
 <p>Universidad Carlos III de Madrid</p>	<p><b>Departamento de Informática</b> <b>Grado en Ingeniería Informática</b> <b>Sistemas Operativos</b></p> <p><b>Prueba de Evaluación Continua</b> <b>18 de octubre de 2010</b></p>	
---	--	---

**ATENCIÓN:**

- Lea atentamente todo el enunciado antes de comenzar a contestar.
- Dispone de **90 minutos** para realizar la prueba.
- No se podrán utilizar libros ni apuntes, ni calculadoras de ningún tipo.
- Los teléfonos móviles deberán permanecer desconectados durante la prueba (apagados, no silenciados).
- Solamente se corregirán los ejercicios contestados con bolígrafo. Por favor no utilice lápiz.

---

**APELLIDOS:**



**NOMBRE:**

**NIA:**

**GRUPO:**

**Teoría (4 puntos)**

- **[1 punto]** Indique las ventajas e inconvenientes de un sistema operativo monolítico.
  - **Ventajas: Mayor eficiencia en el uso de los servicios del SO**
  - **Inconvenientes: Mayor dificultad en el desarrollo y mantenimiento del SO.**
  
- **[1 punto]** Indique las fases necesarias para el arranque de un SO
  - **Iniciador ROM**
  - **Cargador de SO en memoria**
  - **Ejecución de la parte residente de SO**
  - **Fase normal de ejecución del SO**

 <p>Universidad Carlos III de Madrid</p>	<p>Departamento de Informática Grado en Ingeniería Informática Sistemas Operativos</p> <p>Prueba de Evaluación Continua 18 de octubre de 2010</p>	
---	---	---

- **[1 punto]** Describa en que consiste la técnica *Copy-on-Write*

*Copy-on-Write* es una técnica que retrasa o evita la copia de los datos al hacer el fork

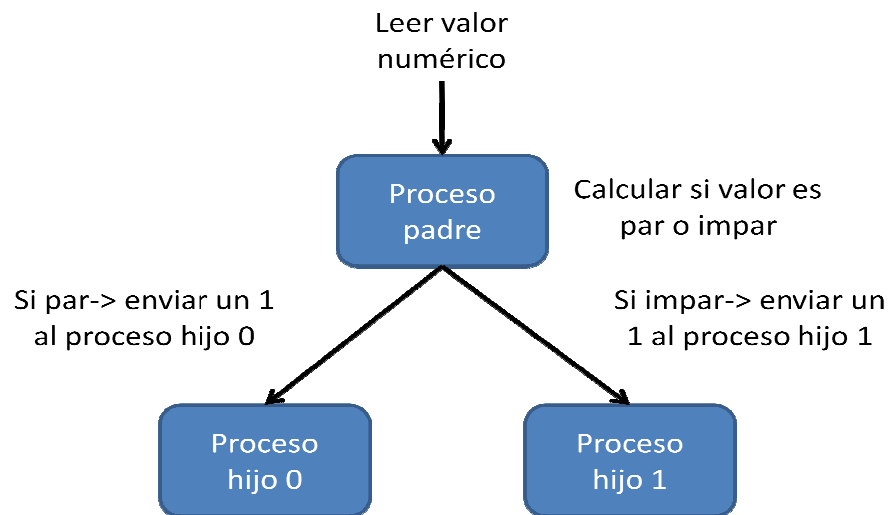
Los datos se marcan de manera que si se intentan modificar se realiza una copia para cada proceso (padre e hijo)

Ahora *fork()* sólo copia la tabla de páginas del padre (no las páginas) y crea un nuevo BCP para el hijo

- **[1 punto]** Indique de que partes básicas se compone un proceso ligero y que comparte con el resto de los procesos ligeros.
  - Cada hilo dispone de:
    - ▣ Identificador de thread
    - ▣ Contador de programa
    - ▣ Conjunto de registros
    - ▣ Pila
  - Comparten con el resto de hilos del proceso:
    - ▣ Mapa de memoria (sección de código, sección de datos, shmem)
    - ▣ Ficheros abiertos
    - ▣ Señales, semáforos y temporizadores.

