

8,0

Nome: Vitor A. Apolinário

- MANTER DESLIGADO E GUARDADO CELULARES, COMPUTADORES E CALCULADORAS.
- A COMPREENSÃO DAS QUESTÕES FAZ PARTE DA AVALIAÇÃO!!!

1. [1,0 ponto] **SO Kid** se autodenomina um dos maiores especialistas em sistemas operacionais. Ele afirma que no escalonamento de processos por loteria (*lottery scheduling*) os processos com apenas um (1) bilhete acabam necessariamente padecendo de inanição (*starvation*). **SO Kid está correto? Explique.**

2. [1,5 ponto] Considerando a solução de exclusão mútua com espera ociosa baseada em chaveamento obrigatório (abaixo exemplo de código para 2 processos: (a) processo 0; (b) processo 1), **SO kid** pede que você implemente uma solução para tratar N processos (assumindo $N > 2$). Apresente a sua solução, explicando-a.

```
1,5
while (TRUE) {
    while (turn != 0)           /* Laço */
        critical_region();
    turn = 1;
    noncritical_region();
}
```

(a)

```
while (TRUE) {
    while (turn != 1)           /* Laço */
        critical_region();
    turn = 0;
    noncritical_region();
}
```

(b)



3. [2,0 pontos] Em um aplicativo *multi-thread* desenvolvido por **SO Kid**, as *threads* concorrem pelo acesso aos arquivos denominados **papu**, **papô**, **papá**, **papi** e **papé**. Os arquivos sempre são acessados na modalidade **leitura e escrita**, exigindo que se controle o acesso exclusivo ao arquivo (*i.e.*, caso o arquivo esteja disponível, a *thread* que conseguir acesso ao arquivo adquira o lock sobre o mesmo, impedindo o acesso às demais *threads*). **SO Kid** definiu algumas regras a serem seguidas pelas *threads* a fim de se evitar *deadlocks*. As regras são:

- I) Caso a *thread* consiga acesso ao arquivo **papá**, poderá tentar acessar o arquivo **papé** e nada mais;
- II) Caso a *thread* tenha conseguido acesso ao arquivo **papu**, poderá somente tentar acesso aos arquivos **papi** e **papé** mas, caso decida primeiro acessar o arquivo **papé**, não poderá mais tentar acessar o arquivo **papi**;
- III) Caso a *thread* tenha conseguido acesso ao arquivo **papô** e **papu** (possível, desde que solicitado/obtido nessa ordem), poderá tentar acessar apenas o arquivo **papi**. **A solução de SO Kid previne impasses? Explique.**

OBS.: a) cada *thread* pode manter abertos múltiplos arquivos simultaneamente (desde que possível segundo as regras estabelecidas); **b)** assume-se que as *threads* acessam os arquivos com frequência mas sempre por um tempo finito (caso consigam acesso, naturalmente).

4. [1,0 ponto] **Para cada um** dos seguintes endereços binários virtuais, calcule o número da página virtual e o deslocamento (*offset*) considerando páginas de 256 bytes. Apresente o desenvolvimento do cálculo em decimal.

(a) 0011 1000 0110 1111 pg = 56 offset = 111

(b) 1001 0000 0001 1011 pg = 144 offset = 27

5. [1,0 ponto] Marque V (verdadeiro) ou F (falso) para cada uma das assertivas abaixo:

- (F) O sistema operacional xv6 adota uma abordagem não preemptiva para o escalonamento de processos.
 (F) A arquitetura *microkernel* exige que todos os *drivers* suportados pelo sistema operacional sejam carregados no momento da inicialização do SO.
 (V) As instruções de configuração do intervalo de interrupção do TIMER pertencem ao conjunto de instruções privilegiadas.
 (V) Quando se emprega gerenciamento de memória baseada em paginação, tem-se fragmentação interna de memória.

