FUNDAMENTOS BÁSICOS DE LOS SISTEMAS OPERATIVOS

Fe de Erratas

Actualizada a 10 de octubre de 2024

Nota: Las erratas en color azul son erratas que han sido añadidas o modificadas con respecto a la actualización anterior realizada el 30 de septiembre de 2022.

Capítulo 0

- **Página 7. Tabla 0.1.**
  - *Errata*: "100 bytes"
  - *Corrección*: "100 bytes/s"
- **Página 14. Sección 0.5.2. Párrafo 4. Línea 2.**
  - *Errata*: "puede"
  - *Corrección*: "pueden"

Capítulo 1

- **Página 30. Párrafo 6. Línea 1.**
  - *Errata*: "Los sistema operativo..."
  - *Corrección*: "Los sistemas operativos..."
- **Página 40. Párrafo 2.**
  - *Errata*: "Los sistemas multiprocesador más comunes utilizan el *multiprocesamiento simétrico*, que consiste en que cada procesador utiliza una copia idéntica del sistema operativo. Dichas copias se comunican entre sí cuando resulta necesario. "
  - *Corrección*: "Los sistemas multiprocesador más comunes utilizan el *multiprocesamiento simétrico*, que consiste en la existencia de una única copia del sistema operativo cargada en la memoria principal, el código del sistema operativo puede ser ejecutado en cualquier procesador. "

Capítulo 2

- **Página 54. Párrafo 3. Línea 1.**
  - *Errata*: "Además el sistemas operativo..."
  - *Corrección*: "Además el sistema operativo"
- **Página 60. Sección 2.3.1. Párrafo 1. Línea 3.**
  - *Errata*: "operativo de un sistema operativo almacena..."
  - *Corrección*: "operativo almacena..."
- **Página 62. Párrafo 7. Línea 4.**
  - *Errata*: "...para que así puede ser planificado."
  - *Corrección*: "...para que así pueda ser planificado."
- **Página 66. Párrafo 2. Línea 6.**
  - *Errata*: "...repercute en el progreso de dicho progreso."
  - *Corrección*: "...repercute en el progreso de dicho proceso."
- **Página 67. Sección 2.5.1. Párrafo 6. Línea 2.**
  - *Errata*: "multhilo"
  - *Corrección*: "multihilo"

Capítulo 3

- **Página 81. Sección 3.1. Párrafo 1. Línea 5.**
  - *Errata*: "...y realiza está tarea..."
  - *Corrección*: "...y realiza esta tarea..."
- **Página 93. Párrafo 4. Línea 4.**
  - *Errata*: "...entonces $M_n + 1 = M_n$..."
  - *Corrección*: "...entonces $M_{n+1} = M_n$..."
- **Página 93. Ecuación (3.4).**
  - *Errata*: $M_{n+1} = \alpha \cdot t_n + (1 - \alpha) \cdot t_{n-1} + \ldots + (1 - \alpha)^i \cdot \alpha \cdot t_{n-i} + \ldots + M_1$
  - *Corrección*: $M_{n+1} = \alpha \cdot t_n + (1 - \alpha) \cdot \alpha \cdot t_{n-1} + \ldots + (1 - \alpha)^i \cdot \alpha \cdot t_{n-i} + \ldots + (1 - \alpha)^n \cdot M_1$
- **Página 96. Figura 3.6.**
  - *Errata*: El proceso que termina en $t = 13$ ut es el proceso C no el proceso D.
- **Página 96. Párrafo 4. Línea 1.**
  - *Errata*: "En $t = 4$ ut llega el proceso D..."
  - *Corrección*: "En $t = 3$ ut llega el proceso D..."
- **Página 98. Figura 3.8.**
  - *Errata*: La flecha de terminación del proceso B situada en $t = 8$ ut debe estar en $t = 7$ ut.
- **Página 103. Párrafo 4**
  - *Errata*: "Cuando un proceso llega a la cola de menor prioridad, cada vez que consuma un cuanto volverá a ella hasta que consiga completarse. En consecuencia, el algoritmo de planificación de la cola de menor prioridad es de tipo turno rotatorio. El resto de colas se pueden planificar internamente con un algoritmo FCFS."
  - *Corrección*: "Cuando un proceso llega a la cola de menor prioridad, cada vez que consuma un cuanto volverá a ella hasta que consiga completarse."
- **Página 103. Párrafo 5. Línea 5**
  - *Errata*: "inanición de proceso."
  - *Corrección*: "inanición de procesos."

- **Página 104. Párrafo 4. Línea 6.**
  - *Errata*: "resta 1 ut"
  - *Corrección*: "restan 2 ut"
- **Página 106. Figura 3.13.**
  - *Errata*: Sobra la flecha del proceso C llegando en $t = 5$ ut

Capítulo 4

- **Página 117. Viñeta 4.**
  - *Errata*: "Conocer el planteamiento y al solución de varios..."
  - *Corrección*: "Conocer el planteamiento y la solución de varios..."
- **Página 118. Párrafo 4. Línea 5.**
  - *Errata*: "...estar ejecutando una sección crítica del programa."
  - *Corrección*: "...estar ejecutando una sección crítica asociada a un mismo recurso compartido."
- **Página 124. Sección 4.2.5. Párrafo 1. Línea 1.**
  - *Errata*: "En esta sección se van describir..."
  - *Corrección*: "En esta sección se van a describir..."
- **Página 127. Figura 4.5. Línea 11.**
  - *Errata*: `while(turno == otro_pid && petición_rec[otro_pid] == TRUE);`
  - *Corrección*: `while(petición_rec[otro_pid] == TRUE && turno == otro_pid);`
- **Página 128. Párrafo 2 y 3.**
  - *Errata*:
    1. `turno==otro_pid`. La variable `turno` coincida con el identificador del otro proceso que se está ejecutando concurrentemente. En este caso el proceso es B luego `turno` vale 1.
    2. `petición_rec[otro_pid]==TRUE`. El otro proceso (en este caso B) haya solicitado previamente usar el recurso compartido.
  - *Corrección*:
    1. `petición_rec[otro_pid]==TRUE`. El otro proceso (en este caso B) haya solicitado previamente usar el recurso compartido.
    2. `turno==otro_pid`. La variable `turno` coincida con el identificador del otro proceso que se está ejecutando concurrentemente. En este caso el proceso es B luego `turno` vale 1.

- **Página 128. Párrafo 8.**
  - *Errata*: "El proceso A y el proceso B invocan prácticamente a la vez la función `acceso_sc`. El proceso que conseguirá acceder a la sección crítica será aquel que primero consiga completar la instrucción `petición_rec[pid]=TRUE`. Nótese que el valor que se quedará en la variable `turno` será el del último proceso que ejecute la instrucción `turno=otro_proceso`."
  - *Corrección*: "El proceso A y el proceso B invocan prácticamente a la vez la función `acceso_sc` y ambos ya han ejecutado la instrucción `petición_rec[pid]=TRUE` de dicha función. El proceso que conseguirá acceder a la sección crítica será el primero que ejecute la instrucción `turno=otro_pid`, ya que aunque inicialmente dicho proceso puede que tenga que esperar en el bucle de espera de la función `acceso_sc`, logrará entrar en cuanto el otro proceso ejecute la instrucción `turno=otro_pid`. Nótese que el valor que se quedará en la variable `turno` será el del último proceso que ejecute la instrucción `turno=otro_pid`."
- **Página 128. Párrafo 9.**
  - *Errata*: "El algoritmo de Peterson garantiza la exclusión mutua ya que el proceso que primero ejecuta la instrucción `peticion_rec[pid]=TRUE` se garantiza el uso del recurso. El otro proceso tendrá que esperar."
  - *Corrección*: "En los escenarios anteriores se observa que el algoritmo de Peterson garantiza la exclusión mutua, ya que si un proceso accede a una sección crítica asociada a un recurso compartido el otro proceso tendrá que esperar para usar dicho recurso hasta que el primero termine de utilizarlo y salga de la sección crítica. "
- **Página 129. Ejemplo 4.4. Párrafo 5. Línea 2.**
  - *Errata*: "ya que ejecuta un bucle hasta que `cerrojo=1`"
  - *Corrección*: "ya que ejecuta un bucle mientras que `cerrojo=1`"
- **Página 130. Párrafo 5. Línea 4.**
  - *Errata*: ", con lo éste es interrumpido..."
  - *Corrección*: ", con lo que éste es interrumpido"
- **Página 132. Párrafo 2. Línea 4.**
  - *Errata*: "...`S=signal=signal_sem`."
  - *Corrección*: "...`V=signal=signal_sem`."

