grapa

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	16/01/2016	15:30

75.097 16 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	16/01/2016	15:30

1. Teoría

Contestad justificadamente las siguientes preguntes.

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

Esta transición típicamente se realiza cuando un proceso invoca una llamada al sistema bloqueante. Otro caso sería si el proceso recibe un signal SIGSTOP.

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 direcciones lógicas diferentes sean traducidas a una misma dirección física?

Sí, porque con paginación bajo demanda, una página puede ser explusada de memoria física y ser remplazado por otra página del mismo proceso. Por tanto, ambas páginas lógicas habrán tenido asoaciado el mismo frame físico.

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

El primer fork crea un nuevo proceso. El segundo fork es ejecutado por dos procesos y crea dos nuevos procesos. Los cuatro procesos ejecutan la llamada exec sobre el programa ls con lo que por stdout aparece la lista de ficheros del directorio actual por cuatriplicado.

d) Definid del concepto de “capability”.

La capability es una tupla (objeto, derechos) asociada a un domino. Para un dominio nos indica qué derechos tiene sobre un objeto.

e) ¿En qué consiste la condición de “Hold and Wait” (Retención y espera) necesaria para la existencia de deadlocks?

Consiste en que los threads tienen asignado un recurso en exclusión mútua (hold) y que está bloqueado esperando que el sistema le asigne otro recurso (wait).

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	16/01/2016	15:30

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:

Región	Dir. lógica de inicio	Tamaño	Tamaño Hex.
Código	00000h	32 KB	8000h
Datos	0A000h	12 KB	3000h
Heap	0F000h	5 KB	1400h
Pila	07B800h	18 KB	4800h

Asumiendo un sistema de gestión de memoria basado en paginación bajo demanda y que el estado actual de la asignación de memoria física al proceso es el siguiente:

Memoria Física

	0H	1000H	2000H	3000H	4000H	5000H	6000H	7000H	8000H	9000H	A000H	B000H	C000H	D000H	E000H	F000H
0H																
10000H							C5									
20000H		C1														
30000H				C3						H1	D1	H2		P4		
40000H															P2	P1
50000H																
60000H					D2				C6							
70000H																

Donde la letra indica una zona de memoria asignada a código (C), datos (D), heap (H) o pila (P) y el número indica el orden. La dirección del bloque de memoria se puede obtener sumando la fila y la columna correspondientes. Por el ejemplo, el primer bloque de datos del proceso 1, empieza en la dirección 3A000h (30000h+A000h) y finaliza en la dirección 3AFFFh (30000h+B000h-1).

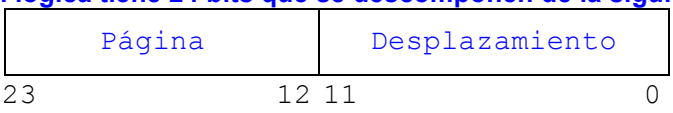
Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	16/01/2016	15:30

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.

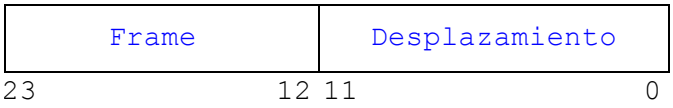
La dirección lógica tiene 24 bits que se descomponen de la siguiente forma:



Desplazamiento página = $\log_2$(Tam_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, que se descomponen de la siguiente forma:



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

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

b) A partir de la asignación de memoria física y el mapa de memoria lógica del proceso rellenar los campos de su tabla de páginas:

Tabla de páginas

	V	P	Frame
00h	1	1	C1 - 21h
01h	1	0	
02h	1	1	C3 - 33h
03h	1	0	
04h	1	1	C5 - 16h
05h	1	1	C6 - 68h
06h	1	0	
07h	1	0	
08h	0	0	
09h	0	0	
0Ah	1	1	D1 - 3Ah
0Bh	1	1	D2 - 64h
0Ch	1	0	
0Dh			
0Eh			
0Fh	1	1	H1 - 39h
10h	1	1	H2 - 3Bh

	V	P	frame
11h			
12h			
13h			
14h			
15h			
16h			
17h			
18h			
78h			
79h			
7Ah			
7Bh	1	0	
7Ch	1	1	P4 - 3Dh
7Dh	1	0	
7Eh	1	1	P2 - 4Eh
7Fh	1	1	P1 - 4Fh

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	16/01/2016	15:30

c) ¿Qué dirección física se corresponde con la dirección lógica 04085H? ¿Qué dirección lógica se corresponde con la dirección física 016234H?

