Primer parcial de Sistemas Operativos

3 de mayo de 2025

Lea detenidamente las siguientes instrucciones. No cumplir los requerimientos puede implicar la pérdida del parcial.

- **Indique su nombre completo, número de cédula y número de parcial en cada hoja** (no se corregirán las hojas sin datos). Numere todas las hojas e indique la cantidad total de hojas en la primera.
- **Escriba las hojas de un solo lado y empiece cada problema en una hoja nueva**.
- Sólo se contestarán dudas de letra y no se aceptarán dudas en los últimos 30 minutos del parcial.
- El parcial es **SIN** material y dura **3 horas**. Al momento de finalizar el parcial **no se podrá escribir absolutamente nada en las hojas**. El estudiante debe ponerse de pie e ir a la fila de entrega.

Debe justificar todas las respuestas, siempre.

Problema 1 (17 puntos)

(a) I. (2 puntos) Describa dos tipos de llamadas al sistema.

Solución: Los tipos de llamadas al sistema son:

- Control de procesos: utilizadas para cargar, ejecutar, finalizar y abortar procesos, obtener atributos, cargar atributos, esperar por tiempo o por un evento señal, obtener o liberar memoria, etc.
- Gestión de archivos: utilizadas para crear, borrar, abrir, cerrar, leer, escribir, obtener o cargar atributos de archivos, etc.
- Gestión de dispositivos: utilizadas para requerir o liberar un dispositivo, leer o escribir, buscar o cargar atributos de un dispositivo, etc.
- Gestión de información del sistema: utilizadas para obtener o cargar la hora del sistema, datos del sistema, datos de procesos, etc.
- Comunicaciones: utilizadas para crear o destruir conexiones, enviar o recibir mensajes, etc.

II. (3 puntos) Describa las acciones que suceden desde la invocación a una llamada al sistema hasta su retorno.

Solución: La invocación a una llamada al sistema incluye las siguientes tareas:

- Cargar los parámetros en el lugar adecuado (stack o registros).
- Cargar el número de llamada al sistema en un registro específico (EAX en Intel).
- Invocar a la interrupción por software (trap) adecuada (manejador).
- Se cambia el bit de modo a monitor e invoca al manejador de la interrupción que controla que el número de llamada al sistema pasado en el registro sea menor que el mayor del sistema y, finalmente, invoca a la llamada al sistema correspondiente.
- El valor retornado por la llamada al sistema se almacena en un registro específico (EAX en Intel).
- Se vuelve el sistema a modo usuario y se retorna el control al proceso que invocó la llamada al sistema (o eventualmente a otro).

III. (3 puntos) Se implementa una llamada al sistema para interactuar con el hardware, que se incorpora a una biblioteca a nivel de usuario. Explique las diferencias que existen en la invocación de esta llamada al sistema entre un sistema operativo monolítico y un sistema operativo diseñado en capas.

Solución: Para una llamada al sistema que interactúa con el hardware, en el caso de un sistema monolítico el manejador puede invocar directamente al código del núcleo que se utiliza para interactuar con el hardware. En el caso de un sistema en capas, se debe interactuar con la capa N-1, N-2, hasta llegar a la capa 0 donde se encuentra el hardware y de esta manera respetar el diseño del sistema.

(b) Considere el grafo de asignación de recursos (R_i) a procesos (P_j) de la figura.

I. (2 puntos) Determine si el sistema se encuentra en un estado seguro

Solución:
Las matrices de asignación, demanda y disponibilidad de recursos son

	asignado					demanda					disponible				
	R_1	R_2	R_3	R_4	R_5	R_1	R_2	R_3	R_4	R_5	R_1	R_2	R_3	R_4	R_5
P_1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
P_2	1	0	0	0	0	0	1	1	0	1					
P_3	0	0	1	0	0	0	0	0	0	1					
P_4	0	1	0	0	0	0	0	0	0	1					
P_5	0	0	0	0	1	0	0	0	1	0					

El único proceso que puede ejecutar es P_4, dado que requiere [0,0,0,0,0] $\leqslant$ [0,0,0,0,0] (disponible). El proceso P_4 finaliza su ejecución, libera una unidad del recurso R_2 y las matrices de asignación, demanda y disponibilidad de recursos quedan

	asignado					demanda					disponible				
	R_1	R_2	R_3	R_4	R_5	R_1	R_2	R_3	R_4	R_5	R_1	R_2	R_3	R_4	R_5
P_1	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0
P_2	1	0	0	0	0	0	1	1	0	0					
P_3	0	0	1	0	0	0	0	0	0	1					
P_5	0	0	0	0	1	0	0	0	1	0					

Ninguno de los procesos restantes puede ejecutar a continuación, ya que todos requieren más que [0,1,0,0,0] (disponible). El sistema no se encuentra en un estado seguro y entrará en deadlock.

