Examen Final de Sistemas Operativos
Temas 1 y 2
6 de junio de 2007

NOTAS:

* La fecha de publicación de las notas, así como de revisión se notificarán por Aula Global
* Para la realización del presente examen se dispondrá de 1 hora.
* El examen se contesta en las hojas dadas con el enunciado.
* No se pueden utilizar libros ni apuntes, ni usar móvil (o similar)
* Será necesario presentar el DNI o carnet universitario para realizar la entrega del examen

---

Ejercicio 1 (5 puntos). Escribir en C o en pseudocódigo un programa que cree un proceso hijo. El hijo deberá ejecutar un mandato recibido por línea de comando. El proceso padre matará al hijo (señal SIGKILL) en caso de que este no haya finalizado su ejecución transcurridos 10 segundos.

Solución:

```
#include <sys/types.h>
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <wait.h>

pid_t pid;

void matar_proceso(void)
{
        kill(pid, SIGKILL);  /* se envía la señal al hijo */
}

void main(int argc, char **argv)
{
        int status;
        char **argumentos;
        struct sigaction act;

        argumentos = &argv[1];
        /* Se crea el proceso hijo */
        pid = fork();
        switch(pid)
        {
                case -1:   /* error del fork() */
                        exit(-1);
```

```
                case 0:    /* proceso hijo */
                        /* El proceso hijo ejecuta el mandato recibido */
                        execvp(argumentos[0], argumentos);
                        perror("exec");
                        exit(-1);
                default:   /* padre */
                        /* establece el manejador */
                        act.sa_handler = matar_proceso; /*función a ejecutar*/
                        act.sa_flags = 0;   /* ninguna acción especifica */
                        sigemptyset(&act.sa_mask);
                        sigaction(SIGALRM, &act, NULL);
                        alarm(10);

                        /* Espera al proceso hijo */
                        wait(&status);
        }

        exit(0);
}
```

Ejercicio 2 (5 puntos). En un determinado sistema operativo los procesos se ejecutan con dos niveles de prioridad (1 para alta prioridad y 2 para baja prioridad). Los procesos de alta prioridad se planifican con la política round-robin y rodaja de tiempo de 100 ms, mientras que los procesos de baja prioridad se planifican con la política FIFO.
En la siguiente tabla se especifica para cada proceso, su prioridad, su tiempo de llegada, el tiempo que necesitan para ejecutarse.

PROCESOS	PRIORIDAD	T. DE LLEGADA	T DE EJECUCIÓN
P1	1	0	500
P2	2	100	300
P3	2	300	400
P4	1	600	1000
P5	1	700	600

Determina el tiempo de finalización de cada proceso.

T	Ejecución	Cola Prioridad 1	Cola Prioridad 2	Termina
0				
100				
300				
500				
600				
700				
800				
900				
1000				
1100				
1200				
1300				
1400				
1500				
1600				
1700				
1800				
2200				
2400				
2800				

Solución:

T	Ejecución	Cola Prioridad 1	Cola Prioridad 2	Termina
0	P1(500)			
100	P1(400)		P2(300)	
300	P1(200)		P2(300), P3(400)	
500	P2(300)		P3(400)	P1
600	P4(1000)		P2(200) P3(400)	
700	P5(600)	P4(900)	P2(200) P3(400)	
800	P4(900)	P5(500)	P2(200) P3(400)	
900	P5(500)	P4(800)	P2(200) P3(400)	
1000	P4(800)	P5(400)	P2(200) P3(400)	
1100	P5(400)	P4(700)	P2(200) P3(400)	
1200	P4(700)	P5(300)	P2(200) P3(400)	
1300	P5(300)	P4(600)	P2(200) P3(400)	
1400	P4(600)	P5(200)	P2(200) P3(400)	
1500	P5(200)	P4(500)	P2(200) P3(400)	
1600	P4(500)	P5(100)	P2(200) P3(400)	
1700	P5(100)	P4(400)	P2(200) P3(400)	
1800	P4(400)		P2(200) P3(400)	P5
2200	P2(200)		P3(400)	P4
2400	P3(400)			P2
2800				P3

