

6,5

Nome:

Leonardo Uerlang

- MANTER DESLIGADO E GUARDADO CELULARES, COMPUTADORES, CALCULADORAS, ETC.
- 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 dois 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.

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

(a)

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

(b)

3. [1,5 ponto] Em um aplicativo *multi-thread* desenvolvido por **SO Kid**, as *threads* concorrem pelo acesso aos arquivos **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 adquire o *lock* sobre o mesmo). **SO Kid** definiu algumas regras a serem seguidas pelas *threads* a fim de se evitar *deadlocks*. As regras são:

- Caso a *thread* consiga acesso ao arquivo **papá**, poderá tentar acessar o arquivo **papé** e nada mais;
- 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**;
- 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); c) operações não previstas nas regras são proibidas.

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

(b) 1001 0000 0001 1011

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.
- (V) A arquitetura *microkernel* exige que todos os *drivers* suportados pelo sistema operacional sejam carregados no momento da inicialização do SO.
- (V) As instruções envolvidas na definição do intervalo de interrupção do **TIMER** pertencem ao conjunto de instruções privilegiadas.
- (V) Quando se emprega gerenciamento de memória baseado em paginação, tem-se fragmentação interna de memória.

6. [2,0 pontos] Agora relembre do seguinte exercício apresentado em aula/laboratório:

- 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. 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 na linguagem C (plataforma Linux) mas, apesar de compilar corretamente, a execução do seu programa não apresenta o resultado esperado. Utilizando-se do fragmento principal do código abaixo, identifique o(s) problema(s) e apresente a(s) respectiva(s) correção(ões), mas sem remover ou acrescentar novas estruturas de dados. Caso deseje, realize a(s) correção(ões) diretamente no 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 rodizio
sem_t turn[N];

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

    pthread_exit(NULL);
}
```

A THREAD 0  
 VAL GLOBAL  
 ANTES DE  
 PM 0  
 POST PM  
 A THREAD 1

SEM\_INIT(&TURN[0], 0, 1)

7. [2,0 pontos] **SO Kid** também se dedicou a outro exercício apresentado em aula/laboratório. 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).**

→ Caso deseje, realize a(s) correção(ões) diretamente no código abaixo.

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

#define N 3 // number of threads
#define MAX 10
int x = 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(x < MAX) {
        pthread_mutex_lock(&count_mutex);
        x++;
        printf("Thread ID%ld: x is now %d.\n", pthread_self(), x);
        sleep(2);
        pthread_mutex_unlock(&count_mutex);
    }

    pthread_exit(NULL);
}
```