UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Departamento de Informática de Sistemas y Computadoras (DISCA)

fSO

Ejercicio de Evaluación
05 de Noviembre de 2012

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

SURNAMES		**NAME**		Group
DNI		**Signature**		

- **Keep exam sheets stapled.**
- **Answer only in the space reserved for this purpose.**
- **Use clear and readable writting. Answer briefly and accurately.**
- **The exam has 9 questions, everyone indicates its grade.**

1) Can you create a user application that does not do any system call? Explain the answer. **0,75 points**

1	

2) During initialization or boot of the operating system of a modern desktop computer, may the processor be found in user mode? Explain the answer. **0,75 points**

2	

3) The following code refers to the generated executable file named "Example1".

```
   /*** Ejemplo1***/
   #include "all_required_header_files.h"
1
2  main()
3  { int i=0;
4    pid_t pid, pid2;
5
6
7    while (i<2)
8    { pid=fork()
9       switch(pid)
10       {case (-1):{printf("Error creating child\n");
11                   break;}
12        case (0):{pid2=fork();
13                  printf("Child %i created\n",i);
14                  sleep(10);
15                  exit(0);
16                 }
17        default: {printf("Parent\n");
18                  sleep(5);}
19      }
20     i++;
21   }
22   exit(0);
23 }
```

Suppose that "Example1" runs successfully

a) Indicate in a reasoned way, the number of processes that will create and draw the generated process tree.
b) Indicate and explain, if zombie processes and/or orphans can appear.

1.5 puntos

3	**a)**
	b)

---

4) Indicate and explain if the following statements, about process states, are true or false:

a) Only the processes that are *running* can be passed to the *finished* state, when the execution of their instructions ends.
b) The *suspended* state can be reached only by processes that have requested or are performing a blocking I/O operation.
c) The state change from *running* to *ready* is only possible in systems with preemptive schedulers.

1.0 point

4	a)
	b)

c)

---

5) A system has an multiqueue scheduler in the short term (STS) with three queues Queue0, Queue 1 and Queue 2, with scheduling algorithms **RR with q = 1, SRTF, and FCFS**, respectively.

Scheduling between **queues is managed with preemptive priorities** being the least priority and the higher priority the ones of Queue2 and Queue0, respectively. Each process has a counter of promotion (CountPro) that the system maintains to manage promotion/demotion between queues, so that processes can get into more priority queues. Whenever a process **goes to the suspended state** his CountPro is incremented by 1 (CountPro = CountPro+1). A process is placed in Queue0 if its CountPro is equal to 0, in Queue1 if its CountPro = 1 and in Queue2 if CountPro > = 2. When processes arrive to the system its CountPro is initialized (CountPro = 0) and they go to Queue0.

Suppose that I/O operations are carried out on the same FCFS scheduled device and that the following processes shown in the table arrive to the system:

Process	Execution profile	Arrival time	CountPro
A	2 CPU + 2 I/O + 1 CPU + 4 I/O + 1 CPU	0	0
B	1 CPU + 2 I/O + 4 CPU + 2 I/O + 2 CPU	1	0
C	3 CPU + 1 I/O + 1 CPU	2	0
D	2 CPU + 1 I/O+ 1CPU	3	0

a) Fill out the following table indicating at each instant in time where the processes are located.
b) Obtain the CPU utilization and the mean waiting time and mean turnaround time.

2.0 point (1.25+0.75)

5a

T	Queue 0 RR q=1	Queue 1 SRTF	Queue 2 FCFS	CPU	I/O queue	I/O	Event
0							A arrives, CountPro(A)=0
1							B arrives, CountPro(B)=0
2							C arrives, CountPro(C)=0
3							D arrives, CountPro(D)=0
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							

5b	Mean waiting time = Mean turnaround time = CPU utilization =

---

6) Given the following code that has been generated the executable file named "Example2".

```
 1 /*** Example2***/
 2 #include <stdio.h>
 3 #include <pthread.h>
 4
 5 void *fun_thread( void *ptr )
 6 { int sec;
 7
 8   sec=(int)ptr;
 9   sleep(sec);
10   printf("I waited %d seconds\n",sec);
11 }
12
13 int main()
14 {
15   pthread_attr_t atrib;
16   pthread_t thread1, thread2, thread3;
17
18   pthread_attr_init( &atrib );
19   pthread_create(&thread1, &atrib, fun_thread, (void *)30);
20   pthread_create(&thread2, &atrib, fun_thread, (void *)1);
21   pthread_create(&thread3, &atrib, fun_thread, (void *)10);
22   pthread_join(thread3, NULL)
23 }
```

Write the strings that prints the program in the terminal after its execution. Explain your answer.

