Departamento de Arquitectura

Primer parcial de Sistemas Operativos

3 de mayo de 2023

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

Formato:

- Indique su nombre completo y número de cédula en cada hoja (no se corregirán las hojas sin nombre). 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.
- Si se entregan varias versiones de un problema solo se corregirá la primera de ellas.

Dudas:

- Sólo se contestarán dudas de letra.
- No se aceptarán dudas en los últimos 15 minutos del parcial.

Material:

- El parcial es **SIN** material (no puede utilizarse ningún apunte, libro ni calculadora). Sólo puede tenerse las hojas del parcial, lápiz, goma y lapicera en su banco. Todas sus demás pertenencias debe colocarlas en el piso debajo de su asiento.

Finalización:

- El parcial dura **3 horas**.
- Al momento de finalizar el parcial no se podrá escribir absolutamente nada en las hojas, debiéndose parar e ir a la fila de entrega. Identificar cada una de las hojas con nombre, cédula y numeración forma parte de la duración del parcial.

Importante: Debe justificar todas las respuestas. Siempre.

Problema 1 (10 pts)

(a) (2 pts) Explique la motivación de implementar protección a nivel de la CPU y cómo se implementa.

> **Solución:** Para esta parte se esperaba que la respuesta incluyera la explicación sobre la posibilidad del *infinite loop*. Con respecto a la implementación se esperaba mención al timer, explicar las interrupciones y qué hace la rutina de atención. Ver diapositivas de los de Estructuras de los Sistemas de Computación como guía.

(b) (2 pts) Describa el funcionamiento de las llamadas al sistema fork() y exec().

> **Solución:** Sobre el fork() se esperaba que se mencionara que crea un nuevo proceso y que este hace una copia pero no comparte memoria con el proceso. Por otro lado con respecto a exec() se esperaba, al menos, que mencionara que sustituye el código ejecutable del proceso actual por el apuntado por excec.

(c) (2 pts) Incluyendo el proceso padre inicial. Cuántos procesos son creados por el siguiente código? Justifique su respuesta.

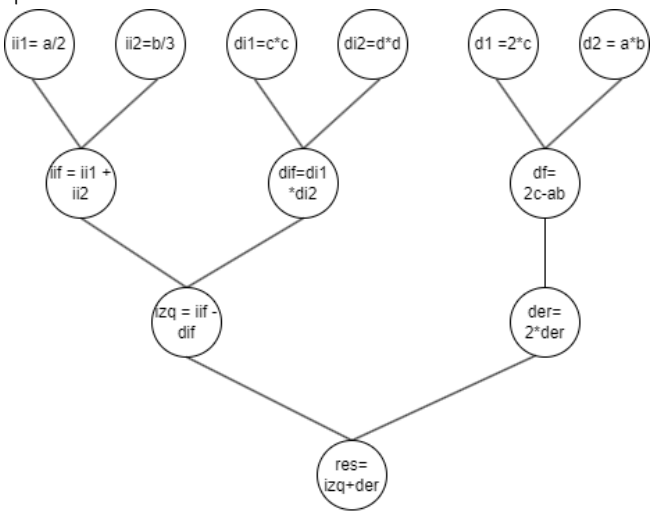
```
#include <stdio.h>
int main(){
    printf("%d\n",getpid());
    fork();
    printf("%d\n",getpid());
    fork();
    printf("%d\n",getpid());
```

```
    fork();
    printf("%d\n",getpid());

    return 0;
}
```

Solución: Son $2^3 = 8$ procesos. Eran válidas diversas justificaciones, xej hacer un árbol mostrando los forks. Lo importante era que se viera que el estudiante entendía el hilo de ejecución de este código y que cada fork duplicaba los procesos.

(d) (4 pts) Implemente la operación $(a/2 - c^2d^2 + b/3) + (2c - ab)d$ utilizando cobegin-coend, logrando el mayor nivel de concurrencia.

