SISTEMAS OPERATIVOS
Solución PED1 (Noviembre 2013)

Solución Ejercicio 1

I) Se denomina *sobrecarga del sistema*, o simplemente *sobrecarga* (overhead), al tiempo que el procesador se encuentra ocupado ejecutando código del sistema operativo asociado a tareas y servicios de administración que no se pueden contabilizar a ningún proceso en particular. En conclusión la afirmación es **FALSA**.

II) El *planificador a largo plazo* debe intentar seleccionar trabajos de tal forma que en el sistema exista una mezcla adecuada de procesos limitados por CPU y procesos limitados por E/S. Si la mayoría de los trabajos seleccionado están limitados por CPU, entonces los dispositivos de E/S se encontrarían la mayor parte del tiempo inactivos. Por el contrario, si la mayoría de los trabajos seleccionados están limitados por E/S, sería el procesador el que se encontraría la mayor parte del tiempo inactivo. En ambos casos se estaría produciendo un desequilibrio no deseable en el uso de los recursos del sistema. En conclusión la afirmación es **FALSA**.

III) El *micronúcleo* se encarga de realizar únicamente los servicios absolutamente esenciales del sistema operativo, aquellos que dependen de la arquitectura de la máquina y que son independientes del tipo de sistema operativo, como por ejemplo, la gestión de memoria a bajo nivel, la comunicación entre procesos, la gestión de la E/S y la gestión de las interrupciones. Dichos servicios se ejecutan en modo núcleo. Los servicios menos esenciales del sistema operativo, aquellos que son independientes de la arquitectura de la máquina y que dependen del tipo de sistema operativo, como por ejemplo la gestión de la memoria virtual, la administración de los sistemas de archivos o servicios de seguridad y protección, etc, se implementan como extensiones del núcleo y se ejecutan en modo usuario. En conclusión la afirmación es **FALSA**.

Solución Ejercicio 2

a) En la Figuras 1 y 2 se muestra una posible solución haciendo uso de semáforos generales.

b) En las Figuras 3, 4 y 5 se muestra una posible solución haciendo uso únicamente de semáforos binarios.

c) En las Figuras 6 y 7 se muestra una posible solución haciendo uso de paso de mensajes con las características indicadas en el enunciado.

Nota 1: Como en el enunciado no se especifica en la solución que se propone se ha considerado que la visita guiada la realizan todos los que se encuentren en el punto de encuentro (PE), es decir, 10 o más visitantes. También sería válido el considerar que la visita se realiza en grupos de 10 visitantes.

Nota 2: Aunque en la solución que se propone no se ha considerado, también sería correcto el hacer que sea el proceso visitante guiado el que avise al guía cuando ya haya diez visitantes en el punto de encuentro (PE) para ello se puede usar otro semáforo S5. De esta forma se reduciría la espera activa del proceso guia.

Nota 3: Aunque en la solución que se propone no se ha considerado, el descanso de al menos 7 minutos del guía también podría haberse colocado al principio del bucle while del guía.

```
int visitantesPE=0;

semáforo_general S1; /*Para regular el acceso a la sala de exposiciones*/
semáforo_general S3; /*Para exclusión mutua sobre variable visitantesPE*/
semáforo_general S4; /*Para sincronizar la espera de los visitantes en el PE*/


void visitante_guiado() /*Código visitante guiado*/
{
  wait_sem(S1); /*Entrada a la sala*/

  ir_punto_encuentro();

  /*Inicio bloque espera en el Punto de Encuentro*/
  wait_sem(S3);
  visitantesPE=visitantesPE+1;
  signal_sem(S3);
  wait_sem(S4); /*Esperar a que le avise el guía*/
  /*Fin bloque espera en el Punto de Encuentro*/

  visita_guiada();

  signal_sem(S1); /*Salida de la sala*/
}

void visitante_libre() /*Código visitante libre*/
{
  wait_sem(S1); /*Entrada a la sala*/
  visita_libre();
  signal_sem(S1); /*Salida de la sala*/
}
```

