EXAMEN SSOO
Febrero 2008

Problema 1. Implementar pipes con mutex y vconds

```
struct {
        int tubo[MAX];
        int posP, posC, nelem, nhuecos;
        int nprods, ncons;
        cond_t esperaC, esperaP;
        mutex_t m_pipe;
} pipe;

desc_t open(mode_t modo) { // modo = O_READ o O_WRITE; desc = TRUE o FALSE
        desc_t res;

        mutex_lock(pipe.m_pipe);
        if ((pipe.nprods + pipe.ncons) == 0) {
                pipe.posC = pipe.posD = pipe.nelem = 0;
                pipe.huecos = MAX;
        }
        if (modo == O_READ) {
                pipe.ncons++; res = TRUE;
        } else {
                pipe.nprods+++; res = FALSE;
        }
        mutex_unlock(pipe.m_pipe);
        return res;
}

void close(desc_t desc) {
        mutex_lock(pipe.m_pipe);
        if (desc == TRUE) {
                pipe.ncons = MAX(pipe.ncons-1, 0);
                broadcast(pipe.esperaP);
        } else {
                pipe.nprods = MAX(pipe.nprods.1, 0);
                broadcast(pipe.esperaC);
        }
        mutex_unlock(pipe.m_pipe);
}
```

```
int read(desc_t desc, int buf[], int n) {
        if (n <= 0) return -1;
        mutex_lock(pipe.m_pipe);
        while ((pipe.nelem == 0) && (pipe.nprods > 0))
                cond.wait(pipe.esperaC, pipe.m_pipe);
        if ((pipe.nelem > 0) {
                if (pipe.nelem < n) n = pipe.nelem;
                for (i=0; i<n; i++)
                        buf[i] = pipe.tubo[(pipe.posC+i)
%TAM] ;
                pipe.posC = (pipe.posC+n)%TAM;
                pipe.nhuecos = pipe.nhuecos + n;
                pipe.nelem = pipe.nelem – n;
                broadcast(pipe.esperaP);        // la creación de n huecos puede despertar a mas de 1
        } else          //  (pipe.nprods == 0)  && (pipe.nelem == 0)
                n = 0;
        }
        mutex_unlock(pipe.m_pipe);
        return n;
}
```

Consumidor

Si nelem==0

- y nprods > 0 ==> BLOQUEO
- y nprods == 0 => 0

Si 0 < nelem >= n => leer n elementos

Si 0 < nelem < n => leer nelem elementos

```
int write(desc_t desc, int buf[], int n) {
        int i;

        if (n <= 0) return -1;
        mutex_lock(pipe.m_pipe);
        while ((pipe.nhuecos < n) && (pipe.ncons > 0))
                cond.wait(pipe.esperaP, pipe.m_pipe);
        if  (pipe.ncons == 0)
                n = -1;
        else {          // pipe.nhuecos >= n
                for (i=0; i<n; i++) pipe.tubo[(pipe.posP+i)
%TAM] = buf[i] ;
                pipe.posP = (pipe.posP+n)%TAM;
                pipe.nhuecos = pipe.nhuecos – n;
                pipe.nelem = pipe.nelem + n;
                broadcast(pipe.esperaC);        // la creación de n elems puede despertar a mas de 1
        }
        mutex_unlock(pipe.m_pipe);
        return n;
}
```

Productor

Si ncons == 0 => -1

Si ncons > 0

- y nhuecos < n ==> BLOQUEO
- y nhuecos >= n => escribe n

2.

tdaa:			tdaa:		
	STDIN (0):	i0		STDIN (0):	i0
	STDOUT(1) = fd2:	i3		STDOUT(1) = fd2:	i3
	STDERR(2):	i2		STDERR(2):	i2
	fd1 (3):	i4		fd1 (3):	i5
data = “Moaxaja”			data = “Moaxaja” / “Alfaqui”		
bufR = ?? / “Alfaquibbb”			bufR = ??		
Proceso Padre			Proceso Hijo		

tin:		pos	modo	cnt	vn	
	i0:	--	R	1/2/1	v0	// +1 fork(),-1 exit P2
	i1:	--	W	1	v0	// suprimido en close(STDOUT)
	i2:	--	W	1/2/1	v0	// +1 fork(), -1 exit P2
	i3:	0	W	1/2/1	v1	// 1 open 1, +1 fork(), -1 close(fd2)
	i4:	0	RW	1	v1	// 1 open 2
	i5	0	RW	1	v1	// 1 open 3, suprimido en (c)

i3.pos: 0 / 8 / 520 / 1032 / 2056 / 2063
i4.pos: 0 / 10
i5.pos: 0 / 7 / 10K / 10k+500 / 12K / 12k+500 / 14K

t-vn		fichero	cnt	tamaño
	v0	“tty”	3/2	--
	v1	“cejel”	1/2/3/2	0 / 8 / 10K+500 / 12K+500

Asignación de bloques de “cejel” y contenido
D0: 0K:“Alfaqui”(8), resto: b's
D1: 1K: b's(8), resto: 0's
D2: 2K: 0's(8), “Moaxaja”(7), resto: 0's
D3: 3K: sin asignar
D4: 4K: sin asignar
D5: indirecto simple ==> 5K,6K,7K,8K,9K – sin asignar
==> 10K: a's(500), resto: 0's
==> 11K – sin asignar
==> 12K: a's(500), FIN_DE_FICHERO
D6: sin asignar (indirecto doble)