Temas 3, 4, 5 y 6
6 de junio de 2007

NOTAS:

* La fecha de publicación de las notas, así como de revisión se notificarán por Aula Global
* Para la realización del presente examen se dispondrá de 2 horas.
* El examen se contesta en las hojas dadas con el enunciado.
* No se pueden utilizar libros ni apuntes, ni usar móvil (o similar)
* Será necesario presentar el DNI o carnet universitario para realizar la entrega del examen

---

Ejercicio 1 (3 puntos)

Se pide:

a) Dado los siguientes fragmentos de pseudocódigo:

```
int i1, i2 ;
th_id th1, th2 ;
th_mutex m1, m2 ;

void f1 ( void *p )   {
  th_lock(&m1); for (i1=0; i1<100; i1++) printf(%d\n,i1); th_unlock(&m1);
}

void f2 ( void *p )   {
  th_lock(&m1); for (i2=0; i2<100; i2++) printf(%d\n,i2); th_unlock(&m1);
}

void f3 ( void *p )   {
  th_lock(&m1); for (i1=0; i1<100; i1++) printf(%d\n,i2) ; th_unlock(&m1);
}

void f4 ( void *p )   {
  th_lock(&m2); for (i2=0; i2<100; i2++) printf(%d\n,i1) ; th_unlock(&m2);
}

int main ( int argc, char *argv[] ) {
     th_create(&th1,f1,NULL) ;  th_create(&th2,f2,NULL) ;
     th_join(&th1,f1,NULL) ;    th_join(&th2,f2,NULL) ;

     th_create(&th1,f3,NULL) ;  th_create(&th2,f4,NULL) ;
     th_join(&th1,f3,NULL) ;    th_join(&th2,f4,NULL) ;
     return (0) ;
}
```

Para conseguir la máxima concurrencia sin problemas modifique en el fragmento de código dado (directamente en la figura). Numere cada modificación y a continuación indique por cada número el por qué de esa modificación. (1,5 puntos)

b) Implementar la siguiente función en código C utilizando llamadas POSIX con variables condicionales y mutex: (1,5 puntos)

```
/* variables globales que se necesiten */
int   numero_hilos_dormidos;
int   numero_total_de_hilos;



int barrera ( void ) {

        /* incrementar el número de hilos dormidos */


        /* si todos los hilos están dormidos, entonces despertar a
           todos los hilos dormidos y dejar todo listo para nuevas barreras */



         /* en caso contrario, dormirse */



}
```

Solución

a) Dado los siguientes fragmentos de pseudocódigo:

```
int i1, i2 ;
th_id th1, th2 ;
th_mutex m1, m2 ;

void f1 ( void *p )   {
  th_lock(&m1); for (i1=0; i1<100; i1++) printf(%d\n,i1); th_unlock(&m1);
}

void f2 ( void *p )   {
  th_lock(&m1); for (i2=0; i2<100; i2++) printf(%d\n,i2); th_unlock(&m1);
}

void f3 ( void *p )   {
  th_lock(&m1); for (i1=0; i1<100; i1++) printf(%d\n,i2) ; th_unlock(&m1);
}

void f4 ( void *p )   {
  th_lock(&m2); for (i2=0; i2<100; i2++) printf(%d\n,i1) ; th_unlock(&m2);
}

int main ( int argc, char *argv[] ) {
     th_create(&th1,f1,NULL) ;  th_create(&th2,f2,NULL) ;
     th_join(&th1,f1,NULL) ;    th_join(&th2,f2,NULL) ;

     th_create(&th1,f3,NULL) ;  th_create(&th2,f4,NULL) ;
     th_join(&th1,f3,NULL) ;    th_join(&th2,f4,NULL) ;
     return (0) ;
}
```

Para conseguir la máxima concurrencia sin problemas modifique en el fragmento de código dado (directamente en la figura). Numere cada modificación y a continuación indique por cada número el por qué de esa modificación. (1,5 puntos)

