Número: Nome:

LEIC/LERC – 2012/13 - 1º Teste de Sistemas Operativos

24 de Novembro de 2011

Responda no enunciado, apenas no espaço fornecido.

Identifique todas as folhas. Justifique todas as respostas.

Duração: 1h30m

Grupo I [7 Val]

1. Considere os seguintes 3 algoritmos de escalonamento num sistema operativo de tempo virtual:

   - Algoritmo A: quantum fixo de 20ms, sem preempção, lista única de processos executáveis, gerida em *round-robin*.
   - Algoritmo B: quantum fixo de 20ms, sem preempção, prioridades dinâmicas de 10 níveis (1-menos prioritário a 10-mais prioritário), processo que executa o quantum completamente desce 1 nível de prioridade, processo que se bloqueie sem terminar quantum sobe 1 nível de prioridade.
   - Algoritmo C: igual ao B mas com preempção.

   a. [0,8v] Comparando os algoritmos A e B, qual acima privilegia os processos *ES-intensivos (IO-bound),* em detrimento dos processos *CPU-intensivos*? Justifique.

Considere três processos (P1, P2 e P3), que executam os seguintes programas, respectivamente

```
//Programa de P1
main() {
  for i=0..100 {
    res = doShortComputation(i);
    writeResultToDisk(res);
}}
```

```
//Programa de P2 e P3
main() {
  for i=0..100 {
    doLongComputation(i);
  }
}
```

Assuma que:

- As funções *doShortComputation* e *doLongComputation* executam apenas operações aritméticas sobre dados em memória primária;
- A função *doShortComputation* demora cerca de 5ms a executar;
- A função *doLongComputation* demora cerca de 20ms a executar;
- A função *writeResultToDisk* envia um pedido de escrita ao disco (assuma que o envio do pedido é practicamente instantâneo) e bloqueia-se imediatamente depois (perdendo o processador); passados cerca de 25ms, a escrita em disco é concluída, o processo é desbloqueado e a função *writeResultToDisk* pode retornar;
- O tempo de escalonamento e comutação são negligenciáveis (ou seja, 0ms).

b. [1v] Considere que P1 executou uma iteração completa do seu programa.
Indique os estados por que P1 passou durante a execução da iteração, justificando cada transição.
Nota: no caso de encontrar várias hipóteses de resposta, escolha a que inclui menos transições para a sua resposta. Preencha apenas as linhas da tabela que achar necessárias.

Estado	**Justificação da transição para este estado (20 palavras máx.)**
Em Execução	Processo anteriormente em execução bloqueou-se ou esgotou quantum.

c. [2v] Pretende-se comparar o desempenho dos 3 algoritmos de escalonamento numa máquina single-core (1 cpu apenas) a executar P1, P2 e P3.
   i. Assumindo que, para cada escalonador diferente, P1, P2 e P3 são iniciados simultaneamente e se observa o seu comportamento ao longo de 100ms, preencha as execuções seguintes.
   No caso dos algoritmos B e C, assuma que a prioridade inicial de P1, P2 e P3 é 5.

   Em cada espaço, indique apenas o estado de cada processo em cada instante.
   Legenda a usar: B-bloqueado; E-em execução; W-executável.

Execução com algoritmo de escalonamento A

	0ms	5ms	10ms	15ms	20ms	25ms	30ms	35ms	40ms	45ms	50ms	55ms	60ms	65ms	70ms	75ms	80ms	85ms	90ms	95ms
P1																				
P2																				
P3																				

Execução com algoritmo de escalonamento B

	0ms	5ms	10ms	15ms	20ms	25ms	30ms	35ms	40ms	45ms	50ms	55ms	60ms	65ms	70ms	75ms	80ms	85ms	90ms	95ms
P1																				
P2																				
P3																				

Execução com algoritmo de escalonamento C

	0ms	5ms	10ms	15ms	20ms	25ms	30ms	35ms	40ms	45ms	50ms	55ms	60ms	65ms	70ms	75ms	80ms	85ms	90ms	95ms
P1																				
P2																				
P3																				

ii. Tendo em conta a sua resposta na alínea anterior, qual foi o algoritmo de escalonamento que conseguiu extrair melhor desempenho global da máquina? Sugestão: relacione com o total de iterações completadas pelos 3 processos.

