Parcialito 1 – 14/09/2015 Tema 2

**Apellido y Nombre.……………………………………………………………………………………………………………………………**

**Cada pregunta correcta suma 2 puntos. Cada pregunta incorrecta resta 1 punto. La opción NS/NC no suma ni resta puntos. La respuesta de cada pregunta debe estar indicada con letra mayúscula en el cuadrado correspondiente a la pregunta.**

1. Se dispone de una cola de requerimientos que deben ser resueltos por 4 empleados. Mientras haya requerimientos pendientes, los empleados sacan uno de la cola, lo resuelven, y al terminar incrementan una variable que lleva el total de requerimientos resueltos. Dado el siguiente código para resolver el problema indicar cual opción es la más adecuada.

|  |
| --- |
| queue R; *//suponer que ya está cargada.*  int tot = 0;  Process Empleado [i:0..3]  { bool ok;  tipo\_requerimiento req;  < ok = not empty(R); if (ok) pop (R, req) >  while (ok)  { Delay(20); *//resuelve el req.*  < tot = tot + 1;>  < ok = not empty(R);  if (ok) pop (R, req) >  }  } |

|  |
| --- |
| 1. NS/Nc 2. La solución es correcta |
| 1. La solución es incorrecta porque el incremento de la variable *tot* y la acción de sacar un requerimiento de la cola se debe hacer en una única acción atómica. 2. La solución es incorrecta porque la variable *tot* puede quedar con un valor diferente a la cantidad de requerimientos resueltos. 3. La solución es incorrecta porque no se garantiza la exclusión mutua al sacar requerimiento de la cola. |

1. Dado el siguiente enunciado y código para resolverlo, indicar cual opción es la más adecuada.

**Enunciado:** suponga que existe una BD que puede ser accedida por 6 usuarios como máximo al mismo tiempo. Además los usuarios se clasifican como usuarios de prioridad alta y usuarios de prioridad baja. Por último la BD tiene la siguiente restricción:

- no puede haber más de 4 usuarios con prioridad alta al mismo tiempo usando la BD.

- no puede haber más de 5 usuarios con prioridad baja al mismo tiempo usando la BD.

**Nota:** maximizar la concurrencia.

|  |
| --- |
| 1. NS/Nc 2. La solución cumple con el enunciado. 3. La solución es incorrecta porque el semáforo mutex está inicializado con valor un mayor a 1. 4. La solución es incorrecta porque puede haber más de 10 personas usando la BD al mismo tiempo. 5. La solución es incorrecta porque no maximiza la concurrencia. |

|  |  |
| --- | --- |
| sem mutex = 6  sem alta = 4  sem baja = 5  Process P\_Alta [I:1..L]::  { P (mutex);  P (alta)  <usa la BD>  V (mutex);  V (alta)  } | Process P\_Baja [I:1..M]::  { P (mutex);  P (baja)  <usa la BD>  V (mutex);  V (baja)  } |

1. En una clase hay N clientes que realizan compras en una casa de comidas. Al llegar dejan el pedido y luego esperan a que se lo entreguen. Hay un cocinero que toma los pedidos de a uno, lo prepara y lo embolsa para entregarlo al cliente. Dado el siguiente código marcar cual afirmación es la más adecuada. 

|  |  |
| --- | --- |
| sem listo = 0;  sem mutex = 1;  sem hayPedido = 0;  queue pedidos;  int comidas[N];  ***Process Cocinero::***  ***{*** int i;  tipo\_pedido ped;  for i := 1 to N  { P(hayPedido);  P(mutex);  pop(pedidos, ped);  V(mutex);  comidas[ped. id\_cliente] = Cocinar(ped.pedido);  V(listo) ;  }  ***}*** | ***Process Cliente(id: 1..N)::***  ***{*** tipo\_pedido ped;  int miComida;    ped.id\_cliente = id;  ped.pedido = *HacePedido;*  P(mutex)  push(pedidos, ped);  V(mutex)  V(hayPedido);  P(listo)  miComida = comidas[id];  ***}*** |

