APELLIDOS		NOMBRE		Grupo
DNI		Firma		

- **No desgrape las hojas.**
- **Conteste exclusivamente en el espacio reservado para ello.**
- **Utilice letra clara y legible. Responda de forma breve y precisa.**
- **El examen consta de 7 cuestiones, cuya valoración se indica en cada una de ellas.**

1. Conteste de forma breve y justificada a las siguientes cuestiones:

(1.4 puntos= 4 x 0.35)

1	a) Indique a qué se hace referencia con concurrencia entre CPU y E/S. Al hecho de que puede existir un proceso ejecutándose en la CPU y simultáneamente otro u otros procesos pueden estar siendo atendidos por los dispositivos de E/S. Estos elementos hardware pueden operar en paralelo.
	b) Justifique por qué los procesadores tienen al menos dos modos de ejecución y describa cuáles son. Los procesadores que dan soporte a los Sistemas Operativos tienen que proporcionar al menos dos modos de funcionamiento. En uno de ellos se pueden ejecutar todas las instrucciones disponibles y suele llamarse modo núcleo o protegido. En el otro modo ciertas instrucciones están restringidas (control interrupciones, acceso a diversas área de memoria, ciertos registros del procesador …), es el llamado modo usuario o regular. Esto permite al Sistema Operativo tener un control total de la máquina al ejecutarse en modo protegido, restringiendo el acceso al hardware a los programas de usuario que se ejecutan en modo regular.
	c) Ordene en dos listas, en función de si forman parte del núcleo del sistema operativo o no, los siguientes elementos: ***gestor procesos*** / ***shell*** / ***gestor memoria*** / ***manejador dispositivos*** / ***comando ls*** / ***interfaz de llamadas al sistema*** / ***navegador internet***. ***Parte del Núcleo SO***: gestor de procesos / gestor de memoria / manejador dispositivos / interfaz de llamadas al sistema ***No Núcleo SO***: shell / comando ls / navegador internet
	d) Defina el concepto de utilización de CPU y exponga mediante expresión matemática cómo se calcula. La utilización de la CPU en un determinado periodo de tiempo T se define como el porcentaje de tiempo que dicha CPU ha estado ejecutando instrucciones. Siendo T el periodo de tiempo estudiado, y sea T_e el periodo de tiempo que se han estado ejecutado instrucciones (CPU ocupada), la fórmula para su cálculo es: $U = T_e / T$

2. Considere que en el directorio actual de trabajo existe un archivo "hello.txt", que contiene el texto "hello\n", de forma que la orden $cat hello.txt imprime una línea con la palabra hello. Suponga que se compila y ejecuta el siguiente programa, para cada uno de los valores X=1, 2, 3 y 4. Exponga para cada uno de los 4 casos (valores de X), qué se muestra en el terminal al ejecutar el programa y justifique cada respuest.

```
 1 #include <...all headers...>
 2 #define X 1    //1,2,3,4
 3 int main(int argc, char *argv[])
 4 { int val= 0;
 5   int parent_pid= getpid();
 6     if (X>=3) val=fork();
 7     if (val==0){
 8       if (X%2==1) //X impar
 9           execl("/bin/cat", "cat", "hello.txt", NULL);
10       else
11           execl("/cat/bin", "cat", "hello.txt", NULL);
12     }
13     if (getpid()==parent_pid)
14       printf("parent\n");
15     else  printf("child\n");
16     return 0;
17 }
```

(1.4 punto=0,35+0,35+0,35+0,35**)**

a) Caso X=1 (#define X 1)

`hello`

No se crean procesos hijos, ya que X=1→ X<3. Dado que la variable val está inicializada a 0 se ejecuta con éxito la llamada `execl("/bin/cat", "cat", "hello.txt", NULL);` y por ende el programa cat , mostrando en el terminal el contenido del archivo "hello.txt". La llamada execl() no retorna.

b) Caso X=2 (#define X 2)