(1) Quitar cerrojos puesto que son variables independientes

(2) Usar el mismo cerrojo, puesto que la sección crítica es común

b) Implementar la siguiente función en código C utilizando llamadas POSIX con variables condicionales y mutex: (1,5 puntos)

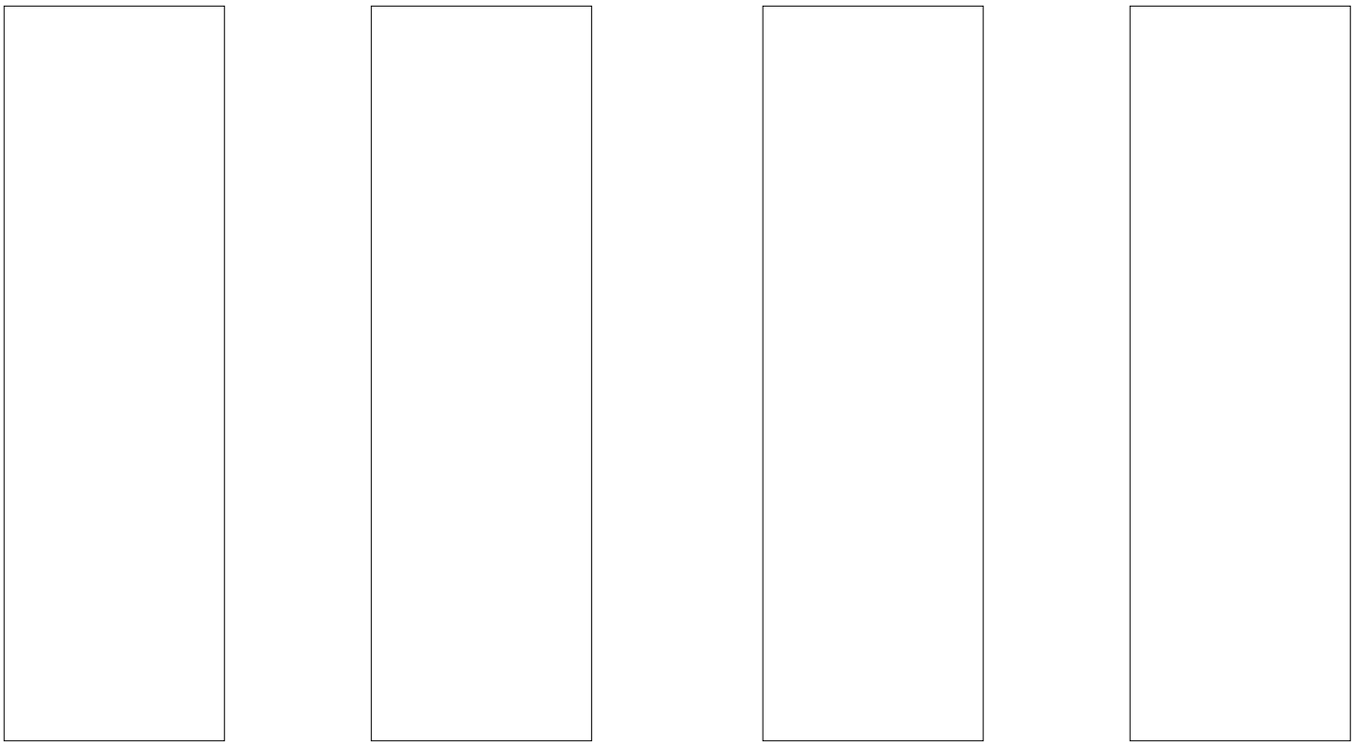
```
/* variables globales que se necesiten */
int   numero_hilos_dormidos;
int   numero_total_de_hilos;

pthread_mutex mutex ;
pthread_condition cond;

int barrera ( void ) {
        pthread_mutex_lock(&mutex) ;

        /* incrementar el número de hilos dormidos */
        numero_hilos_dormidos ++ ;

         if (numero_hilos_dormidos == numero_total_de_hilos) {
        /* si todos los hilos están dormidos, entonces despertar a
           todos los hilos dormidos y dejar todo listo para nuevas barreras */
              numero_hilos_dormidos = 0 ;
              pthread_cond_broadcast(&cond) ;
              pthread_mutex_unlock(&mutex) ;
        } else {
         /* en caso contrario, dormirse */
              pthread_cond_wait(&cond,&mutex) ;
              pthread_mutex_unlock(&mutex) ;
        }

}
```