Figura 1 – Solución apartado a) Ejercicio 2 (continúa en las Figura 2)

```
void guia() /*Código guia*/
{
  int OK;

  while(TRUE)
  {
    /*Inicio bloque esperar a que haya al menos 10 visitantes en el PE*/
    OK=0;
    while(OK==0)
    {
      wait_sem(S3);
      if(visitantesPE >= 10)
      {
        OK=1;
        signal_sem(S3);
      }
      else signal_sem(S3);
    }
    /*Fin bloque esperar a que haya al menos 10 visitantes en el PE*/

    levantarse();

    /*Inicio bloque aviso visitantes en punto de encuentro*/
    wait_sem(S3);
    while(visitantesPE > 0)
    {
      signal_sem(S4);
      visitantesPE=visitantesPE-1;
     }
    signal_sem(S3);
    /*Fin bloque aviso visitantes en punto de encuentro*/

    guiar_visitantes();

    sentarse_7min();
  }
}

main() /*Inicialización semáforos y ejecución concurrente*/
{
  init_sem(S1,V);
  init_sem(S3,1);
  init_sem(S4,0);
  ejecución_concurrente(visitantes_guiados, visitantes_libres, guia);
}
```

Figura 2 – Continuación solución apartado a) Ejercicio 2

```
int visitantes=V;
int visitantesPE=0;

semáforo_binario S1; /*Para exclusión mutua sobre variable visitantes*/
semáforo_binario S2; /*Para sincronizar la entrada de visitantes a la
                       sala de exposiciones*/
semáforo_binario S3; /*Para exclusión mutua sobre variable visitantesPE*/
semáforo_binario S4; /*Para sincronizar la espera de los visitantes en
                       el PE hasta que los avise el guia*/


void visitante_guiado() /*Código visitante guiado*/
{
  /*Inicio bloque entrada a la sala*/
  wait_sem(S1);
  visitantes=visitantes+1;
  if(visitantes > V)
  {
    signal_sem(S1);
    wait_sem(S2); /*Esperar a que salga un visitante de la sala*/
  }
  else signal_sem(S1);
  /*Fin bloque entrada a la sala*/

  ir_punto_encuentro();

  /*Inicio bloque espera en el Punto de Encuentro*/
  wait_sem(S3);
  visitantesPE=visitantesPE+1;
  signal_sem(S3);
  wait_sem(S4); /*Esperar a que le avise el guía*/
  /*Fin bloque espera en el Punto de Encuentro*/

  visita_guiada();

  /*Inicio bloque salida de la sala*/
  wait_sem(S1);
  if(visitantes > V) signal_sem(S2); /*Puede entrar otro visitante*/
  visitantes=visitantes-1;
  signal_sem(S1);
  /*Fin bloque salida de la sala*/
}
```

Figura 3 – Solución apartado b) Ejercicio 2 (continúa en la Figura 4)

```
void visitante_libre() /*Código visitante libre*/
{
  /*Inicio bloque entrada a la sala*/
  wait_sem(S1);
  visitantes=visitantes+1;
  if(visitantes > V)
  {
    signal_sem(S1);
    wait_sem(S2); /*Esperar a que salga un visitante de la sala*/
  }
  else signal_sem(S1);
  /*Fin bloque entrada a la sala*/

  visita_libre();

  /*Inicio bloque salida de la sala*/
  wait_sem(S1);
  if(visitantes > V) signal_sem(S2); /*Puede entrar otro visitante*/
  visitantes=visitantes-1;
  signal_sem(S1);
  /*Fin bloque salida de la sala*/
}
```

