# Curso de Tecnologia em Sistemas de Computação Disciplina: Sistemas Operacionais - AP1X - 1° semestre de 2020.

# Glauber de Souza Faria

# Angra dos Reis - RJ

# 17213050160

# Questão 1

Sabemos que:

* Processo A executa por 12ms.
* Processo A executa por: 3,7ms e depois 4,9ms.
* Processo A faz operações de E/S de 2,5ms e depois 3,2ms.

Somando os tempos que A executa teremos:

* **3,7*ms* + 4,9*ms* = 8,6*ms*.**

Porém A executa por 12ms, logo A executará mais uma vez por:

* 12*ms* – 8,6*ms* = 3,4*ms*.

Organizando, temos que:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EXEC** | **3,7*ms*** |  | **4,9*ms*** |  | **3,4*ms*** |
| **PROCESSO** | A | A | A | A | A |
| **E/S** |  | **2,5*ms*** |  | **3,2*ms*** |  |

Não podemos esquecer que o processo B também executará, entre os processos A, pois na multiprogramação, quando um o programa faz operações de entrada e saída outro está em execução.

Reorganizando teremos:

**Se B não realizar operações de saída, teremos:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **3,7*ms*** | **X1** | **4,9*ms*** | **X2** | **3,4*ms*** |
| **EXEC** | A | B | A | B | A |
| **E/S** |  | A |  | A |  |
|  |  | **2,5*ms*** |  | **3,2*ms*** |  |

Logo, para evitarmos ociosidade teremos os valores de X como:

Concluimos que o tempo minimo de execução de B no processador será dado por:

2,5*ms* + 3,2*ms* = **5,7*ms***

**Logo 5,7*ms* será o tempo minimo de execução do processo B no processador**

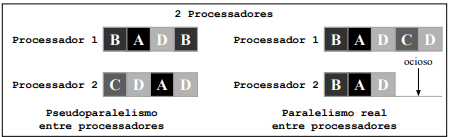
# Questão 2

1. **V**
2. **V**
3. **F**
4. **F**
5. **V**

# Questão 3

1. Árvore de Processos.
2. Background.
3. Escalonamento não preemptivo.

# Questão 4



Na afirmação acima, no lado esquerdo o nome do processo é paralelismo real e não pseudoparalelismo entre processadores, pois o paralelismo real executa os processos em mais de um processador. Já no canto direito temos dois processadores executando os mesmos processos ao mesmo tempo. Portanto a afirmação do aluno está incorreta.

# Questão 5

'''

AP1X - SISTEMAS OPERACIONAIS

ANO: 2020.1

AUTOR: GLAUBER FARIA

SINCE: 24/04/2020

LANGUAGE: PYTHON 3.X

Utilizei o parâmetro mutex para:

    1 = Recurso liberado.

    0 = Recurso em utilização.

OBS.: FOI UTILIZADO O PYTHON PARA REPRESENTAR A SOLUÇÃO DO PROBLEMA DESCRITO ABAIXO

Suponha que um conjunto possa armazenar ate n numeros. Suponha ainda que ele seja compartilhado por dois processos A e B, e que

Inicialmente possua x numeros, 0 ≤ x ≤ n. O processo A continuamente coloca dois numeros no conjunto caso ele ainda possa armazenar

Dois numeros adicionais. Ja o processo B continuamente espera que

O conjunto tenha pelo menos dois numeros, para depois remover dois

Números do conjunto e colocar o produto deles no conjunto. Como

Dois sem´aforos de contagem e um semáforo binário podem ser usados

Para garantir que os processos executem sem condi¸c˜oes de corrida ou

Impasses? Justifique a sua resposta.