- **Página 134. Párrafo 2**
  - *Errata*: " El núcleo implementa las operaciones `wait_sem` y `signal_sem` como primitivas o funciones elementales de su código que se ejecutan de forma atómica, es decir, se ejecutan en un único ciclo de instrucción que no puede ser interrumpido, por lo que su ejecución siempre se completa. "
  - *Corrección*: " El núcleo implementa las operaciones `init_sem`, `wait_sem` y `signal_sem` como primitivas o funciones elementales de su código. "
- **Página 136. Figura 4.8.**
  - *Errata*: `"int ocupado;"`
  - *Corrección*: `"int ocupado=0;"`
- **Página 136. Figura 4.8.**
  - *Errata*: En el código de la operación `wait_sem` falta añadir al final la línea

```
else S.ocupado=0;
```

- **Página 138. Párrafo 1. Línea 4.**
  - *Errata*: "A partir de momento cualquier..."
  - *Corrección*: "A partir de ese momento cualquier..."
- **Página 138. Párrafo 8. Línea 2.**
  - *Errata*: "ceso B ejecutará..."
  - *Corrección*: "ceso B ejecutara..."
- **Página 142. Comentario de leer_buffer**
  - *Errata*: "Escribir dato en el buffer"
  - *Corrección*: "Leer dato en el buffer"
- **Página 145. Párrafo 9. Línea 1.**
  - *Errata*: "Cuando escritor..."
  - *Corrección*: "Cuando un escritor..."

Página 149. Párrafo 5. Línea 3.

  - *Errata*: "....,entonces el productor..."
  - *Corrección*: "....,entonces el consumidor..."

Página 149. Párrafo 6. Línea 7.

- *Errata*: "consumidor que estuviera..."
- *Corrección*: "productor que estuviera..."

Página 150. Figura 4.15. Línea 14.

- *Errata*: "`void leer(int dato);`"
- *Corrección*: "`void leer(int dato)`"

- **Página 150. Figura 4.15. Línea 19.**
  - *Errata*: "`signal(lleno);`"
  - *Corrección*: "`signal_mon(lleno);`"
- **Página 156. Párrafo 2. Línea 3.**
  - *Errata*: "...que se borra de cola)"
  - *Corrección*: "...que se borra de la cola)"
- **Página 156. Párrafo 2. Línea 4.**
  - *Errata*: "...y puede sea entrar en su sección crítica."
  - *Corrección*: "...y puede entrar en su sección crítica."
- **Página 160. Párrafo 5. Líneas 6 y 7.**
  - *Errata*: " ...de las variables de condición, que tiene asociado una cola de procesos..."
  - *Corrección*: " ...de las variables de condición, que tienen asociadas unas colas de procesos..."
- **Página 161. Párrafo 3. Líneas 4.**
  - *Errata*: ": envío y recepción sin bloque,"
  - *Corrección*: " : envío y recepción sin bloqueo,"

Página 168. Párrafo 5. Líneas 3.

- *Errata*: "a otro proceso del mismo conjunto."
- *Corrección*: "a los otros procesos del mismo conjunto."

Capítulo 5

- **Página 170. Párrafo 2. Línea 7**
  - *Errata*: "...las instancias de dichos recursos deben forman parte de caminos que sean ciclos."
  - *Corrección*: "...las instancias de dichos recursos deben forman parte de caminos que sean ciclos o encontrarse asignadas a procesos interbloqueados en otros ciclos."
- **Página 172. Párrafo 2. Línea 4.**
  - *Errata*: "...de las dos instancia del recurso R1"
  - *Corrección*: "...de las dos instancias del recurso R1"
- **Página 172. Párrafo 2. Línea 4.**
  - *Errata*: "...a los proceso"
  - *Corrección*: "...a los procesos"
- **Página 173. Párrafo 2. Línea 6.**
  - *Errata*: "...mediante la eliminación de la existencia de expro-"
  - *Corrección*: "...mediante la eliminación de la existencia de no expro-"
- **Página 179. Sección 5.5.2. Párrafo 2. Línea 1.**
  - *Errata*: "La implementación más conocida y usada de esta técnica..."
  - *Corrección*: "La implementación más conocida de esta técnica..."
- **Página 184. Ecuación 2.**
  - *Errata*: $\mathbf{X} = \mathbf{X} + \mathbf{A}_3$
  - *Corrección*: $\mathbf{X} = \mathbf{X} + \mathbf{A}_1$
- **Página 184 Ecuación 5.**
  - *Errata*: $\mathbf{X} = \mathbf{X} + \mathbf{A}_3$
  - *Corrección*: $\mathbf{X} = \mathbf{X} + \mathbf{A}_2$
- **Página 191. Problema 5.4. Línea 2.**
  - *Errata*: "En un cierto instante de..."
  - *Corrección*: "En un cierto instante..."

Capítulo 6

- **Página 201. Párrafo 3. Línea 3.**
  - *Errata*: "...de 1192 Kbytes."
  - *Corrección*: "...de 1192 KiB."
- **Página 211. Párrafo 2. Línea 1.**
  - *Errata*: "En la parte izquierda de la"
  - *Corrección*: "En la parte derecha de la"
- **Página 211. Párrafo 3. Línea 1.**
  - *Errata*: "En la parte derecha de la"
  - *Corrección*: "En la parte izquierda de la"
- **Página 222. Pie Figura 6.15.**
  - *Errata*: "Formato de una dirección lógica de proceso..."
  - *Corrección*: "Formato de una dirección lógica de un proceso..."
- **Página 224. Ejemplo 6.14. Línea 4.**
  - *Errata*: "...las tablas de página..."
  - *Corrección*: "...las tablas de páginas..."
- **Página 253. Figura 6.34.**
  - *Errata*: El sentido de las flechas inferiores es hacía la izquierda en lugar de hacía la derecha.

Capítulo 7

- **Página 259. Sección 7.2.1. Párrafo 1. Línea 5.**
  - *Errata*: "libre o ocupado"
  - *Corrección*: "libre u ocupado"

  Página 266. Párrafo 3. Línea 3.
  - *Errata*: "disco indica..."
  - *Corrección*: "disco indican..."

  Página 267. Párrafo 1. Línea 2.
  - *Errata*: "...ya dichas páginas..."
  - *Corrección*: "...ya que dichas páginas..."