3.

a. Código : 1 página ==> acceso a disco-SF
Pila M: 1 página ==> creación / acceso a disco swap
Pila H1: 1 página ==> creación / acceso a disco swap
Pila H2: 1 página ==> creación / acceso a disco swap
D.BSS: 1 página ==> creación / acceso a disco swap
D.iniciales: 6 páginas ==> acceso a disco-SF / acceso a disco swap

b. Dos discos independientes, paginación por demanda

Main: C(20)
Hilo1: A0(20) cpu(16) A1(20) cpu(16) A2*(20) cpu(16)
Hilo2: A3(20) cpu(16) A4(20) cpu(16) A5(20)* cpu(16)
Total: 7 x 20 + 16 = 156

c. Dos discos independientes y prepaginación de 2

Main: C(20)
Hilo1: A01(25) cpu(32) A2(20)* cpu(16)
Hilo2: A34(25) cpu(32) A4(20)* cpu(16)
Total: 20 + 25+ 32 x 2 + 20 + 16 = 145

d. Dos particiones en un mismo disco, paginación por demanda

Main: C(20)
Hilo1: A0(20) cpu(16) A1(20) cpu(16) A0'(20) A2(20) cpu(16)
Hilo2: A3(20) cpu(16) A4(20) cpu(16) A3'(20) A5(20) cpu(16)
Total: 9 x 20 + 16 = 196

TEORIA

1.

i=0 => a=0+1=1, b=1+1=2
i=1 => a=1+2=3, b=2+1=3
i=2 => a=3+3=6, b=3+1=4 Este el resultado mostrado por pantalla

El proceso hijo y su hilo asociado no influencian para nada la operación del padre
(Tan solo se ve afectado por la sincronización)

b) Las relaciones señal<=>acción
de padre e hijo son idénticas inmediatamente después de fork()
de proceso antes y después de exec(), son las mismas para las acciones IGN y DFL
pero las acciones-función dejan de ser válidas al cambiar de programa
(y por tanto de mapa de memoria) y revierten a DFL

2.

a) Es mecanismo de sincronización

b)	wait(s) = {
	while (!TS(s.lock)) ;
	s.pid = getpid();
	}
		signal(s) = {
		s.pìd = NULL; s.lock = 0;
		}

c)	wait(s) = {
	for (i=0; i< 100; i++) {
	while(TS(s.lock)) ;
	if (s.pid == NULL) break;
	s.lock = 0;
	}
	if (i==100) {
	while(TS(s.lock) ;
	bloquear el proceso;
	despachar y s.lock=0;
	} else {
	s.pid = getpid();
	s.lock = 0;
	}
	}
		signal(s) = {
		while (TS(s.lock) ;
		if (procesos bloqueados) {
		desbloquear uno;
		s.pid = pid_nuevo;
		} else {
		spid = NULL;
		}
		s.lock = 0;
		}

d. EXCLUSION MUTUA => garantiza que solo un proceso tiene la propiedad del mutex
PROGRESO => un proceso no espera si el mutex está libre
ESPERA ACOTADA => la política FIFO de espera hace que un proceso solo
espera a los que tiene delante

3.

a) Espacio de trabajo: conjunto de páginas del espacio virtiual de un programa, que se referenciarán en memoria desde la referencia actual hasta dentro de Θ (ventana) referencias. Se trata teóricamente como una función $\omega(\Theta,t)$
b) Espacio residente: conjunto de páginas del espacio virtual de un programa, actualmente

cargadas en marcos de memoria

c) CREAR => exec(), mmap(), dlopen()
DUPLICAR: fork() -- optimizado con COW
CAMBIAR TAMAÑO: fallo de página por expansión de pila ó break() - aumento del heap
LIBERAR: exit(), munmap(), dlclose()

4.

a) La política de planificación de disco pretende ordenar las peticiones teniendo en cuenta la distribución "espacial" de sus localizaciones basándose en que, dado que el acceso en disco es del tipo "directo" (una petición de localización numérica "vecina" a la última accedida está más cercana físicamente que otra petición numéricamente "alejada") será mucho más rápido atender a las peticiones en orden de vecindad.
La numeración de bloque relacionada con la vecindad física, y por tanto la potencial aplicación provechosa de una política de planificación de disco, se da en:
- el gestor de bloques: numeración de bloque lógico del sistema de fichero
- el manejador de dispositivo: numeración física LBA del disco
- el controlador de disco h/w: numeración física CHS del disco

b) - petición expresa mediante llamada: open(); read(), write(), exec()
- fallo de página mayor
- actualización periódica de marcos en disco (actualización por buffering)
- retirada de un proceso a swap (control de carga)

5.

a) Directorio UNIX => nombre + número de inodo (+ longitud del nombre)
Directorio FAT => nombre + atributos (tipo, tamaño, protección,...) + cluster inicial

b) Un enlace físico consiste en la compartición directa de los atributos de un fichero desde dos nombres distintos. En FAT cada nombre va asociado a sus propios atributos, por lo que la comportación de sus valores supondría duplicarlos en dos antradas distintas con enormes problemas de mantenimeinto de coherencia.

c) ext3 => en el Mapa de Bits de Inodos contar el número de bits con valor OCUPADO (0)
fat => en la FAT, contar el número de entradas con valor EOF