**Ejercicio 1** [3 puntos]:

**a)[1 punto]** Se pretende codificar un programa (en lenguaje C), que genere dos procesos hijos, siguiendo el siguiente esquema:



1. El proceso principal debe leer por teclado un número entero, y calcular si el valor es par o impar. Si el valor es 0, ir al **punto 5**.
2. Si el valor:
  - es par, el proceso padre debe enviar un 1 al proceso hijo 0
  - en caso contrario, enviará un 1 al proceso hijo 1.
3. El hijo que reciba la comunicación sumará 1 a un contador interno de números.
4. Volver al **punto 1**.
5. Enviar un 0 a ambos hijos para que terminen.

Complete el programa que aparece a continuación para que el programa funcione como se ha establecido.

En las comunicaciones, se utilizarán tuberías.



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

**(1)**

```
int impares, pares, valor, valor1;
```

```
void hijo0(){
```

```
    int valor;
```

```
    pares=-1;
```

**(5)**

```
    do{
```

**(6)**

```
        par++;
```

```
    }while(valor !=0);
```

**(7)**

```
    printf("pares %d\n", pares);
```

**(8)**

```
    exit(0);
```

```
}
```

```
void hijo1(){
```

```
    int valor;
```

```
    impares=-1;
```

**(2)**

```
    do{
```

**(3)**

```
        impar++;
```

```
    }while(valor !=0);
```

```
    printf("impares %d\n",
```

```
    impares);
```

**(4)**

```
    exit(0);
```

```
}
```

```
int main (int argc, char *argv[]) {
    impares = pares = 0;
```

**(9)**

```
    if (((10)) /* codigo del hijo */
```

```
        hijo0();
```

```
    }else{
```

```
        if (((11)) { /* codigo del hijo */
```

```
            hijo1();
```

```
        }else{
```

**(12)**

```
            while (0 != scanf("%d",&valor1)){
```

```
                if (valor1 == 0)
```

```
                    break;
```

```
                valor = 1;
```

```
                if (valor1%2 != 0){
```

**(13)**

```
                    }else{
```

**(14)**

```
                }
```

```
            }
```

```
            valor = 0;
```



**(15)**



```
        }
```

```
    }
```

```
    return 0;
```



```
}
```

 <p data-bbox="386 181 617 244">Universidad Carlos III de Madrid</p>	<p data-bbox="678 104 1079 213"><b>Departamento de Informática</b> <b>Grado en Ingeniería Informática</b> <b>Sistemas Operativos</b></p> <p data-bbox="683 250 1075 320"><b>Prueba de Evaluación Continua</b> <b>18 de octubre de 2010</b></p>	
---	--	---

 <p>Universidad Carlos III de Madrid</p>	<p><b>Departamento de Informática</b>  <b>Grado en Ingeniería Informática</b>  <b>Sistemas Operativos</b></p> <p><b>Prueba de Evaluación Continua</b>  <b>18 de octubre de 2010</b></p>	
---	---	---

**b) [1 punto]** Indique que modificaciones se deben realizar sobre el código anterior para que el proceso padre capture la señal SIGINT (ctrl+c). En este caso, el programa padre enviará un 0 a cada uno de los hijos y estos imprimirán el número de valores pares e impares leídos.

Los procesos hijos ignoran la señal SIGINT en todo momento.

 <p>Universidad Carlos III de Madrid</p>	<p><b>Departamento de Informática</b> <b>Grado en Ingeniería Informática</b> <b>Sistemas Operativos</b></p> <p><b>Prueba de Evaluación Continua</b> <b>18 de octubre de 2010</b></p>	
---	--	---

**c) [1 puntos]** Indique que modificaciones se deben realizar sobre el código para que el proceso padre lea los datos a través de un fichero de texto, llamado *f1.txt*, que contiene valores enteros (redirección de entrada estándar). Una vez que se han leído todos los valores por fichero, el programa padre enviará un 0 a cada uno de los hijos y estos imprimirán el número de valores pares e impares leídos.

