7,9

Universidade Federal da Fronteira Sul (UFFS)
Campus Chapecó - Curso de Ciência da Computação - ***Prof. Marco Aurélio Spohn***
CCR: ***Sistemas Operacionais – 2025.2 - Primeira Prova (P1) - 24/10/2025***

Nome: Erickson Giesel Miller

- MANTER DESLIGADOS E GUARDADOS EQUIPAMENTOS ELETRÔNICOS.
- A COMPREENSÃO DAS QUESTÕES FAZ PARTE DA AVALIAÇÃO!!!
- **Marque apenas uma alternativa para cada questão.**
- Cada questão de múltipla escolha vale **0,4 ponto**.
- A questão 21 vale **2 pontos**
- Não é permitido o uso de material de consulta.

1. Qual é a principal função de um sistema operacional?
( ) a) Traduzir código de alto nível para linguagem de máquina
( ) b) Controlar redes externas e dispositivos móveis
(x) c) Gerenciar recursos de hardware e fornecer serviços aos programas
( ) d) Executar aplicativos de usuário diretamente
( ) e) Realizar *backup* automático dos dados do sistema

2. O que caracteriza um processo em um sistema operacional?
( ) a) Um conjunto de instruções armazenadas na ROM
(x) b) Um programa em execução com seu próprio espaço de memória
( ) c) Um arquivo armazenado em disco
( ) d) Uma função matemática executada pelo processador
( ) e) Um *driver* de dispositivo

3. Qual é a diferença entre processo e *thread*?
( ) a) *Threads* são mais lentas que processos
( ) b) *Threads* compartilham o mesmo espaço de memória do processo
(x) c) Processos compartilham memória, *threads* não
( ) d) Processos não podem ser executados em paralelo
( ) e) *Threads* não possuem contexto de execução

4. Qual das alternativas representa uma condição necessária para ocorrência de impasse (*deadlock*)?
( ) a) Compartilhamento ilimitado de recursos
( ) b) Execução sequencial
( ) c) Preempção obrigatória
(x) d) Espera circular
( ) e) Uso exclusivo de memória virtual

5. O que é escalonamento de processos?
( ) a) Criação de novos arquivos no sistema
( ) b) Atribuição de memória virtual
( ) c) Monitoramento de dispositivos de entrada
(x) d) Distribuição de tempo de CPU entre processos
( ) e) Compressão de dados em tempo real

6. Qual algoritmo de escalonamento favorece processos com menor tempo de execução?
( ) a) Round Robin
(x) b) *Shortest Job Next*
( ) c) *First-Come, First-Served*
( ) d) *Multilevel Queue*
( ) e) Escalonamento por loteria

7. Qual técnica pode ser usada para evitar impasses?

( ) a) Remoção de todos os semáforos
( ) b) Execução simultânea de todos os processos
( ) c) Alocação ilimitada de recursos
(X) d) Negação da condição de espera circular
( ) e) Uso de memória cache compartilhada

8. O que é memória virtual?

( ) a) Memória usada exclusivamente por *threads*
( ) b) Memória que armazena dados em tempo real
( ) c) Memória que existe apenas em dispositivos móveis
(X) d) Memória que simula espaço maior do que o físico disponível
( ) e) Memória alocada por processos em modo *kernel*

9. Qual estrutura é usada para mapear endereços virtuais para físicos?

( ) a) Tabela de arquivos
( ) b) Tabela de processos
( ) c) Tabela de semáforos
(X) d) Tabela de páginas
( ) e) Tabela de usuários

10. O que é troca de contexto?

( ) a) Mudança de usuário no sistema
( ) b) Suspensão de um processo e ativação de outro
( ) c) Execução de comandos em modo *kernel*
( ) d) Alteração de permissões de arquivos
(X) e) Redirecionamento de entrada e saída