- **Página 272. Párrafo 1.Línea 4.**
  - *Errata*: "...valores en el rango (0, ti]."
  - *Corrección*: "...valores en el rango (0, t_i]."

  Página 274. Párrafo 1. Línea 3.
  - *Errata*: "...del conjunto de página que"
  - *Corrección*: "...del conjunto de páginas que"

- **Página 276. Párrafo 3. Línea 1.**
  - *Errata*: "Otras implementaciones del algoritmo LRU requieren el apoyo del hardware."
  - *Corrección*: "Otras implementaciones del algoritmo LRU requieren el uso de hardware adicional al utilizado para implementar la memoria virtual."

- **Página 280. Párrafo 2. Línea 3.**
  - *Errata*: "(ver Ejemplo 7.8)"
  - *Corrección*: "(ver Ejemplo 7.9)"

- **Página 285. Ejemplo 7.14. Párrafo 3. Línea 3.**
  - *Errata*: "En la posición de la cola ocupada por $i = 2$..."
  - *Corrección*: "En la posición de la cola ocupada por $i = 5$..."

- **Página 286. Figura 7.14.**
  - *Errata*: "Instante t_4, página $i = 7$, $r = 0$ $m = 0$"
  - *Corrección*: "Instante t_4, página $i = 7$, $r = 1$ $m = 0$"
- **Página 286. Párrafo 1. Línea 3.**
  - *Errata*: "...se examina el bi *r*..."
  - *Corrección*: "...se examina el bit *r*..."
- **Página 288. Figura 7.15. Instantes t_1, t_2 y t_3. Página $i = 5$.**
  - *Errata*: "$r = 0$ $m = 0$"
  - *Corrección*: "$r = 0$ $m = 1$"
- **Página 288. Figura 7.15. Instantes t_1, t_2 y t_3. Página $i = 6$.**
  - *Errata*: "$r = 0$ $m = 1$"
  - *Corrección*: "$r = 1$ $m = 1$"
- **Página 288. Figura 7.15. Instante t_3. Página $i = 1$**
  - *Errata*: "$r = 0$ $m = 1$"
  - *Corrección*: "$r = 0$ $m = 0$"
- **Página 289. Párrafo 1. Línea 2.**
  - *Errata*: "cercana a la producido..."
  - *Corrección*: "cercana a la producida..."
- **Página 291. Píe de la Figura 7.17**
  - *Errata*: "Influencia de grado..."
  - *Corrección*: "Influencia del grado..."
- **Página 296. Párrafo 3. Línea 3**
  - *Errata*: "Además el campo válida..."
  - *Corrección*: "Además el bit validez..."

Capítulo 8

- **Página 316. Párrafo 1. Líneas 3.**
  - *Errata*: "transfiendo..."
  - *Corrección*: "transfiriendo..."
- **Página 318. Sección 8.6.1. Párrafo 1. Línea 4.**
  - *Errata*: ". El periodo..."
  - *Corrección*: ". La frecuencia..."
- **Página 321. Sección 8.6.2. Párrafo 1. Línea 1.**
  - *Errata*: "...de la memoria secundaría.."
  - *Corrección*: "...de la memoria secundaria.."
- **Página 331. Párrafo 8. Línea 1.**
  - *Errata*: "...de la memoria secundaría..."
  - *Corrección*: "...de la memoria secundaria..."
- **Página 331. Párrafo 8. Línea 2.**
  - *Errata*: "...en un tiempo de relativamente pequeño..."
  - *Corrección*: "...en un tiempo relativamente pequeño..."
- **Página 331. Párrafo 8. Línea 8.**
  - *Errata*: "...tripleta cilndro-cabeza-sector."
  - *Corrección*: "...tripleta cilindro-cabeza-sector."
- **Página 333. Enunciado pregunta 8.8. Línea 2.**
  - *Errata*: "Respuesta en sección 8.3.2"
  - *Corrección*: "Respuesta en sección 8.3.1"

Capítulo 9

- **Página 358. Ejemplo 9.7. Párrafo 2. Línea 4.**
  - *Errata*: "B_F = 193"
  - *Corrección*: "B_F = 103"
- **Página 363. Párrafo 2. Línea 8.**
  - *Errata*: "bloque físico de discos..."
  - *Corrección*: "bloque físico de disco..."
- **Página 366. Cálculo de N_{MB}. Primer denominador.**
  - *Errata*: "2 (KB/bloque)"
  - *Corrección*: "2 (KiB/bloque)"
- **Página 372. Figura 9.13**
  - *Errata*: sobra la flecha que une la implementación (b) y (c).
- **Página 374. Párrafo 8. Línea 2.**
  - *Errata*: "en añadir al archivo a la lista de bloques libres"
  - *Corrección*: "en añadir el bloque a la lista de bloques libres"
- **Página 375. Párrafo 9. Línea 2.**
  - *Errata*: "añadiendo al archivo a la lista de bloques libres."
  - *Corrección*: "añadiendo el bloque a la lista de bloques libres"

Capítulo 10

- **Página 390. Párrafo 6. Línea 1.**
  - *Errata*: "...que usan todo..."
  - *Corrección*: "...que usan toda..."
- **Página 400. Párrafo 1. Línea 6.**
  - *Errata*: "...del Ejemplo 10.1..."
  - *Corrección*: "...del Ejemplo 10.5..."
- **Página 401. Párrafo 8. Línea 1.**
  - *Errata*: "existirá"
  - *Corrección*: "existirán"

Apéndice C

- **Página 423. Figura C.3.**
  - *Errata*: el diagrama mostrado no contempla la distinción entre los estados ejecutándose en modo usuario y ejecutándose en modo supervisor tal y como se pide en el enunciado. En la Figura 11 se muestra la Figura C.3 corregida.

