**Curso de Tecnologia em Sistemas de Computação**

**Disciplina: Sistemas Operacionais**

**AD 1 - 1° semestre de 2020.**

Glauber de Souza Faria

17213050160

**1)**

Sabemos que teremos:

* 3 programas (A, B e C), 3 anagramas, logo teremos uma permutação P³ = 3! = 3\*2\*1 = 6 possibilidades.
* A só pode executar após C terminar por completo

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EXEC** | A | B | C |  |
| **E/S** |  | A | B | C |

Logo:

* A-B-C
* A-C-B

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EXEC** | A | C | B |  |
| **E/S** |  | A | C | B |

* B-C-A
* B-A-C

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EXEC** | B | C | A |  |
| **E/S** |  | B | C | A |

* C-A-B
* **C-B-A**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EXEC** | B | A | C |  |
| **E/S** |  | B | A | C |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EXEC** | C | A | B |  |
| **E/S** |  | C | A | B |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EXEC** | C | B | A |  |
| **E/S** |  | C | B | A |

**Portanto apenas um caso satisfaz as regras (C-B-A).**

Sabemos que:

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | 1/4 a ms | **E/S** | 3/4 a ms |
| **B** | 2/3 b ms | **E/S** | 1/3 b ms |
| **C** | 1/2 c ms | **E/S** | 1/2 c ms |

* A = a ms
* B = b ms
* C = c ms

Portanto teremos dois casos com os dados acima:

**CASO 1 (C-B-A):**

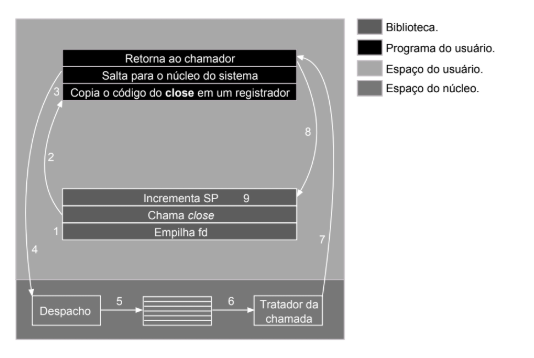
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1/2 c ms | 2/3 b ms | 1/4 a ms |  |
| **EXECUTANDO:** | C | B | A |  |
| **E/S:** |  | C | B | A |
|  |  | 1/2 c ms | 1/3 b ms | 3/4 a ms |

Logo teremos os seguintes resultado nesse caso:

Concluímos que:

* **A = *a ms***

**2)**



1. O processo do usuário que executou a função close empilha o único parâmetro fd desta função.
2. Chama a função da biblioteca close.
3. Coloca, em um lugar pré-determinado pelo sistema operacional, o código que identifica a chamada ao sistema operacional close. Depois disso.
4. Esta função executa a instrução TRAP do processador, o que mudara o processador do modo usuário para o modo supervisor, e fara com que o controle seja transferido para o endereço do núcleo responsável pelo tratamento das chamadas ao sistema operacional.
5. A parte do núcleo responsável por tratar as chamadas obtém, usando o código (passado pela biblioteca) como um índice em uma tabela com os endereços das funções que executam as chamadas, o endereço da função do núcleo que executa a chamada close.
6. O sistema operacional executa esta função, denominada de tratador da chamada close. Depois deste tratador executar as tarefas necessárias para fechar o arquivo.
7. O processador será alternado do modo supervisor para o modo usuário, e o controle ser ́a passado a instrução, da função close da biblioteca, posterior a instrução TRAP. Após fazer as finalizações necessárias ao fechamento do arquivo, a função da biblioteca então passa.
8. O controle novamente ao processo do usuário, na instrução seguinte a que chamou a função.
9. O processo do usuário incrementa o ponteiro da pilha SP, para remover o parâmetro fd colocado na pilha antes de chamarmos a função close.