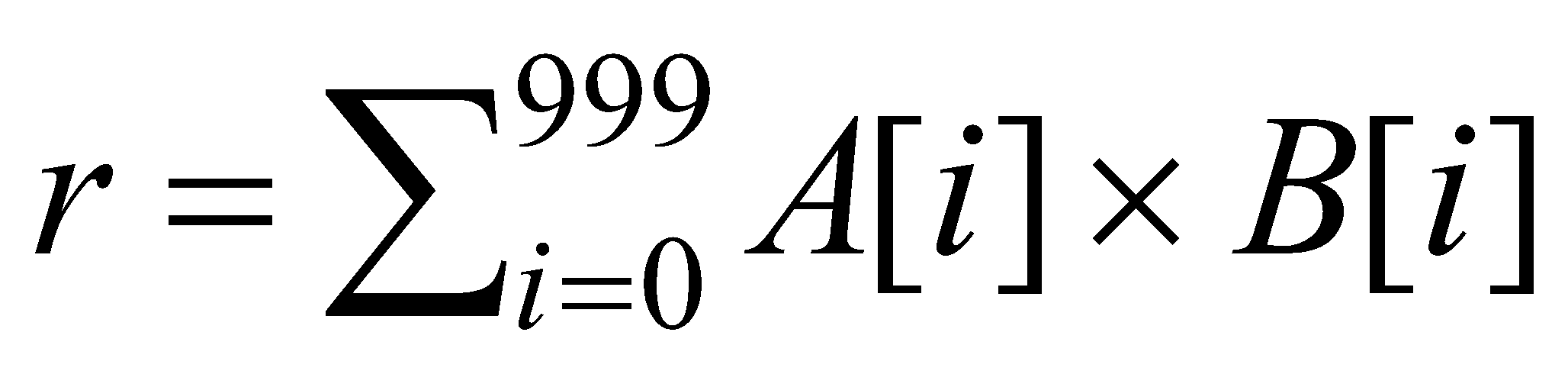
|  |
| --- |
| 1. NS/NC |
| 1. El código es correcto. |
| 1. El código es incorrecto porque un cliente puede llevarse la comida de otro**.** |
| 1. El código es incorrecto porque reduce la concurrencia al usar el semáforo *hayPedido*, ya que este no es necesario. 2. El código es incorrecto porque el cocinero hace un *pop* de la cola *pedidos* sin verificar que no esté vacía. |

1. Existe un túnel ferroviario en el cual sólo puede pasar un tren por vez, y de acuerdo al orden en que fueron llegando al mismo. Dado el siguiente código para resolver el problema indicar cual opción es la más adecuada.

|  |
| --- |
| queue espera;  bool libre = True;  sem mutex = 1;  sem paso[N] = ([N] 0);  ***Tren[id: 1..N]:: {***  int aux;  P(mutex);  if (not libre) push (espera, id)  else  { libre = false;  V(paso[id]);}  V(mutex);  P(paso[id]);  *//Pasa por el túnel*  *Delay (100);*  *//Sale del túnel*  P(mutex)  libre = True;  if (not empty (espera))  { pop(espera, aux);  V(paso[aux]);}  V (mutex);  **}** |

|  |
| --- |
| 1. NS/NC |
| 1. El código es correcto. |
| 1. El código es correcto pero sería más eficiente si se usa un único semáforo para dormir a los procesos ***Tren*** en lugar del arreglo ***paso***. 2. El código es incorrecto porque no cumple la propiedad de exclusión mutua en el uso del túnel. |
| 1. El código es incorrecto porque puede ocurrir que estando vacío el túnel no se deje pasar a ningún proceso ***Tren*** que está esperando hacerlo. |
|  |



1. Se debe realizar el producto escalar de dos vectores *A* y *B* de 1000 números (como se muestra en la fórmula ), utilizando 10 procesos. Dados los siguientes códigos marcar cual afirmación es la más adecuada.

|  |  |  |
| --- | --- | --- |
| **SOLUCIÓN 1**  int A[1000], B[1000], r =0;  Process P[id:0..9]::  { int i, aux = 0;  int inicial = id\*100;  int final = inicial+99;  for i := inicial to final  aux = aux + (A[i]\*B[i]);  < r = r + aux>  } | **SOLUCIÓN 2**  int A[1000], B[1000], r =0;  Process P[id:0..9]::  { int i;  int inicial = id\*100;  int final = inicial+99;  for i := inicial to final  < r = r + (A[i]\*B[i]) >;  } | **SOLUCIÓN 3**  int A[1000], B[1000], r =0;  Process P[id:0..9]::  { int i, aux = 0;  int inicial = id\*100;  int final = inicial+99;  for i := inicial to final  aux = aux + (A[i]\*B[i]);  r = r + aux;  } |

|  |
| --- |
| 1. NS/NC |
| 1. Las 3 soluciones resuelven lo pedido. |
| 1. La solución 1 resuelve lo pedido y es la mejor de las 3. 2. La solución 2 resuelve lo pedido y es la mejor de las 3. 3. La solución 3 resuelve lo pedido y es la mejor de las 3. 4. Ninguna solución resuelve lo pedido. |