`parent`

No se crean procesos hijos, ya que X=2→ X<3. Dado que la variable val está inicializada a 0 y X es par se ejecuta `execl("/cat/bin", "cat", "hello.txt", NULL);` , la ejecución de cat falla porque la ruta no es correcta. La llamada execl retorna con error -1. El proceso imprime finalmente "parent".

c) Caso X=3 (#define X 3)

`hello`
`parent`

Se crea un proceso hijo, ya que X=3→ X>=3. El proceso hijo ejecuta la llamada `execl("/bin/cat", "cat", "hello.txt", NULL;` y por ende el programa cat con éxito, mostrando en el terminal el contenido del archivo "hello.txt" . La llamada execl no retorna. Sólo el padre alcanza la línea 14 e imprime "parent".

d) Caso X=4 (#define X 4)

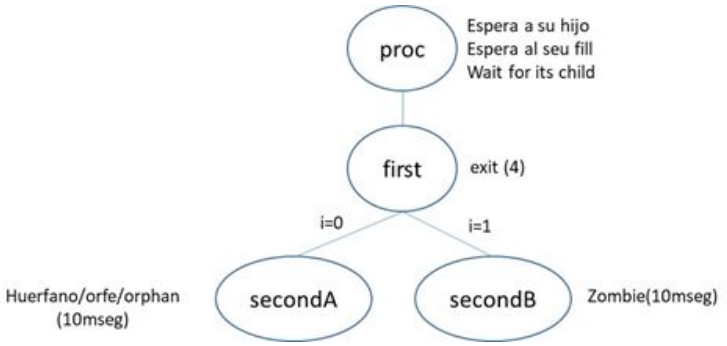
`parent`
`child`

Se crea un proceso hijo, ya que X=4→ X>=3. El proceso hijo ejecuta la llamada `execl("/cat/bin", "cat", "hello.txt", NULL);` La ejecución de cat falla porque la ruta no es correcta. La llamada execl retorna con error -1. Ambos, padre e hijo, alcanzan la línea 13..

3. Complete el código propuesto para *proc.c,* cuyo ejecutable denominaremos ***proc,*** de manera que:

- Al ejecutar ***proc*** se cree un proceso ***first*** que a su vez crea dos hijos ***secondA*** y ***secondB***.
- El proceso ***proc*** debe esperar a su hijo ***first*** y ***first*** debe finalizar con ***exit(4)***.
- Los procesos ***secondA*** y ***secondB***, debe quedarse huérfano y zombi respectivamente durante al menos un tiempo de 10seg. Para ello se le sugiere utilizar ***sleep(),*** con ***sleep(10)***, ***sleep(20)*** y ***sleep(30),*** cuando lo considere adecuado.

El esquema siguiente muestra el parentesco y las acciones de los diferentes procesos.



(1.2 punto)

3.

```
//Programa proc.c
#include <.....los necesarios.h…..>
int main(int argc, char *argv[])
{ int val1, val2, status;
  int i, pid;
   printf("Proceso proc\n");
     val1=fork();

    if (val1==0){
        printf("First\n");
     for (i=0; i<2; i++)
     { printf("Creating Childs\n");
         val2=fork();
         if (val2==0)
         {
             if (i==0)
               {printf("SecondA: Orphan\n");
                   sleep(30);
                   exit(0);}
             if(i==1)
               { printf("SecondB: zombie\n");
                sleep(10);
                exit(0);}
         }
     }//for
       sleep(20);
       exit(4);
    } //if val1

      while ((pid=wait(&status)) >0)
        printf("Child waited %d, status %d\n", pid, status/256);
    exit(0);
}
```

4. En un sistema de tiempo compartido se tiene un único dispositivo de E/S que se gestiona con FCFS. A dicho sistema llegan 3 procesos A, B y C, cuyos instantes de llegada, prioridad (siendo 1 la prioridad más alta, 3 la menor) y esquema de solicitud de ráfagas de CPU y E/S es el siguiente:

Proceso	Llegada	Prioridad	Ráfagas de CPU y E/S
A	0	3 (-)	4 CPU + 1 E/S + 3 CPU + 1 E/S + 4 CPU
B	1	2	2 CPU + 2 E/S + 3 CPU + 1 E/S + 3 CPU
C	2	1 (+)	1 CPU + 5 E/S + 1 CPU + 5 E/S + 1 CPU

(2.0 puntos =1.1+ 0.3+0.3+0.4)

4 a) Represente mediante diagrama temporal la ocupación de CPU, del periférico de E/S y de la cola de preparado para un planificador basado **en prioridades expulsivo**. En cada instante de tiempo T ponga entre paréntesis (ej. (A(2)), unidades de tiempo de CPU o E/S pendientes para finalizar dicha ráfaga.

T	Preparados	CPU	Cola E/S-->	E/S	Evento
0		A(3)			Llega A
1	A	B(1)			Llega B
2	B,A	C(0)			Llega C
3	A	B(0)		C(4)	
4		A(2)	B	C(3)	
5		A(1)	B	C(2)	
6		A(0)	B	C(1)	
7			A,B	C(0)	
8		C(0)	A	B(1)	
9			C,A	B(0)	
10		B(2)	C	A(0)	
11	A	B(1)		C(4)	
12	A	B(0)		C(3)	
13		A(2)	B	C(2)	
14		A(1)	B	C(1)	
15		A(0)	B	C(0)	
16		C(0)	A	B(0)	
17		B(2)		A(0)	Fin C
18	A	B(1)			
19	A	B(0)			
20		A(3)			Fin B
21		A(2)			
22		A(1)			
23		A(0)			
24					Fin A
25					-

4 b) Indique los tiempos de espera (de CPU) y de retorno de cada proceso (**Tabla PRIO_1**)

PRIO_1	T.espera	T.retorno
A	**7**	24-0 = **24**
B	**1**	20-1 = **19**
C	**0**	17-2 = **15**

4 c) Utilizando el mismo planificador de prioridades expulsivas y carga se obtiene los siguiente valores de tiempos de espera y de retorno. Determine qué prioridades se han asignado a los procesos para obtener estos valores. (**Tabla PRIO_2**)

PRIO_2	T.espera	T.retorno
A	0	13
B	7	26
C	14	28

Hay que invertir el orden de prioridades: A = 1; B = 2, C = 3

4 d) La siguiente tabla muestra los resultados obtenidos para la misma carga utilizando un planificador Round-Robin con quantum q=2ut. (Tabla **RR**)

RR	T.espera	T.retorno
A	8	22
B	8	19
C	6	21

A partir de las tablas PRIO_1, PRIO_2 y RR, calcule la tasa de rendimiento (productividad) para cada uno de las tres planificaciones,

Tasa de rendimiento= Número de procesos por unidad de tiempo

PRIO1 = 3/24 = 0,125;

PRIO2 = 3/30 = 0,1; (*)

RR = 3/23 = 0,13;

(*) Nota: el Proc C tiene un t.retorno de 28, pero empieza en el tiempo 2, por lo que el tiempo total es 30. Para RR será 21+2=23

Indique a partir de las tablas PRIO_1, PRIO_2 y RR qué tipo de procesos, limitados por CPU o E/S, son priorizados en cada planificación..

Procesos priorizados cada planificador

PRIO1: A los procesos limitados por E/S, es decir que usan más E/S que CPU.

PRIO2: A los procesos limitados por CPU

RR: Hay un reparto equitativo de la CPU, y por lo tanto no hay una priorización por tipo de proceso

5. **D**ado el siguiente programa ***Hilos.c*** cuyo ejecutable es ***Hilos***.

