UFFS – Campus Chapecó - Curso de Ciência da Computação
Sistemas Operacionais – Prof. Marco Aurélio Spohn
2016.2 – Primeira Avaliação (P1) – 26/09/2016

Nome: _____

- **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**.

*SO Kid errou ao afirmar enfáticamente (i.e., ele diz "**necessariamente**") a ocorrência de inanição para processos com apenas um bilhete. Isso não é verdade porque, mesmo com apenas um bilhete, há chances do processo ser sorteado e, consequentemente, ser executado.*

2. [1,0 ponto] ***SO Kid*** afirma que, caso a CPU suporte isolamento de memória via MMU (unidade de gerenciamento de memória), não se torna necessário no mínimo dois modos de operação da CPU (i.e., modo *kernel* e modo usuário) para suportar um sistema protegido/seguro. **SO Kid está correto? Explique**.

Não está correto, porque é imprescindível a existência de um modo privilegiado (i.e., modo kernel) para que se proceda corretamente, e com segurança, o gerenciamento de memória.

3. [1,5 pontos] Em um aplicativo *multi-thread* desenvolvido por ***SO Kid***, as *threads* concorrem pelo acesso aos arquivos **babu, babô, babá, babi e babé**. 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, 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 **babá**, poderá tentar acessar o arquivo **babé** e nada mais;
II) Caso a *thread* tenha conseguido acesso ao arquivo **babu**, poderá somente tentar acesso aos arquivos **babi** ou **babé** mas, caso decida primeiro acessar o arquivo **babé,** não poderá mais tentar acessar o arquivo **babi**;
III) Caso a *thread* tenha conseguido acesso ao arquivo **babô** e **babi** (possível, desde que solicitado/obtido nessa ordem), poderá tentar acessar apenas o arquivo **babu**. **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 máximo pré-definido.

Uma possível solução seria: enumerar os recursos (i.e., arquivos) e verificar se a requisição/alocação ocorre estritamente em ordem crescente de identificadores dos recursos; caso negativo, a solução não previne impasses.
Uma solução alternativa, consiste em apresentar apenas um caso (i.e., prova por contradição) em que existe a possibilidade de deadlock. Por exemplo: Thread 1 (T1) conseguiu acessar os arquivos babô e babi; Thread 2 (T2) conseguiu acessar o arquivo babu; T1 requisita o arquivo babu e bloqueia; T2 requisita o arquivo babi e bloqueia; deadlock!

4. [1,5 pontos] Assuma um computador com apenas **4** molduras (*frames*) de página. O instante de carregamento de página na memória, o instante do último acesso e os bits R (referenciada) e M (modificada) para cada página são mostrados a seguir (os tempos estão em tiques de relógio):

Página	Carregamento	Último acesso	R	M
0	**106**	**260**	**1**	**0**
1	**210**	**245**	**0**	**1**
2	**120**	**250**	**0**	**0**
3	**90**	**265**	**1**	**1**

Responda:
(a) Qual página será trocada segundo a abordagem **Não Usada Recentemente** (**NUR**)? 2
(b) Qual página será trocada pelo **FIFO**? 3
(c) Qual página será trocada pelo **MRU** (Menos Recentemente Usada)? 1 (último acesso em 245)
(d) Qual página será trocada segundo a abordagem **segunda chance**? 2

OBS.: Para cada uma das políticas, assuma sempre o mesmo cenário inicial conforme descrito na tabela acima.

5. [1,0 ponto] **Para cada um** dos seguintes endereços binários virtuais, calcule o número da página virtual e o deslocamento considerando páginas de **512 bytes**. **Apresente o cálculo e o resultado em decimal**.

(a) 0101 1000 1010 1011

Página → 0101 100 = 44
Deslocamento → 0 1010 1011 = 171

(b) 1011 0000 0011 1111

Página → 1011 000 = 88
Deslocamento → 0 0011 1111 = 63

6. [1,0 ponto] *SO Kid* colaborou no projeto de uma determinada *motherboard* baseada em um processador com arquitetura de **64 bits**. No entanto, a *motherboard* deverá suportar, no máximo, **128 Gbytes** de memória RAM. Pensando nessa limitação e no custo associado ao barramento de memória, SO Kid decidiu adotar um barramento de memória de **37** linhas (1 linha = 1 bit). **SO Kid acertou nesse ajuste? Explique**.

Acertou! Com endereçamento de 37 bits pode-se tratar adequadamente os 128 Gbytes (i.e., 2^37 = 128G).

7. [3,0 pontos] ***SO Kid*** codificou na linguagem C, na plataforma Linux, uma solução para o problema do "jantar dos filósofos". No entanto, ao executar o programa, percebeu que não está funcionando como esperado. **Identifique o(s) problema(s) e apresente a(s) respectiva(s) correção(ões).**

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/time.h>
#include <errno.h>
#include <semaphore.h>
#include <fcntl.h>

#define N 5
int left(int id);
int right(int id);
void *philosopher(void *data);
void take_forks(int id);
void put_forks(int id);
void test(int id);

#define THINKING 0
#define HUNGRY 1
#define EATING 2

int state[N];
sem_t mutex;
sem_t s[N];

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

  sem_init(&mutex, 0, 1);
  for(i=0;i<N;i++) {
    sem_init(&s[i], 0, 0);
    state[i]=THINKING;
  }
  for(i=0; i<N; i++) {
    int *j = malloc(sizeof(int));
    *j=i;
    pthread_create(&tids[i], NULL, philosopher, (void *)j);
  }

  for(i=0; i<N;i++) {
    pthread_join(tids[i], NULL);
  }
  return(1);
}
```

```
void *philosopher(void *data){
    int id = *((int *) data);
    while(1){
      printf("\n Philosopher %d is thinking\n",id);
      sleep(2);
      take_forks(id);
      printf("\n Philosopher %d is eating\n",id);
      sleep(3);
      put_forks(id);
    }
    pthread_exit(NULL);
}

int left(int id){
  return((id+N-1)%N);
}

int right(int id){
   return((id+1)%N);
}

void take_forks(int id){
   sem_wait(&mutex);
   state[id]= HUNGRY;
   test(id);
   sem_post(&mutex);
   sem_wait(&s[id]);
}

void put_forks(int id){
   sem_wait(&mutex);
   state[id]=THINKING;
   test(left(id));
   test(right(id));
   state[id]=THINKING;
   sem_post(&mutex);
}

void test(int id){
    if(state[id]==HUNGRY && state[left(id)]!=EATING &&
state[right(id)]!=EATING)
        {
          state[id]=EATING;
          sem_post(&s[id]);
        }
}
```