El resultado obtenido también puede fundamentarse mediante la propiedad que indica que "Si existe solo una instancia de cada recurso, un ciclo en el grafo de asignación de recursos indica que el sistema no se encuentra en un estado seguro". En este caso, la existencia del ciclo P_2, R_5, P_5, R_4, P_1, R_1, P_2 implica la situación de deadlock.

II. (2 puntos) ¿Qué sucede si se agrega una unidad del recurso R_5?

Solución: Si se agrega una unidad del recurso R_5, el vector de recursos disponibles pasa a ser [0,1,0,0,1]. El proceso P_3 puede ejecutar, dado que requiere [0,0,0,0,1] $\leqslant$ [0,1,0,0,1] (disponible).
El proceso P_3 finaliza su ejecución, libera una unidad del recurso R_3 y las matrices de

asignación, demanda y disponibilidad de recursos quedan

	asignado					demanda					disponible				
	R_1	R_2	R_3	R_4	R_5	R_1	R_2	R_3	R_4	R_5	R_1	R_2	R_3	R_4	R_5
P_1	0	0	0	1	0	1	0	0	0	0	0	1	1	0	1
P_2	1	0	0	0	0	0	1	1	0	0					
P_5	0	0	0	0	1	0	0	0	1	0					

A continuación puede ejecutar el proceso P_2, que requiere [0,1,1,0,1] $\leqslant$ [0,1,1,0,1] (disponible). El proceso P_2 finaliza su ejecución, libera una unidad del recurso R_1 y el vector de recursos disponibles pasa a ser [1,1,1,0,1]. Luego puede ejecutar P_1, que requiere [1,0,0,0,0] $\leqslant$ [1,1,1,0,1] (disponible).
El proceso P_1 finaliza su ejecución, libera una unidad del recurso R_4 y el vector de recursos disponibles pasa a ser [1,1,1,1,1]. Finalmente, el proceso P_5 puede ejecutar y finalizar.
En el caso planteado, el sistema se encuentra en un estado seguro.

(c) I. (2 puntos) Explique el concepto de cambio de contexto y su importancia en sistemas multitarea.

Solución: El cambio de contexto es una serie de acciones por las cuales se guarda el estado de un proceso o hilo que está ejecutándose, para poder cargar y ejecutar otro proceso o hilo.
El cambio de contexto es clave para que varios programas puedan ejecutarse "al mismo tiempo" en un sistema multitarea. En la realidad, la CPU está alternando rápidamente de uno a otro programa.

II. (3 puntos) Indique qué módulo del sistema operativo se encarga de ejecutar los cambios de contexto. Describa qué otras tareas realiza este módulo.

Solución: El módulo que se encarga de realizar los cambios de contexto es el despachador. Este módulo es invocado por el planificador cuando necesita cambiar el proceso que se encuentra ejecutando por otro que se encuentra listo. Además del cambio de contexto, el despachador se encarga de: i) cambiar el bit de modo a usuario y ii) saltar a la instrucción adecuada que había quedado el proceso al cual se le asignó a la CPU (registro Program Counter, PC). El valor del PC lo toma del campo registros del PCB del proceso entrante.

Problema 2 (15 puntos)

Considere un sistema operativo multiprogramado con dos CPUs. El planificador utiliza el algoritmo *Round Robin* con un quantum de 2 unidades de tiempo. En un tiempo dado, se crea un proceso asociado a la siguiente rutina:

```
1  void main() {
2      int a = fork ();
3      int b = fork ();
4      if (a > 0) {
5          obtener_recurso("IMPRESORA");
6          escribir_buffer ("IMPRESORA", "datos");
7          liberar_recurso("IMPRESORA");
8          otras_tareas_1 ();
9      } else {
```

```
10                  wait(NULL);
11                  otras_tareas_5();
12              }
13          }
```

Las llamadas al sistema obtener_recurso y liberar_recurso permiten obtener y liberar un recurso dado su nombre. Un proceso que intente obtener un recurso que ha sido tomado por otro proceso permanecerá bloqueado hasta que se complete la ejecución de la llamada al sistema liberar_recurso. En caso que varios procesos ejecuten obtener_recurso, se debe definir un criterio para la asignación del recurso. De manera similar, si dos procesos ingresan a la cola de listos al mismo tiempo, se debe establecer una regla para determinar su orden.