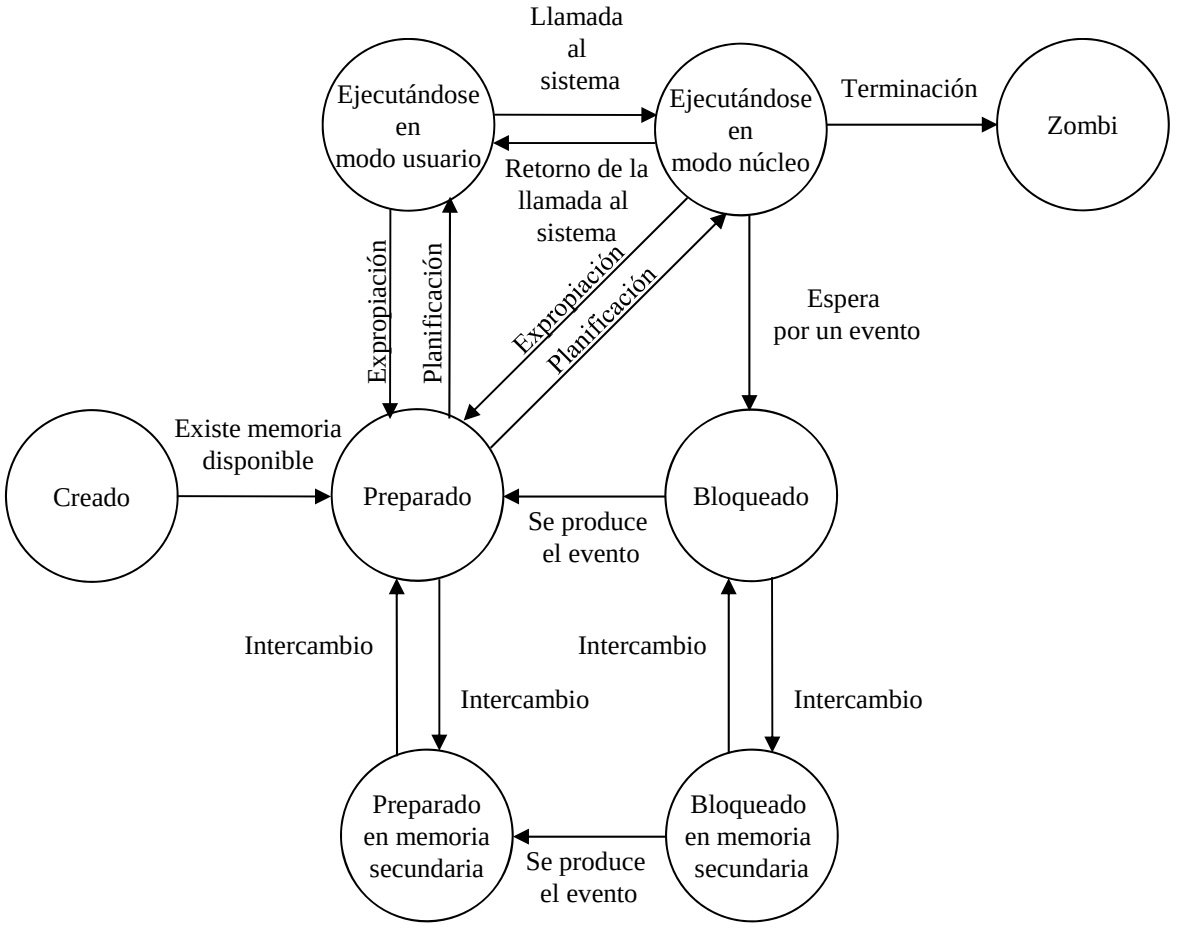

Figura 1 – Figura C.3 corregida

- **Página 428. Párrafo 6. Línea 1.**
  - *Errata*: "...si se considera un cuanto $q = 4$ ut se producen 4 cambios de proceso,"
  - *Corrección*: "...si se considera un cuanto $q = 4$ ut se producen 3 cambios de proceso,"
- **Página 429. Figura C.8a.**
  - *Errata*: Falta la flecha de terminación del proceso C en $t = 11$ ut.
- **Página 429. Figura C.8d.**
  - *Errata*: Falta la flecha de terminación del proceso C en $t = 11$ ut.
- **Página 435. Figura C.20.**
  - *Errata*: en el diagrama del uso del procesador el rótulo de la sexta flecha debe ser C en lugar de B, ya que el proceso que finaliza en t =14 ms es el proceso C.

Página 438. Solución Problema 4.1. Apartado a.

- *Errata*: La solución que se propone no es correcta ya que para acceder a una ventanilla no se respeta el orden de llegada a la oficina. En el Anexo 1 de esta fe de erratas se encuentra la solución del apartado a) del Problema 4.1 corregida.

Página 438. Solución Problema 4.1. Apartado c.

- *Errata*: La solución que se propone no es correcta ya que solo permite usar una única ventanilla de las cinco existentes. En el Anexo 1 de esta fe de erratas se encuentra la solución del apartado c) del Problema 4.1 corregida.

Página 441. Solución Problema 4.3.

- *Errata*: La solución que se propone no es correcta ya que no modela adecuadamente la espera de los caballeros para entrar al baño. En el Anexo 1 de esta fe de erratas se encuentra la solución del problema 4.3 corregida.

Página 442. Solución Problema 4.4.

- *Errata*: La solución que se propone no es correcta ya que para cortar el pelo no se respeta el orden de llegada a la peluquería. En el Anexo 1 de esta fe de erratas se encuentra la solución del problema 4.4 corregida.

Página 445. Solución Problema 4.5.

- *Errata*: La solución que se propone contiene algunas erratas. En el Anexo 1 de esta fe de erratas se encuentra la solución del problema 4.5 corregida.

Página 447. Solución Problema 4.6.

- *Errata*: La solución que se propone contiene algunas erratas. En el Anexo 1 de esta fe de erratas se encuentra la solución del problema 4.6 corregida.

Página 464. Solución Problema 7.1. Apartado c.

- *Errata*: E = 15 + 1 + 1 + 1 = 18
- *Corrección*: El tamaño del campo marco de página, de acuerdo con el apartado (a), es de 16 bits. Luego los tres bits y el campo marco de página ocupan un total de 16 + 1 + 1 + 1 = 19 bits.

  Por otra parte, para poder direccionar una entrada de tabla de páginas, ésta debe ocupar un número entero positivo de unidades direccionables. En el enunciado se indica que la unidad direccionable es la palabra de 16 bits. Por lo tanto, para almacenar la información de una entrada de la tabla de páginas, 19 bits, necesitamos usar

  $$\text{ceil}\left(\frac{19}{16}\right) = \text{ceil}(1.1875) = 2 \text{ palabras} = 32 \text{ bits}$$

  Luego el tamaño E que ocupa en memoria una entrada de tabla de páginas es E = 32 bits.
- *Nota*: Esta errata afecta y modifica los restantes cálculos de este apartado.

- **Página 465. Párrafo 1. Linea 4**
  - *Errata*: t_{gp}
  - *Corrección*: t_{gf}
- **Página 465. Ecuaciones (2), (4) y (5)**
  - *Errata*: Donde pone t_p
  - *Corrección*: Debe poner t_{am}

Página 473. Línea 12

- *Errata*: Donde pone 218 bytes
- *Corrección*: Debe poner 2^{18} bytes

- **Página 476. Último párrafo. Línea 1**
  - *Errata*: ", el cluster del primer bloque,"

  - *Corrección*: ", el primer cluster,"

- **Página 476. Último párrafo. Línea 2**
  - *Errata*: "Como cada bloque ocupa"
  - *Corrección*: "Como cada cluster ocupa"

ANEXO A: SOLUCIONES PROBLEMAS CAPITULO 4

Actualizado a 10 de octubre de 2024

NOTA: Se marcan en azul los cambios con respecto a la última actualización

---

4.1. En una oficina municipal de atención al ciudadano existen 5 ventanillas. Cuando un ciudadano entra en la oficina para realizar alguna gestión debe guardar una única cola hasta que alguna ventanilla queda libre. Escribir el pseudocódigo de un programa que coordine la actividad de los ciudadanos en la oficina usando: a) Semáforos binarios. b) Semáforos generales. c) Un monitor de nombre `oficina`, considerar la solución de Hansen en el comportamiento de la operación `signal_mon`. d) Paso de mensajes, suponer que la comunicación es indirecta a través de buzones y que se dispone de la operación `send` sin bloqueo y de la operación `receive` con bloqueo.

