grapa

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	23/01/2016	09:00

75.097 23 01 16 EX

Espacio para la etiqueta identificativa con el código personal del **estudiante**. Examen

Ficha técnica del examen

- **Comprueba que el código y el nombre de la asignatura corresponden a la asignatura de la cual estás matriculado.**
- **Debes pegar una sola etiqueta de estudiante en el espacio de esta hoja destinado a ello.**
- **No se puede añadir hojas adicionales.**
- **No se puede realizar las pruebas a lápiz o rotulador.**
- **Tiempo total** 2 **horas**
- **En el caso de que los estudiantes puedan consultar algún material durante el examen, ¿cuál o cuáles pueden consultar?:** Un chuletario tamaño folio/DIN-A4 con anotaciones por las dos caras.
- **Valor de cada pregunta:** 2,5 puntos
- **En el caso de que haya preguntas tipo test: ¿descuentan las respuestas erróneas?** NO **¿Cuánto?**
- **Indicaciones específicas para la realización de este examen**

Enunciados

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	23/01/2016	09:00

1. Teoría

Contestad justificadamente las siguientes preguntes.

a) Indicad qué puede provocar que un proceso pase del estado Run (ejecución) al estado Ready (preparado).

Típicamente, esta transición sería provocada por un planificador de la cpu basado en round robin cuando finaliza el quantum asociado al proceso.

b) Sea un sistema de gestión de memoria basado en paginación bajo demanda. A lo largo de la ejecución de un proceso, ¿es posible que una dirección lógica sea traducida a diferentes direcciones físicas?

Sí. Gracias a la paginación bajo demanda, una página lógica puede ser intercambiada de memoria a disco y de disco a memoria diversas veces a lo largo de la ejecución de un proceso, pudiendo ocupar un frame diferente cada vez que vuelve a memoria.

c) Indicad cuál será el resultado de ejecutar el siguiente código (jerarquía de procesos creada, información mostrada por la salida estándar). Podéis asumir que ninguna llamada al sistema devolverá error.

```
main() {
  if (fork() == 0) fork();
  fork();
  execl("/bin/ls", "ls", NULL);
  exit(0);
}
```

El primer fork crea un proceso hijo. Como la condición del if sólo será cierta para el proceso hijo, únicamente éste ejecutará el fork interior al if, creando un nuevo proceso (nieto del inicial).
Los tres procesos ejecutarán el siguiente fork, creando tres nuevos procesos (un hijo, un nieto y un biznieto del proceso inicial).
Los seis procesos ejecutarán el exec, con lo que aparecerá seis veces la lista de ficheros del directorio actual.

d) Indicad un ejemplo de utilización de “lista de control de acceso” y de “capability” en Unix.

Un ejemplo de llista de control de acceso son los bits de protección rwxrwxrwx asociados a los ficheros.
Un ejemplo de capability es el file descriptro retornado por la llamada al sistema open.

e) ¿En qué consiste la condición de “No preemption” (No expropiación) necesaria para la existencia de deadlocks?

Consiste en que un recurso que ha sido asignado a un thread sólo puede ser liberado por ese mismo thread. Ningún otro proceso puede arrebatarle el recurso.

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	23/01/2016	09:00

2. Gestión Memoria

Tenemos un sistema que tiene las siguientes características:

- Tamaño dirección física/lógica: 24 bits.
- Tamaño memoria física: 256KB.
- Tamaño página/frame: 4KB

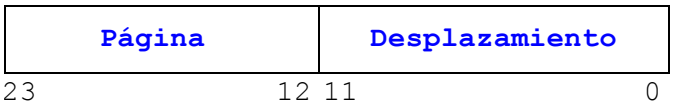
Tenemos la siguiente información de la ubicación de un proceso en memoria:

Regió	Adreça lògica d'inici	Mida	Mida Hex.
Codi	00000h	32 KB	8000h
Dades	0A000h	12 KB	3000h
Heap	0F000h	5 KB	1400h
Pila	07B800h	18 KB	4800h

Asumiendo un sistema de gestión de memoria por paginación bajo demanda, contestar, justificando las respuestas, a las siguientes preguntas:

a) Calcular el tamaño de cada uno de los campos de la dirección lógica y de la dirección física.

