Sistemas Operacionais
Segunda Prova - 12/07/2022

NOME: _____

1. Um certo conjunto de processos concorrentes entra em *deadlock* **toda vez** que é executado, independentemente da máquina em que a execução ocorre. Qual das condições de existência de *deadlock* deve ser tratada para evitar essa situação? Por quê? (VALOR 2 pontos)

   Essa situação indica que existe um erro de programação nesses processos, por levarem sempre ao *deadlock*. Com isso, a melhor solução é impedir a formação de ciclos de espera, o que pode ser feito corrigindo o erro de programação.

2. Um projetista de um S.O. pensa em usar o algoritmo FIFO para gerenciamento de disco. Que tipo de aplicações/sistema podem justificar essa escolha? (VALOR 1,5 pontos)

   Isso é justificável p/ sistemas em que a taxa de requisições de acesso ao disco seja relativamente pequena em relação à velocidade de acesso, ou seja, situações em que não tenham filas de requisições. Também é adequado para aplicações que demandem acesso em blocos de páginas, sem intercalação com requisições de outras aplicações.

3. Considerando E/S para um dispositivo SSD, qual seria a vantagem em acumular, quando possível, várias operações de escrita para serem feitas de uma vez? (VALOR 2 pontos)

   Isso permite que a escrita envolva um bloco completo, em vez de várias páginas de forma separada, o que consumiria mais tempo para as escritas.

4. Sistemas como o skype fazem uso de colaboração entre sistemas, permitindo que milhares de diálogos ocorram simultaneamente. O que é preciso modificar no S.O. para se ter um gerenciamento da E/S para rede eficiente? (VALOR 1,5 pontos)

   Uma modificação interessante é colocar prioridades diferentes para acesso ao dispositivo (placa de rede), o que significa dar maior prioridade (maior fluxo de tráfego) para as aplicações que demandam tráfego mais intenso, como o skype.

5. Considerando o problema dos filósofos (compartilhamento dos hashis), imagine uma solução em que em vez de pegar um hashi por vez, cada filósofo peça os dois hashis para um garçom, que lhe entregará dois hashis se houver disponibilidade. Após comer o filósofo devolve os hashis ao garçom. Construa os processos filosofo() e garcom() usando semáforos para controlar o acesso aos hashis. **Observação:** o garçom tem que atender tanto os pedidos como as devoluções de hashi. (VALOR 3 pontos)

A solução é transformar o garçom em um processo semelhante ao barbeiro (problema do barbeiro dorminhoco). Assim, o garçom espera que um filósofo faça uma chamada, quando avisa se está pedindo ou devolvendo os hashis. O garçom trata cada chamada avaliando quantos hashis estão disponíveis e o tipo da chamada. Se for requisição de hashi ele verifica se tem pelo menos dois hashis e libera o filósofo se for o caso ou o coloca para esperar. Se for devolução, então libera algum filósofo se tiver alguém esperando ou apenas atualiza os hashis.

```
process garcom(){
    while (true) {
        P(s);
        P(mutex);
        if (tipo == 1)
          { filo = buffer[k];
            k = (k+1)%n;
            if (nhashi > 2)
              { nhashi -= 2;
                V(sem[filo]);
              }
            else
              coloca_na_fila(filo);
          }
        else
          { if (tem_alguem_na_fila())
              { filo = primeiro_fila();
                V(sem[filo]);
              }
            else
              nhashi += 2;
          {
        V(mutex);
    }
}
```

```
process filosofo(i=1 to n-1){
  while (true) {
        think();
        P(mutex);
        buffer[k] = i;
        tipo = 1;
        V(s);
        V(mutex);
        P(sem[i]);
        eat();
        P(mutex);
        tipo = 2;
        V(s);
        V(mutex);
  }
}
```

Boa prova,