2. Considere o seguinte programa multi-tarefa:

```
int main(int argc, char **argv) {
      for(i = 0; i < 3; i++) {
            if (xthread_create(calculaResultados, (void *)i) == NULL) {
                  exit(1);
            }
      }
      //Imprime no ecrã resultados que as 3 tarefas colocaram num vector partilhado
      printResults();
      exit(0);
}
```

a. [0,8v] Por vezes os resultados apresentados no ecrã são incompletos. Indique qual o erro do programa que está na origem desse comportamento e indique como o corrigiria.

b. [0,8v] Consoante a API xthread seja implementada por tarefas reais ou por pseudo-tarefas, o número de chamadas sistema é diferente? Justifique.

3. Em Linux, considere uma tarefa em execução que está a meio do seu quantum.
   a. [0,8v] Essa tarefa chama a função *pthread_mutex_lock* e deixa de se executar. Esta situação é possível? Se sim, ilustre com um exemplo; se não, justifique.

b. [0,8v] Essa tarefa chama a função *pthread_muted_unlock* e deixa de se executar (apesar de ainda não ter expirado o seu quantum). Esta situação é possível? Se sim, ilustre com um exemplo; se não, justifique.

Grupo II [8 Val]

Numa oficina de pintura automóvel existe um *robot* que pinta as carroçarias dos automóveis. Estas são colocadas num armazém, uma de cada vez, para serem pintadas. O armazém está inicialmente vazio e tem capacidade para N carroçarias que aí aguardam a sua vez para serem pintadas pelo *robot*.

Quando não há carroçarias para pintar, o *robot* entra em modo "descanso" (*stand-by*) para minimizar o consumo de energia. Assim que houver carroçarias para pintar no armazém, o *robot* deve sair do estado "descanso" e passar ao estado "pintar" movendo uma dessas carroçarias para o local de pintura apropriado.

Se, enquanto o *robot* estiver a pintar, forem entregues mais carroçarias no armazém, estas aguardarão a sua vez no armazém. Se o armazém estiver completamente cheio, a entrega de mais carroçarias deve ser cancelada.

Em resumo, os procedimentos efectuados pelo *robot* e por cada uma das carroçarias, cada um representado por um processo distinto, devem fazer com que:

- *Robot* (procedimento "robot"): deve ser bloqueado (estado "descanso") sempre que não há carroçarias para pintar; deve ser acordado (i.e. passa ao estado "pintar") assim que existem carroçarias para pintar.

- Carroçaria (procedimento "colocaNovaCarrocaria"): deve ser bloqueado (fica no armazém a aguardar) quando o *robot* está ocupado; deve ser acordado quando o robot termina de pintar.

Considere ainda que o procedimento "moveCarrocariaParaPintura" respeita a ordem FIFO (relativamente às carroçarias que são colocadas no armazém) e que os semáforos são FIFO.

Assim, considere a solução em pseudo-código que se apresenta de seguida, que está incompleta.

```
#define DIM_ARMAZEM N

mutex_t mutex;

int carrocarias_espera = 0;

semaforo_t carrocarias, robot;
```

```
main{ /*…*/

carrocarias = CriarSemaforo (     );

robot = CriarSemaforo (     );

mutex = CriarMutex(          );

/*...*/}
```