1.0 point

6	

7) The following C code corresponds to a process with two threads that share memory and have to synchronize using the **busy waiting mechanism**. In order to design the **input and output protocols** to the critical section, the shared variables **flag** and **turn** are declared.

JUMP THIS QUESTION
IT CORRESPONDS TO REMOVED CONTENT

```
1  //***** Example 3 **//
2  #include <pthread.h>
3  …
4  int flag[2];
5  int turn;
6
7  void* thread(void* id)
8  { int i;
9    i = (int)id;
10
11   while ( 1 ) {
12
13     remaining_section();
14
15     /** b)Assign values to flag[i] and turn**/
16     while(/** c)Input protocol condition **/);
17
18     critical_section();
19
20     flag[i] = 0;
21   }
22 }
23 int main() {
24   pthread_t th0, th1;
25   pthread_attr_t atrib;
26   pthread_attr_init(&atrib);
27   /** a)Initialization of flag variable **/
28   pthread_create(&th0, &atrib, thread, (void *)0);
29   pthread_create(&th1, &atrib, thread, (void *)1);
30   pthread_join(th0, NULL);
31   pthread_join(th1, NULL);
32 }
```

Answer to the following items:

a) Replace "`/**a) Initialization of flag variable **/`" by its C code.
b) Replace "`/**b) Assign values to flag[i] and turn **/`" by its C code.
c) Replace "`/**c) Input protocol condition **/`" by its C code.

1.0 point

7	**a)**
	b)
	c)

Indicate all the possible values that can reach variable x after the concurrent execution of the processes A, B and C. Explain your answer indicating for each reached value of x the execution order of every section.

// Shared variables
int x=1;
Semaphores S1=3, S2=0, S3=0;		

// Process A	//Process B	//Process C

P(S1)
P(S3)
x = 2*x + 1; // section 1
V(S2)
	P(S1)
	P(S2)
	x = x*3; // section 2
	V(S1)
		P(S1)
		x = x + 2; // section 3
		V(S3)
		P(S1)
		x = x + 3; // section 4

1.0 point

8	

9) In 'Example4' complete the code of functions **fth_ONE** and **fth_TWO**, with the operations on **semaphores** needed to display on the screen the first ten unsigned integers sorted (0 1 2 3 4 5 6 7 8 9).

```
 1  /*** Example4***/
 2  #include <stdio.h>
 3  #include <pthread.h>
 4
 5  void *fth_ONE( void *ptr )
 6  { int i;
 7    for (i=1; i<10; i+=2)
 8    {
 9      printf("%d ",i);
10
11    }
12  }
```

```
13  void *fth_TWO( void *ptr )
14  { int i;
15  for (i=0; i<10; i+=2)
16    {
17      printf("%d ",i);
18
19    }
20  }
```

```
21  int main()
22  { pthread_attr_t atrib;
23    pthread_t th1, th2;
24
25    pthread_attr_init( &atrib );
26    pthread_create(&th1,&atrib,fth_ONE, NULL);
27    pthread_create(&th2,&atrib,fth_TWO, NULL);
28    pthread_join( th1, NULL);
29    pthread_join( th2, NULL);
30    printf ("\n");
  }
```

Thread 'th1' should display the odd numbers and "th2" the even numbers. To synchronize the threads execution use as many semaphores as needed and specify their initial values. Use the P and V names for semaphore operations.

1.0 puntos

9	

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Departamento de Informática de Sistemas y Computadoras (DISCA)

fSO

Ejercicio de Evaluación
05 de Noviembre de 2012

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

SOLUCIONES

---

1) ¿Se puede crear una aplicación de usuario cuyo código no incluya ninguna llamada al sistema? Justifique su respuesta. **0,75 puntos**

1	*No sería posible.* *Las aplicaciones de usuario necesitan acceder a datos del disco (read/write) y mostrar información por pantalla. Se trataría de una aplicación que no accede a los recursos hardware de la máquina.* *Se podría crear una aplicación que en ejecución, no realizara ningún acceso de entrada/salida, además dicha aplicación no podría terminar su ejecución normal con exit(). La única utilidad de un programa así sería la de ocupar tiempo de CPU (recuerde el programa "tragón" de practicas)*

