Instituto Politécnico Nacional

Escuela Superior de Cómputo

Tarea 2

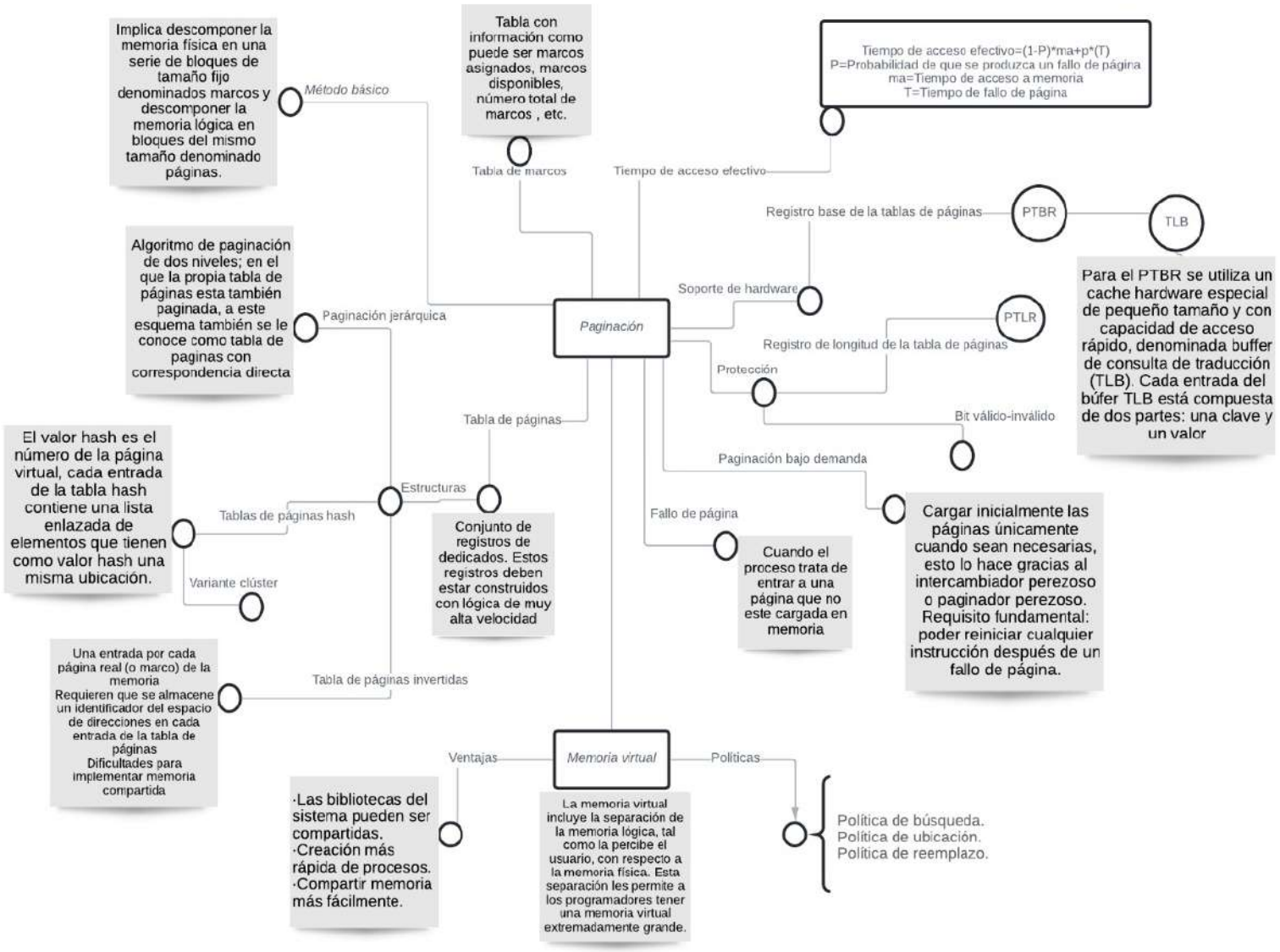
Unidad de aprendizaje: Sistemas operativos

Grupo: 2CM8

Alumnos:

- Martínez Coronel Brayan Yosafat
- Monteros Cervantes Miguel Angel
- Ramírez Olvera Guillermo
- Sánchez Méndez Edmundo Josué

