**[Algoritmos de Dekker](https://bourneshell.wordpress.com/2012/10/26/algoritmos-de-dekker/)**

Publicado el [octubre 26, 2012](https://bourneshell.wordpress.com/2012/10/26/algoritmos-de-dekker/)por [bourneshell](https://bourneshell.wordpress.com/author/bourneshell/" \o "Lee todas las entradas de bourneshell)

El algoritmo de Dekker es un [algoritmo](http://es.wikipedia.org/wiki/Algoritmo) de [programación concurrente](http://es.wikipedia.org/wiki/Programaci%C3%B3n_concurrente) para [exclusión mutua](http://es.wikipedia.org/wiki/Exclusi%C3%B3n_mutua_(inform%C3%A1tica)), que permite a dos [procesos](http://es.wikipedia.org/wiki/Procesos) o [hilos de ejecución](http://es.wikipedia.org/wiki/Hilo_en_sistemas_operativos) compartir un recurso sin conflictos. Fue uno de los primeros algoritmos de exclusión mutua inventados, implementado por [Edsger Dijkstra](http://es.wikipedia.org/wiki/Edsger_Dijkstra" \o "Edsger Dijkstra).

Si ambos procesos intentan acceder a la sección crítica simultáneamente, el algoritmo elige un proceso según una variable de turno. Si el otro proceso está ejecutando en su sección crítica, deberá esperar su finalización.

Existen cinco versiones del algoritmo Dekker, teniendo ciertos fallos los primeros cuatro. La versión 5 es la que trabaja más eficientemente, siendo una combinación de la 1 y la 4.

Primer algoritmo

Garantiza la exclusión mutua, pero su desventaja es que acopla los procesos fuertemente, esto significa que los procesos lentos atrasan a los procesos rápidos.

|  |  |  |
| --- | --- | --- |
| PROGRAMA UNO;  Variables            turno: Entero;  Inicialización            turno = 1;     |  |  | | --- | --- | | Repeat           Hace\_Cosas();      While turno =2 Do;               REGION\_CRITICA();           turno = 2;           Hace\_mas\_cosas();      Until Fin | Repeat           Hace\_Cosas();           While turno = 1 Do;               REGION\_CRITICA();           turno = 1;           Hace\_mas\_cosas();      Until Fin | |

Segundo algoritmo

Problema interbloqueo. No existe la alternancia, aunque ambos procesos caen a un mismo estado y nunca salen de ahí.

|  |  |  |
| --- | --- | --- |
| PROGRAMA DOS; Variables           P1QE, P2QE: Bool; Inicialización           P1QE = false;           P2QE = false;   |  |  | | --- | --- | | Repeat       Hace\_Cosas();       P1QE = true;       While P2QE Do;            REGION\_CRITICA();       P1QE = False;      Hace\_mas\_cosas();  Until Fin | Repeat       Hace\_Cosas();       P2QE = true;       While P1QE Do;            REGION\_CRITICA();       P2QE = False;       Hace\_mas\_cosas();  Until Fin | |

Tercer algoritmo

Colisión región crítica no garantiza la exclusión mutua. Este algoritmo no evita que dos procesos puedan acceder al mismo tiempo a la región crítica.

|  |  |  |
| --- | --- | --- |
| PROGRAMA TRES;  Variables            P1A, P2A: Bool;  Inicialización            P1A = false;            P2A = false;   |  |  | | --- | --- | | Repeat           Hace\_Cosas();           While P2A Do;           P1A = true;                REGION\_CRITICA();                                                              P1A = False;           Hace\_mas\_cosas();      Until Fin | Repeat           Hace\_Cosas();           While P1A Do;           P2A = true;                REGION\_CRITICA                ();           P2A = False;           Hace\_mas\_cosas();      Until Fin | |

Cuarto algoritmo

Postergación indefinida. Aunque los procesos no están en interbloqueo, un proceso o varios se quedan esperando a que suceda un evento que tal vez nunca suceda.

|  |  |  |
| --- | --- | --- |
| PROGRAMA CUATRO;  Variables            P1QE, P2QE: Bool;  Inicialización            P1QE = false;            P2QE = false;   |  |  | | --- | --- | | Repeat                                                        Hace\_Cosas();           P1QE = true;           While P2QE Do       Begin                 P1QE = false;                Delay (random());                P1QE = true;       end;                REGION\_CRITICA();           P1QE = False;           Hace\_mas\_cosas();      Until Fin | Repeat      Hace\_Cosas ();                                                              P1QE = true;           While P2QE Do       Begin                 P2QE = false;                Delay (random());                P2QE = true;       end;                REGION\_CRITICA();           P2QE = False;           Hace\_mas\_cosas();      Until Fin | |

Quinto Algoritmo

El quinto algoritmo de Dekker es la versión optimizada y que no presenta problemas como las cuatro versiones anteriores, para su estructuración se hace una combinación de dos algoritmos de acuerdo al orden de prioridad de desempeño y funcionamiento de las cuatro versiones conocidas.

|  |  |  |
| --- | --- | --- |
| PROGRAMA CINCO;  Variables            P1QE, P2QE: Bool;            turno: Entero;  Inicialización            P1QE = false;            P2QE = false;            turno = 1;   |  |  | | --- | --- | | Repeat       Hace\_Cosas();                                            P1QE = true;                                                While P2QE Do       Begin              if(turno = 2)             Begin                  P1QE = false;                Delay (random());                P1QE = true;             end;       end;            REGION\_CRITICA();       turno = 2;      P1QE = False;       Hace\_mas\_cosas();  Until Fin | Repeat       Hace\_Cosas();       P2QE = true;  While P1QE Do                                                     Begin              if(turno = 1)             Begin                  P2QE = false;                Delay (random());                P2QE = true;             end;       end;            REGION\_CRITICA();       turno = 1;      P2QE = False;       Hace\_mas\_cosas();  Until Fin | |