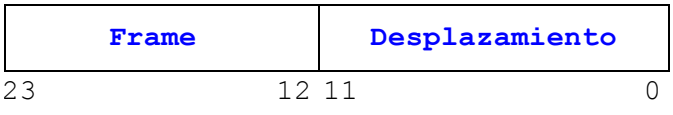
La dirección lógica tiene 24 bits que se descomponen de la siguiente forma:



Desplazamiento página = $\log_2$(Tamaño_pag) = $\log_2$(4096) = 12 bits

Página = $\log_2$(Número_pags) = $\log_2$(2^24/4096) = 12 bits

Para acceder a toda la memoria física necesitamos 24 bits, $\log_2$(32KB), que se descomponen de la siguiente forma:



Desplazamiento frame = $\log_2$(Tamaño_frame) = $\log_2$(4096) = 12 bits

Frame = $\log_2$(Número_frames) = $\log_2$(2^24/4096) = 12 bits

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	23/01/2016	09:00

b) Indicar en la siguiente tabla la memoria física asigna al proceso, indicando si la página pertenece a la región de código (C), datos (D), heap (H) o a la pila (P) y el número de página dentro de esa región (por ejemplo, C1, C2, D1, H1, P2).

Tabla de páginas

	V	P	Frame
00h	1	1	21h
01h	1	1	31h
02h	1	0	41h
03h	1	1	51h
04h	1	1	61h
05h	1	0	71h
06h	1	0	81h
07h	1	0	91h
08h	0	0	25h
09h	0	0	21h
0Ah	1	0	33h
0Bh	1	1	44h
0Ch	1	1	53h
0Dh	0	0	15h
0Eh	0	0	31h
0Fh	1	0	11h
10h	1	1	54h

	V	P	Frame
11h	0	0	41h
12h	0	0	44h
13h	0	0	21h
14h	0	0	45h
17h	0	0	33h
18h	0	0	12h
19h	0	0	25h
1Ah	0	0	66h
78h	0	0	01h
79h	0	0	62h
7Ah	1	0	49h
7Bh	1	1	47h
7Ch	1	0	57h
7Dh	1	1	48h
7Eh	1	0	67h
7Fh	1	1	47h

Memoria Física

	0H	1000H	2000H	3000H	4000H	5000H	6000H	7000H	8000H	9000H	A000H	B000H	C000H	D000H	E000H	F000H
0H																
10000H																
20000H		C1														
30000H		C2														
40000H					D2			P1, P5	P3							
50000H		C4		D3	H2											
60000H		C5														
70000H																

La dirección del bloque de memoria se puede obtener sumando la fila y la columna correspondientes. Por el ejemplo, un bloque de memoria ubicado en las coordenadas (3,3), empieza en la dirección 22200h ($20000h+2000h) y finaliza en la dirección 22FFFh ($20000h+3000h -1).

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	23/01/2016	09:00

c) Indicar si dicho sistema puede generar fragmentación, de que tipo y su cantidad.

Si, el sistema puede generar fragmentación interna. En concreto 3KB en la región de heap y 2KB en la región de pila.