Profesor:
Cortés Galicia Jorge


Implica descomponer la memoria física en una serie de bloques de tamaño fijo denominados marcos y descomponer la memoria lógica en bloques del mismo tamaño denominado páginas.
Método básico
Tabla con información como puede ser marcos asignados, marcos disponibles, número total de marcos , etc.
Tabla de marcos
Tiempo de acceso efectivo=(1-P)*ma+p*(T)
P=Probabilidad de que se produzca un fallo de página
ma=Tiempo de acceso a memoria
T=Tiempo de fallo de página
Tiempo de acceso efectivo
Registro base de la tablas de páginas
PTBR
TLB
Para el PTBR se utiliza un cache hardware especial de pequeño tamaño y con capacidad de acceso rápido, denominada buffer de consulta de traducción (TLB). Cada entrada del búfer TLB está compuesta de dos partes: una clave y un valor
Algoritmo de paginación de dos niveles; en el que la propia tabla de páginas esta también paginada, a este esquema también se le conoce como tabla de paginas con correspondencia directa
Paginación jerárquica
Paginación
Soporte de hardware
PTLR
Registro de longitud de la tabla de páginas
Protección
Bit válido-inválido
Tabla de páginas
Paginación bajo demanda
Cargar inicialmente las páginas únicamente cuando sean necesarias, esto lo hace gracias al intercambiador perezoso o paginador perezoso. Requisito fundamental: poder reiniciar cualquier instrucción después de un fallo de página.
El valor hash es el número de la página virtual, cada entrada de la tabla hash contiene una lista enlazada de elementos que tienen como valor hash una misma ubicación.
Estructuras
Tablas de páginas hash
Variante clúster
Conjunto de registros de dedicados. Estos registros deben estar construidos con lógica de muy alta velocidad
Fallo de página
Cuando el proceso trata de entrar a una página que no este cargada en memoria
Una entrada por cada página real (o marco) de la memoria
Requieren que se almacene un identificador del espacio de direcciones en cada entrada de la tabla de páginas
Dificultades para implementar memoria compartida
Tabla de páginas invertidas
Memoria virtual
Ventajas
Políticas
·Las bibliotecas del sistema pueden ser compartidas.
·Creación más rápida de procesos.
·Compartir memoria más fácilmente.
La memoria virtual incluye la separación de la memoria lógica, tal como la percibe el usuario, con respecto a la memoria física. Esta separación les permite a los programadores tener una memoria virtual extremadamente grande.
Política de búsqueda.
Política de ubicación.
Política de reemplazo.

Algorithm 1: Sustitución de páginas FIFO

Entrada: Número de Marcos numM, Número de paginas numP, Arreglo de paginas P
Salida: Número de fallos numF
Función: BuscaFalloPagina($numM, numP, P$)

```
   /* Supongamos creada una estructura de datos tipo cola                    */
1  Marco = nuevaCola();
2  numF = 0;
3  for k = 0 hasta numP do
4      if Marco.length <= numM then
5          if P[k] no esta en Marco then
6              Marco.formar(Marco,P[k]);
7              numF = numF + 1;
8      else
9          if P[k] no esta en Marco then
10             Marco.desformar(Marco);
11             Marco.formar(Marco,P[k]);
12             numF = numF + 1;
13 return numF;
```

Algorithm 2: Sustitución óptima de páginas

Entrada: Número de Marcos numM, Número de paginas numP, Arreglo de paginas P
Salida: Número de fallos numF

```
/* Se usaran dos funciones, la que nos dara el numero de fallos y la que buscara la pagina con el mayor
   periodo de tiempo en no ser utilizada                                                              */
```

Función: Prediccion$P, Marco, numP, inicioPrediccion$)

```
1  prediccion = −1;
2  masLejano = inicioPrediccion;
3  for i = 0 hasta Marco.length do
4      for j = inicioPrediccion hasta numP do
5          if Marco[i] == P[j] then
6              if j > masLejano then
7                  masLejano = j;
8                  prediccion = i;
9              break;
10     if j == numP then
11         return i;
12 return (prediccion == -1) ? 0 : prediccion;
```

Función: BuscaFalloPagina($numM, numP, P$)

```
13 Marco = nuevoVector();
14 numF = 0;
15 for k = 0 hasta numP do
16     if P[k] esta en Marco then
17         continuar;
18     numF=numF+1;                                   // Si no esta, entonces hay fallo
19     if Marco.length < numM then
20         S.aniadirFinal(P[k]);                      // Añade al final del vector el valor
21     else
22         indicePrediccion = Prediccion(P, Marco, numP, i + 1);
23         Marco[indicePrediccion]=P[k];
24 return numF;
```