Ejercicio 2 (3,5 puntos)

Atendiendo al siguiente código:

```
float a;  float b=2;

void f1(int c) {
        float i;
        int j;

        b = i+ 5;
        .......
B       f2(j)
        return;
}

void f2(int w) {
        static f = 2;
        a=a+w+f;
        .......
        return;
}

main (int argc, char **argv) {
        int *p;
        char *q;

A       p = (int *) malloc (512)

        f1(b);

C       q=mmap (fichero)
        .......
        free (p)
        .......
        free (q)
        ....
        exit (0)
}
```

Indica como evolucionan las regiones o segmentos de memoria del programa cuando se lanza su ejecución, y en los puntos del código indicados con flechas rellenando las siguientes figuras. Indica así mismo el contenido de cada segmento.



Inicio de la ejecución | A | B | C

Solución:

Inicio de la ejecución: Código, Datos con valor inicial (b=2, f=2), Datos sin valor inicial (a=?), Pila (var. Entorno, argc, argv, retorno main, p=?, q=?)

A: Código, Datos con valor inicial (b=2, f=2), Datos sin valor inicial (a=?), Pila (var. Entorno, argc, argv, retorno main, q=?) y Heap(p)

B: Código, Datos con valor inicial (b=2, f=2), Datos sin valor inicial (a=?), Pila (var. Entorno, argc, argv, retorno main, q=?, Bloque de activacion de f1(i,j,c, retorno f1), Bloque de activacion de f2 (w,retorno f2) y Heap(p)

C: Código, Datos con valor inicial (b=2, f=2), Datos sin valor inicial (a=?), Pila (var. Entorno, argc, argv, retorno main) , Heap(p) y Proyeccion de fichero (q)

Ejerci

Se dispone de un Sistema de Ficheros estilo UNIX donde el tamaño de bloque es de 512 bytes. El i-nodo 0 corresponde al directorio raíz del sistema.

Se

Bloq: 200	Bloq: 201	Bloq: 202	Bloq: 203	Bloq: 204
i-nodo: 0 0 Enlaces: 0 4 Nombre: work 1 Tipo: dir Bloque1: 2 200	i-nodo: 0 1 Enlaces: 2 Nombre: es Tipo: dir Bloque1: 204	i-nodo: 2 2 Enlaces: 0 2 Nombre: pr1.txt 4 Tipo: dir Bloque1: 202	i-nodo: 3 Enlaces: 1 Nombre: pr.asm Tipo: symbolic Bloque1: 203	i-nodo: 14 Enlaces: 02 Nombre: Pruebas 3 Tipo: file Bloque1: 201

1.- Dibujar mediante un árbol la estructura de directorios actual en el sistema (0,5 puntos).

2.- Ejecutar las siguientes instrucciones en el orden indicado, indicando qué y para qué se usa la información del sistema de ficheros en cada momento, y cuales son las modificaciones que se producen a nivel de i-nodos y bloques de datos.

Asumir que la operación anterior puede influir en la siguiente.

a) cat /work/pr1.asm (0,75 puntos)
b) cat /work/pr.asm (0,75 puntos)
c) rm /work/pr.asm (0,75 puntos)
d) ln /work/pr1.asm /work/pr.asm (0,75 puntos)

NOTA:

ln source_file target_file

Por defecto establece un enlace duro entre source_file (debe existir) y target_file

Solución:

1.-

2.-

A) cat /work/pr1.asm

a) Leer el inodo del directorio raiz (inodo 0 - bloque 100) y luego se lee el primer bloque de datos de dicho directorio (bloque 200), como ahí se encuentra la entrada de work (inodo 1) no hay que leer más bloques.
b) Leer el inodo de work (inodo 1 - bloque 101), se ve que es un directorio y se lee el primer bloque de datos (bloque 204),. Se lee la entrada pr.asm y a continuación la entrada de pr1.asm (inodo 4) no hay que leer más bloques.
c) Leer el inodo de pr1.asm (inodo 4 - bloque 104), se ve que es un fichero y se lee el primer bloque de datos (bloque 201) y se muestra por pantalla (si hubiera más bloques habría que mostrarlos todos).