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	23/01/2016	09:00

3. Gestión de Procesos

En el segundo ejercicio de la segunda práctica de la asignatura nos indicaban que durante la ejecución de los procesos, estos pueden recibir señales por parte del sistema operativo, otro procesos o del propio usuario.

a) Indicar, mediante un ejemplo real, como se podría forzar que el SO enviase una señal a si mismo sin utilizar la llamada al sistema kill.

```
alarm(10);
```

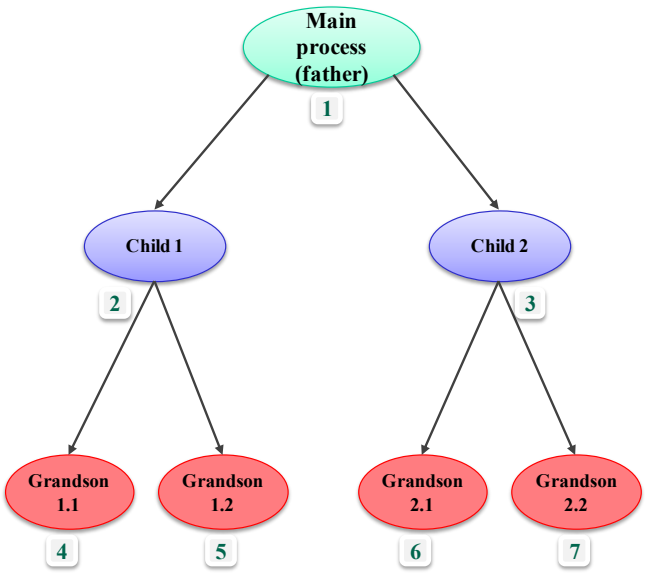
Mediante la llamada al sistema alarm, le indicamos al SO que queremos que nos envíe la señal SIGALRM dentro de un determinado número de segundos (en este caso 10 segs).

b) Indicar, mediante un ejemplo real, como puede un proceso de una aplicación enviar una señal a otro proceso.

Para enviar una señal a otro proceso, tenemos que utilizar la llamada al sistema kill, especificando el número de señal que queremos enviar y a que proceso/s queremos enviarla.

```
kill(getppid(), SIGUSR1);    // Enviamos señal SIGUSR1 a nuestro padre.
```

c) Atendiendo a la siguiente jerarquía de procesos y asumiendo que los procesos se han creado en orden (primero el proceso 1 y último el 7):



Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	23/01/2016	09:00

¿Sería posible que el proceso 2 pueda enviar una señal a proceso 4? ¿Y que el procesos 4 se la pueda enviar al proceso 7? Justificar vuestras respuestas.

El proceso 2 si que puede enviarle una señal al proceso 4, ya que tienen parentesco directo (el proceso 4 es hijo del proceso 2) y por lo tanto, el proceso 2 puede conocer el pid del proceso al que quiere enviarle la señal.

En el caso de los procesos 4 y 7, esto no es posible ya que el proceso 4 no conoce el pid del proceso 7, por lo que no podría enviarle una señal directamente.

d) En el siguiente fragmento de código de la solución del apartado 2.2 de la segunda práctica:

```
1.  void programing_signals()
2.  {
3.  struct sigaction myaction, oldaction;
4.
5.  myaction.sa_handler=handler_signals;
6.  myaction.sa_flags=SA_RESTART;
7.
8.  if (sigaction(SIGTERM, &myaction, &oldaction)<0)
9.              Error("Installing SIGTERM signal action.");
10.
11. if (sigaction(SIGQUIT, &myaction, &oldaction)<0)
12.             Error("Installing SIGQUIT signal action.");
13.
14. if (sigaction(SIGUSR1, &myaction, &oldaction)<0)
15.             Error("Installing SIGUSR1 signal action.");
16.
17. /*
18. if (sigaction(SIGKILL, &myaction, &oldaction)<0)
19.             Error("Installing SIGKILL signal action.");
20. */
21.
22. if (sigaction(SIGALRM, &myaction, &oldaction)<0)
23.             Error("Installing SIGALARM signal action.");
24.
25. }
```

i. Justificar porque es necesario asignar el flag SA_RESTART.

Para que el proceso que está bloqueado en una llamada al sistema (en este caso wait), pueda reanudar la ejecución de esta llamada al sistema una vez recibida la señal.

ii. ¿Por qué están comentadas las líneas 18-19?

