Propiedades de la estrategia del reloj

- Caso múltiples procesos: se consideran todas las páginas por igual, sin distinguir a qué proceso pertenecen
- Tiempo que toma el cursor en dar una vuelta es el tiempo que se considera suficiente para reemplazar una página si no ha sido referenciada en toda la vuelta
- Depende de la cantidad de page faults, que depende a su vez de la localidad de los accesos
  - buena localidad → mayor tiempo entre vueltas
  - mala localidad → menor tiempo entre vueltas
- Situación anómala: todas las páginas fueron referenciadas
  - se da la vuelta completa
  - no es relevante porque se da cuando hay pocos page faults
- Ventajas:
  - simple de implementar
  - sobrecosto cero cuando la memoria es abundante

El problema de la estrategia del reloj

Thrashing

- Hay demasiados page faults
- Todos los procesos están en espera del reemplazo de una página
- Apenas se retoma un proceso, sufre un page fault y vuelve al estado de espera
- La CPU tiene un porcentaje de ocupación cercano a 0%
- El disco para paginamiento se ocupa al 100%
- La única solución es que el administrador mate algunos procesos pero el shell de comandos de *root* tampoco avanza
- Se produce cuando hay múltiples procesos, porque si bien cada proceso exhibe localidad de accesos, cuando el scheduler los ejecuta en tajadas de tiempo, pierden su localidad
- Solución: el *working set*

La estrategia del working set

- Combina paginamiento en demanda con swapping
- Básada en el cálculo del working set para cada proceso
- La principal ventaja es que evita el thrashing
- Pero agrega un sobrecosto fijo: calcular los working set
- Idealmente es mejor usar la estrategia del reloj, pero si sube demasiado la tasa de page faults, cambiar a working set
- Definición:

  El working set de un proceso P en el intervalo de tiempo virtual [t, t'[es el conjunto de páginas referenciadas por P

- El working set de P se recalcula cada Δt segundos para los últimos Δt segundos de uso de tiempo de CPU por parte de P
- Estrategia del working set:
  - Mantener working set de cada proceso en memoria
  - En caso de page fault se reemplazan una página que no pertenezca al working set de ningún proceso
  - Si no hay suficiente memoria para la suma de los working set: recurrir a swapping, llevar a disco procesos completos

Implementación del working set

Cada Δt segundos de uso de tiempo de CPU por parte de *P*, calcular su working set:

Para toda página $q \in P$ residente en memoria real {

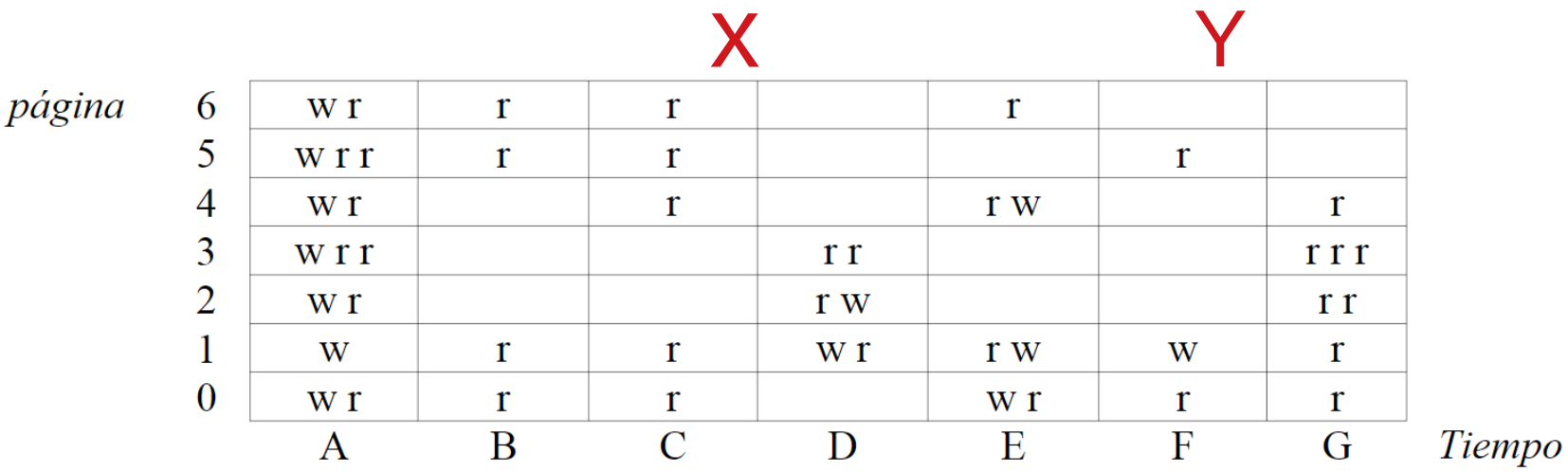
```
if ( bitR(q) ≡ 1 ) {
    setBitWS(q, 1)
    setBitR(q, 0)
}
else {
    setBitWS(q, 0)
}
```

Cuando ocurre un page fault se elige la página *q* que será reemplazada con:

```
for (;;) {
    Sea una página q cualquiera tal que bitWS(q) ≡ 0
    Si ∄ q {
        swapping( )
    }
    else {
        if ( bitR(q) ≡ 0 )
            return q
        setBitWS(q, 1)
    }
}
```

Un proceso no puede ejecutarse eficientemente si su working set no cabe en memoria, pero todavía se puede achicar Δt para disminuir el tamaño de su working set

Ejemplo de working set



- Working sets

  A : todas las páginas

  B : 0 1 5 6

  C : 0 1 4 5 6

  D : 1 2 3

  E : 0 1 4 6

  F : 0 1 5

  G : 0 1 2 3 4

Ejercicio:

- Considerar que todas las páginas de este proceso residen en memoria
- En X e Y se producen 2 page faults asociados a otros procesos, ¿Cuáles páginas de este proceso se podrían reemplazar para el page fault X? ¿Y para el page fault Y?

Optimizaciones para paginamiento en demanda

- Grabar en disco solo páginas que se hayan modificado
  - Al recuperar una página *q* de disco, colocar *bit dirty* en 0
  - La MMU coloca automáticamente bit *dirty* en 1 cuando se escribe en *q*
  - Al reemplazar *q*, grabar solo si bit dirty ≡ 1
- Si el disco de paginamiento está desocupado, elegir cualquier página *q* con bitWS(q) ≡ 0 y bitR(q) ≡ 0:
  - grabar *q* en disco
  - colocar bit dirty en 0
- Tratar de que el 20 % de las páginas reales disponibles para procesos no pertenezca al working set de ningún proceso
  - Si es menor que 20 % recurrir a swapping
  - Si es mayor que 20 % recuperar procesos de disco
- Tratar de que la tasa de page faults de cada proceso sea por ejemplo 10 page faults por segundo:
  - Un disco moderno permite atender unos 100 page faults por segundo, cuando no hay que escribir la página de reemplazo
  - 10 page faults significarían aproximadamente un 10 % de sobrecosto en tiempo de ejecución
  - Si es mayor que 10 page faults : agrandar Δt para ese proceso
  - Si es menor que 10 page fauls: disminuir Δt para ese proceso