11. O que caracteriza o escalonamento por loteria em sistemas operacionais?

( ) a) Os processos são executados em ordem de chegada, sem prioridade definida
( ) b) Cada processo recebe uma quantidade fixa de tempo de CPU por rodada
( ) c) Os processos são organizados em filas múltiplas com diferentes níveis de prioridade
(X) d) A escolha do próximo processo é feita por sorteio, proporcional ao número de bilhetes atribuídos
( ) e) O processo com menor tempo de execução é sempre escolhido primeiro

12. Qual técnica de gerenciamento de memória reduz fragmentação externa?

(X) a) Segmentação
( ) b) Paginação
( ) c) Memória cache
( ) d) Alocação contígua
( ) e) Compactação de arquivos

13. O que caracteriza um impasse?

( ) a) Processos que são finalizados automaticamente
( ) b) Processos que compartilham recursos livremente
(X) c) Processos que aguardam indefinidamente por recursos
( ) d) Processos que não usam memória virtual
( ) e) Processos que executam em tempo real

14. Qual é a função do *bit* de presença na tabela de páginas?

( ) a) Indicar se o processo está ativo
( ) b) Indicar se o arquivo está aberto
(X) c) Indicar se a página está na memória física
( ) d) Indicar se o processo está em modo *kernel*
( ) e) Indicar se o processo está em espera

15. Qual é a principal vantagem do uso de *threads*?

( ) a) Maior consumo de memória
(X) b) Compartilhamento eficiente de recursos
( ) c) Execução sequencial garantida
( ) d) Isolamento completo entre tarefas
( ) e) Redução da taxa de transferência

16. Qual das alternativas representa um tipo de escalonamento de CPU?

( ) a) FIFO
( ) b) LIFO
(X) c) Round Robin
( ) d) Bubble Sort
( ) e) Merge Sort

17. O que é fragmentação interna?

(X) a) Espaço desperdiçado dentro de blocos de memória alocados
( ) b) Espaço livre entre blocos de memória
( ) c) Arquivos corrompidos no disco
( ) d) Dados duplicados na memória virtual
( ) e) Perda de dados durante troca de contexto

18. Qual das opções abaixo é uma técnica de prevenção de impasse?

( ) a) Permitir espera circular
(X) b) Alocar todos os recursos antecipadamente
( ) c) Usar escalonamento por prioridade
( ) d) Compartilhar recursos sem controle
( ) e) Executar processos em ordem aleatória

19. O que é uma *thread* do kernel?

( ) a) Uma *thread* que roda fora do sistema operacional
( ) b) Uma *thread* que executa em modo usuário
(X) c) Uma *thread* que é gerenciada diretamente pelo sistema operacional
( ) d) Uma *thread* que não possui contexto de execução
( ) e) Uma *thread* que é criada por *drivers* de dispositivos

20. Qual é a principal desvantagem da segmentação de memória?

( ) a) Aumento da fragmentação interna
(X) b) Dificuldade de acesso direto ao disco
( ) c) Necessidade de tabelas de páginas múltiplas
( ) d) Possibilidade de fragmentação externa
( ) e) Incompatibilidade com sistemas multitarefa

1,0

21. **SO Kid** implementou o problema do jantar dos filósofos em linguagem C no sistema operacional *Linux*. No entanto, ele observou que os resultados obtidos não correspondem ao esperado. **Identifique e comente, brevemente, o(s) problema(s), apresentando solução(ões) no próprio código abaixo.**

```
#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, 2);
  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;
    printf("\n creating philosopher %d \n",*j);
        pthread_create(&tids[i], NULL, philosopher,
(void *)j);
  }


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

}
```

```
void *philosopher(void *data){
    int id = *((int *) data);
    while(1){
      printf("\n Philosopher %d is thinking\n",id);
      sleep(2); // apenas para  passar tempo "pensando"
      take_forks(id);
      sleep(2); // apenas para passar tempo "comendo"
      printf("\n Philosopher %d is eating\n",id);
      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);
    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));
    sem_post(&mutex);
}

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

// mudar o estado do
// filósofo para HUNGRY,
// para que o test() possa
// funcionar

{state[id] = HUNGRY;

// não é necessário fazer o down do s

// em vez de down, é um up, o down
// foi feito no início da função.