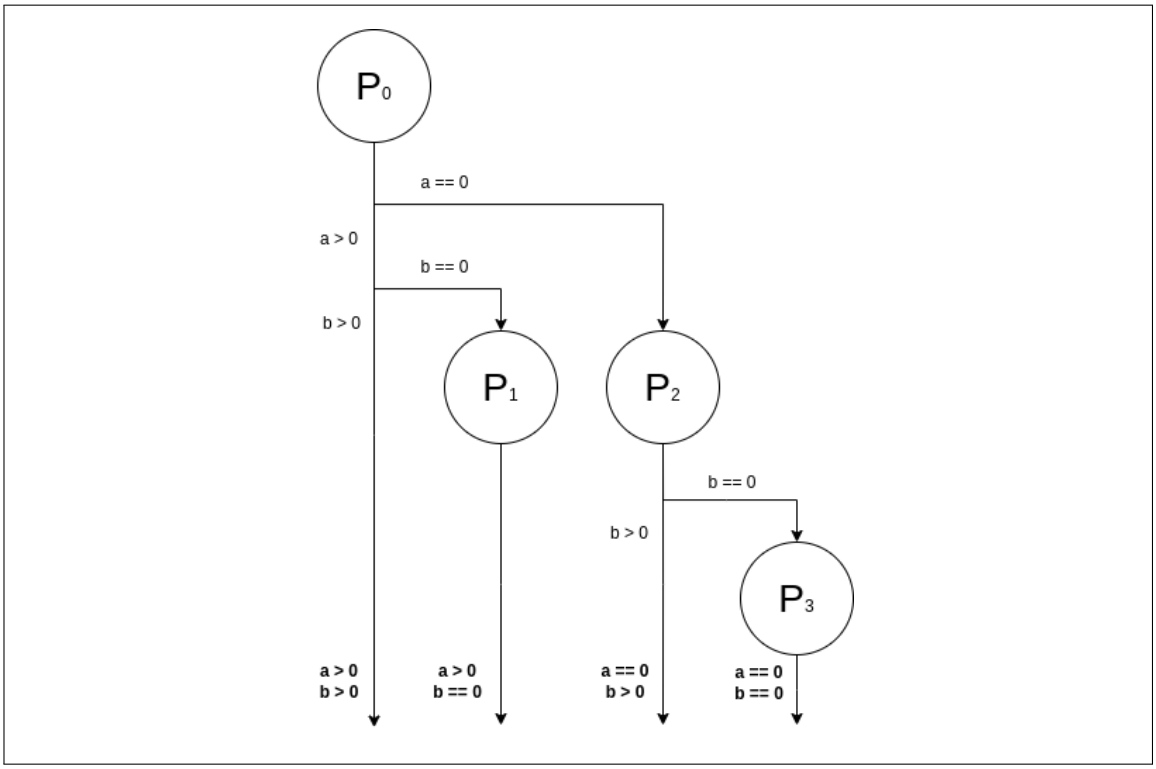
Toda operación que bloquee al proceso lo hará de forma inmediata, sin ejecutar ninguna parte de la operación en cuestión ni consumir el quantum. Una vez que el proceso se reanude tras el bloqueo, procederá a ejecutar las unidades de tiempo correspondientes a la operación que lo bloqueó. Si a un proceso se le expropia la CPU antes de comenzar una operación bloqueante, el proceso permanecerá en estado listo. En la siguiente asignación de CPU, ejecutará la operación y, en caso de bloquearse, lo hará de forma inmediata, sin consumir el quantum.

Las operaciones demandan el tiempo que se reporta a continuación.

Operación	*Tiempo*
fork()	1
obtener_recurso(string nombreRecurso)	1
escribir_buffer(string nombreRecurso, string datos)	1
liberar_recurso(string nombreRecurso)	2
otras_tareas_1()	1
wait(int *s)	0
otras_tareas_5()	5
asignación y evaluación booleana	0

(a) (3 puntos) Presente un diagrama de los procesos creados, donde pueda apreciar la relación entre ellos. Asigne un nombre a cada proceso y utilícelo en el diagrama de planificación.

