situações, indique de forma detalhada o evento que causou a ativação do núcleo.

c. [0,8 val] Este escalonador tende a dar prejudicar os processos ES-intensivos (*IO-bound*). Justifique a afirmação com um exemplo.

d. [0,8 val] Apresente 2 diferenças substanciais entre este algoritmo de escalonamento e aquele usado no Linux.

Grupo IV [3,4 Val]

1. Considere o ficheiro com o nome /a/b/c.c.
   a. [0,8 val] Quantos inodes são acedidos quando a seguinte linha é executada com sucesso?

      ```
      open ("/a/b/c/d/d.c", ...);
      ```

      Justifique a sua resposta indicando a que corresponde cada inode que refere. Na sua resposta, assuma que não existe cache de nomes.

   b. [0,8 val] O valor de retorno da função é um inteiro; este valor indica o quê?

2. [1 val] Do ponto de vista da rapidez de acesso ao ficheiro, qual a razão principal para que a operação de "open" retorne um inteiro que depois é usado nas operações de leitura e escrita nesse ficheiro? Justifique a sua resposta.

3. [0,8 val] Considere agora um sistema de ficheiros do tipo Linux / Unix em que o sistema de ficheiros tem as seguintes caraterísticas:
   i) inode com 10 entradas directas + 3 níveis de indirecção (uma entrada para cada nível);
   ii) referências para blocos com 4 bytes;
   iii) blocos com 1024 bytes.

   Quantos blocos de dados, com conteúdo útil de ficheiros, podem ser endereçados através do 3º nível de indirecção? Justifique a sua resposta apresentando uma figura ilustrativa de um inode e respectivos blocos.

Grupo V [3,6 Val]

Numa estação de metro, existe um ecrã onde são apresentadas mensagens com avisos aos passageiros na plataforma.
Esse ecrã é gerido por um processo que recebe mensagens (cadeiras de caracteres) de outros processos locais através de um *pipe* com nome. O sistema operativo é Linux.

1. [0,6 val] Indique que função sistema foi usada para criar o *pipe* com nome. No caso de conhecer mais que uma alternativa, indique apenas uma.

2. [0,6 val] Invente e apresente um possível nome para este *pipe* com nome.

3. [0,8 val] Complete o seguinte programa (em pseudo- código), que é executado por um processo cliente que envia avisos para o *pipe* com nome. Assuma que a função auxiliar *obtemProximoAviso* espera até que haja novo aviso para publicar; assim que tal aconteça, a função preenche o texto do próximo aviso no buffer passado como argumento e retorna. Assuma também que o pipe tem o nome que indicou na alínea acima.

4. [0,8 val] Assuma agora uma variante mais sofisticada do sistema, em que o processo receptor devolve uma resposta "ack" ao processo cliente confirmando que o aviso enviado por este tenha sido apresentado no ecrã. Considere uma solução em que: i) processo cliente envia aviso pelo pipe com nome; ii) processo receptor recebe aviso e devolve resposta "ack" pelo mesmo pipe; iii) processo cliente lê a resposta e avança para próxima iteração. Esta solução é correta? Justifique.

5. [0,8 val] Apresente 2 diferenças substanciais entre um *pipe com nome* e um *socket datagram*.

Grupo VI [3,6 Val]

1. [0,8 val] Considere um sistema operativo com memória virtual paginada. Diga se concorda ou não com a frase seguinte. *"Para cada página partilhada entre dois processos P1 e P2, o endereço virtual é sempre o mesmo em ambos os processos."* Responda "Sim" ou "Não" e justifique.

2. [0,8 val] Considere a TLB e diga como varia a sua ocupação (mais ou menos entradas preenchidas) em função do quantum dos processos. Justifique a sua resposta.

3. [0,8 val] Explique porque é que os sistemas com paginação não necessitam de usar algoritmos como o "BestFit" ou "FirstFit", que são necessários nos sistemas com segmentação. Responda relacionando com a noção de fragmentação externa.

4. Considere um sistema com uma arquitectura paginada de memória virtual de 32 bits. Neste sistema, cada endereço virtual é composto em 22 bits (mais significativos) que indicam o nº de página e 10 bits (menos significativos) que indicam o deslocamento.
    a. [0,6 val] Qual a dimensão das páginas deste sistema? Justifique.

    b. [0,6 val] Quantas linhas pode ter a tabela de páginas de um dado processo? Justifique.