Porque el SO no permite definir una rutina de tratamiento para la señal SIGKILL y al intentarlo la llamada al sistema nos devuelve un error.

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	23/01/2016	09:00

4. Concurrencia

Una barrera (barrier) es una herramienta que permite sincronizar un conjunto de threads: los threads se bloquearán en la barrera hasta que todos los threads del conjunto lleguen a la barrera.
Asumiremos que la barrera tendrá la siguiente interfície:

- `create_barrier(int N);` Inicializa una barrera para N threads.
- `barrier();` Punto de sincronización.

Un posible ejemplo de utilización sería:

```
/* Initializes a barrier for 3 threads */
create_barrier(3);

/* Thread creation */
pthread_create(..., thread1, ...);
pthread_create(..., thread2, ...);
pthread_create(..., thread3, ...);
...
```

```
thread1 ()
{
  ...

  printf ("Before 1\n");
  barrier ();
  printf ("After 1\n");
  pthread_exit (NULL);
}
```

```
thread2 ()
{
  ...

  printf ("Before 2\n");
  barrier ();
  printf ("After 2\n");
  pthread_exit (NULL);
}
```

```
thread3 ()
{
  ...

  printf ("Before 3\n");
  barrier ();
  printf ("After 3\n");
  pthread_exit (NULL);
}
```

Gracias a la barrera podemos garantizar que el resultado de ejecutar el código mostrará en primer término los tres mensajes “Before 1/2/3” (en cualquier orden) y a continuación los tres mensajes “After 1/2/3” (en cualquier orden).

a) Nos proponen esta posible implementación. Indicad justificadamente qué error(es) presenta.

```
struct{
  int count;
  int n;
  sem_t sem_mutex;
  sem_t sem_barrier;
} brr;
```

```
void create_barrier(int n) {
  brr.count = 0;
  brr.n = n;
  sem_init(brr.sem_mutex, 1);
  sem_init(brr.sem_barrier, 0);
}
```

```
void barrier()
{
  int i;

  sem_wait(brr.sem_mutex);
  brr.count++;
  if (brr.count == brr.n) {
    for (i=0; i < brr.n; i++) {
      sem_wait(brr.sem_barrier);
    }
  }
  sem_signal(brr.sem_mutex);

  sem_signal(brr.sem_barrier);
}
```

El problema es que las operaciones sobre sem_barrier están invertidas. Donde se hace el sem_wait debería hacerse el sem_signal, y viceversa. Ésto provoca que los primeros threads no se bloqueen en la barrera y que el último thread se quede bloqueado esperando un sem_signal sobre el semáforo.

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	23/01/2016	09:00

b) Indicad justificadamente si la siguiente implementación es correcta. Si es correcta, indicad cuál es el estado final de la barrera `brr` cuando los tres threads del ejemplo abandonen la barrera.

```
struct{
  int count;
  int n;
  sem_t sem_mutex;
  sem_t sem_barrier;
} brr;
```

```
void create_barrier(int n) {
  brr.count = 0;
  brr.n = n;
  sem_init(brr.sem_mutex, 1);
  sem_init(brr.sem_barrier, 0);
}
```

```
void barrier()
{
  int i;

  sem_wait(brr.sem_mutex);
  brr.count++;
  if (brr.count == brr.n) {
    for (i=0; i < brr.n; i++) {
      sem_signal(brr.sem_barrier);
    }
    sem_signal(brr.sem_mutex);
    return;
  }
  sem_signal(brr.sem_mutex);

  sem_wait(brr.sem_barrier);
}
```

Es correcta porque se actualiza el campo count en exclusión mútua, los threads (excepto el último) se bloquean en el semáforo sem_barrier y el último thread envía N signals sobre el semáforo para desloquejar a los threads.
El estado final es n=count=3, el contador de sem_mutex tendrá el valor 1 y el de sem_barrier también (se han hecho N sem_signals pero sólo se han hecho N-1 sem_waits).