Corrección: B/M (considerar excepcional un 0.5 → Sabe de que va pero no estáis seguros por como lo expone)

---

2) Durante la inicialización o arranque del sistema operativo de un computador de sobremesa moderno, el procesador ¿podría encontrarse en modo usuario? Justifique la respuesta. **0,75 puntos**

2	*No podría.* *Durante el arranque del sistema es necesario cargar el núcleo del sistema operativo en memoria, para ello se debe acceder al disco o memoria secundaria y por tanto es necesario ejecutar instrucciones de E/S. Las instrucciones de E/S son privilegiadas y solo se ejecutan en modo núcleo.* *En UNIX tampoco sería posible en el caso hipotético de que el código del sistema operativo estuviese siempre cargado en memoria principal. Al accionar el interruptor se ejecuta un programa que reside en memoria ROM, cuya misión es cargar en la memoria RAM otro programa con capacidad para localizar en el disco duro el código del núcleo del sistema operativo, llevarlo a memoria y posteriormente cederle el control. A partir de este momento el núcleo se encarga de inicializar los diferentes drivers, y crear un primer proceso que será el encargado de crear el resto de los procesos. La creación de procesos sólo se puede llevar a cabo en modo núcleo.*

Corrección: B/M (considerar excepcional un 0.5 → Sabe de que va pero no estáis seguros por como lo expone)

---

3) El siguiente código corresponde al archivo ejecutable generado con el nombre "Ejemplo1".

```
    /*** Ejemplo1***/
 1  #include "todas_las_cabeceras_necesarias.h"
 2
 3  main()
 4  { int i=0;
 5    pid_t pid, pid2;
 6
 7    while (i<2)
 8    { pid=fork()
 9       switch(pid)
10        {case (-1):{printf("Error creando hijo\n");
11                    break;}
```

```
12          case (0):{pid2=fork();
13                    printf("Hijo %i creado\n",i);
14                    sleep(10);
15                    exit(0);
16                   }
17          default: {printf("Padre\n");
18                    sleep(5);}
19        }
20     i++;
21    }
22    exit(0);
23  }
```

Suponga que "Ejemplo1" se ejecuta correctamente:

c) Indique de forma razonada, el número de procesos que creará y dibuje el árbol de procesos generado.

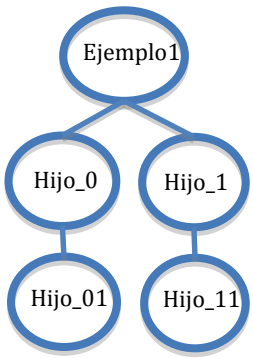
d) Indique de forma justificada, si pueden producirse procesos zombies y/o huérfanos.

1.5 puntos

3

a)

Se crean un total de 5 procesos al ejecutar el código ejemplo1.
El proceso Ejemplo1 crea dos procesos hijos: con la llamada pid=fork() e i=0 crea Hijo0 a continuación ejecuta la opción "default" e incrementa i, vuelve a repetir el bucle y con i=1 crea otro proceso Hijo1.
Además tanto el Hijo 0 como Hijo1 se convierten en padres ya que ellos ejecutan el código de la opción "case(0)" dentro del switch, donde aparece otra llamada fork(), a estos hijos ejecutan y sus padre ejecutan exit() y finalizan.



b)

En el código Ejemplo1, no aparece la llamada wait(). Los procesos padres no esperan a sus procesos hijos y por tanto podrían quedar procesos huérfanos, si los procesos padre finalizan su ejecución antes que sus hijos.
sleep() hace que un proceso se suspendan voluntariamente.
El proceso inicial Ejemplo1 se suspende 5 segundos (sleep(5)) después de crear al Hijo_0 y otros 5 segundos después de crear a su Hijo_1, en total 10 segundos. Hijo_0 e Hijo_1 se suspende 10 segundos (sleep(10)), existe una alta probabilidad de que Ejemplo1 finalice su ejecución antes de que Hijo_1 acabe. Por tanto Hijo_1 queda huérfano y es adoptado por init().

Es bastante improbable que haya procesos zombies, ya que los procesos padres (Ejemplo1, Hijo_0, Hijo_1) aparentemente finalizan antes que sus hijos.