Algorithm 3: Sustitución de páginas LRU

Entrada: Número de Marcos numM, Arreglo de paginas P
Salida: Número de fallos numF
/* Recordar que podemos usar tanto contadores como pilas, asi que el desarrollo de este algoritmo es la idea general */
Función: BuscaFalloPagina($numM, P$)
/* Supongamos una estructura de datos tipo lista creada */

```
1  Marco = vacia();
2  numF = 0;
3  auxiliar = 0;
4  foreach i en P do
5      if !Marco.Busca(i) then
6          if Marco.length == numM then
7              Marco.Eliminar(0);
8              Marco.Añadir(numM-1,i);          // Indice, Número a introducir
9          else
10             Marco.Añadir(auxiliar,i);
11         numF=numF+1;
12         auxiliar=auxiliar+1;
13     else
14         Marco.Eliminar(i);                   // Indice, Número a introducir
15         Marco.add(Marco.length,i);           // Indice, Número a introducir
16 return numF;
```

Algorithm 4: Sustitución de páginas mediante aproximación LRU "Bits de referencia adicionales"

Entrada: Número de Marcos numM, Número de paginas numP, Arreglo de paginas P
Salida: Número de fallos numF
Función: MoverBits($BitsReferencia, posMarco, numM$)

```
1  copiaArreglo[numM][8];
2  for i = 0 hasta numM do
3  |  for j = 0 hasta 8 do
4  |  |  copiaArreglo[i][j] = BitsReferencia[i][j];
5  for i = 0 hasta numM do
6  |  for j = 1 hasta 8 do
7  |  |  BitsReferencia[i][j] = copiaArreglo[i][j-1];
8  for i = 0 hasta numM do
9  |  BitsReferencia[i][0] = 0;
10 BitsReferencia[posM][0]=1;
11 return BitsReferencia;
```

Función: menosReciente($BitsReferencia, numM$)

```
12 indiceMenosReciente = 0;
13 arregloValores[numM];
14 for i = 0 hasta numM do
15 |  for j = 0 hasta 8 do
16 |  |  arregloValores[i] = arregloValores[i] + BitsReferencia[i][7-j]*pow(2,j);
17 auxiliar = 0;
18 for i = 0 hasta numM do
19 |  if arregloValores[i] < auxiliar then
20 |  |  auxiliar = arregloValores[i];
21 |  |  indiceMenosReciente = i;
22 return indiceMenosReciente;
```

Función: BuscaFalloPagina($numM, numP, P$)

```
/* Supongamos una estructura de datos tipo lista creada                    */
23 Marco = vacia();
24 BitsReferencia[numM][8] con valores 0;
25 numF = 0;
26 Auxiliar = 0 for k = 0 hasta numP do
27 |  if Marco.length() <= numM then
28 |  |  if P[k] no esta en Marco then
29 |  |  |  Marco.Añadir(Auxiliar,P[k]);
30 |  |  |  BitsReferencia = MoverBits(BitsReferencia, Auxiliar, numM);
31 |  |  |  numF = numF + 1
32 |  |  |  Auxiliar = Auxiliar + 1;
33 |  else
34 |  |  if P[k] no esta en Marco then
35 |  |  |  Auxiliar = menosReciente(BitsReferencia,numM)
36 |  |  |  Marco.Añadir(Auxiliar,P[k]);
37 |  |  |  BitsReferencia = MoverBits(BitsReferencia, Auxiliar, numM);
38 |  |  |  numF = numF + 1
39 return numF;
```

Algorithm 5: Sustitución de páginas segunda oportunidad (forma básica, no es reloj)