En las indicaciones se ha de poner donde se realizan las operaciones

a)

<pre>#include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt;  int tuberia0[2], tuberia1[2]; int impares, pares, valor;  void hijo0(){     int valor;     pares=-1;     close(tuberia1[1]);     close(tuberia0[0]);     close(tuberia0[1]);      do{         read(tuberia1[1], &amp;valor, sizeof(int));         par++;     }while(valor !=0);     printf("pares %d\n", pares);     close(tuberia0[0]);     exit(0); }  void hijo1(){     int valor;     impares=-1;     close(tuberia0[1]);     close(tuberia1[0]);     close(tuberia1[1]);      do{         read(tuberia0[0], &amp;valor, sizeof(int));         impar++;     }while(valor !=0);     printf("impares %d\n", impares);     close(tuberia0[0]);     exit(0); }</pre>	<pre>int main (int argc, char *argv[]) {     impares = pares = 0;      pipe(tuberia0);     pipe(tuberia1);      if (fork()==0) { /* codigo del hijo */         hijo0();     }else{         if (fork()==0) { /* codigo del hijo */             hijo1();         }else{             close(tuberia0[0]);             close(tuberia1[0]);              while (0 != scanf("%d",&amp;valor)){                 if (valor == 0)                     break;                 valor = 1;                 if (valor%2 != 0){                     write(tuberia1[1], &amp;valor, sizeof(int));                 }else{                     write(tuberia0[1], &amp;valor, sizeof(int));                 }             }             valor = 0;             write(tuberia0[1], &amp;valor, sizeof(int));             write(tuberia1[1], &amp;valor, sizeof(int));             while(wait(&amp;valor)!=-1);             close(tuberia0[1]);             close(tuberia1[1]);         }     }      return 0; }</pre>
---	--



b)

<pre> #include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt;  int tuberia0[2], tuberia1[2]; int impares, pares, valor,valor1;  void hijo0(){     int valor;     pares=-1;     signal(SIGINT,SIG_IGN);     close(tuberia1[1]);     close(tuberia0[0]);     close(tuberia0[1]);      do{         read(tuberia1[1], &amp;valor, sizeof(int));         par++;     }while(valor !=0);     printf("pares %d\n", pares);     close(tuberia1[0]);     exit(0); }  void hijo1(){     int valor;     impares=-1;     signal(SIGINT,SIG_IGN);     close(tuberia0[1]);     close(tuberia1[0]);     close(tuberia1[1]);      do{         read(tuberia0[0], &amp;valor, sizeof(int));          impar++;     }while(valor !=0);     printf("impares %d\n", impares);     close(tuberia0[0]);     exit(0); } </pre>	<pre> void manejador(){     int valor = 0;     write(tuberia0[1], &amp;valor, sizeof(int));     write(tuberia1[1], &amp;valor, sizeof(int));     while(wait(&amp;valor)!=-1);     exit(0); }  int main (int argc, char *argv[]) {     impares = pares = 0;      pipe(tuberia0);     pipe(tuberia1);      if (fork()==0) { /* codigo del hijo */         hijo0();     }else{         if (fork()==0) { /* codigo del hijo */             hijo1();         }else{             signal(SIGINT,manejador);             close(tuberia0[0]);             close(tuberia1[0]);              while (0 != scanf("%d",&amp;valor1)){                 if (valor1 == 0)                     break;                 valor = 1;                 if (valor1%2 != 0){                     write(tuberia1[1], &amp;valor, sizeof(int));                 }else{                     write(tuberia0[1], &amp;valor, sizeof(int));                 }             }             valor = 0;             write(tuberia0[1], &amp;valor, sizeof(int));             write(tuberia1[1], &amp;valor, sizeof(int));             while(wait(&amp;valor)!=-1);             close(tuberia0[1]);             close(tuberia1[1]);          }     }     return 0; } </pre>
---	---

c)