6. [2,0 pontos] Considere o seguinte problema envolvendo programação multithread, mutexes e semáforos:

“- A partir da thread principal criar N threads;
- Cada thread executa, basicamente, a mesma tarefa que consiste em incrementar uma variável global inicializada com valor zero (0); no entanto, a cada rodada envolvendo todas as threads, cada thread incrementa a variável global uma única vez. Além disso, a alternância entre as threads dá-se sempre em ordem crescente de identificadores. Assumir identificadores das threads incrementais iniciando-se a primeira thread com ID=0. O incremento se encerra quando for atingido um valor máximo (MAX). Exemplo: assumindo que existem 3 threads (ids 0, 1 e 2), ter-se-á a seguinte apresentação de incremento da variável global:

- thread 0: global = 1;
- thread 1: global = 2;
- thread 2: global = 3;
- thread 0: global = 4;
- thread 1: global = 5;
- ...”

Pois bem, **SO Kid** codificou em linguagem C (plataforma Linux) mas, apesar de compilar corretamente, a execução do programa dele não apresenta o resultado esperado. Utilizando-se do fragmento principal do código dele (abaixo), identifique o(s) problema(s) e apresente a(s) respectiva(s) correção(ões), mas sem remover ou acrescentar novas estruturas de dados, mutexes e semáforos. APRESENTE A(S) CORREÇÃO(ÕES) NO PRÓPRIO CÓDIGO ABAIXO!

```
...
void *mythread(void *data);

#define N 3 // number of threads
#define MAX 10
// vetor de semáforos (uma entrada por thread)
// utilizado para controlar o rodízio/rodada
sem_t turn[N];

int global = 0;

int main(void) {
    pthread_t tids[N];
    int i=0;

    // inicializa vetor de semáforos
    for(i=0; i<N; i++) {
        sem_init(&turn[i], 0, 0);
    }
    sem_post(&turn[0]); // permite que T0 execute primeiro

    for(i=0; i<N; i++) {
        int *j = malloc(sizeof(int));
        *j = i;
        pthread_create(&tids[i], NULL, mythread, (void *)j);
    }

    for(i=0; i<N; i++) {
        pthread_join(tids[i], NULL);
        printf("Thread id %ld retornou \n", tids[i]);
    }

    return(1);
}
```

```
void *mythread(void *data) {

    int id;
    id = *((int *) data);

    while(global < MAX) {

        sem_wait(&turn[id]);
        global++;
        printf("\n thread %d: global = %d", id, global);
        sem_post(&turn[(id+1)%N]);
        sleep(2);
    }

    pthread_exit(NULL);
}
```

✓

(1,5)

7. [1,5 ponto] Semelhante ao exercício anterior, agora há um conjunto de *threads* tentando manipular uma variável global mas sem nenhuma ordem preestabelecida. No entanto, **apenas exige-se que se garanta a exclusão mútua ao se atualizar a variável global**. Após inicializar a execução do programa de SO Kid, observa-se que o mesmo não produz o resultado esperado e sequer finaliza. Utilizando-se do fragmento principal do código de SO Kid (abaixo), identifique o(s) problema(s) e apresente a(s) respectiva(s) correção(ões) NO PRÓPRIO CÓDIGO!

```
void *mythread(void *data);
pthread_mutex_t count_mutex = PTHREAD_MUTEX_INITIALIZER;

#define N 3 // number of threads
#define MAX 10
int global = 0;

int main(void) {
    pthread_t tids[N];
    int i;

    for(i=0; i<N; i++) {
        pthread_create(&tids[i], NULL, mythread, NULL);
    }

    for(i=0; i<N; i++) {
        pthread_join(tids[i], NULL);
        printf("Thread id %ld returned\n", tids[i]);
    }
    return(1);
}

void *mythread(void *data) {

    while(global < MAX) {
        pthread_mutex_lock(&count_mutex);
        global++; X
        printf("Thread ID%ld: global is now %d.\n", pthread_self(), global);
        sleep(2); X
    → pthread_mutex_unlock (&count_mutex);
    }

    pthread_exit(NULL);
}
```



