Sistemas Distribuídos - COS470
2023/1

Segunda Lista de Exercícios

Dica: Para ajudar no processo de aprendizado responda às perguntas integralmente, mostrando o desenvolvimento das respostas.

Questão 1: Descreva sucintamente as duas principais abordagens utilizadas para oferecer IPC (Inter-Process Communication). Quais as vantagens e desvantagens de cada abordagem?

Questão 2: Considere um *pipe* entre dois processos (ou duas threads) P_1 e P_2, cada qual no *write end* e *read end* do pipe, respectivamente. Considere o seguinte trecho de código:
P_1:

```
while(1) {
  ...
  write(pipe, ...)
  ...
}
```

P_2:

```
while(1) {
  ...
  read(pipe, m, ...)
  process(m)
  ...
}
```

Considere que a função *process()* pode demorar alguns segundos para retornar. Considere as duas variações das chamadas ao sistema (*system call*) para as funções de *read* e *write*: bloqueante e não-bloqueante. Para cada combinação, descreva o que pode acontecer com o programa acima, focando no estado do pipe e no parâmetro m passado para a função *process* (ex. *write* bloqueante e *read* não-bloqueante).

Questão 3: Quais são as vantagens de construir um sistema *multi-threaded* em comparação a *multi-processed*? Cite uma possível desvantagem.

Questão 4: Considere um sistema *multi-threaded* baseado em *user-level threads*. O desempenho deste sistema executando em um processador multi-processado (ex. múltiplos núcleos) será superior ao mesmo sistema executando em um processador mono-processado? Explique sua resposta.

Questão 5: Explique o que é uma *condição de corrida*? Ilustre com algum pseudo-código os problemas que ela pode causar.

Questão 6: Considere as seguintes funções de *acquire()* e *release()* para implementação de exclusão-mútua a uma região crítica compartilhada por duas threads:

```
struct lock {
  bool interested[2] = [0,0];
}

void acquire(lock) {
  lock->interested[this_thread] = 1
  while(lock->interested[other_thread]);
}

void release(lock) {
  lock->interested[this_thread] = 0;
}
```

Explique o que pode acontecer de errado mesmo assumindo que cada instrução é executada atomicamente.

Questão 7: Considere a seguinte função, que troca o valor de duas variáveis:

```
void swap (bool *a, *b) {
  bool temp = *a;
  *a = *b;
  *b = temp
}
```

Assuma que esta função é implementada atomicamente (sem interrupção). Mostre como esta função pode ser utilizada para implementar locks, definindo o pseudo-código das funções de *acquire()* e *release()* adequadamente.

Questão 8: Considere o seguinte trecho de código sendo executado por um conjunto de threads:

```
while(1) {
    wait(s1);
    print("a");
    wait(s2);
    print("b");
    signal(s2);
    signal(s1);
}
```

onde s1 e s2 são semáforos binários. Determine os padrões que podem ser impressos no terminal. Determine ainda se pode haver algum *deadlock*. Explique sua resposta.

Questão 9: Considere o problema dos Produtores-Consumidores utilizando semáforos conforme visto em aula. Explique o que pode acontecer se trocarmos a ordem das chamadas *wait()* no Produtor.

Questão 10: Qual a diferença entre a chamada *signal()* associada a semáforos e a mesma chamada associada a monitores?

Questão 11: Considere o seguinte trecho de código sendo executado por um conjunto de threads:

```
Monitor Operacao {
    double a;
    Condition positive;

    double raiz() {
      if (a < 0) wait(positive);
      return sqrt(a);
    }

    void inc() {
      a = a + 1;
      if (a > 0) signal(positive)
    }

    void dec() {
      a = a - 1;
    }
}
```

Considere que muitas threads estão usando este monitor. Explique o comportamento do mesmo para a semântica Mesa e Hoare. Em particular, o que pode acontecer na semântica Mesa?