E. U. de Informática | Departamento de Informática Aplicada

Examen Final de Sistemas Operativos I **2 de Junio de 2008**

Apellidos.....

Nombre .....

Nº de Matrícula .....

EJERCICIO 1 (1' 5 puntos) 15 minutos

Escriba un programa en C denominado `micp.c` usando llamadas al sistema POSIX, que permita invocar el fichero ejecutable `/bin/cp` para copiar el contenido de un fichero origen a un fichero destino. Utilice una llamada al sistema en la que no haga falta especificar el camino (/bin) al fichero ejecutable cp. Los argumentos deberán leerse de la linea de comandos. Controle los errores de ejecución de las llamadas al sistema que use, de tal forma que si se produce un error se imprima el valor de errno y la tira de caracteres asociada por la salida de errores.

E. U. de Informática | Departamento de Informática Aplicada

Examen Final de Sistemas Operativos I **2 de Junio de 2008**

Apellidos.....

Nombre .....

Nº de Matrícula .....

EJERCICIO 2 (3 puntos) 45 minutos

Escriba un programa `control_tiempo.c` en el que un padre crea un hijo que se encargará de ejecutar el fichero `timeout1` que no admite argumentos y que se encuentra en el directorio actual. El padre deberá esperar como mucho 5 segundos a que el hijo acabe. Si el hijo acaba antes de los 5 segundos, el padre escribirá por la salida estanadar el literal "el proceso hijo ha teminado a tiempo" junto con el estatus de la muerte del hijo.
Si por el contrario, el hijo sigue vivo tras los 5 segundos, el padre matará al proceso hijo y escribirá por la salida estandar "el proceso hijo ha excedido los 5 segundos" junto con su estatus correspondiente. (**Nota: no está permitido el uso de la llamada sleep. Suponga que las llamadas no producen errores.**)

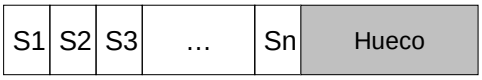
E. U. de Informática — Departamento de Informática Aplicada

Examen Final de Sistemas Operativos I — **2 de Junio de 2008**

Apellidos.....
Nombre .....
Matrícula .....

EJERCICIO 3 (2' 5 puntos)

Considere una memoria de **c** palabras en la que segmentos contiguos S1, S2, …, Sn se disponen en estricto orden de creación desde un extremo de la memoria al otro extremo (ver figura).



Cuando se crea el segmento Sn+1, se ubica inmediatamente después del segmento Sn incluso aunque algunos de los segmentos S1, S2, …, Sn ya hayan sido borrados. Cuando la frontera entre segmentos y el "Hueco" alcance el otro extremo de la memoria, se compactan los segmentos en uso. El tamaño medio del segmento es de **s** palabras y su tiempo medio de vida son **t** referencias a memoria; es decir, cada t referencias a memoria se crea/destruye un segmento. Sea **f** la fracción de memoria no usada. Considere que copiar una palabra de una posición de memoria a otra implica dos referencias a memoria. ¿Cuál es la fracción de tiempo empleada en la compactación?

NOTA: Por término medio se crea 1 segmento de tamaño s a la misma velocidad que se destruye un segmento de tamaño s. Luego, en equilibrio, f es constante.

E. U. de Informática — Departamento de Informática Aplicada

Examen Final de Sistemas Operativos I — **2 de Junio de 2008**

Apellidos.....

Nombre .....

Nº de Matrícula .....

EJERCICIO 4 (3 puntos) **60 minutos**

Se ha generado un sistema de ficheros Minix (tipo Unix) sobre un disquette utilizando para ello el comando mkfs. Tras realizar ciertas operaciones sobre dicho sistemas de ficheros, se ha volcado el estado actual del sistema de ficheros (utilizando el comando od) a pantalla siendo parte de dicho volcado el que se ilustra en la Figura 1.

```
0000000 0000 0000 0000 0000 0000 0000 0000 0000
                        *
0000400 0020 0000 0000 0001 0001 0005 0000 0000
0000410 3800 2010 0096 0000 4d5a 0000 0800 0000
0000420 0000 0000 0000 0000 0000 0000 0000 0000
                        *
0001000 000f 0000 0000 0000 0000 0000 0000 0000
0001010 0000 0000 0000 0000 0000 0000 0000 0000
                        *
0001800 000f 0000 0000 0000 0000 0000 0000 0000
0001810 0000 0000 0000 0000 0000 0000 0000 0000
                        *
0002000 41ff 0002 0002 0002 0100 0000 afff 483e
0002010 afed 483e afed 483e 0005 0000 0000 0000
0002020 0000 0000 0000 0000 0000 0000 0000 0000
                        *
0002040 81a4 0001 0000 0000 000e 0000 afe4 483e
0002050 afe4 483e afe4 483e 0006 0000 0000 0000
0002060 0000 0000 0000 0000 0000 0000 0000 0000
                        *
0002080 81a4 0001 0000 0000 0012 0000 afe4 483e
0002090 afe4 483e afe4 483e 0007 0000 0000 0000
00020a0 0000 0000 0000 0000 0000 0000 0000 0000
                        *
0002800 0001 0000 002e 0000 0000 0000 0000 0000
0002810 0000 0000 0000 0000 0000 0000 0000 0000
                        *
0002840 0001 0000 2e2e 0000 0000 0000 0000 0000
0002850 0000 0000 0000 0000 0000 0000 0000 0000
                        *
0002880 0002 0000 3166 0000 0000 0000 0000 0000
0002890 0000 0000 0000 0000 0000 0000 0000 0000
                        *
00028c0 0003 0000 3266 0000 0000 0000 0000 0000
00028d0 0000 0000 0000 0000 0000 0000 0000 0000
                        *
0003000 6f68 616c 202c 7571 2065 6174 0a6c 0000
0003010 0000 0000 0000 0000 0000 0000 0000 0000
                        *
0003800 756d 2079 6962 6e65 202c 7267 6361 6169
0003810 0a73 0000 0000 0000 0000 0000 0000 0000
0003820 0000 0000 0000 0000 0000 0000 0000 0000
                        *
```

Figura 1.