- **Dir. Lógica 04085H -> Dir. física: 16085h**

  Página: 4h Despl: 085h

  Frame: TablaPáginas[Página] = TablaPáginas[4] = 16h

  Dir. física: 16h & 085h = 16085h

- **Dir. Física 016234H -> Dir. Lógica: 04234h**

  Frame: 16h Despl: 234h

  Entrada tabla páginas que contiene frame 16h: 04h

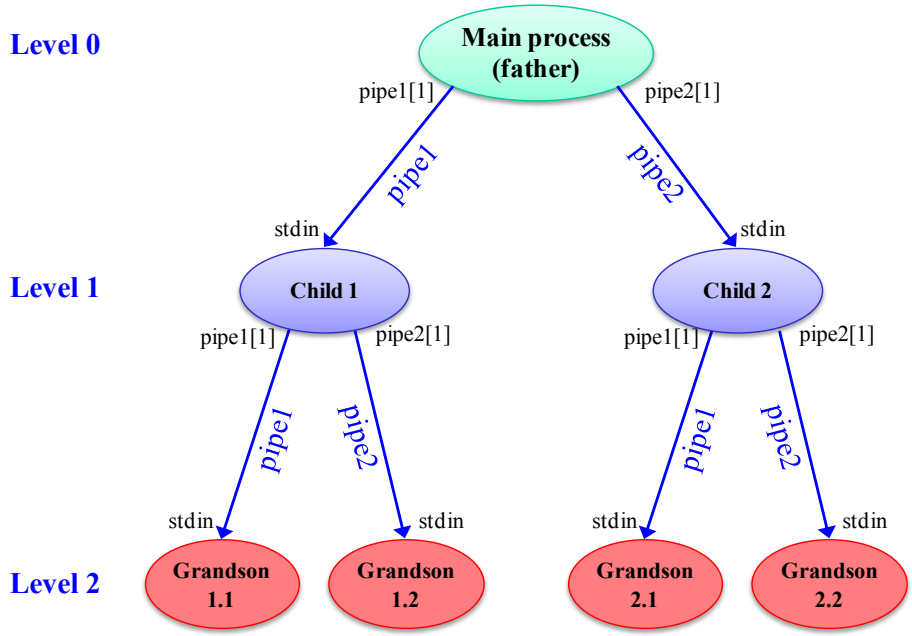
  Dir. lógica: 04h & 234h = 04234h

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	16/01/2016	15:30

3. Gestión de Procesos

En el ejercicio 3 de la práctica nos pedían realizar la siguiente comunicación mediante pipes para implementar una búsqueda binaría.




```
void create_childs(int level, int max_level)
{
    char cad[128];
    pid_t pid, pid_h1, pid_h2;
    int ret_h1, ret_h2;

    // Have to create childs?
    if (level<max_level)
    {
        int res_h1, res_h2;

        if (pipe(pipe_childleft)<0)
             Error("Creating pipe.");

        // Create left child.
        pid_h1=fork();
        if (pid_h1<0)
             Error("Fork process.");
        else
            if (pid_h1==0)
            {
                 char level_str[10], max_level_str[10];

                 close(0);
                 dup(pipe_childleft[0]);
                 close(pipe_childleft[0]);
                 close(pipe_childleft[1]);
```

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	16/01/2016	15:30

```
                sprintf(level_str,"%d",level+1);
                sprintf(max_level_str,"%d",max_level);
                execlp("./pas3_child","pas3_child",level_str, max_level_str, NULL);
                Error("Execlp Left Child");
            }

        close(pipe_childleft[0]);

        if (pipe(pipe_childright)<0)
             Error("Creating pipe.");

        // Create right child.
        pid_h2=fork();
        if (pid_h2<0)
            Error("Fork process.");
        else
            if (pid_h2==0)
            {
                char level_str[10], max_level_str[10];

                close(0);
                dup(pipe_childright[0]);
                close(pipe_childright[0]);
                close(pipe_childright[1]);
                close(pipe_childleft[1]);

                sprintf(level_str,"%d",level+1);
                sprintf(max_level_str,"%d",max_level);
                execlp("./pas3_child","pas3_child",level_str, max_level_str, NULL);
                Error("Execlp Right Child");
            }

        close(pipe_childright[0]);
    }
}

void do_search(int level, int max_level)
{
    int search_key, pos_key;

    if (read(0, &search_key,sizeof(int))<0)
          Error("Read search numbers");

    if (level<max_level)
    {   // Inner tree node -> propagate search.
          if (search_key<Pivot)
          {
                if (write(pipe_childleft[1],&search_key,sizeof(int))<0)
                      Error("Write numbers");
          }
          else
          {
                if (write(pipe_childright[1],&search_key,sizeof(int))<0)
                      Error("Write numbers");
          }
    }
    else
    {     // Local binary Search
          if ((pos_key=local_binary_search(search_key))>=0)
          {
                char cad[128];
```

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	16/01/2016	15:30

```
                    sprintf(cad,"Process %d -> Key %d found with value %s.\n", getpid(),
                            search_key, Keys[pos_key].value);
                    write(1,cad,strlen(cad));
                }
            }
        }
```

a) Indicar por que se necesita cerrar los descriptores de los pipes, tanto en el proceso padre como en los hijos. Justificar la respuesta.