Completar com valor inicial dos dois semáforos e o mutex

```
void robot() {
  while(TRUE) {
    [                                ]
    [                                ]
    moveCarrocariaParaPintura ();
    carrocarias_espera--;
    [                                ]
    pintaCarrocaria ();
    [                                ]
  }
}
```

```
int colocaNovaCarrocaria() {
  [                                ]
  if (carrocarias_espera < DIM_ARMAZEM) {
    carrocarias_espera++;  /*mais uma carrocaria*/
    [                                ]
    [ /*avisar robot que há carrocarias */ ]
    [ /* esperar que robot termine */ ]
  } else {
    Abrir(mutex);
    return 0; //desistir se não há espaço no armazem
  }}
```

1. [1 Val] Complete o procedimento “main” com o valor inicial dos dois semáforos e do mutex. Responda no local indicado sem rasuras; pode usar lápis; justifique nas linhas em baixo.

2. [2 Val] Complete o procedimento “robot” preenchendo as caixas sabendo que cada uma se destina a uma e apenas uma instrução. Responda no local indicado sem rasuras; pode usar lápis.

3. [2 Val] Complete o procedimento “colocaNovaCarrocaria” preenchendo as caixas sabendo que cada uma se destina a uma e apenas uma instrução. Responda no local indicado sem rasuras; pode usar lápis.

4. [1 Val] Imagine que não usava o mutex indicado. O que poderia suceder (descreva um cenário concreto) ?

5. [1 Val] O que poderia suceder se a instrução "Abrir(mutex);" no final do procedimento "colocaNovaCarrocaria" não existisse (descreva um cenário concreto) ?

6. [1 Val] Imagine uma situação inicial na qual o armazém, em vez de estar inicialmente vazio, continha M carroçarias (em que M≤N); qual o valor com que devia inicializar o semáforo "carrocarias" ?

Grupo III [5 Val]

1. Numa partição ext2/ext3 com blocos de 4kBytes, existe um ficheiro com nome "/usr/home/so.txt", cujos atributos são:
   - Número de i-node (i-number): 1204
   - Criado em 19/11/2012, 12:00
   - Tamanho: 57 kBytes
   - Dados armazenados nos blocos cujos identificadores são: 42, 43, 44, 45, 46, 47, 48, 49, 50, 60, 61, 89, 90, 91, 92
   - Dono: UID=100, GUID=20
   - Permissões: `RW-RW----`

a. [1,1v] Apresente, sob a forma de uma tabela, o conteúdo da directoria "/usr/home".
Caso necessite da informação, o i-number de "/usr/home" é 800 e o i-number de "/usr" é 54.

b. [1v] Considere os dados deste ficheiro, armazenados nesta partição em blocos de 4kBytes. Existe fragmentação interna? Se sim, quantifique-a (em número de kBytes desperdiçados). Se não, justifique.

c. [1,1v] Apresente o conteúdo do vector de índices de blocos de dados do i-node do ficheiro "/usr/home/so.txt".
Notas:
- Caso lhe falte alguma informação (não indicada nos atributos acima) para responder, invente essa informação;
- Caso o ficheiro necessite de blocos de índice, apresente o seu conteúdo também;
- O vector de entradas de blocos de dados num i-node no ext3 inclui: 12 entradas directas, uma indirecta de nível 1, uma indirecta de nível 2 e uma indirecta de nível 3.

d. [0,7v] Tendo em conta os atributos indicados neste enunciado, que outros campos do i-node de "/usr/home/so.txt" cujos valores conhece? Indique cada campo e valor.

e. [1,1v] Assuma que os processos P1 e P2 executaram as seguintes instruções, respectivamente:

```
P1:                                          P2:
f = open("/usr/home/so.txt", "r");           g = open("/usr/home/so.txt", "rw");
read(f, buffer, 60);
```

Apresente o conteúdo das estruturas voláteis do sistema de ficheiros imediatamente após P1 e P2 terem executado as instruções apresentadas acima.
Na sua resposta, inclua apenas as tabelas de ficheiros abertos e a tabela de i-nodes em RAM (também chamada cache de i-nodes).