<pre>#include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;unistd.h&gt;  int tuberia0[2], tuberia1[2]; int impares, pares, valor, valor1;  void hijo0(){     int valor;     pares=-1;     close(tuberia1[1]);     close(tuberia0[0]);     close(tuberia0[1]);      do{         read(tuberia1[1], &amp;valor, sizeof(int));         par++;     }while(valor !=0);     close(tuberia1[0]);     exit(0); }  void hijo1(){     int valor;     impares=-1;     close(tuberia0[1]);     close(tuberia1[0]);     close(tuberia1[1]);      do{         read(tuberia0[0], &amp;valor, sizeof(int));         impar++;     }while(valor !=0);     close(tuberia0[0]);     exit(0); }</pre>	<pre>int main (int argc, char *argv[]) {     impares = pares = 0;      pipe(tuberia0);     pipe(tuberia1);      if (fork()==0) { /* codigo del hijo */         hijo0();     }else{         if (fork()==0) { /* codigo del hijo */             hijo1();         }else{             int fd=open("f1.txt",O_RDONLY);             close(0);             dup(fd);             close(fd);             close(tuberia0[0]);             close(tuberia1[0]);              while (0j=scanf("%d",&amp;valor1)){                 if (valor1 == 0)                     break;                 valor = 1;                  if (valor1%2 != 0){                     write(tuberia1[1], &amp;valor, sizeof(int));                 }else{                     write(tuberia0[1], &amp;valor, sizeof(int));                 }             }             valor = 0;             write(tuberia0[1], &amp;valor, sizeof(int));             write(tuberia1[1], &amp;valor, sizeof(int));             while(wait(&amp;valor)!=-1);             close(tuberia0[1]);             close(tuberia1[1]);         }     }     return 0; }</pre>
---	---

**Ejercicio 2** [2.5 puntos] Un sistema operativo utiliza un planificador cíclico (*round-robin*). En un instante determinado no hay ningún trabajo en ejecución y se desean ejecutar trabajos cuyos tiempos de llegada al sistema son los siguientes:

Proceso	Tiempo de llegada al sistema	Tiempo de ejecución	Prioridades
A	0	5	3
B	1	7	3
C	2	5	2
D	4	3	2
E	6	4	1
F	7	2	1

Las prioridades son inversas al valor que tienen. Así un proceso con prioridad 1 es prioritario respecto a otro con prioridad 2 o 3.

Se pide rellenar las siguientes tablas en los siguientes casos:

- a) Política de planificación *round-robin* con rodaja de 1 **[1 punto]**
- b) Política de planificación *round-robin* con rodaja de 4 **[1 punto]**
- c) Política de planificación SJF (*Shortest Job First*) (No expulsivo) **1 punto]**

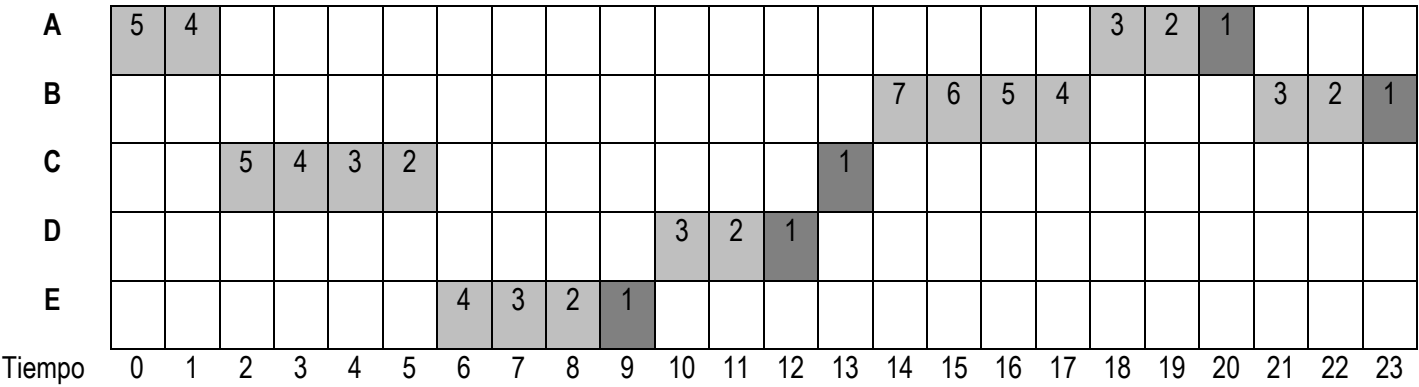
NOTA: Si la rodaja de ejecución de un proceso termina en el mismo instante que llega un nuevo proceso al sistema, entonces el nuevo proceso se coloca en la cola de listos para ejecutar antes que el proceso que le expira la rodaja.