Solución: Se adjunta el grafo, esto no era necesario pero es recomendable como guía y para que el docente entienda el razonamiento




```
BEGIN
    Cobegin
    BEGIN   //Rama der
        Cobegin
        d1 =2*c
        d2 = a*b
        Coend
        df = 2c-ab
        der = df*d
    END
    BEGIN   //rama izq
        Cobegin
        BEGIN   //rama der izq
            Cobegin
                di1 = c*c
                di2 = d*d
            Coend
            dif = di1*di2
        END
        BEGIN   //rama izq izq
```

```
            Cobegin
                ii1 = a/2
                ii2 = b/3
            Coend
            iif = ii1 + ii2
        END
        Coend
        izq = iif - dif
    END
    Coend
    res = izq + der
END
```

Problema 2 (15 pts)

Considere un sistema con un único procesador sobre el que ejecuta un sistema operativo con un planificador Round Robin con un cuanto de 10 ms y un modelo de hilos M×1. Inicialmente en el sistema no se tiene ningún proceso de usuario ejecutando y en el tiempo t = 0 se lanza la ejecución del siguiente programa de usuario:

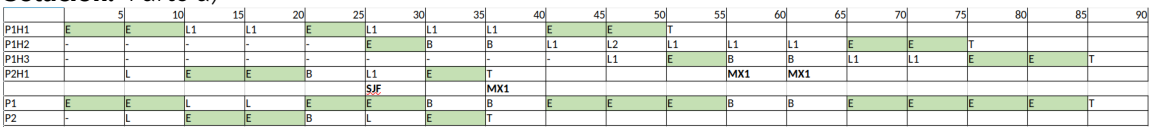
```
begin
    pid = fork();
    if pid = 0 then
        Ejecuta 5ms
        Bloquea 5ms
        Ejecuta 5ms
    else
        create_thread(proc1)
        create_thread(proc1)
        print(pid)
    end if
end

procedure proc1()
begin
    Ejecuta 5ms
    Bloquea 10ms
    Ejecuta 10ms
end
```

La operación fork() es estilo Unix. La operación create_thread() crea un nuevo hilo de ejecución y comienza su ejecución en el procedimiento recibido como argumento. Salvo que se indique explícitamente, el tiempo de ejecución de todas las funciones (fork, create_thread, condición del if, etc.) es de 5 ms. La asignación en pid = fork() se considera instantánea (solo cuenta el tiempo de ejecución del fork). La planificación definida para los hilos a nivel de usuario es SJF expropiativo. En caso de empates en las políticas de reemplazo se desempata por menor identificador (si los procesos Pi y Pj empatan, se selecciona como próximo proceso a ejecutar Pi si $i < j$ y Pj en caso contrario.

(a) (12 pts) Realice un diagrama de planificación (tiempo vs. hilos y tiempo vs. procesos), comenzando en el tiempo t = 0 e indicando el estado de cada uno de los hilos y procesos.

(b) (3 pts) Calcule el tiempo de espera y de retorno de cada proceso.

Solución: Parte a)



	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90
P1H1	E	E	L1	L1	E	L1	L1	L1	E	E	T							
P1H2	-	-	-	-	-	E	B	B	L1	L2	L1	L1	L1	E	E	T		
P1H3	-	-	-	-	-	-	-	-	-	L1	E	B	B	L1	L1	E	E	T
P2H1		L	E	E	B	L1	E	T				MX1	MX1					
						SJF		MX1										
P1	E	E	L	L	E	E	B	B	E	E	E	B	B	E	E	E	E	T
P2	-	L	E	E	B	L	E	T										

Parte b)

- Tiempo de espera P1 = 10
- Tiempo de espera P2 = 10
- Tiempo de retorno P1 = 85
- Tiempo de retorno P2 = 30

Problema 3 (15 pts)

(a) (9 pts) Explique las dos funciones principales de los semáforos presentadas en el curso. Ejemplifique presentando el código de tres procesos P1, P2, P3 que deben ejecutar una secuencia de unidades funcionales U1, U2, U3 (cada unidad funcional puede ser utilizada por un único proceso a la vez).

(b) (2 pts) ¿Qué modificaciones deben realizarse al código si se considera un cuarto proceso P4 que debe utilizar las unidades funcionales?

(c) (4 pts) ¿Qué modificaciones deben realizarse al código si el proceso P4 solamente debe usar la unidad U1 luego que el proceso P2 haya hecho uso de la unidad U2?

Solución: a) Las dos funciones de los semáforos presentadas en el curso son: i) control de acceso a recursos y secciones críticas en mutua exclusión y ii) señalización/secuenciación.

