UFES – Departamento de Informática

1ª. Prova de Sistemas Operacionais – 2018/1 – Profª. Roberta L. Gomes

Aluno: _____

1) **(2,5)** Em algumas implementações do UNIX, o kernel é não-preemptivo. O que isto significa? Quais as vantagens e desvantagens dessa abordagem?

2) **(2,0)** Considere um sistema com um escalonador por fatia de tempo e processos em sua maioria interativos (um editor de texto por exemplo). Suponha que, na média, um processo seja executado por 10 ms até gerar uma solicitação de E/S com bloqueio. Suponha que 90% dos ciclos de processador tem uma duração entre 8 ms e 15 ms. Do ponto de vista do escalonamento, qual o efeito (overhead e tempo de resposta) de:

a) Uma fatia de tempo pequena, como por exemplo 1 ms.

b) Uma fatia de tempo grande, como por exemplo 100 ms.

3) **(2,5)** O código a seguir pode gerar a sequência de mensagens ilustradas no quadro "SAÍDA" exibido à direita do código? Explique sua resposta construindo um grafo de precedência em que os nós desse grafo são os rótulos associados às instruções "printf(...)" do código (isto é, as letras 'A' até 'G'). Dessa forma, o grafo de precedência representará a ordem em que as funções "printf(...)" podem ser executadas. Suponha que todas as chamadas "fork" e "exec" são bem sucedidas.

```
int main()
{ int f1,f2;
  printf("Hello, I'm the boss!\n"); //A
  f1=fork();
  if(f1==0) {
     printf("No... I AM the boss!\n"); //B
     execlp("prog1","prog1",NULL);
     printf("See, I've told you!\n"); //C
  }
  printf("Sorry, you are a dead BOSS!\n"); //D
  f2=fork();
  if(f2==0){
     waitpid(f1,NULL,NULL);
     execlp("prog2","prog2",NULL);
  }
  waitpid(f2,NULL,NULL);;
  printf("All right! All dead now!\n"); //E
  exit(0);
}
```

****** SAÍDA ******

```
Hello, I'm the boss!
No... I AM the boss!
Sorry, you are a dead BOSS!
Hi from 2...
All right! All dead now!
Hi from 1...
```

```
//*** prog1
int main()
{  printf("Hi from 1...\n"); //F
   exit(0); }
```

```
//*** prog2
int main()
{  printf("Hi from 2...\n"); //G
   exit(0); }
```

OBSERVAÇÃO

Um grafo de precedência é um grafo direcionado em que a relação (x)→(y) indica que 'x' precede 'y'. Para este problema, os nós do grafo terão uma letra (A...F) representando chamadas "printf":

*(A)→(B) indica que o print realizado na linha **A**, sempre aparecerá na saída antes do print realizado na linha **B** ..*

4) **(3,0)** O código abaixo tenta resolver o problema do "Produtor-Consumidor" em um buffer compartilhado, usando as primitivas `sleep/wakeup`.

a) Qual o problema com essa solução?

```
#define N 100                   /* number of slots in the buffer */
int count = 0;                  /* number of items in the buffer */

void producer(void) {
 while (true){
   produce_item();              /* generate next item */
   if (count == N)
      sleep();                  /* if buffer is full, go to sleep */
   enter_item();                /* put item in buffer */
   count = count + 1;           /* increment count of items in buffer*/
   if (count >= 1){             /* was buffer empty? */
     wakeup(consumer);}
 }
}

void consumer(void){
 while (true){
   if (count == 0)
      sleep();                  /* if buffer is empty, got to sleep */
   remove_item();               /* take item out of buffer */
   count = count - 1;           /* decrement count of items in buffer*/
   if (count <= N-1){
      wakeup(producer);}        /* was buffer full? */
   consume_item();              /* print item */
 }
}
```

b) Corrija a solução alterando o código apresentado no quadro abaixo, usando os semáforos `s1`, `s2` e `s3` (foram retiradas as chamadas `sleep/wakeup`). Considere que as funções `enter_item()` e `remove_item()`, ambas acessam todos os índices de inserção e remoção do buffer (compartilhados), ou seja, esses índicies são acessados de forma concorrente.

```
#define N 100
int count = 0;
semaphore s1 = ___ ;
semaphore s2 = ___ ;
semaphore s3 = ___ ;

void producer(void) {
  while (true){

    produce_item();

    if (count == N){

       enter_item();

    }

    count = count + 1;

    if (count >= 1){

    }
  }
}
```

```
void consumer(void){
  while (true){

    if (count == 0) {

      remove_item();

    }

    count = count - 1;

    if (count <= N-1){

    }

    consume_item();
  }
}
```

Boa prova!!!