a) Política de planificación *round-robin* con rodaja de 1

A	5													4		3		2		1				
B		7													6		5		4		3	2	1	
C			5	4		3					2		1											
D					3					2		1												
E						4	3	2	1															
Tiempo	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

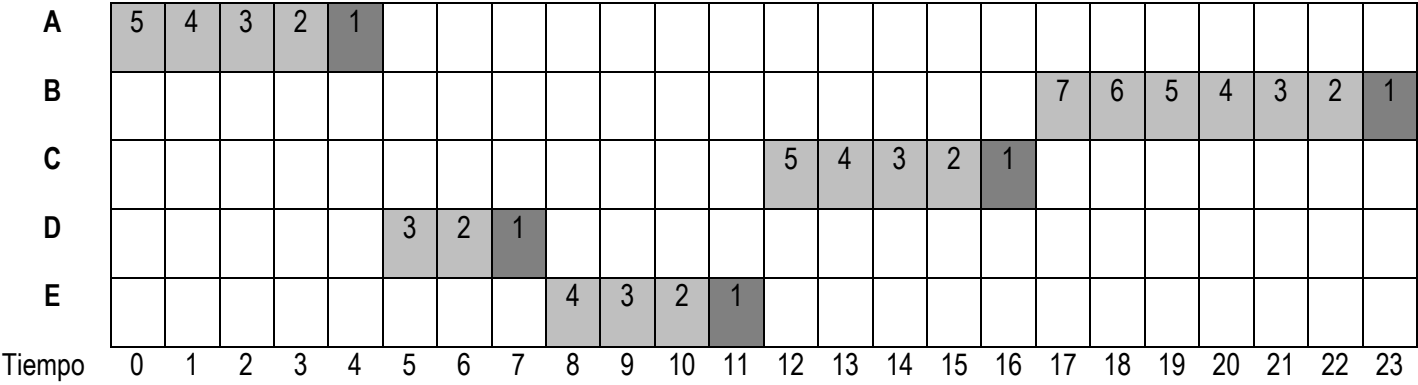
Proceso	Tiempo de finalización	Tiempo de retorno	Tiempo de servicio	Tiempo de espera	Tiempo de retorno normalizado
A	21	$21-0=21$	5	$21-5=16$	$21/5=4,2$
B	24	$24-1=23$	7	$23-7=16$	$23/7=3,3$
C	14	$14-2=12$	5	$12-5=7$	$12/5=2,4$
D	13	$13-4=9$	3	$9-3=6$	$9/3=3$
E	10	$10-6=4$	4	$4-4=0$	$6/6=1$
Valores medios		13,8	4,8	9	2,78

b) Política de planificación *round-robin* con rodaja de 4



Proceso	Tiempo de finalización	Tiempo de retorno	Tiempo de servicio	Tiempo de espera	Tiempo de retorno normalizado
A	21	$21-0=21$	5	$21-5=16$	$21/5=5,25$
B	24	$24-1=23$	7	$23-7=16$	$23/7=3,3$
C	14	$14-2=12$	5	$12-5=7$	$12/5=2,4$
D	13	$13-4=9$	3	$9-3=6$	$9/3=3$
E	10	$10-6=4$	4	$4-4=0$	$4/4=1$
Valores medios		13,4	4,8	9	3

c) Política de planificación SJF (*Shortest Job First*)



Proceso	Tiempo de finalización	Tiempo de retorno	Tiempo de servicio	Tiempo de espera	Tiempo de retorno normalizado
A	5	$5-0=5$	5	$5-5=0$	$5/5=1$
B	24	$24-1=23$	7	$23-7=16$	$23/7=3,3$
C	17	$17-2=15$	5	$15-5=10$	$15/5=3$
D	8	$8-4=6$	3	$6-3=3$	$6/3=2$
E	12	$12-6=6$	4	$6-4=2$	$6/4=1,5$
Valores medios		11	4,8	6,2	2,16