CRITERIO
Cada apartado ½ de la nota de la pregunta
Apartado a) 0.75 Bien, 0.35 número explicación+0.4 por el diagrama.
Apartado b) Toda la nota del apartado si correcto

4) Indique de forma justificada si son ciertas o falsas las siguientes afirmaciones sobre los estados de los procesos:

d) Sólo los procesos que están en *Ejecución* pueden pasar al estado *Terminado*, una vez finaliza la ejecución de sus instrucciones.

e) El estado *Suspendido* sólo puede ser alcanzado por aquellos procesos que hayan solicitado o estén realizando una operación de E/S bloqueante

f) El cambio de estado de en Ejecución al de *Preparado* sólo es posible en sistemas con planificadores expulsivos

1.0 puntos

4

a)
FALSA
Desde el estado de Preparado y Suspendido un proceso puede pasar a Terminado si se le envía una señal. Un proceso puede finalizar de forma anormal mediante la señal SIGKILL. Los procesos reciben las señales incluso en estado suspendido, la señal SIGKILL no se puede enmascarar.

b)
FALSA
Los procesos pueden pasar al estado suspendido al ejecutar una instrucción bloqueante como wait()o sleep(). También pasan a SUSPENDIDO cuando solicitan un recurso SW que se encuentra ocupado (sem_wait, pthread_mutex_lock)

c)
CIERTA
Con planificadores expulsivos cuando un proceso es expulsado de la CPU, por otro mas prioritario, antes de finalizar su ráfaga de CPU, se inserta en la cola de preparados. En planificadores no expulsivos los procesos sólo abandonan la CPU para ir a los estados de acabado o suspendido.

CRITERIO
Cada apartado 1/3 de la nota de la pregunta sólo B/M

---

5) Un sistema dispone de un planificador multicolas a corto plazo (PCP) con tres colas Cola0, Cola1 y Cola2, cuyos algoritmos de planificación son **RR con q=1, SRTF, y FCFS**, respectivamente.
La planificación entre **colas es gestionada con prioridades expulsivas** siendo la más prioritaria la Cola2 y la menos prioritario la Cola0. Cada proceso dispone de un contador de promoción (ContPro) que el sistema mantiene para establecer su promoción entre colas, de manera que los procesos puedan alcanzar colas más prioritarias. Cada vez que un proceso **pasa al estado suspendido** su ContPro se incrementa en 1 (ContPro= ContPro+1). Un proceso es ubicado en la Cola0 si su ContPro es igual a 0, en la Cola1 si su ContPro=1 y en la Cola2 si ContPro>=2. Los procesos que llegan al sistema se les asigna un ContPro=0 y van a la Cola0.
Suponga que las operaciones de E/S se efectúan sobre el mismo dispositivo gestionado con FCFS y que al sistema llegan los procesos mostrados en la tabla:

Proceso	Perfil de ejecución	Instante de llegada	ContPro
A	2 CPU + 2 E/S + 1 CPU + 4 E/S + 1 CPU	0	0
B	1 CPU + 2 E/S + 4 CPU + 2 E/S + 2 CPU	1	0
C	3 CPU + 1 E/S + 1 CPU	2	0
D	2 CPU + 1 E/S+ 1CPU	3	0

c) Rellene la siguiente tabla indicando en cada instante de tiempo donde se encuentran los procesos.
d) Indique la utilización de CPU para esta carga y los tiempos medios de espera y retorno.

2.0 puntos (1.25+0.75)

5a

T	Cola 0 RR q=1	Cola 1 SRTF	Cola 2 FCFS	CPU	Cola E/S	E/S	Evento
0	A			A			Llega A , ContPro(A)=0

	1	A B			B			Llega B, ContPro(B)=0
	2	CA			A		B	LlegaC, ContPro(C)=0 ContPro(B)=1
	3	D C			C	A	B	Llega D
	4	CD	B		B		A	ContPro(A)=1
	5	CD			B		A	
	6	CD	B A		A			
	7	CD	B		B		A	ContPro(A)=2
	8	CD			B		A	
	9	CD			D	B	A	ContPro(B)=2
	10	D C			C	B	A	
	11	CD		A	A		B	
	12	CD			D		B	FIN A
	13	C		B	B		D	ContPro(D)=1
	14	C	D		B			
	15	C			D			FIN B
	16	C			C			FIN D
	17				---		C	ContPro(C)=1
	18				C			
	19							FIN C
	20							
	21							
	22							
	23							
	24							