(1.5 puntos=0.6 +0.3+0.3+0.3)

```
#include <... all headers...>
#define NTHREADS 3
pthread_t Th[NTHREADS];
pthread_attr_t atrib;
int N=0;

void *Func(void *arg)
{ int i= (int) arg;
  N=N+i;
  printf("Hilo %d,N = %d\n",i ,N);
  pthread_exit(0);
}
```

```
int main (int argc, char *argv[]) {
  pid_t pid;
  int i;
  printf("START MAIN \n");
  pthread_attr_init(&atrib);

  pid=fork();
  for (i=0;i<NTHREADS;i++)
   {pthread_create(&Th[i],&atrib, Func,(void *)i);
    pthread_join(Th[i],NULL); }

  if (pid == 0) pthread_exit(0);
  else exit(0);
  printf("END MAIN \n");
}
```

5 **a)** Indique la secuencia (una de las posibles) que imprime el programa en la Terminal al ejecutarlo. Justifique su respuesta.

START MAIN Hilo 0,N=0 Hilo 1,N=1 Hilo 2,N=3 Hilo 0,N=0 Hilo 1,N=1 Hilo 2,N=3	El proceso *Hilos* comienza y muestra en pantalla "START MAIN", al ejecutar pid=fork() se crea un proceso hijo. Padre e hijo continúan su ejecución de forma concurrente con la siguiente instrucción al fork(), ejecutando el bucle for(i=0,...) y la llamada pthread_create() y pthread_join(). Cada proceso tendrá su propia imagen de memoria y por tanto sus propias variables globales. Los hilos creados por cada uno de los procesos se lanzan de forma secuencial , ya que hasta que no finaliza un hilo no se crea el siguiente y ejecutan la función *Func*.

Es por esto que todos los hilos finalizan correctamente. Los dos procesos se ejecutan concurrentemente por lo que lo que imprime cada uno de ellos podría intercalarse con lo que imprime el otro. El proceso hijo finaliza hilo main con pthread_exit(), mientras el proceso padre hace exit(0), por lo tanto ninguno de ellos imprime "END MAIN".

b) Indique el número máximo de hilos del programa ***Hilos*** que podrían estar ejecutándose concurrentemente. Justifique su respuesta.

El lanzamiento de los hilos por parte de cada proceso se realiza secuencialmente, ya que cada vez que se crea un hilo, el hilo main() queda suspendido en pthread_join() hasta que ese hilo finaliza. Los dos procesos se ejecutan concurrentemente y mientras que cada proceso ejecuta el hilo main y uno de los hilos creados dentro del bucle de forma concurrente. Por tanto, el máximo número de hilos que se ejecutarán concurrentemente será de 4, 2 por cada proceso.

c) Indique si existe riesgo de condición de carrera al ejecutar ***Hilos***. Justifique adecuadamente su respuesta.

No existe riesgo de condición de carrera ya que el acceso a la variable global N por parte de cada uno de los hilos se realiza de forma secuencial. Y en cuanto a los dos procesos generados cada uno de ellos tiene una copia propia de la variable N.

d) Suponga que los hilos de ***Hilos*** se soporten a nivel usuario (por el *runtime*), e indique qué debe gestionar en dicho caso el planificador a corto plazo del sistema. Justifique su respuesta.

Si los hilos no son soportados por el Sistema Operativo, el planificador a corto plazo que gestiona el uso de la CPU sólo verá dos procesos compitiendo por la CPU. Cuando la CPU se asignada por el planificador a corto plazo a un proceso, el runtime se encarga de decidir a qué hilo de dicho proceso se le asigna la CPU.

6. En un computador con una sola CPU dotado de un sistema operativo con planificador de *turno rotatorio (RR)*, se desea resolver el problema de acceso a sección crítica de **dos** hilos utilizando variantes de *test_and_set()*. Indique **de forma justificada** para cada caso particular propuesto si se cumple la condición de espera limitada y anilice si puede ser considerada como espera activa o espera no activa:

(1.2 puntos=0.4+0.4+0.4)

6

a)

Protocolo de entrada:	Protocolo de salida:
`while (test_and_set(&llave)) /*bucle vacio*/ ;`	`llave = 0;`

Espera limitada

No se cumple espera limitada. Esta solución no garantiza que el número de veces que un proceso puede acceder a su SC (sección crítica), desde que otro proceso solicita el acceso a SC, está limitado. Cuando un hilo finaliza SC, ejecuta el protocolo de salida (llave=0) y por tanto podría volver a entrar en SC mientras le quede tiempo de CPU, es decir, antes de que venza su quantum. Por tanto, cuando finaliza su quantum puedes estar de nuevo dentro de la SC, impidiendo que otros proceso puedan entrar a SC (llave=1) a pesar de disponer del recurso CPU. Esto se podría repetir de forma indefinida.