Solución:

a) En la Figura 2 se propone una solución que utiliza las siguientes variables globales y semáforos binarios:
   - `contador`. Variable global de tipo entero para llevar la cuenta del número de ciudadanos en la cola o en la ventanilla.
   - `S1`. Semáforo binario que se utiliza para garantizar la exclusión mutua en el uso de la variable global `contador`.
   - `S2`. Semáforo binario que se utiliza para sincronizar el acceso a las ventanillas de la oficina.

```
int contador=0; /* Definición e inicialización de la variable global */
semáforo_binario S1, S2; /* Definición semáforos binarios */
void ciudadano() /* Proceso ciudadano */
{
        wait_sem(S2); /* Esperar si todas las ventanillas están ocupadas */
        wait_sem(S1);
          contador = contador + 1;
          if (contador < 5) signal_sem(S2); /* Existen ventanillas libres */
        signal_sem(S1);

        realizar_gestión();

        wait_sem(S1);
          contador = contador - 1;
          if (contador == 4) signal_sem(S2); /* Queda una ventanilla libre */
        signal_sem(S1);
}

main() /* Inicialización semáforos y ejecución concurrente */
{
        init_sem(S1,1);
        init_sem(S2,1);
        ejecución_concurrente(ciudadano,...,ciudadano);
}
```

Figura 2 – Solución del apartado a) del Problema 4.1

b) En la Figura 3 se propone una solución que utiliza un único semáforo general `S1` que se inicializa con el número de ventanillas de la oficina.

```
semáforo S1;  /* Definición semáforo */

void ciudadano()  /* Proceso ciudadano */
{
        wait_sem(S1);
        realizar_gestión();
        signal_sem(S1);
}

main() /* Inicialización semáforo y ejecución concurrente */
{
        init_sem(S1,5);
        ejecución_concurrente(ciudadano,...,ciudadano);
}
```

Figura 3 – Solución del apartado b) del Problema 4.1

c) En la Figura 4 se propone una definición de monitor que utiliza las siguientes variables globales y variables de condición:

   - `contador`. Variable global de tipo entero para llevar la cuenta del número de ciudadanos en la cola o en las ventanillas.
   - `ventanilla_disponible`. Variable de condición para bloquear en su cola a los procesos hasta que exista alguna ventanilla disponible.

```
#define N 5 /* Número de ventanillas */
monitor oficina /* Definición del monitor */
    condición ventanilla_disponible;
    int contador;

    int obtener_ventanilla()  /* Procedimiento del monitor */
    {
      if (contador == 5) wait_mon(ventanilla_disponible);
      contador=contador+1;
    }

    void dejar_ventanilla() /* Procedimiento del monitor */
    {
      contador = contador - 1;
      signal_mon(ventanilla_disponible);
    }

    { /* Inicialización del monitor */
      contador=0;
    }
end monitor

void ciudadano() /* Proceso ciudadano */
{
    oficina.obtener_ventanilla();
    realizar_gestión();
    oficina.dejar_ventanilla();
}

main() /* Ejecución concurrente */
{
    ejecución_concurrente(ciudadano,...,ciudadano);
}
```

Figura 4 – Solución del apartado c) del Problema 4.1

Nótese que, a diferencia de los semáforos, la operación `wait_mon` de las variables de condición de los monitores siempre bloquea al proceso que la invoca. Por ese motivo antes de hacer `wait_mon` en el procedimiento `obtener_ventanilla` del monitor de la Figura 4 es necesario comprobar que el contador es igual a 5 y que por lo tanto es necesario esperar. De lo contrario podría haber ventanillas disponibles y quedarse el proceso bloqueado.

En la Figura 5 se propone otra posible definición de monitor que permite identificar cada ventanilla. Este monitor utiliza las siguientes variables globales y variables de condición:

- `contador`. Variable global de tipo entero para llevar la cuenta del número de ventanillas ocupadas.
- `ventanilla[N]`. Array de N variables globales de tipo entero. Si la ventanilla `k` con k=0,..,N-1 está libre se almacena en el elemento `k` de este array el valor `-1`, mientras que si está ocupada se almacena el identificador `PID` del proceso que está usando dicha ventanilla. El PID de un proceso es un número entero positivo que el sistema operativo asigna a cada proceso existente para poder identificarlo de manera univoca.
- `ventanilla_disponible`. Variable de condición para bloquear en su cola a los procesos hasta que exista alguna ventanilla disponible.

```
#define N 5 /* Número de ventanillas */
monitor oficina /* Definición del monitor */
    condición ventanilla_disponible;
    int k, contador, ventanillas[N];

    int obtener_ventanilla(int PID)  /* Procedimiento del monitor */
    {
      int h=0;
      if (contador == 5) wait_mon(ventanilla_disponible);
      for(h=0;h<N;h++)
      {
       if (ventanilla[h]==-1)
       {
        contador=contador+1;
        ventanilla[h]=PID;
        return h;
       }
      }
    }

    void dejar_ventanilla(int numero_ventanilla) /* Procedimiento del monitor */
    {
      contador = contador - 1;
      ventanilla[numero_ventanilla]=-1;
      signal_mon(ventanilla_disponible);
    }

    { /* Inicialización del monitor */
      contador=0;
      for(k=0;k<N;k++) ventanilla[k]=-1;
    }
end monitor

void ciudadano() /* Proceso ciudadano */
{
    int PID, numero_ventanilla;

    PID=obtener_PID();
    numero_ventanilla=oficina.obtener_ventanilla(PID);
    realizar_gestión();
    oficina.dejar_ventanilla(numero_ventanilla);
}

main() /* Ejecución concurrente */
{
    ejecución_concurrente(ciudadano,...,ciudadano);
}
```

Figura 5 – Solución alternativa del apartado c) del Problema 4.1

d) En la solución mediante paso de mensajes que se propone en la Figura 6, se utiliza un único buzón, `buzón1`, que se inicializa con 5 mensajes, un mensaje por cada ventanilla existente.

```
void ciudadano() /* Proceso ciudadano */
{
        mensaje X;
        receive(buzón1,X);
        realizar_gestión();
        send(buzón1,X);
}

void main() /* Inicialización del buzón y ejecución concurrente */
{
        mensaje M;
        crear_buzón(buzón1);
        for(h == 1; h <= 5; h = h+1) /* Inicialización del buzón */
        {
          send(buzon1,M);
        }
        ejecución_concurrente(ciudadano,...,ciudadano);
}
```

Figura 6 – Solución del apartado d) del Problema 4.1

4.2. Dos procesos A y B se ejecutan concurrentemente en un determinado sistema. El proceso A ejecuta unas tareas (`tareas_1()`) y alcanza un punto de encuentro. Posteriormente realiza otras tareas (`tareas_2()`) y finaliza. Por su parte el proceso B ejecuta unas tareas (`tareas_3()`) y llega al punto de encuentro. Posteriormente realiza otras tareas (`tareas_4()`) y finaliza. El primer proceso que llega al punto de encuentro no puede continuar su ejecución hasta que no llegue el otro proceso. No se sabe qué proceso comienza a ejecutarse primero o cuál es el primero que termina. Escribir el pseudocódigo de un programa que coordine la actividad de los procesos A y B usando: a) Semáforos. b) Paso de mensajes, suponer que la comunicación es indirecta a través de buzones y que se dispone de la operación `send` sin bloqueo y de la operación `receive` con bloqueo.