'''

#Semaforo1 - Conta se o conjunto está cheio

def semaforoCont1(conjunto, n):

    comprimentoConjunto = len(conjunto)

    if comprimentoConjunto >= n-1:

        #Conjunto Cheio

        #Bloqueia Processo A

        return False

    else:

        #Conjunto Não esta cheio

        #Pode-se Inserir

        return True

#Semaforo2 - Conta se o semaforo possui no minimo dois elementos

def semaforoCont2(conjunto, n):

    comprimentoConjunto = len(conjunto)

    if comprimentoConjunto >= 2:

        #Conjunto Com no minimo dois elementos

        return True

    else:

        #Bloqueia o Processo B

        #Não se pode remover elementos de 2 em 2

        return False

#Semaforo Binario - Defini se podemos ou não executar tal operação

def semaforoBinario(mutex):

    if mutex == 1:

        return #Processo Executa

    else:

        return #Processo Dorme

#Altera o estado do mutex para liberado.

def up(mutex):

    #Bloqueia Recurso

    mutex = 0

    return mutex

#Altera o estado do mutex para bloqueado.

def down(mutex):

    mutex = 1

    return mutex

#Processo A Adiciona +2 Numeros se puder

def ProcessoA(conjunto,n):

    #Se o recurso estiver disponivel.

    if semaforoBinario(mutex):

        #Bloqueia o Recurso

        donw(mutex)

        if semaforoCont1(conjunto,n) == true:

            #Adiciona dois elementos no conjunto

            conjunto.append(x1,x2)

        else:

            #Conjunto Cheio Não é possível inserir

        #libera o recurso

        up(mutex)

    else:

        #Não é possivel utilizar o recurso

#Processo B - Remove 2 numeros e adiciona o produto deles, se  possivel

def ProcessoB(conjunto,n):

    #Se o recurso estiver disponivel.

    if semaforoBinario(mutex):

        #Bloqueia o Recurso

        donw(mutex)

        if semaforoCont2(conjunto,n) == True:

            #Pega 2 valores remova-os e armazene o produto deles

            #i é a posição do conjunto e

            #i+1 é seu sucessor que foi adotado neste exemplo

            del(conjunto[i])

            del(conjunto[i+1])

            conjunto[i] = conjunto[i]\*conjunto[i+1]

        else:

            #Não é possivel realizar operação pois o comprimento do conjunto é menor que 2.

        #libera o recurso

        up(mutex)

    else:

        #Não é possivel utilizar o recurso

# Questão 6

Round Robin descrito na Questão:

Quantum de 4ms.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 35 | 39 | 41 | 45 | 46 |
| A | C | B | A | C | B | A | C | B | A | C | A | A | - |

Baseado na tabela apresentada, podemos extrair as seguintes informações:

|  |  |  |
| --- | --- | --- |
| Processo | Tempo | Tempo Final |
| A | 21ms | 46ms |
| B | 11ms | 35ms |
| C | 14ms | 41ms |

Agora executando os processos por ordem de prioridade e respeitando as regras, temos:

**Regras:**

* Quantum 5ms.
* A cada 5ms um processo perde 3 de prioridade.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 24 | 21 | 19 | 18 | 17 | 16 | 14 | 13 | 11 | 8 | 5 |  |
| 0 | 5 | 10 | 15 | 16 | 21 | 26 | 31 | 35 | 40 | 45 | 46 |
| B | B | C | B | A | C | A | C | A | A | A | - |

* Na primeira linha temos a prioridade atual dos processo.
* Na segunda linha temos o tempo, em ms, antes do processo dado nessa coluna executar.
* Na terceira linha temos a ordem de execução dos processos.

Agora podemos extrair as informações solicitadas:

|  |  |  |  |
| --- | --- | --- | --- |
| Processo | Tempo | Tempo Final | Prioridade Final |
| A | 21ms | 46ms | 5 |
| B | 11ms | 16ms | 18 |
| C | 14ms | 35ms | 13 |

Podemos concluir que:

* Tempo final de A é o mesmo no Round Robin e no processo por prioridade.
* B tem o tempo final menor no processo por prioridade.
* C tem o tempo final menor no processo por prioridade.