Sistemas Operacionais
Terceira Lista de Exercícios

Norton Trevisan Roman
Clodoaldo Aparecido de Moraes Lima

19 de setembro de 2022

1. Diferencie processo e thread
2. No modelo de threads, o conjunto de registradores é relacionado como um item por thread, e não por processo. Por que? (Afinal, a máquina tem somente um conjunto de registradores)
3. O que faria uma thread no espaço do usuário desistir voluntariamente da CPU, chamando *thread _yield*? Afinal, como não há interrupção na thread, eal pode nunca mais obter a CPU de volta.
4. Uma thread específica pode sofrer preempção por uma interrupção do relógio? (Considere os modelos de threads no espaço do usuário e do núcleo)
5. Considere a leitura de um arquivo usando um servidor de arquivos. São necessário 15ms para obter uma requisição de trabalho, despachá-la e fazer o restante do processamento necessário, presumindo que os dados essenciais estejam na cache (de blocos). Se for necessária uma operação de disco – como ocorre em um terço das vezes –, será preciso um tempo adicional de 75ms, durante o qual a thread dorme. Quantas requisições/segundo o servidor pode tratar se for monothread? E se for multithread?
6. Qual a maior vantagem de implementar threads no espaço do usuário? Qual a maior desvantagem?
7. No programa abaixo:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NUMERO_DE_THREADS 10

void *ola_mundo(void *tid) {
  printf("Ola mundo. Saudacoes da thread %d\n",tid);
  pthread_exit(null);
}

int main(int argc, char *argv[]) {
  pthread_t threads[NUMERO_DE_THREADS];
  int status, i;

  for (i=0; i<NUMERO_DE_THREADS; i++) {
    printf("Thread principal. Criando thread %d\n",i);
    status = pthread_create(&threads[i], null,
                                ola_mundo, (void *)i);
    if (status != 0) {
      printf("pthread_create retornou o codigo de erro
```

```
                                        %d\n",status);
      exit(-1);
    }
  }
  exit(null);
}
```

as criações de threads e as mensagens impressas por elas são intercaladas aleatoriamente. Há algum modo de impor que a ordem seja estritamente thread 1 criado, thread 1 imprime mensagem, thread 1 sai, thread 2 criado, thread 2 imprime a mensagem, thread 2 sai e assim por diante? Em caso de resposta afirmativa, qual é esse modo? Em caso de resposta negativa, por que não?

8. Porque podemos afirmar que threads distintos em um processo não são tão independentes quanto processos distintos?

9. Porque cada thread precisa ter sua própria pilha?

10. Em um sistema com threads, quando são utilizadas threads de usuário, há uma pilha por thread ou uma pilha por processo? E quando se usam threads de núcleo? Explique.

11. O que significam as condições de disputa (ou de corrida)? O que fazer para evitar esse problema?

12. Quando um computador está sendo desenvolvido, ele é antes simulado por um programa que executa uma instrução por vez. Mesmo os multiprocessadores são simulados de modo estritamente sequencial. É possível que ocorra uma condição de corrida quando não há eventos simultâneos como nessas simulações?

13. O que significa a expressão Exclusão Mútua (EM)? Como ela pode ser empregada?

14. Quais as vantagens e desvantagens de desabilitar interrupções?

15. A solução de espera ociosa usando a variável *turn* funciona quando os dois processos estão executando em um multiprocessador de memória compartilhada, isto é, duas CPUs compartilhando uma memória comum?

16. A solução de Peterson para o problema da exclusão mútua funciona quando o escalonamento do processo for preemptivo? E quando o escalonamento não for preemptivo?

17. Duas threads podem, no mesmo processo, sincronizar a partir do uso de um semáforo de núcleo se as threads forem implementadas pelo núcleo? E se fossem implementadas no espaço do usuário? (Suponha que nenhuma thread em qualquer outro processo tenha acesso ao semáforo). Comente suas respostas.

18. Em uma aplicação concorrente que controla saldo bancário em contas correntes, dois processos compartilham uma região de memória onde estão armazenados os saldos dos clientes A e B. Os processos executam, concorrentemente os seguintes passos:

```
Processo 0 (Define os semáforos compartilhados)
int S1 = ?;
int S2 = ?;
```

```
Processo 1 (Cliente A)                  Processo 2 (Cliente B)
/* saque em A */                        /* saque em A */
1a. x = saldo_do_cliente_A;             2a. y = saldo_do_cliente_A;
1b. x = x - 200;                        2b. y = y - 100;
1c. saldo_do_cliente_A = x;             2c. saldo_do_cliente_A = y;

/* deposito em B */                     /* deposito em B */
1d. x = saldo_do_cliente_B;             2d. y = saldo_do_cliente_B;
1e. x = x + 100;                        2e. y = y + 200;
1f. saldo_do_cliente_B = x;             2f. saldo_do_cliente_B = y;
```

Supondo que S1 e S2 são compartilhadas (tendo sido definidas pelo Processo 0 antes dos outros dois processos); que saldo_do_cliente_A e saldo_do_cliente_B são variáveis compartilhadas que refletem, em uma base de dados, os saldos dos clientes A e B, respectivamente; e que os valores dos saldos de A e B sejam, respectivamente, 500 e 900, antes de os processos executarem, pede-se:

(a) Quais os valores corretos esperados para os saldos dos clientes A e B após o término da execução dos processos?

(b) Quais os valores finais dos saldos dos clientes se a sequência temporal de execução das operações for: 1a, 2a, 1b, 2b, 1c, 2c, 1d, 2d, 1e, 2e, 1f, 2f?

(c) Utilizando dois semáforos (S1,S2), proponha uma solução que garanta a integridade dos saldos e permita o maior compartilhamento possível dos recursos entre os processos, não esquecendo a especificação da inicialização dos semáforos (Feita no Processo 0 acima).

19. Considere o problema do consumidor e produtor. Para o código abaixo, coloque os semáforos na posição correta

```
#define N 100 /* número de posições do buffer*/
typedef int semaphore;
semaphore mutex = 1; /*controla o acesso a RC*/
semaphore empty = N; /*conta as posições vazias do buffer*/
semaphore full = 0; /*conta as posições ocupadas do buffer*/
void producer (void)
{
   int item;
   while (1){
    item = produce_item( ); /*produz um novo item*/
    --------------------
    ---------------------
    enter_item(item); /*coloca novo item no buffer*/
    ---------------------
    ---------------------
    }
}

void consumer(void)
{
int item;
while (1){
   ----------------------
   ----------------------
   item = remove_item( ); /*retira 1 item do buffer*/
   ---------------------
   ---------------------
   consume_item(item); /*consome um novo item no buffer*/
   }
```

```
}
```

20. Um restaurante de fast-food tem quatro tipos de empregados: (1) anotadores de pedido, que anotam o pedido dos clientes; (2) cozinheiros, que preparam a comida; (3) embaladores, que colocam a comida nas sacolas; e (4) caixas, que entregam as sacolas para os clientes e recebem o dinheiro deles. Cada empregado pode ser observado como um processo sequencial de comunicação. Qual forma de comunicação entre processos eles usariam? Relacione esse modelo aos processos no UNIX.

21. Imagine um sistema por troca de mensagens que use caixas postais. Quando se envia para uma caixa postal cheia ou tenta-se receber de uma caixa postal vazia, um processo não bloqueia. Na verdade, ele obtém um código de erro. O processo responde ao código de erro apenas tentando novamente, sucessivamente, até que ele consiga. Esse esquema leva a condições de corrida?

22. O que é starvation e como podemos solucionar esse problema?

23. Por que podemos afirmar que a utilização de Semáforos não é eficaz quando aplicada em SOs distribuídos?

24. Explique, dando exemplos, o mecanismo rendez-vous.

25. Explique, dando exemplos, o mecanismo rendez-vous estendido. Diferencie-o do rendez-vous tradicional

26. Explique o que é um pipeline na comunicação entre processos