**No Hardware:**

1 E/S 1,25 ms

4.500 E/S X ms

*Logo 4500\*1,25 = 5625 ms, portanto o tempo de execução será de (X-5625)ms.*

**Na Máquina Virtual:**

Sabemos que:

1. Deveremos reduzir a velocidade do processador em 55%.
2. Deveremos reduzir a velocidade das operações de E/S em 50%.
3. São executadas 1500 instruções de E/S a menos.
4. O valor de X se o tempo de execução do processo for de 20.000ms.

Logo:

1. 100% - 55% = 45% é executado.
2. 4.500E/S – 1.500 E/S = **3.000 E/S**.
3. 3000 E/S \* 2,5ms = 7500ms dos 20.000 ms.

20.000 – 7.500 = 12.500 ms.

1 E/S 2,5 ms

3.000 E/S 12.500 ms

**X = 11250 ms.**

**4)**

O diagrama de estado de processos está incorreto, as transições não estão em ordem e as instruções incorretas. O ideal será utilizar o diagrama abaixo:

**1**

**6**

**2**

**5**

**3**

**4**

BLOQUEADO

BLOQUEADO

NOVO

PRONTO

EXECUTANDO

BLOQUEADO

TERMINADO

* 1. O novo processo passa ao estado pronto.
  2. O processo é bloqueado esperando por algum evento.
  3. O escalonador suspende a execução do processo.
  4. O processo é escolhido pelo escalonador.
  5. O processo é desbloqueado.
  6. O processo termina a sua execução.

**5)**

#Q5 - AD1 - SO

#Author Glauber Faria

#Since 11/04/2020

#Language: Python 3.X

'''

-> Utilizado o parametro mutex para:

    #1 = Recurso liberado.

    #0 = Recurso em utilização.

-> Método down e up:

    #up = altera o estado do mutex para liberado.

    #down = altera o estado do mutex para bloqueado.

-> Valor aleatório pode ser definido pelo professor, só utilizamos um exemplo.

'''

#importa biblioteca que gera números randômicos.

from random import randint

#Processo A

def ProcessoA(mutex, vetor, pilha):

    #Se o recurso estiver liberado faça.

    if mutex == 1:

        #Bloqueie o recurso.

        down(mutex)

        #Separa os valores do vetor na variável i.

        for i in vetor:

            #Gerar e armazenar um inteiro aleatorio inteiro entre 0 e 100.

            valorAleatorio = (randint(0,100))

            #Inserir na pilha.

            inserePilha(i,valorAleatorio)

        #Libera o recurso.

        up(mutex)

        #Retorne que o processo foi executado com sucesso.

        return "Processo A executado"

    #Senão

    else:

        #Retorne que o processo não foi executado com sucesso.

        Return "Processo A em Sleep."

#Processo B

def ProcessoB(mutex, vetor, pilha):

    #Se a pilha tiver um tamanho maior ou igual a 1.

    if len(pilha) >= 1:

        #Se o recurso estiver liberado faça.

        if mutex == 1:

            #Bloqueie o recurso.

            down(mutex)

            #Armazenar o valor do topo da pilha em uma variável.

            topoPilha = len(pilha)-1

            #Gerar e armazenar um inteiro aleatorio inteiro entre 0 e 100

            valorAleatorio = (randint(0,100))

            #Vetor de posição aleatória recebe a soma deles mesmo com o valor no topo da pilha.

            vetor[valorAleatorio] += pilha[topoPilha]

            #Libera o recurso.

            up(mutex)

            #Retorne que o processo foi executado com sucesso.

            return "Processo B executado"

        #Senão

        else:

            #Retorne que o processo não foi executado com sucesso.

            return "Processo B em Sleep."

#Processo C

def ProcessoC(mutex, vetor, pilha):

    #Se o recurso estiver liberado faça.

    if mutex == 1:

        #Bloqueie o recurso.

        down(mutex)

        #Separa os valores do vetor na variável i.

        for i in vetor:

            #Se i for diferente de 0 então

            if i != 0:

                #imprimir o i.

                print i

        #Libera o recurso.

        up(mutex)

        #Retorne que o processo foi executado com sucesso.

        return "Processo C executado"

    #Senão

    else:

        #Retorne que o processo não foi executado com sucesso.

        return “Processo C em Sleep."