La primera columna de la Fig.1 indica una dirección que es el desplazamiento (en bytes) desde el comienzo del disco. El resto es el contenido del disco. Todo está expresado en hexadecimal. El carácter * indica que se repite el contenido anterior. Para la información mantenida en 4 bytes primero aparecen los 2 bytes menos significativos y luego los dos más significativos.

La estructura del sistema de ficheros de Minix en disco es la siguiente:

1. Bloque de boot (1K).
2. Superbloque (1K).
3. Bloques para soportar el mapa de bits de i-nodes.
4. Bloques para soportar el mapa de bits de zonas.
5. Bloques que contienen los i-nodes.
6. Bloques de datos

La estructura (en disco) del superbloque es la siguiente:

- Número de i-nodos (4 bytes).
- No usado (2 bytes).
- Número de bloques para el mapa de bits de i-nodos (2 bytes).
- Número de bloques para el mapa de bits de zonas (2 bytes).
- Primera zona de datos (2 bytes).
- $\log_2$ (número de bloques por zona) (2 bytes).
- No usado (2 bytes).
- Tamaño máximo del fichero (4 bytes).
- Número de zonas (4 bytes).
- Número mágico (2 bytes).
- No usado (2 bytes).
- Tamaño del bloque (2 bytes).
- FS subversión (1 byte).

Una entrada al directorio tiene 64 bytes: 4 para indicar el número de i-nodo y 60 para el nombre del fichero.

La estructura del i-nodo en Minix es la siguiente.

```
/* Declaration of the V2 inode as it is on the disk (not in core). */
typedef struct {            /* V2.x disk inode */
  mode_t d2_mode;               /* file type, protection, etc. (2 bytes) */
  u16_t d2_nlinks;          /* how many links to this file. HACK! (2 bytes) */
  uid_t d2_uid;             /* user id of the file's owner. (2 bytes) */
  u16_t d2_gid;             /* group number HACK! (2 bytes) */
  off_t d2_size;            /* current file size in bytes (4 bytes) */
  time_t d2_atime;          /* when was file data last accessed (4 bytes) */
  time_t d2_mtime;          /* when was file data last changed (4 bytes) */
  time_t d2_ctime;          /* when was inode data last changed (4 bytes) */
  zone_t d2_zone[V2_NR_TZONES]; /* block nums for direct, ind, and dbl ind */
                        /* 4 bytes cada puntero a bloque */
} d2_inode;

#define V2_NR_TZONES 10         /* 7 directos, 1 indirecto, 1 indirecto doble, no usado */

/* Flag bits for i_mode in the inode. */
```

```
#define I_SYMBOLIC_LINK 0120000        /* file is a symbolic link */
#define I_REGULAR       0100000     /* regular file, not dir or special */
#define I_BLOCK_SPECIAL 0060000        /* block special file */
#define I_DIRECTORY     0040000        /* file is a directory */
#define I_CHAR_SPECIAL  0020000     /* character special file */
#define I_NAMED_PIPE0010000 /* named pipe (FIFO) */
#define I_SET_UID_BIT   0004000        /* set effective uid_t on exec */
#define I_SET_GID_BIT   0002000        /* set effective gid_t on exec */
#define RWX_MODES       0000777        /* mode bits for RWX only */
#define R_BIT           0000004          /* Rwx protection bit */
#define W_BIT           0000002          /* rWx protection bit */
#define X_BIT           0000001          /* rwX protection bit */
```

Siendo:

```
#define ROOT_INODE  1  /* inode number for root directory */
```

Tabla de códigos ASCII

ASCII	Hex	Símbolo
0	0	NUL
1	1	SOH
2	2	STX
3	3	ETX
4	4	EOT
5	5	ENQ
6	6	ACK
7	7	BEL
8	8	BS
9	9	TAB
10	A	LF
11	B	VT
12	C	FF
13	D	CR
14	E	SO
15	F	SI
16	10	DLE
17	11	DC1
18	12	DC2
19	13	DC3
20	14	DC4
21	15	NAK
22	16	SYN
23	17	ETB
24	18	CAN
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	FS
29	1D	GS
30	1E	RS
31	1F	US
32	20	(espacio)
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	

Responda a las siguientes cuestiones en relación con el sistema de ficheros cuyo contenido se muestra en la figura 1:

1. ¿Cuántos i-nodos tiene?
2. ¿Cuál es el tamaño máximo de un fichero? ¿A partir de qué datos se obtiene este resultado?
3. ¿Cuál es el tamaño del bloque?
4. ¿Qué bloques contienen los i-nodos?

5. ¿Cuál es el tamaño del directorio raíz? ¿Por qué?
6. ¿Cuál es el contenido del fichero f2?