Solución: Al ejecutar el código, se crean cuatro procesos:



(b) (12 puntos) Realice un diagrama de planificación (tiempo vs. procesos), comenzando en el tiempo t=0. Para cada proceso, indique su estado (E–ejecutando, L–listo, B–bloqueado), la línea de código que ejecuta y la justificación. Además, defina los criterios de resolución de prioridad para los casos de empate.

Solución:

	P0 (a>0, b>0)		*P2 (a==0, b>0)*		*P1 (a> 0, b==0)*		*P3 (a==0, b==0)*	
t	*Estado*	*Línea/justificación*	*Estado*	*Línea/justificación*	*Estado*	*Línea/justificación*	*Estado*	*Línea/justificación*
0	E	2/Ejecuta por ser el único proceso, crea a P2	-	-	-	-	-	-
1	E	3/Crea a P1, agota su quantum y pasa a listo junto a P1 y P3	E	3,4/Ejecuta utilizando el CPU libre, crea a P3	-	-	-	
2	E	4,5/Ejecuta por ser el primer proceso en cola de listos y obtiene el recurso	B	10/Se bloquea hasta que P3 termine	B	4,5/Como segundo proceso en cola de listos se bloquea al instante al perder la obtención del recurso contra P0	E	4,10,11/Ejecuta dado que P1 se encuentra bloqueado
3	E	6/Agota su quantum, pasa a listo	B		B		E	11/Agota su quantum, pasa a listo
4	E	7	B		B		E	11
5	E	7/Luego de este instante, el recurso está libre y por tanto P1 ya no estará bloqueado. Agota quantum, pasa a listo	B		B	Sigue bloqueado por la obtención del recurso. Luego de este instante, el recurso queda libre, por lo que P1 deja de estar bloqueado. Pasa a listo junto a P0 y P3	E	11/Agota su quantum, pasa a listo
6	E	8/Ejecuta como primer proceso en cola de listos	B		E	5/Ejecuta como segundo proceso en cola de listos la llamada para obtener el recurso	L	Estado listo por ocupar el tercer lugar de la cola de listos
7	T		B		E	6/Agota su quantum	E	11/ Ejecuta porque hay un CPU libre
8	T		E	11/P3 terminó su ejecución, P2 pasa a listo y ejecuta porque hay un CPU libre	E	7	T	P3 termina, por ello P2 ya no está bloqueado
9	T		E	11/Agota su quantum	E	7/Agota su quantum	T	
10	T		E	11	E	8	T	
11	T		E	11/Agota su quantum	T		T	
12	T		E	11	T		T	
13	T		T		T		T	

El criterio para los desempates utiliza el índice que identifica al proceso: da prioridad a los valores más bajos. Si varios procesos ejecutan obtener_recurso, el de menor índice obtiene el recurso. Si varios procesos ingresan a la cola de listos al mismo tiempo, ocuparán el final de la lista, ordenados por su índice.

Problema 3 (18 puntos)

Una ciudad ha implementado un sistema de alquiler de bicicletas con 15 estaciones distribuidas en zonas estratégicas. Cada estación tiene 10 anclajes para bicicletas. Al comienzo, todas las estaciones están llenas de bicicletas. Los usuarios pueden retirar bicicletas de cualquier estación y devolverlas en cualquier estación. Si la estación de la cual quiere retirar una bicicleta está vacía, el usuario debe esperar hasta que se devuelva una bicicleta allí. Si la estación en la que quiere devolver la bicicleta está llena (tiene los 10 anclajes ocupados), el usuario debe esperar hasta que se libere un espacio.

Tres empleados de mantenimiento revisan las estaciones cuando éstas lo solicitan. Durante la revisión, los empleados verifican los anclajes y las bicicletas, y reparan lo que este dañado. Los empleados pueden revisar una estación en cualquier momento, pero no pueden revisar una estación que ya está siendo (o va a ser) revisada por otro empleado. Mientras un empleado está revisando una estación,

los usuarios no pueden retirar ni devolver bicicletas en ella. Los empleados tienen prioridad sobre los usuarios.

Una estación que solicitó ser revisada no vuelve a solicitarlo hasta que se complete la revisión. Mientras espera a ser revisada, la estación puede seguir funcionando normalmente (los usuarios pueden retirar y devolver bicicletas).

Implemente en ADA las tareas usuario, empleado y estación. Se permite implementar tareas auxiliares.