En el mecanismo de control de acceso acceso a recursos y secciones críticas, se inicializa un semáforo con el número de recursos disponibles o con el número máximo de procesos que se permite ingresar simultáneamente a la región crítica. Cada proceso debe realizar una operación wait (P) sobre el semáforo antes de usar el recurso o ingresar a la sección crítica y una operación signal (V) sobre el semáforo luego de usar el recurso o salir de la sección crítica.

En el mecanismo de señalización, para garantizar que un proceso P1 ejecute una parte S1 antes que un proceso P2 ejecute una parte S2, se inicializa un semáforo con el valor 0. El proceso P2 debe realizar una operación wait (P) antes de ejecutar la parte S2, hasta recibir la señal del proceso P1. Luego de ejecutar S1, el proceso P1 lo indica al proceso P2 ejecutando signal (V) sobre el semáforo, permitiendo a P2 continuar su ejecución (Silberschatz, paginas 214–215).

Ejemplificando sobre el caso de estudio planteado, el código para que tres procesos P1, P2, P3 utilicen tres unidades funcionales U1, U2, U3 bajo la restricción que las unidades funcionales no pueden utilizarse simultáneamente por más de un proceso es:

```
procedure MAIN
    init(S1,1);          ▷ S1 se utiliza para controlar el acceso exclusivo a la unidad U1
    init(S2,1);          ▷ S2 se utiliza para controlar el acceso exclusivo a la unidad U2
    init(S3,1);          ▷ S3 se utiliza para controlar el acceso exclusivo a la unidad U3
    cobegin
            P1()
            P2()
            P3()
    coend
```

end procedure

procedure P1
P(S1)
usar_U1()
V(S1)
P(S2)
usar_U2()
V(S2)
P(S3)
usar_U3()
V(S3)
end procedure

procedure P2
P(S1)
usar_U1()
V(S1)
P(S2)
usar_U2()
V(S2)
P(S3)
usar_U3()
V(S3)
end procedure

procedure P3
P(S1)
usar_U1()
V(S1)
P(S2)
usar_U2()
V(S2)
P(S3)
usar_U3()
V(S3)
end procedure

Los procesos son idénticos (en lo referente al uso de los mecanismos de sincronización y uso de las unidades funcionales). Los procesos utilizan las unidades funcionales en régimen de exclusión mutua, en el orden en que se presentan en la letra del problema (cualquier otro orden para el uso de las unidades funcionales es también correcto). Eventualmente, los procesos podrían realizar otras operaciones intercaladas que no son relevantes para el problema propuesto.

b) No se necesitan cambios en el código presentado, porque los semáforos se definen para controlar el acceso a los **recursos**. Para resolver el problema es suficiente con tres semáforos, independientemente del número de procesos en ejecución. Solamente se debe incluir el nuevo proceso P4, que tiene el mismo código que los procesos P1, P2 y P3, y su ejecución concurrente con los demás procesos, en el cobegin-coend.

c) Para implementar la secuencialización indicada, se debe incluir un nuevo semáforo S4 para señalización, inicializado en 0 y utilizado por los procesos involucrados en la secuencialización (P2 y P4):

procedure MAIN
init(S1,1); ▷ S1 se utiliza para controlar el acceso exclusivo a la unidad U1
init(S2,1); ▷ S2 se utiliza para controlar el acceso exclusivo a la unidad U2
init(S3,1); ▷ S3 se utiliza para controlar el acceso exclusivo a la unidad U3
init(S4,0); ▷ S4 se utiliza para la señalización/secuencialización de P2 y P4
cobegin
P1()
P2()
P3()
P4()
coend
end procedure

procedure P2
P(S1)
usar_U1()
V(S1)
P(S2)
usar_U2()
V(S2)
V(S4)
P(S3)
usar_U3()
V(S3)
end procedure

procedure P4
P(S4)
P(S1)
usar_U1()
V(S1)
P(S2)
usar_U2()
V(S2)
P(S3)
usar_U3()
V(S3)
end procedure

Se admite una solución donde el proceso P4 solamente usa la unidad funcional U1 (sin hacer uso de las otras, que no se mencionan explícitamente en la parte c) del problema)