Los procesos necesitan cerrar los descriptores de los pipes que no necesitan para que el SO les pueda informar correctamente cuando no haya ningún proceso en el otro extremo para leer o escribir la información. Esto lo controla el SO mediante el número de descriptores abiertos de cada uno de los extremos del pipe. Cuando este contador llega a cero, si un proceso quiere leer de un pipe en el cual no hay descriptores abiertos de escritura, entonces recibe un EOF como código de retorno de la lectura. Si un proceso intenta escribir en un pipe que no tiene descriptores abiertos de lectura, entonces el SO envía la señal SIGPIPE al proceso que realiza la escritura notificándole que los datos escritos nadie los va a poder leer.

b) En la práctica el proceso hoja era el encargado de mostrar el resultado de la búsqueda. Ahora queremos modificar el código para que ahora el resultado se devuelva al padre utilizando un pipe adicional que comunicará todos los procesos (salvo el principal) con el proceso principal.

¿Explicar que cambios habría realizar en el código para hacer esta modificación?

El proceso padre crearía un pipe adicional para notificar al proceso padre si han encontrado la clave y devolver su valor asociado. Este el descriptor de escritura de este pipe estará redirigido a la stdout de cada uno de los procesos hijos. Para ello, basta con redirigir la stdout del proceso principal al descriptor de escritura de su pipe. A partir de ese momento, al heredar todos sus procesos hijos/nietos/ect sus descriptores estándar, todos ellos podrían enviar información al proceso principal escribiendo en su stdout:

```
// Crear pipe.
if (pipe(pipe_main)<0)
   Error("Creating pipe.");

close(1);
dup(pipe_main[1]);
close(pipe_main[1]);
```

También es posible redirigir explícitamente la stdin del proceso padre a cada uno de los dos hijos creados.

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	16/01/2016	15:30

A la hora de enviar el resultado al proceso principal, los procesos hoja podrían escribir una estructura de datos con la siguiente información:

```
struct Result {
    int     pid;        //Pid del proceso que devuelve el resultado.
    boolean key_found; //¿Ha encontrado la clave?
    int     value;     //Valor asociado con esa clave.
}
```

El hijo escribiría el resultado con el siguiente código:

```
struct Result res;

res.pid = getpid();
// Local binary Search
if ((pos_key=local_binary_search(search_key))>=0)
{
    res.key_found = true;
    res.value = Keys[pos_key].value
}
else
    res.key_found = false;

if (write(1,&res,sizeof(struct Result))<0)
    Error("Write result");
```

El padre leería los resultados con el siguiente:

```
for (x=0;x<Leave_nodes;x++)
{
    // Receive result.
    if (read(pipe_main[0],&res,sizeof(struct Result))<0)
        Error("Read result");
    if (res.key_found)
    {
        char cad[128];
        sprintf(cad,"Process %d -> Key %d found with value %s.\n", res.pid,
                    search_key, res.value);
        write(1,cad,strlen(cad));
    }
}
```

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	16/01/2016	15:30

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, 1);
}
```

```
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);

  sem_wait(brr.sem_barrier);
}
```

El error es que el semáforo sem_barrier está inicializado a 1 con lo que el primer thread en llegar a la barrera no se bloqueará en el sem_wait sobre sem_barrier con lo que no esperará al resto de threads.

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	16/01/2016	15:30

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-1; i++) {
      sem_signal(brr.sem_barrier);
    }
    return;
  }
  sem_signal(brr.sem_mutex);

  sem_wait(brr.sem_barrier);
}
```

La solución es correcta porque se accede en exclusión mútua a brr.count, todos los threads excepto el último se bloquean en el sem_wait sobre sem_barrier y el último envia N-1 signals sobre sem_barrier y finaliza la ejecucion de la rutina.

El estado final será N y count = 3, el contador de sem_barrier tendrá el valor 0 y el de sem_mutex también (el último thread no ha hecho el sem_signal).