No produce cambios en el sistema de ficheros.

B) cat /work/pr.asm

a) Leer el inodo del directorio raiz (inodo 0 - bloque 100) y luego se lee el primer bloque de datos de dicho directorio (bloque 200), como ahí se encuentra la entrada de work (inodo 1) no hay que leer más bloques.
b) Leer el inodo de work (inodo 1 - bloque 101), se ve que es un directorio y se lee el primer bloque de datos (bloque 204),. Se lee la entrada pr.asm (inodo 3) no hay que leer más bloques.
c) Leer el inodo de pr1.asm (inodo 4 - bloque 104), se ve que es un enlace simbólico y se lee su bloque de datos (bloque 203).
d) Al ser un enlace simbólico se ejecutaría el conjunto de operaciones con el contenido del bloque de datos (cat /ssoo/practica2.asm).
e) Leer el inodo del directorio raiz (inodo 0 - bloque 100) y luego se lee el primer bloque de datos de dicho directorio (bloque 200). No se encuentra la entrada ssoo luego finaliza la llamada al sistema y se retorna ERROR debido a que no existe la ruta del fichero.

No produce cambios en el sistema de ficheros.

C) rm /work/pr.asm

a) Leer el inodo del directorio raiz (inodo 0 - bloque 100) y luego se lee el primer bloque de datos de dicho directorio (bloque 200), como ahí se encuentra la entrada de work (inodo 1) no hay que leer más bloques.
b) Leer el inodo de work (inodo 1 - bloque 101), se ve que es un directorio y se lee el primer bloque de datos (bloque 204). Se lee la entrada pr.asm (inodo 3

) no hay que leer más bloques. Se lee el i-nodo 3 y se procede a borrar la

c)

d) nodo 103 (bloque 203)

Bloq: 200	Bloq: 201	Bloq: 202	Bloq: 203	Bloq: 204
i-nodo: 0 0 Enlaces: 0 4 Tipo: dir Bloque1: 200	i-nodo: 1 1 Enlaces: 2 2 Tipo: dir Bloque1: 204	i-nodo: 2 2 Enlaces: 0 2 Tipo: dir Bloque1: 202		i-nodo: 14 Enlaces: 02 Tipo: file Bloque1: 201

D)

a) Leer el inodo del directorio raiz (inodo 0 - bloque 100) y luego se lee el primer bloque de datos de dicho directorio (bloque 200), como ahí se encuentra la entrada de work (inodo 1) no hay que leer más bloques.
b) Leer el inodo de work (inodo 1 - bloque 101), se ve que es un directorio y se lee el primer bloque de datos (bloque 204).. Se lee la entrada pr1.asm (inodo 4) no hay que leer más bloques.
c) Leer el inodo de pr1.asm (inodo 4 - bloque 104), se ve que es un fichero y se procede a incrementar el contador de enlaces a 2. Apuntamos el inodo 4 para insertarlo como entrada de pr1.asm. En los siguientes pasos procedemos a localizar el emplazamiento de pr.asm
d) Leer el inodo del directorio raiz (inodo 0 - bloque 100) y luego se lee el

e)

Bloq: 200	Bloq: 201	Bloq: 202	Bloq: 203	Bloq: 204
i-nodo: 0 0 Enlaces: 0 4 Tipo: dir Bloque1: 200	i-nodo: 1 1 Enlaces: 2 2 Tipo: dir Bloque1: 204	i-nodo: 2 2 Enlaces: 0 2 Tipo: dir Bloque1: 202		i-nodo: 14 Enlaces: 03 Tipo: file Bloque1: 201

que va apuntar al inodo 4 resultado de la operación de lectura producida en los pasos a,b y c.