Solución:

a) La solución que se muestra en la Figura 7 para modelar el punto de encuentro entre los procesos A y B utiliza dos semáforos `S1` y `S2`. Ambos semáforos son inicializados al valor 0 ya que se utilizan para sincronizar.

```
semáforo S1, S2; /* Definición semáforos */
void proceso_A() /* Proceso A */
{
        tareas_1();
        signal_sem(S2);
        wait_sem(S1);
        tareas_2();
}

void proceso_B() /* Proceso B */
{
        tareas_3();
        signal_sem(S1);
        wait_sem(S2);
        tareas_4();
}

main() /* Inicialización de semáforos y ejecución concurrente */
{
        init_sem(S1,0);
        init_sem(S2,0);
        ejecución_concurrente(proceso_A,proceso_B);
}
```

Figura 7 – Solución del apartado a) del Problema 4.2

b) La solución que se muestra en la Figura 8 para modelar el punto de encuentro entre los procesos A y B mediante paso de mensajes utiliza dos buzones (`buzón1` y `buzón2`) que inicialmente están vacíos.

```
void proceso_A() /*Proceso A*/
{
        mensaje X;
        tareas_1();
        send(buzón1,X);
        receive(buzón2,X);
        tareas_2();
}

void proceso_B() /*Proceso B*/
{
        mensaje X;
        tareas_3();
        send(buzón2,X);
        receive(buzón1,X);
        tareas_4();
}

main()/*Creación de buzones y ejecución concurrente*/
{
        crear_buzón(buzón1);
        crear_buzón(buzón2);
        ejecución_concurrente(proceso_A,proceso_B);
}
```

Figura 8 – Solución del apartado b) del Problema 4.2

4.3. El baño de caballeros de un centro comercial posee una capacidad para seis caballeros. Cuando el servicio está completo los caballeros que desean pasar deben esperar fuera haciendo cola al lado de la puerta. Además si el operario de limpieza está limpiando el baño no puede pasar ningún caballero. Por otra parte, el operario solo pasa a limpiar el baño si éste está vacío. Escribir el pseudocódigo de un programa que usando semáforos binarios coordine la actividad de los caballeros y del operario de limpieza.

Solución: La solución que se propone en la Figura 9 para este problema utiliza las siguientes variables globales y semáforos binarios:

- `contador`. Variable global de tipo entero para llevar la cuenta del número de caballeros.
- `S1`. Semáforo binario que se utiliza para garantizar la exclusión mutua en el uso de la variable global `contador`.
- `S2`. Semáforo binario que se utiliza para sincronizar la actividad del operario de limpieza y los caballeros.
- `S3`. Semáforo binario que se utiliza para sincronizar la entrada al baño de los caballeros.

```
/* Definición variables y semáforos */
semáforo_binario S1, S2, S3;
int contador=0;

void caballero() /* Proceso caballero */
{

        wait_sem(S3); /* Esperar si el baño está completo */
        wait_sem(S1);
        contador = contador + 1;
        if (contador==1)
        {
                wait_sem(S2);   /* Esperar si el operario está en el baño */
                signal_sem(S3); /* Hay puestos en el baño */
        }
        else if (contador < 6) signal_sem(S3); /* Hay puestos en el baño */
        signal_sem(S1);

        usar_baño();

        wait_sem(S1);
        if (contador == 1) signal_sem(S2); /* Avisa al operario para que pase */
        else if (contador == 5) signal_sem(S3);      /* Avisa a un caballero para que pase *
        contador = contador - 1;
        signal_sem(S1);
}

void operario_limpieza() /* Proceso operario_limpieza */
{
        wait_sem(S2); /* Esperar si el baño está ocupado */
        limpiar_baño();
        signal_sem(S2); /* Avisa al primer caballero para que pase */
}

void main() /* Inicialización de semáforos y ejecución concurrente */
{
        init_sem(S1,1);
        init_sem(S2,1);
        init_sem(S3,1);
        ejecución_concurrente(caballero,...,caballero,operario_limpieza);
}
```

Figura 9 – Solución Problema 4.3

4.4. Una peluquería de caballeros tiene una capacidad para 5 clientes (4 sillas para esperar y un sillón para cortarse el pelo). Si un cliente entra en la peluquería y ve que está llena entonces abandona la peluquería. En caso contrario, se sienta en una silla y espera a que llegue su turno y el peluquero le indique que se puede sentar en el sillón. El peluquero tiene que esperar a que el cliente se siente y le diga el corte que desea antes de empezar a cortarle el pelo. Cuando termina, el peluquero avisa al cliente para que se levante y se queda esperando a que le pague. A su vez, el cliente cuando paga al peluquero se queda esperando a que el peluquero le devuelva el cambio y la factura. Posteriormente, el cliente abandona la peluquería. Por su parte, el peluquero avisa al siguiente cliente si queda alguno. Si no hay clientes a quienes atender el peluquero se pone a dormir en el sillón. El primer cliente que entra en la peluquería despierta al peluquero y se queda de pie esperando a que el peluquero se prepare y le indique que se puede sentar en el sillón. Escribir el pseudocódigo de un programa que usando semáforos binarios coordine la actividad del peluquero y de sus clientes.

Solución: La solución que se propone en las Figura 10 a 13 utiliza las siguientes variables globales y semáforos binarios:

- `contador`. Variable global de tipo entero para llevar la cuenta del número de clientes dentro de la peluquería. Se inicializa a 0.
- `turno`. Variable global de tipo entero para indicar el número de turno al que le toca cortarse el pelo. Se usa solamente para los clientes que esperan en silla, no se aplica para el primer cliente que entra en la peluquería cuando estaba vacía. Se inicializa a 0.
- `número`. Variable global de tipo entero para indicar el número de turno que le corresponde a los clientes que esperan sentados en sillas. El primer cliente que entra en la peluquería cuando estaba vacía no coge número. Este número se va aumentando secuencialmente durante toda la jornada. Se inicializa a 0.
- `sillas`. Variable global de tipo array entero de 4 elementos para almacenar el número de turno del cliente que se sienta en cada silla. Si una silla está vacía entonces se indica almacenando en el elemento del array correspondiente el valor -1. Inicialmente todas las sillas están vacías. Recuérdese que en C los elementos de un array se empiezan a numerar desde 0.
- `S0`. Semáforo binario que se utiliza para garantizar la exclusión mutua en el uso de la variable global `contador`. Se inicializa a 1.
- `S1`. Semáforo binario que se utiliza para garantizar la exclusión mutua en el uso de las variables globales `turno`, `número` y `silla`. Se inicializa a 1. Como estas variables están relacionadas, para simplificar el código, se ha preferido usar un único semáforo para protegerlas a todas juntas, en lugar de usar un semáforo para cada una.

- `S2`. Semáforo binario que se utiliza para implementar las esperas del peluquero. Se inicializa a 0.
- `S3`. Semáforo binario que se utiliza para implementar las esperas del cliente que le toca ser atendido. Se inicializa a 0.
- `S40`. Semáforo binario que se utiliza para implementar la espera de un cliente en la silla 0. Se inicializa a 0.
- `S41`. Semáforo binario que se utiliza para implementar la espera de un cliente en la silla 1. Se inicializa a 0.
- `S42`. Semáforo binario que se utiliza para implementar la espera de un cliente en la silla 2. Se inicializa a 0.
- `S43`. Semáforo binario que se utiliza para implementar la espera de un cliente en la silla 3. Se inicializa a 0.

```
/* Definición variables globales y semáforos */
#define TRUE 1
semáforo_binario S0, S1, S2, S3, S40, S41, S42, S43;
int contador=0; turno=0; número=0; sillas[]={-1,-1,-1,-1};

void cliente() //Proceso cliente
{

        entrar_en_peluqueria();
        wait_sem(S0);
        if(contador <5)
        {
                contador=contador + 1;
                if contador==1
                {
                        signal_sem(S0);
                        signal_sem(S2); //Despertar al peluquero
                        wait_sem(S3);   //Esperar de pie a que el peluquero le diga
                                        //que se siente en el sillón
                }
                else
                {
                        signal_sem(S0);
                        esperar_sentado_en_silla(); //Ver pseudocódigo en FIGURA 12
                }

                sentarse_en_sillón();
                indicar_corte();
                signal_sem(S2); //Avisar al peluquero para que comience el corte
                wait_sem(S3);   //Esperar a que el peluquero termine de cortarle el pelo

                levantarse_y_pagar();
                signal_sem(S2); //Avisar al peluquero que espera por el pago
                wait_sem(S3);   //Esperar el cambio y la factura

                //Disminuir el contador de clientes
                wait_sem(S0);
                contador=contador-1;
                signal_sem(S0);
        }
        else signal_sem(S0);

        abandonar_peluquería();
}
```