Entrada: Número de Marcos numM, Número de paginas numP, Arreglo de paginas P
Salida: Número de fallos numF
Función: BuscaFalloPagina($numM, numP, P$)

```
   /* Supongamos creada una estructura de datos tipo cola                    */
 1 Marco = nuevaCola();
 2 numF = 0;
 3 bitReferencia[numM] con valores false;
 4 auxiliar = 0; // Auxiliar para encontrar el primer elemento que no tiene el bit de referencia en verdadero
 5 for k = 0 hasta numP do
 6     if P[k] no esta en Marco then
 7         if Marco.length <= numM then
 8             Marco.formar(Marco,P[k]);
 9         else
10             auxiliar = 0;
11             while !esnueva(Marco) do
12                 if bitReferencia[auxiliar % numM] then
13                     bitReferencia[auxiliar % numM] = !bitReferencia[auxiliar % numM];
14                 else
15                     break;
16                 auxiliar = 0;
17             if esnueva(Marco) then
18                 Marco.desformar(Marco);
19                 Marco.formar(Marco,P[k]);
20             else
21                 j = 0;
22                 while j < (auxiliar % numM ) do
23                     t1 = Marco.primero();
24                     Marco.desformar(Marco);
25                     Marco.formar(Marco,P[t1]);
26                     temp = bitReferencia[0];
27                     for contador = 0 hasta numM -1 do
28                         bitReferencia[contador] = bitReferencia[contador + 1];
29                     bitReferencia[numM - 1] = temp;
30                     j = j + 1;
31                 Marco.desformar(Marco);
32                 Marco.formar(Marco,P[k]);
33             numF = numF + 1;
34     else
35         tempMarco = Marco;
36         contador = 0;
37         while !esnueva(Marco) do
38             if Marco.primero() == P[k] then
39                 bitReferencia[contador] = true;
40                 contador = contador + 1;
41                 Marco.desformar(Marco);
42         Marco = tempMarco;
43 return numF;
```

Algorithm 6: Sustitución de páginas segunda oportunidad implementación reloj

Entrada: Número de Marcos numM, Número de paginas numP, Arreglo de paginas P
Salida: Número de fallos numF
Función: ActualizarBitSegundaOportunida($numM, pagina, arregloSegunda, Marco$)

```
1  for i = 0 hasta numM do
2      if Marco[i] == pagina then
3          arregloSegunda[i] = verdadero;
4          return verdadero;
5  return falso;
```

Función: BuscaActualizaBitSegundaOportunida($numM, pagina, arregloSegunda, Marco, puntero$)

```
6  while true do
7      if !arregloSegunda[puntero] then
8          Marco[puntero] = pagina;
9          return (puntero + 1) % numM;
10     arregloSegunda[puntero] = falso;
11     puntero = (puntero + 1) % numM;
12 return falso;
```

Función: BuscaFalloPagina($numM, numP, P$)

```
13 Marco = nuevaCola();
14 numF = 0;
15 arregloSegunda[numM];
16 puntero = 0;  // Auxiliar para encontrar el primer elemento que no tiene el bit de referencia en verdadero
17 for k = 0 hasta numP do
18     if !ActualizarBitSegundaOportunida(numM, P[i], arregloSegunda, Marco) then
19         puntero = BuscaActualizaBitSegundaOportunida(numM, P[i], arregloSegunda, Marco, puntero);
20     numF = numF + 1;
21 return numF;
```

Algorithm 7: Sustitución de páginas segunda oportunidad mejorado o NRU (No Usada Recientemente)

Entrada: Número de Marcos numM, Número de paginas numP, Arreglo de paginas P
Salida: Número de fallos numF

```
/* Recordemos que necesitamos tener un periodo de tiempo para poder poner RM[numero marco][0] = 0 para
   garantizar la existencia de algún reemplazo futuro, pero esto se hace con base en el SO que se utilice
   */
```

Función: BuscaVictima($numM, RM$)

```
1  auxSegundo = 0;
2  for i = 0 hasta numM do
3      if RM[i][0] = 0 y RM[i][1] = 0 then
4          return i
5      else if RM[i][0] = 1 y RM[i][1] = 0 then
6          auxSegundo = i;
7  return auxSegundo;
```

Función: BuscaFalloPagina($numM, numP, P$)

```
8  Marco[numM];
9  RM[numM][2]conen0;
10 numF = 0;
11 auxMarco = 0;
12 for k = 0 hasta numP do
13     if Marco.length <= numM then
14         if P[k] no esta en Marco then
15             Marco[auxMarco] = P[k];
16             numF = numF + 1;
17             RM[auxMarco][0] = 1;
18             RM[auxMarco][1] = 0;
19             auxMarco = auxMarco + 1;
20         else
21             RM[Marco.buscar(P[k])][0] = RM[Marco.buscar(P[k])][0];
22             RM[Marco.buscar(P[k])][1] = 1;
23             auxMarco = auxMarco + 1
24     else
25         if P[k] no esta en Marco then
26             Marco[BuscaVictima(numM, RM)] = P[k];
27             numF = numF + 1;
28         else
29             RM[Marco.buscar(P[k])][0] = RM[Marco.buscar(P[k])][0];
30             RM[Marco.buscar(P[k])][1] = 1;
31             auxMarco = auxMarco + 1
32 return numF;
```