Figura 4 – Solución apartado b) Ejercicio 2 (continúa en la Figura 5)

```
void guia() /*Código guía*/
{
  int OK;

  while(TRUE)
  {
    /*Inicio bloque esperar a que haya al menos 10 visitantes en el PE*/
    OK=0;
    while(OK==0)
    {
      wait_sem(S3);
      if(visitantesPE >= 10)
      {
        OK=1;
        signal_sem(S3);
      }
      else signal_sem(S3);
    }
    /*Fin bloque esperar a que haya al menos 10 visitantes en el PE*/

    levantarse();

    /*Inicio bloque aviso visitantes en punto de encuentro*/
    wait_sem(S3);
    while(visitantesPE > 0)
    {
      signal_sem(S4);
      visitantesPE=visitantesPE-1;
     }
    signal_sem(S3);
    /*Fin bloque aviso visitantes en punto de encuentro*/

    guiar_visitantes();

    sentarse_7min();
  }
}

main() /*Inicialización semáforos y ejecución concurrente*/
{
  init_sem(S1,1);
  init_sem(S2,0);
  init_sem(S3,1);
  init_sem(S4,0);
  ejecución_concurrente(visitantes_guiados, visitantes_libres, guía);
}
```

Figura 5 – Continuación solución apartado b) Ejercicio 2

```
int visitantesPE=0;

void visitante_guiado() /*Código visitante guiado*/
{
  mensaje m;

  receive(S1,m); /*Entrada a la sala*/

  ir_punto_encuentro();

  /*Inicio bloque espera en el Punto de Encuentro*/
  receive(S3,m);
  visitantesPE=visitantesPE+1;
  send(S3,m);
  receive(S4,m); /*Esperar a que le avise el guía*/
  /*Fin bloque espera en el Punto de Encuentro*/

  visita_guiada();

  send(S1,m);   /*Salida de la sala*/
}

void visitante_libre() /*Código visitante libre*/
{
  mensaje m;

  receive(S1,m); /*Entrada a la sala*/

  visita_libre();

  send(S1,m); /*Salida de la sala*/
}
```

Figura 6 – Solución apartado c) Ejercicio 2 (continúa en la Figura 7)

```
void guia() /*Código guía*/
{
  int OK;
  mensaje m

  while(TRUE)
  {
    /*Inicio bloque esperar a que haya al menos 10 visitantes en el PE*/
    OK=0;
    while(OK==0)
    {
      receive(S3,m);
      if(visitantesPE >= 10)
      {
        OK=1;
        send(S3,m);
      }
      else send(S3,m);
    }
    /*Fin bloque esperar a que haya al menos 10 visitantes en el PE*/

    levantarse();

    /*Inicio bloque aviso visitantes en punto de encuentro*/
    receive(S3,m);
    while(visitantesPE > 0)
    {
      send(S4,m);
      visitantesPE=visitantesPE-1;
     }
    send(S3,m);
    /*Fin bloque aviso visitantes en punto de encuentro*/

    guiar_visitantes();

    sentarse_7min();
  }
}

main() /*Inicialización buzones y ejecución concurrente*/
{
  int h;
  mensaje nulo;

  crear_buzón(S1); /* Para regular acceso a la sala de exposiciones */
  for (h=1;h<=V;h++) send(S1, nulo); /*Inicializar S1 con V mensajes*/

  crear_buzón(S3); /*Para exclusión mutua sobre variable visitantesPE*/
  send(S3,nulo)    /*Inicializar S3 con 1 mensaje*/

  crear_buzón(S4); /*Para sincronizar la espera de los visitantes en
                     el PE hasta que los avise el guía*/

  ejecución_concurrente(visitantes_guiados, visitantes_libres, guia);
}
```

Figura 7 – Continuación solución apartado c) Ejercicio 2

Solución Ejercicio 3

En $t = 0$ ut llega el trabajo T0 y se le asigna una prioridad igual a 100. Como es el único trabajo pendiente pasa a ejecutarse en la CPU su primera ráfaga.

En $t = 1$ ut llega el trabajo T1 y se le asigna una prioridad igual a 100.

En $t = 3$ ut llega el trabajo T2 y se le asigna una prioridad igual a 100. Además finaliza la primera ráfaga de T0 con lo que se procede a recalcular las prioridades de todos los trabajos. Puesto que T0 acaba de usar la CPU su prioridad (P) se recalcula de la siguiente forma:

$$PT0 = \min\{100 + 3, 169\}=103$$

Su tiempo acumulado de uso de CPU (Tcpu) es TcpuT0= 3 ut.