Se cuenta con las siguientes funciones:

- elegir_estacion(): Integer: ejecutada por los usuarios para elegir una estación de donde retirar o devolver una bicicleta
- retirar_bicicleta(id: Integer): ejecutada por los usuarios para retirar una bicicleta de la estación id
- usar_bicicleta(): ejecutada por los usuarios mientras utilizan una bicicleta
- devolver_bicicleta(id: Integer): ejecutada por los usuarios para devolver una bicicleta a la estación id
- revisar_estacion(id: Integer): ejecutada por los empleados para inspeccionar y reparar anclajes y bicicletas en la estación id
- revisar(): Boolean: ejecutada por las estaciones para verificar si necesitan mantenimiento. Devuelve True si necesitan mantenimiento y False en caso contrario.

Solución:

```
1   task type estacion is
2       entry retirar ();
3       entry fin_retirar ();
4       entry devolver();
5       entry fin_devolver();
6       entry iniciar_mantenimiento();
7       entry finalizar_mantenimiento();
8       entry set_id(num : Integer);
9   end estacion;
10
11  task body estacion is
12      bicicletas : Integer;
13      usuarios: Integer;
14      id : Integer;
15      esperando_revision: Boolean;
16  begin
17      accept set_id(num: Integer) do
18          id := num;
19      end set_id;
20      bicicletas := 10;
21      usuarios := 0;
22      esperando_revision := False;
23      loop
24          select
25              when iniciar_mantenimiento'Count = 0 and bicicletas > 0 =>
26                  accept retirar ();
27                  bicicletas := bicicletas - 1;
28                  usuarios := usuarios + 1;
29          or
30              accept fin_retirar ()
31              usuarios := usuarios - 1;
```

```
32                  or
33                      when iniciar_mantenimiento'Count = 0 and bicicletas < 10 =>
34                          accept devolver();
35                          bicicletas := bicicletas + 1;
36                          usuarios := usuarios + 1;
37                  or
38                      accept fin_devolver();
39                      usuarios := usuarios - 1;
40                  or
41                      when usuarios = 0 =>
42                          accept iniciar_mantenimiento();
43                          accept finalizar_mantenimiento();
44                          esperando_revision := False;
45                  end select;
46                  if revisar() and not esperando_revision then
47                          admin.estacion_revisar(id);
48                          esperando_revision := True;
49                  end if;
50              end loop;
51          end estacion;
52
53          task type usuario;
54
55          task body usuario is
56              estacion_id: Integer;
57          begin
58              estacion_id := elegir_estacion();
59              estaciones[estacion_id].retirar();
60              retirar_bicicleta(estacion_id);
61              estaciones[estacion_id].fin_retirar();
62              usar_bicicleta();
63              estacion_id := elegir_estacion();
64              estaciones[estacion_id].devolver();
65              devolver_bicicleta(estacion_id);
66              estaciones[estacion_id].fin_devolver();
67          end usuario;
68
69          task type empleado;
70
71          task body empleado is
72              estacion_id: Integer;
73          begin
74              loop
75                  admin.asignar_estacion(estacion_id);
76                  estaciones[estacion_id].iniciar_mantenimiento();
77                  revisar_estacion(estacion_id);
78                  estaciones[estacion_id].finalizar_mantenimiento();
79              end loop;
80          end empleado;
81
82          task admin is
83              entry estacion_revisar(id: Integer);
84              entry asignar_estacion(id: Integer);
85          end admin;
86
87          task body admin is
```

```
88              estacion_id: Integer;
89              estaciones_revisar: array [0..14] of Boolean;
90          begin
91              for id in 0..14 loop
92                  estaciones_revisar[id] := False;
93              end loop;
94              loop
95                  estacion_id := -1;
96                  for id in 0..14 loop
97                      if estaciones_revisar[id] then
98                          estacion_id := id;
99                          exit;
100                     end if;
101                 end loop;
102                 select
103                     accept estacion_revisar(id: Integer) do
104                         estaciones_revisar[id] := True;
105                     end accept;
106                 or
107                     when estacion_id <> -1 =>
108                         accept asignar_estacion(id: Integer) do
109                             id := estacion_id;
110                         end accept;
111                         estaciones_revisar[id] := False;
112                 end select;
113             end loop;
114         end admin;
115
116         estaciones: array[15] of estacion;
117         empleados: array[3] of empleado;
118
119         begin
120             for id in 0..14 loop
121                 estaciones[id].set_id(id);
122             end loop;
123         end;
```