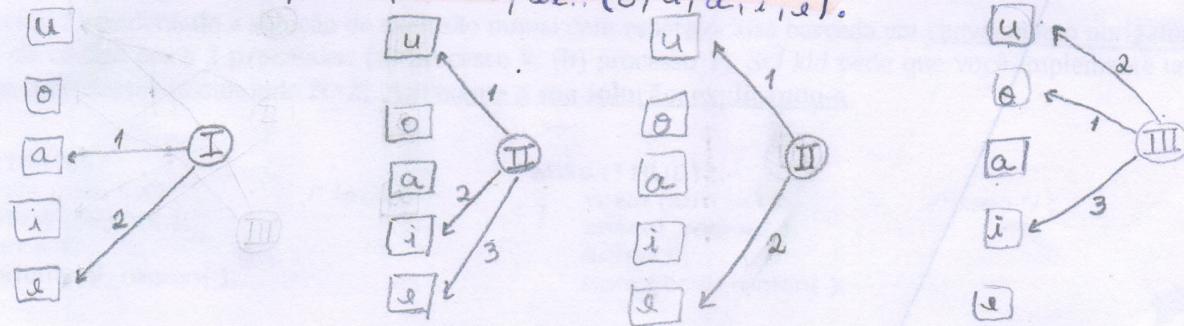
SO kid definitivamente não parou de bugar. Ele esquece que desbloquear o mutex, assim os threads paralelos (^{subsequentes}) sempre bloqueiam e não executam.

① Não, pois mesmo com uma chance pequena ele ainda pode ser recarregado, e não "morrer" por falta de CPU.

② `while (TRUE) {`
 `M é o número de cada processo (identificador)`
 `while (turn != M) de cada processo (identificador) 1)`
 `critical-region () N é a q.t. de processos`
 `turn = (turn+1)%N; N=3`
 `noncritical-region (); (0,1,2)`
 `} à ser chavado e a ser chavado`
 `...`

O chavamento ocorre p/ um processo de identificador subsequente. Repetindo threads, turn = 2, o próximo processo proc = (2+1)%3 \Rightarrow proc = 0

③ Reposta: A solução é livre de ímpares. Pode ser possível recarregar os arquivos de uma forma que uma thread nunca pegue um recurso de nível inferior ao que ela possui, ex.: (0, u, a, i, e).



Utilizando o conceito de espera circular a solução não é livre de ímpares. Considerando que papel, papo, papa, papé e papé não arranjados e enumerados nesta ordem, a regra III permite que a thread pegue um recurso de ordem menor do que ela já possui, possibilitando arrumos como ímpares.

④ a) $2^8 = 256$
~~offset = 8 lists~~

$$Rg = \frac{32}{\begin{array}{r} +16 \\ +48 \\ +8 \\ \hline 56 \end{array}}$$

$$\text{offset} = \frac{64}{\begin{array}{r} +32 \\ +196 \\ +8 \\ \hline 104 \\ +4 \\ \hline 111 \end{array}}$$

b) $Rg = \frac{128}{\begin{array}{r} +16 \\ +444 \\ \hline \end{array}}$
 $\text{offset} = \frac{16}{\begin{array}{r} +11 \\ +27 \\ \hline \end{array}}$

10) para que sejam em uso todos os bytes da memória virtual, calcule o tamanho de página virtual e o deslocamento (offset) da página na tona base, apontando o deslocamento da página de endereço 1000.

11) para que sejam em uso todos os bytes da memória virtual, calcule o tamanho de página virtual e o deslocamento (offset) da página na tona base, apontando o deslocamento da página de endereço 1000.

12) para que sejam em uso todos os bytes da memória virtual, calcule o tamanho de página virtual e o deslocamento (offset) da página na tona base, apontando o deslocamento da página de endereço 1000.

13) para que sejam em uso todos os bytes da memória virtual, calcule o tamanho de página virtual e o deslocamento (offset) da página na tona base, apontando o deslocamento da página de endereço 1000.

14) para que sejam em uso todos os bytes da memória virtual, calcule o tamanho de página virtual e o deslocamento (offset) da página na tona base, apontando o deslocamento da página de endereço 1000.

15) para que sejam em uso todos os bytes da memória virtual, calcule o tamanho de página virtual e o deslocamento (offset) da página na tona base, apontando o deslocamento da página de endereço 1000.