Algorithm 8: Sustitución de páginas NFU (No Frecuentemente Usado)

Entrada: Número de Marcos numM, Número de paginas numP, Arreglo de paginas P
Salida: Número de fallos numF
Función: BuscaFalloPagina($numM, numP, P$)

```
1  Marco tipo arreglo;
2  BitAuxiliar tipo diccionario;
3  numF = 0;
4  contador = 0;
5  valor = 0;
6  auxmin = 0;
7  victima = 0;
8  while (contado < numP) y (P[contador]!=-1) do
9      if Marco.contiene(P[contador]) then
10         valor = BitAuxilar.obtener(P[contador]);
11         BitAuxilar.añadir(P[contador], valor + 1);
12         contador = contador + 1;
13     else
14         auxmin = 9999;        // Este valor puede ser modificado al gusto solo tiene que cumplir en ser un
            número grande
15         if marco.tamaño() == numM then
16             foreach pagina en Marco do
17                 fr = pagina.siguiente();
18                 freq = BitAuxilar.obtener(fr);
19                 if freq < min then
20                     min = freq;
21                     victima = fr;
22                 indice = marco.indicede(victima);
23                 marco.eliminar(indice);
24                 marco.añadir(P[contador]);
25                 if !BitAuxiliar.contienellave(P[contador]) then
26                     valor = BitAuxiliar.obtener(P[contador]);
27                     BitAuxiliar.añadir(P[contador] , valor + 1);
28                 else
29                     BitAuxiliar.añadir(P[contador] , 1);
30                 numF = numF + 1;
31                 contador = contador + 1;
32         else
33             Marco.añadir(P[contador]);
34             BitAuxiliar.añadir(P[contador], 1);
35             numF = numF + 1;
36             contador = contador + 1;
37 return numF;
```

Algorithm 9: Sustitución de páginas LFU (*Least Frequently Used*)

Entrada: Número de Marcos numM, Número de paginas numP, Arreglo de paginas P
Salida: Número de fallos numF
/* Recordemos que necesitamos tener un periodo de tiempo */
Función: $romperEmpate(Marco, P, c, numM)$
1 $contador$[1000] con valores en 0;
2 max
3 $valormax$
4 **for** i *hasta* $numM$ **do**
5 **for** $j = c; j >= 0$ **do**
6 contador[i]++;
7 **if** *Marco[i] == P[j]* **then**
8 break;
9 **for** i *hasta* $numM$ **do**
10 **if** $i == 0$ **then**
11 max = contador[0];
12 valormax = Marco[0];
13 **else**
14 max = contador[i]; valormax = Marco[i];
15 **return** *valormax;*
Función: BuscaFalloPagina($numM, numP, P$)
16 $Marco[numM]$ con valores -1;
17 $numF = 0$;
18 $arregloFrecuencia$[100] con valores 0;
19 $frecuencia$;
20 $bandera$;

```
21 for i hasta numP do
22     temporalC = 0;
23     for j hasta numM do
24         if Marco[j]==-1 then
25             Marco[j] = P[i];
26             frecuencia = P[i];
27             arregloFrecuencia[frecuencia]++;
28             break;
29         else
30             if Marco[j] == P[i] then
31                 frecuencia = P[i];
32                 arregloFrecuencia[frecuencia]++;
33                 break;
34             else
35                 temporalC = temporalC + 1;
36     if temporalC == numM then
37         numF= numF + 1;
38         min
39         frecuenciamin
40         empate = 0
41         for k hasta numM-1 do
42             frecuencia = Marco[k];
43             frecuencia1 = Marco[k+1];
44             bandera = 1;
45             if arregloFrecuencia[frecuencia] < arregloFrecuencia[frecuencia1] then
46                 min = arregloFrecuencia[frecuencia];
47                 frecuenciamin = frecuencia;
48             else
49                 min = arregloFrecuencia[frecuencia1];
50                 frecuenciamin = frecuencia1;
51         for k hasta numM do
52             frecuencia = Marco[k];
53             if arregloFrecuencia[frecuencia] == min] then
54                 bandera = 1;
55                 empate++;
56                 if empate > 1 then
57                     break;
58         if empate > 1 then
59             val = romperEmpate(Marco, P, i, numM);
60             empate = 0;
61             for k hasta numM do
62                 if val == Marco[k] then
63                     Marco[k] = P[i];
64         else if bandera == 1 then
65             for k hasta numM do
66                 if frecuenciamin == Marco[k] then
67                     Marco[k] = P[i];
68         empate = 0;
69 return numF;
```