Para el trabajo T1 su prioridad se recalcula de la siguiente forma:

$$PT1 = \max\{0, 100-1\}=99$$

Su tiempo acumulado de uso de CPU (Tcpu) es TcpuT1= 0 ut.

Para el trabajo T2 su prioridad se recalcula de la siguiente forma:

$$PT2 = \max\{0, 100-1\}=99$$

Su tiempo acumulado de uso de CPU (Tcpu) es TcpuT2= 0 ut.

Los trabajos T1 y T2 son los que tienen la prioridad más alta, como ambos no tienen acumulado tiempo de uso de CPU entonces se planifica la primera ráfaga del proceso que ha llegado antes, es decir, T1.

En $t = 5$ ut termina la primera ráfaga de CPU de T1. Las nuevas prioridades de los tres trabajos y sus tiempos acumulados de CPU son los siguientes:

$PT0 = \max\{0, 103-1\}=102$	TcpuT0 = 3 ut
$PT1 = \min\{99 + 2, 169\}=101$	TcpuT1 = 2 ut
$PT2 = \max\{0, 99-1\}=98$	TcpuT2 = 0 ut

Se planifica el trabajo T2 ya que es que el tiene mayor prioridad.

En $t = 13$ ut termina la primera ráfaga de CPU de T2. Las nuevas prioridades de los trabajos y sus tiempos acumulados de CPU son los siguientes (a partir de aquí puesto que el valor de la prioridad nunca va a estar fuera del rango [0,169] por simplificar se van omitir los máximos y mínimos de las expresiones de cálculo de las prioridades):

$PT0 = 102 - 1 = 101$	TcpuT0 = 3 ut
$PT1 = 101 - 1 = 100$	TcpuT1 = 2 ut
$PT2 = 98 + 8 = 106$	TcpuT2 = 8 ut

Se planifica el trabajo T1 ya que es que el tiene mayor prioridad.

En $t = 16$ ut termina la segunda ráfaga de CPU de T1. Las nuevas prioridades de los trabajos y sus tiempos acumulados de CPU son los siguientes:

$PT0 = 101 - 1 = 100$	TcpuT0 = 3 ut
$PT1 = 100 + 3 = 103$	TcpuT1 = 5 ut
$PT2 = 106 - 1 = 105$	TcpuT2 = 8 ut

Se planifica el trabajo T0 ya que es el que tiene mayor prioridad.

En $t = 18$ ut termina la segunda ráfaga de CPU de T0. Las nuevas prioridades de los trabajos y sus tiempos acumulados de CPU son los siguientes:

PT0 = 100 + 2 = 102 TcpuT0 = 5 ut
PT1 = 103 − 1 = 102 TcpuT1 = 5 ut
PT2 = 105 − 1 = 104 TcpuT2 = 8 ut

Los dos trabajos de mayor prioridad son T0 y T1, como ambos tienen el mismo tiempo de uso de CPU entonces se planifica el proceso de menor tiempo de llegada, es decir, T0.

En t = 20 ut termina la tercera ráfaga de CPU de T0. Las nuevas prioridades de los trabajos y sus tiempos acumulados de CPU son los siguientes:

PT0 = 102 + 2 = 104 TcpuT0 = 7 ut
PT1 = 102 − 1 = 101 TcpuT1 = 5 ut
PT2 = 104 − 1 = 103 TcpuT2 = 8 ut

Se planifica el trabajo T1 ya que es el que tiene mayor prioridad.

En t = 21 ut termina la tercera ráfaga de CPU de T1. Las nuevas prioridades de los trabajos y sus tiempos acumulados de CPU son los siguientes:

PT0 = 104 − 1 = 103 TcpuT0 = 7 ut
PT1 = 101 + 1 = 102 TcpuT1 = 6 ut
PT2 = 103 − 1 = 102 TcpuT2 = 8 ut

Los dos trabajos de mayor prioridad son T1 y T2, pero como T1 tienen menor tiempo acumulado de uso de CPU entonces se planifica T1.