5b	Tiempo medio de espera= 1+1+11+9 /4 =22/4 =5.5 Tiempo medio de retorno= 12+15+19+16/4 = 62/4 =15.5 Utilización de CPU = 18/19

CRITERIO

5ª) Todo bien 1.25. Todo bien con un error no conceptual y puntual 1.1 puntos (por ejemplo en E/S)
0.25 puntos →solo bien hasta t=4, Hacen bien RR
0.75 puntos→ bien hasta t=9 Hacen bien RR, SRTF e intercolas
1.25 puntos →bien hasta t=19 Hacen bien RR,SRTF, FCFS e intercolas

5b) 0.25 cada tiempo.
Se corrige en función de la tabla que ha desarrollado, siempre que sea razonable. En este caso 0.2 por cada tiempo

---

6) Dado el siguiente código cuyo archivo ejecutable ha sido generado con el nombre "Ejemplo2".

```
1  /*** Ejemplo2***/
2  #include <stdio.h>
3  #include <pthread.h>
4
5  void *fun_hilo( void *ptr )
6  { int sec;
```

```
 7
 8    sec=(int)ptr;
 9    sleep(sec);
10    printf("Yo he esperado %d segundos\n",sec);
11  }
12
13  int main()
14  {
15    pthread_attr_t atrib;
16    pthread_t hilo1, hilo2, hilo3;
17
18    pthread_attr_init( &atrib );
19    pthread_create( &hilo1, &atrib, fun_hilo, (void *)30);
20    pthread_create( &hilo2, &atrib, fun_hilo, (void *)1);
21    pthread_create( &hilo3, &atrib, fun_hilo, (void *)10);
22    pthread_join( hilo3, NULL)
23  }
```

Indique las cadenas que imprime el programa en la terminal tras su ejecución. Justifique su respuesta

1.0 puntos

6

Yo he esperado 1 segundos
Yo he esperado 30 segundos

El hilo main únicamente espera con join() a hilo3. Por tanto el main terminara inmediatamente después de que termine el hilo3. Cuando el hilo main acaba se liberan todos los recursos del proceso, memoria principal, etc. y por tanto finalizan todos los hilos, independientemente de si han acabado de ejecutar todas sus instrucciones o no.
Dada la diferencia de tiempos en los sleep() que ejecutan los hilos, hilo3 finaliza antes que hilo1, por lo que nunca saldrá el mensaje del hilo1 que se suspende durante 30 segundos.

CRITERIO
0.25→ sólo join(). Saben que en main solo hay un join y puede dar problemas, pero no lo tienen bien
0.25→ Paso de Parámetros→ Han escrito los tres mensajes con PARAMETROS correctos
0.25→ detecten que al terminar el hilo3 finaliza main, abortando cualquier hilo que esté en marcha,

---

7) El siguiente código C corresponde a un proceso con dos hilos que comparten memoria y se han de sincronizar utilizando el **mecanismo de espera activa**. Para diseñar su **protocolo de entrada y salida** a la sección crítica utilizan las variables compartidas **flag** y **turn** declaradas.

```
 1  //***** Ejemplo 3 **//
 2  #include <pthread.h>
 3  …
 4  int flag[2];
 5  int turn;
 6
 7  void* thread(void* id)
 8  { int i;
 9    i = (int)id;
10
11   while ( 1 ) {
12
```

```
13      remaining_section();
14
15      /**b)Asigne valores a flag y a turn**/
16      while(/**c)Condición del Protocolo de Entrada**/);
17
18      critical_section();
19
20      flag[i] = 0;
21    }
22  }
23  int main() {
24    pthread_t th0, th1;
25    pthread_attr_t atrib;
26    pthread_attr_init(&atrib);
27    /**a)Inicialice la variable flag **/
28    pthread_create(&th0, &atrib, thread, (void *)0);
29    pthread_create(&th1, &atrib, thread, (void *)1);
30    pthread_join(th0, NULL);
31    pthread_join(th1, NULL);
32  }
```

Se pide:

a) Reemplace "`/**a)Inicialice la variable flag **/`" por el código C correspondiente.

b) Reemplace "`/**b)Asigne valores a flag[i] y a turn**/`" por el código C correspondiente.

c) Reemplace "`/**c)Condicion del protocolo de entrada**/`" por el código C correspondiente.