Algorithm 10: Sustitución de páginas MFU (*Most Frequently Used*)

Entrada: Número de Marcos numM, Número de paginas numP, Arreglo de paginas P
Salida: Número de fallos numF

```
1  lleno = 0;
2  frecuencia[numM] con valores 0;
3  remIndice;
   Función: buscaRemplazo
4  max = 0
5  for i hasta numM do
6      if frecuencia[max] < frecuencia[i] then
7          max = i;
8  remIndice = max;
9  return remIndice;
   Función: remplazarPagina(pagina)
10 temporal;
11 remIndice = buscaRemplazo() ;
12 temporal = Marco[remIndice];
13 Marco[remIndice] = pagina; frecuencia[remIndice] = 1;
   Función: falloPagina(pagina)
14 bandera;
15 if lleno != numM then
16     frecuencia[lleno]++; Marco[lleno++] = pagina;
17 else
18     remplazarPagina(pagina);
   Función: Busca(pagina)
19 bandera;
20 if lleno != 0 then
21     for i hasta lleno do
22         if pagina == Marco[i] then
23             bandera = 1;
24             frecuencia[i]++;
25             break;
26 return bandera;
   Función: BuscaFalloPagina(numP, P)
27 numF = 0;
28 auxiliar;
29 for k = 0 hasta numP do
30     auxiliar = k;
31     if Busca(P[k]) !=1 then
32         falloPagina(P[k]);
33         numF++;
34 return numF;
```

Algorithm 11: Sustitución de páginas PFF (*Page Fault Frequency*)

Entrada: Número de Marcos numM, Número de paginas numP, Arreglo de paginas P
Salida: Número de fallos numF
Función: BuscaFalloPagina($numM, numP, P$)

```
   /* Supongamos creada una estructura de datos tipo cola                    */
 1 diccionarioAuxiliar tipo diccionario numF = 0;
 2 tiempo = 0;
 3 for i = 0 hasta numP do
 4     pagina = P[i];
 5     booleano = diccionarioAuxiliar.contienellave(pagina);
 6     if booleano == falso then
 7         numF++;
 8         if tiempo < numM then
 9             diccionarioAuxiliar.añadir(x, 1);
10         else
11             diccionarioAuxiliar.valor().borrartodos(0);
12             foreach llave en diccionarioAuxiliar.llaves() do
13                 diccionarioAuxiliar.añadir(llave, 0);
14         tiempo = 0;
15     else
16         tiempo++;
17 return numF;
```

Algorithm 12: Sustitución de páginas Working Set (Medio multiprogramado)

Entrada: Número de Marcos numM, Número de paginas numP, Arreglo de paginas P
Salida: Número de fallos numF
Función: eliminarPasado ($auxMarco, P, numM, ultimo$

```
1  MarcoActualizado[numM] con valores -1;
2  for i = k hasta k - numM do
3      for j = 0 hasta numM do
4          if auxMarco[j] == P[k] then
5              MarcoActualizado[auxMarco.indice(P[k])] = auxMarco.valor(P[k]);
6  for i = 0 hasta numM do
7      if MarcoActualizado[i] == -1 then
8          MarcoActualizado[i] = auxMarco[i];
9  return MarcoAcualizado;
   Función: BuscaFalloPagina(numM, numP, P)
10 Marco[numM] es un arreglo;
11 Marco[numM + 1] es un arreglo;
12 numF = 0;
13 auxM = 0;
14 for k = 0 hasta numP do
15     if k == 0 then
16         Marco[auxM] = P[k];
17         auxM ++;
18     else
19         if auxM ¡ numM then
20             if P[k] no esta en Marco then
21                 Marco[auxM] = P[k];
22                 auxM++
23                 numF ++;
24         else
25             if P[k] no esta en Marco then
26                 auxMarco = Marco;
27                 auxMarco[numM + 1] = P[k];
28                 Marco = eliminarPasado(auxMarco, P, numM, k);
29                 numF ++;
30 return numF;
```