En t = 23 ut termina la cuarta y última ráfaga de CPU de T1. Las nuevas prioridades de los trabajos y sus tiempos acumulados de CPU son los siguientes:

PT0 = 103 − 1 = 102 TcpuT0 = 7 ut
PT2 = 102 − 1 = 101 TcpuT2 = 8 ut

Se planifica el trabajo T2 ya que es el que tiene mayor prioridad.

En t = 24 ut termina la segunda ráfaga de CPU de T2. Las nuevas prioridades de los trabajos y sus tiempos acumulados de CPU son los siguientes:

PT0 = 102 − 1 = 101 TcpuT0 = 7 ut
PT2 = 101 + 1 = 102 TcpuT2 = 9 ut

Se planifica el trabajo T0 ya que es el que tiene mayor prioridad.

En t = 26 ut termina la cuarta ráfaga de CPU de T0. Las nuevas prioridades de los trabajos y sus tiempos acumulados de CPU son los siguientes:

PT0 = 101 + 2 = 103 TcpuT0 = 9 ut
PT2 = 102 − 1 = 101 TcpuT2 = 9 ut

Se planifica el trabajo T2 ya que es el que tiene mayor prioridad.

En t = 27 ut termina la tercera y última ráfaga de CPU de T2. Se planifica T0 ya que es el único trabajo pendiente.

En t = 28 ut termina la quinta y última ráfaga de CPU de T0.

De acuerdo con la explicación anterior en la Figura 8 se representa el diagrama de uso de la CPU. Los tiempos de retorno y de espera de cada trabajo son:

T0: T_R = 28 − 0 = 28 ut T_E = 28 − (3 + 2 + 2 + 2 + 1) = 18 ut
T1: T_R = 23 − 1 = 22 ut T_E = 22 − (2 + 3 + 1 + 2) = 14 ut
T2: T_R = 27 − 3 = 24 ut T_E = 24 − (8 + 1 + 1) = 14 ut

NOTA: Si se hubiera considerado en $t = 3$ ut que la prioridad asociada al trabajo T2 es 100 y que dicha prioridad no se recalcula por haber coincidido el instante de llegada del trabajo con el de finalización de una ráfaga entonces se hubiera obtenido el diagrama de uso de CPU que se muestra en la Figura 9. Los tiempos de retorno y espera serían los mismos que en el caso anterior.

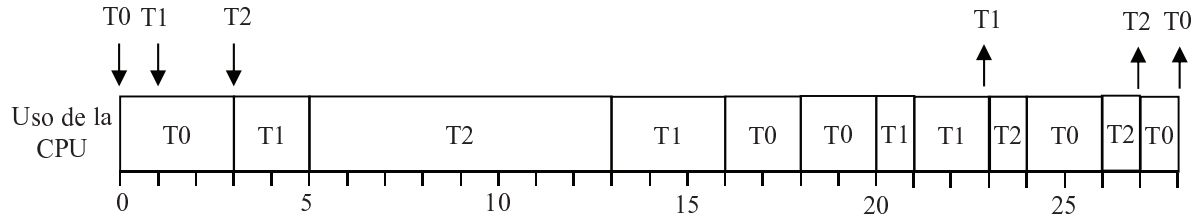

Figura 8 – Diagrama de uso de la CPU

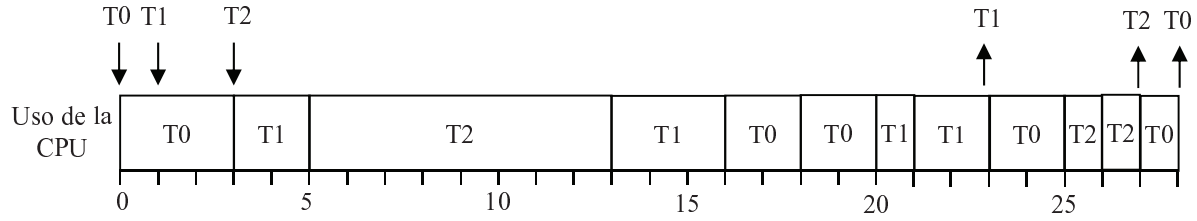

Figura 9 – Otro posible diagrama de uso de la CPU