1.0 puntos

7

a)

```
flag[0]=0;  flag[1]=0;
```

b)

```
flag[i] = 1;
turn =(i+1)%2;
```

c)

```
while (flag[(i+1)%2]==1) && (turn==(i+1)%2);
```

CRITERIO

a) 0.2 b) 0.4 c) 0.4 ; Creo que lo dejaran en blanco o harán burradas

---

8) Indique todos los posibles valores que puede alcanzar la variable x tras la ejecución concurrente de los procesos A, B y C. Justifique su respuesta indicando para cada uno de valores el orden de ejecución de las distintas secciones.

// Variables compartidas
int x=1;
Semáforos S1=3, S2=0, S3=0;

// Proceso A	//Proceso B	//Proceso C

P(S1)
P(S3)
x = 2*x + 1; // sección 1
V(S2)
	P(S1)
	P(S2)
	x = x*3; // sección 2
	V(S1)
		P(S1)
		x = x + 2; // sección 3
		V(S3)
		P(S1)
		x = x + 3; // sección 4

1.0 puntos

8

Existen tres posibles ordenes de ejecución que son:
X=1
C P(S1), sección3, V(S3) -> A P(S1)P(S3) sección1, V(S2) -> B P(S1), P(S2) sección2 , V(S1) -> C P(S1) sección4 → valor de x=24
*x=3 → x= 3*2 +1 =7 → x= x*3= 21 → x= x+3=24*

x=1
C P(S1),sección3,V(S3), P(S1), sección4 → A P(S1), P(S3), sección1 → Proceso B suspendido en el semáforo S1
x=3 →x=6 → x=13

x=1
C P(S1), sección3;V(S3)→ A P(S1), P(S3), sección1, V(S2)→ C P(S1)sección4 → Proceso B suspendido en el semáforo S1
x=3 →x=7 → x=10

CRITERIO
Deben detectar al menos los dos posibles órdenes de ejecución con→ 0.4 puntoscada orden correcto.
1.0 punto para el que detecte los tres

---

9) Complete en "Ejemplo4" el código de las funciones fth_UNO y fth_DOS, con las operaciones sobre **semáforos** necesarias, para que al ejecutarlo muestre por pantalla de manera ordenada, los diez primeros números enteros, es decir, " 0 1 2 3 4 5 6 7 8 9" .

```
 1  /*** Ejemplo4***/
 2  #include <stdio.h>
 3  #include <pthread.h>
 4
 5  void *fth_UNO( void *ptr )
 6  { int i;
 7    for (i=1; i<10; i+=2)
 8    {
 9      printf("%d ",i);
10
11    }
12  }

13  void *fth_DOS( void *ptr )
14  { int i;
15  for (i=0; i<10; i+=2)
16    {
17      printf("%d ",i);
18
19    }
20  }

20  int main()
21  { pthread_attr_t atrib;
22    pthread_t th1, th2;
23
24    pthread_attr_init( &atrib );
25    pthread_create(&th1,&atrib,fth_UNO, NULL);
26    pthread_create(&th2,&atrib,fth_DOS, NULL);
27    pthread_join( th1, NULL);
28    pthread_join( th2, NULL);
29    printf ("\n");
30  }
```

El hilo "th1" debe mostrar los números impares y "th2" los pares. Para sincronizar la ejecución de los hilos utilice tantos semáforos como sea necesario y diga su valor de inicialización. Utilice la nomenclatura P y V.

1.0 puntos

9

Semáforo: sinc, smutex;
sinc=0; smutex=1;

```
void *fth_UNO( void *ptr )
{ int i;
  for (i=1; i<10; i+=2)
  { P(sinc);
    P(smutex);
    printf("%d ",i);
    V(smutex);
  }
}
```

```
void *fth_DOS( void *ptr )
{ int i;
for (i=0; i<10; i+=2)
  { P(smutex);
    printf("%d ",i);
   V(smutex);
   V(sinc);
  }
}
```

CRITERIO:
0.5 Sincronización y 0.5 exclusión mutua
0.25 Inicializan semáforos correctamente y después se lian
No ser estricto con la nomenclatura, pueden usar POSIX. Las operaciones deben ser correctas para valorar con 1.0puntos.