Espera activa o no activa

Espera ACTIVA. El hilo espera en un bucle consumiendo CPU (todo el quantum que tiene asignado), cuando llave es igual a 1.

b)

Protocolo de entrada:	Protocolo de salida:
`while (test_and_set(&llave)) usleep(100) ;`	`llave = 0;`

Espera limitada

No cumple espera limitada. Si se produce un cambio de contexto estando un hilo en SC, el segundo hilo no podrá acceder a su SC y se suspenderá y cuando se despierte no hay garantía de poder entrar a SC. Esto se podría repetir un número indeterminado de veces.

Espera activa o no activa

Espera NO ACTIVA. El hilo que no puede acceder a SC no consume CPU, ya que ejecuta usleep y se suspende dejando que el resto de procesos puedan hacer uso de la CPU.

c)

Protocolo de entrada:	Protocolo de salida:
`while (test_and_set(&llave)) usleep(100);`	`llave = 0;` `usleep(105);`

Espera limitada

Si cumple espera limitada. Si un hilo está en SC, otro hilo quedará suspendido por 100 useg al ejecutar el protocolo de entrada. Al salir el hilo de SC se suspende por 105 useg, lo que obliga al planificador RR a asignar la CPU al otro proceso. Con usleep(105) el hilo que termina de ejecutar SC estará más tiempo suspendido que él que haya sido suspendido previamente en el usleep(100), garantizado así los cambios de contexto fuera de la SC. Esta solución siempre suspende a un hilo tras finalizar SC alargando su tiempo de ejecución innecesariamente..

Espera activa o no activa

Espera NO ACTIVA. El hilo que no puede acceder a SC no consume CPU, ya que ejecuta usleep y se suspende dejando que el resto de procesos puedan hacer uso de la CPU.

7. Describa una posible secuencia de la ejecución concurrente de los hilos ThA, ThB y ThC

```
//valores iniciales de las variables
 int x=0, y=0;
 Semaphore: S1=0, S2=1, S3=0;
```

ThA	**ThB**	**ThC**

P(S3);
P(S2);
x = x + 1;
y = 2*x + y;
V(S2);
V(S1);
	P(S1);
	P(S2);
	x = 2*x;
	y = x + y;
	V(S2);
		P(S2);
		x = x - 2;
		y = x - y:
		V(S2);
		V(S3);

Utilice la tabla siguiente indicando una operación y el hilo al que corresponde en cada línea, así como el valor de las variables y semáforos tras cada operación.

(1.3 puntos)

7

Inicio	ThX-Operación	S1=0	S2=1	S3=0	X=0	Y=0
1	ThA--> P(S3)--> susp A			-1, A		
2	ThB-->P(S1)-->susp B	-1, B				
3	ThC-->P(S2)		0			
4	ThC-->x=x-2=-2				-2	
5	ThC-->y=x-y=-2-0					-2
6	ThC-->V(s2)		1			
7	ThC--> V(S3)			0		
8	ThA-->P(S2)		0			
9	ThA-->X=X+1=-2+1=-1				-1	
10	ThA-->y=2*x+y-2-2=-4					-4
11	ThA-->V(S2)		1			
12	ThA-->V(S1)	0				
13	ThB-->P(S2)		0			
14	ThB-->X=2*x=-2				-2	
15	ThB-->Y=x+y=-2-4=-6					-6
16	ThB-->V(S2)		1			
Valores Finales de las variables		0	1	0	-2	-6