Figura 10 – Primera parte solución Problema 4.4

```
void esperar_sentado_en_silla()
{
        //Esperar sentado en una silla libre
        wait_sem(S1);
        número=número+1;
        if (sillas[0]==-1){
                sillas[0]=número;
                signal_sem(S1);
                wait_sem(S40);
        }
        else if (sillas[1]==-1){
                sillas[1]=número;
                signal_sem(S1);
                wait_sem(S41);
        }
        else if (sillas[2]==-1){
                sillas[2]=número;
                signal_sem(S1);
                wait_sem(S42);
        }
        else if (sillas[3]==-1){
                sillas[3]=número;
                signal_sem(S1);
                wait_sem(S43);
        }
}
```

Figura 11 – Segunda parte solución Problema 4.4

```
void avisar_cliente_sentado()
{
        wait_sem(S1);
        turno=turno+1;
        if (sillas[0]==turno){
                signal_sem(S40);
                sillas[0]=-1;
        }
        else if (sillas[1]==turno){
                signal_sem(S41);
                sillas[1]=-1;
        }
        else if (sillas[2]==turno){
                signal_sem(S42);
                sillas[2]=-1;
        }
        else if (sillas[3]==turno)
        {
                signal_sem(S43);
                sillas[3]=-1;
        }
        signal_sem(S1);
}
```

Figura 12 – Tercera parte solución Problema 4.4

```
void peluquero() //Proceso peluquero
{
        int primer_cliente=0;

        while(TRUE)
        {
                primer_cliente=0;

                wait_sem(S0);
                if (contador==0)
                {
                        signal_sem(S0);
                        wait_sem(S2); //Se pone a dormir
                        primer_cliente=1;
                }
                else signal(S0);

                prepararse_para_cortar_pelo();

                //Avisar al cliente para que se siente en el sillón
                if(primer_cliente==1) //Caso primer cliente que espera de pie
                {
                        signal_sem(S3);
                }
                else //Caso cliente que espera en silla
                {
                        avisar_cliente_sentado(); //Ver pseudocódigo en FIGURA 12
                }

                wait_sem(S2); //Esperar a que el cliente se siente en el sillón
                              //y le indique el tipo de corte
                cortar_pelo();

                signal_sem(S3);//Avisar al cliente para que se levante y le pague
                wait_sem(S2);  //Esperar a que el cliente se levante y le pague

                cobrar_cliente();

                signal_sem(S3);//Avisar al cliente que está esperando el cambio y factura
        }
}

void main() /* Inicialización de semáforos y ejecución concurrente */
{
        init_sem(S0,1); init_sem(S1,1);
        init_sem(S2,0); init_sem(S3,0);
        init_sem(S40,0); init_sem(S41,0); init_sem(S42,0); init_sem(S43,0);
        ejecución_concurrente(cliente,...,cliente,peluquero);
}
```

Figura 13 – Cuarta y última parte solución Problema 4.4

4.5. En un restaurante autoservicio inicialmente vacío existen 10 puestos para comer. Cuando un cliente llega al restaurante lo primero que hace es buscar un puesto libre, si no encuentra ninguno se marcha. Si encuentra algún puesto libre lo reserva dejando allí sus cosas. A continuación coge, sin necesidad de esperar ninguna cola, lo que desea comer de unos mostradores. Luego se pone en una cola para que le cobre un dependiente. Finalmente vuelve a su puesto para comer. El dependiente solo atiende a los clientes de uno en uno y debe ser avisado por el cliente para que le cobre. Cuando no está cobrando a los clientes el dependiente se dedica a reponer los mostradores. Escribir el pseudocódigo de un programa que usando semáforos binarios coordine la actividad de los clientes y del dependiente.

Solución:

La solución que se propone en las Figuras 14 y 15 utiliza las siguientes variables globales y semáforos binarios:

- `contador`. Variable global de tipo entero para llevar la cuenta del número de puestos para comer ocupados. Esta variable se inicializa a 0.
- `aviso`. Variable global de tipo entero para avisar al dependiente cuando un cliente espera para ser cobrado. Recuérdese que cada cliente debe avisar al dependiente para que le cobre, ya que éste puede estar realizando tareas de reposición. Por tarea de reposición se entiende, por ejemplo, sacar un producto de una caja y colocarlo en un mostrador, llevar una caja de un sitio a otro, etc. Una tarea de reposición se ejecuta rápidamente, por lo que cuando el dependiente es avisado por el cliente, bien de viva voz o pulsando un llamador, el dependiente tarda poco en estar listo para cobrarle. Nótese que el dependiente no permanece sin hacer nada, es decir, bloqueado en espera de que le avise un cliente, sino que realiza una espera activa. Por ello no es posible utilizar un semáforo para avisar al dependiente y se tiene que usar una variable global. El cliente si desea ser atendido debe hacer `aviso=TRUE`. El dependiente cuando termina de atender a un cliente hace `aviso=FALSE`. Inicialmente está variable se configura a `FALSE`, ya que el dependiente está haciendo tareas de reposición.
- `S1`. Semáforo binario que se utiliza para garantizar la exclusión mutua en el uso de la variable global `contador`. Este semáforo se inicializa a 1.
- `S2`. Semáforo binario que se utiliza para garantizar la exclusión mutua en el uso de la variable global `aviso`. Este semáforo se inicializa a 1.
- `S3`. Semáforo binario que se utiliza para implementar la espera de los clientes en cola para pagar. Este semáforo se inicializa a 1.
- `S4`. Semáforo binario para implementar la sincronización entre el cliente que le toca pagar y el dependiente.Este semáforo se inicializa a 0.

Nótese que en esta solución se utilizan dos semáforos `S3` y `S4`, uno para esperar en la cola y otro para los pagos, podría parecer que uno de ellos es redundante pero es necesario para evitar que se pierdan los avisos. Si no hubiese espera en `S3`, dos clientes podrían poner `aviso=TRUE` y el dependiente sólo detectaría un aviso y dejaría esperando indefinidamente al segundo cliente. Una alternativa a usar dos semáforos sería por ejemplo usar un contador en vez de un aviso binario.

Además en esta solución, como no se indica lo contrario en el enunciado, se ha supuesto por simplificar que la cola para pagar se forma de forma aleatoria por aquellos clientes que ya han cogido la comida. O dicho de otra forma, que un proceso que ejecuta `coger_comida()` no tiene garantizada una determinada posición en la cola para pagar, dependerá del orden en que el planificador ejecute la instrucción `wait_sem(S3)` en cada proceso, lo cual se ajusta a una situación real: si dos clientes terminan de coger la comida a la vez, llegará antes a la cola el cliente que ande más deprisa.

```
/* Definición constantes, variables y semáforos */
#define TRUE 1
#define FALSE 0
int contador=0, aviso=FALSE;
semáforo_binario S1,S2,S3,S4;

void cliente() /* Proceso cliente */
{
        entrar_en_restaurante();
        wait_sem(S1);
        if (contador<10)
        {
          contador=contador+1;
          dejar_pertenencias_para_reservar();
          signal_sem(S1);

          coger_comida();

          wait_sem(S3); /* Esperar en cola para pagar */

          wait_sem(S2);
          aviso = TRUE; /* Cliente esperando para pagar */
          signal_sem(S2);

          wait_sem(S4); /* Esperar a que el dependiente le cobre */

          signal_sem(S3); /* Avanzar cola de espera */

          comer();

          wait_sem(S1);
          contador = contador - 1;
          signal_sem(S1);
        }
        else signal_sem(S1);

        abandonar_restaurante();

}
```

Figura 14 – Primera parte de la solución del Problema 4.5

```
void dependiente() /* Proceso dependiente */
{
        while(TRUE)
        {
          wait_sem(S2);
          if (aviso == TRUE)
          {
            cobrar_cliente();
            signal_sem(S4); /* Desbloquear al cliente que esperaba para ser cobrado */

            aviso = FALSE;
            signal_sem(S2);
          }
          else
          {
            signal_sem(S2);
            tarea_de_reposición();
          }
        }
}

/* Inicialización de semáforos y ejecución concurrente */
void main()
{
        init_sem(S1,1);
        init_sem(S2,1);
        init_sem(S3,1);
        init_sem(S4,0);
        ejecución_concurrente(cliente,...,cliente,dependiente);
}
```

Figura 15 – Segunda y última parte de la solución del Problema 4.5

4.6. Cinco filósofos están sentados en una mesa circular con cinco platos de espaguetis y cinco tenedores. Cada tenedor está situado en la mesa entre dos platos. Un filósofo cuando se cansa de pensar y le entra hambre tiene que coger dos tenedores para comer su plato de espaguetis. No puede coger los dos tenedores simultáneamente, es decir, primero coge uno (el de su derecha o el de su izquierda) y después el otro. Cuando sacia su apetito deja los dos tenedores sobre la mesa y continúa pensando. Escribir el pseudocódigo de un programa que usando semáforos coordine la actividad de los cinco filósofos evitando posibles interbloqueos.

Solución:

La solución que se propone [Tanenbaum, 2009] en la Figuras 17 y la Figura 18, para evitar la posible existencia de interbloqueos, establece que cada filósofo pueda coger los tenedores solo si los dos se encuentran disponibles. Además, puesto que solo hay cinco tenedores disponibles el máximo grado de concurrencia se corresponde a permitir que dos filósofos puedan estar comiendo simultáneamente. Cada filósofo utiliza dos tenedores, luego se utilizan cuatro tenedores simultáneamente y quedaría uno sin usar.

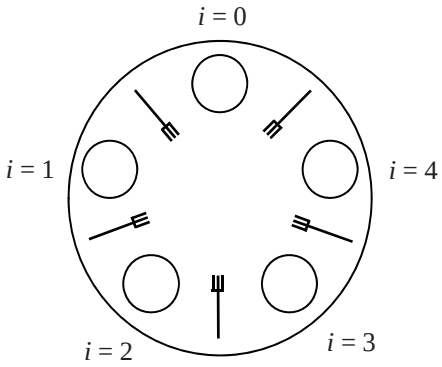

Figura 16 – Ubicación de los filósofos en la mesa

La solución utiliza un vector de cinco semáforos binarios denominado `comensales`, uno por filósofo, para bloquear a un filósofo en el caso de que no estén disponibles los dos tenedores que necesita. La solución supone que un filósofo puede encontrarse en tres estados: `pensando`, `hambriento` y `comiendo`. Inicialmente se supone que todos los filósofos están pensando. Para poder consultar y modificar el estado de un filósofo con exclusión mutua se utiliza el semáforo binario `exm`.

También es importante tener en cuenta la ubicación de cada filósofo en la mesa. Sea N el número de filósofos, si se asigna a cada filósofo un identificador numérico i, con $i = 0, 1, \ldots, N - 1$ en sentido contrario a las agujas del reloj (ver Figura 16), entonces el filósofo i tiene a su izquierda al

filósofo (i + N − 1) %N y a su derecha al filósofo (i + 1) %N. Recuérdese (ver Apéndice A) que % hace referencia a la operación módulo. En la Tabla C.1 se muestra para el caso N = 5 los vecinos a la izquierda y a la derecha que tiene cada filósofo i.

Filosofo a la izquierda (i + N − 1) %N	**Filosofo i**	**Filosofo a la derecha (i + 1) %N**
4	0	1
0	1	2
1	2	3
2	3	4
3	4	0

Tabla 1 – Vecinos de cada filosofo para N = 5

Cuando un filósofo termina de pensar invoca a la función `coger_ten` para coger los dos tenedores. Si los consigue puede comer. Cuando termina de comer invoca a la función `dejar_ten` para dejar los tenedores sobre la mesa y avisar a sus vecinos por si estaban esperando para utilizarlos.

El procedimiento `coger_ten` en primer lugar realiza una operación `wait_sem(exm)` que bloqueará al filósofo `i` en la cola del semáforo si dicho semáforo estaba a 0. Si el semáforo estaba a 1 entonces cambia el estado del filósofo i a `hambriento` e invoca a la función `comprobar` que comprueba si los vecinos del filósofo i no se encuentran comiendo. En caso afirmativo cambia el estado del filósofo i a `comiendo` y realiza una operación `signal_sem(comensales[i])` para desbloquear al filósofo i. En caso negativo, es decir, alguno de los vecinos del filósofo i estaban comiendo, entonces la función `comprobar` no realiza ninguna acción. Al retornar de ella se ejecuta una operación `signal_sem(exm)` y a continuación una operación `wait_sem(comensales[i])` para bloquear al filósofo i mientras quedan disponibles los tenedores.

El procedimiento `dejar_ten` en primer lugar realiza una operación `wait_sem(exm)` que bloqueará al filósofo en la cola del semáforo si dicho semáforo estaba a 0. Si el semáforo estaba a 1 entonces cambia el estado del filósofo i a `pensando` e invoca dos veces a la función `comprobar`. La primera vez con el identificador del vecino de la izquierda y la segunda con el identificador del vecino de la derecha. Con ello lo que se pretende es desbloquear a dichos filósofos por si estaban esperando para comer. El proceso finaliza con una operación `signal_sem(exm)`.

```
#define TRUE 1
#define N 5
#define pensando 0
#define hambriento 1
#define comiendo 2
semáforo_binario exm, comensales[N];
int estado[N];

void filósofo(int i)
{
        while(TRUE)
        {
          pensar();
          coger_ten(i);
          comer();
          dejar_ten(i);
        }
}

void coger_ten(int i)
{
        wait_sem(exm);
        estado[i] = hambriento;
        comprobar(i);
        signal_sem(exm);
        wait_sem(comensales[i]);
}

void dejar_ten(int i)
{
        wait_sem(exm);
        estado[i] = pensando;
        comprobar((i+N-1)%N);
        comprobar((i+1)%N);
        signal_sem(exm);
}

void comprobar(int i)
{
        if (estado[i]==hambriento && estado[(i+N-1)%N]!=comiendo ...
            && estado[(i+1)%N]!=comiendo)
        {
          estado[i] = comiendo;
          signal_sem(comensales[i]);
        }
}
```

Figura 17 – Primera parte de la solución del Problema 4.6

```
void main()
{
        init_sem(exm,1);
        for(i=0,i<N,i=i+1)
        init_sem(comensales[i],0);
        ejecución_concurrente(filósofo(0),filósofo(1),...,filósofo(N-1));
}
```

Figura 18 – Segunda y última parte de la solución del Problema 4.6