grapa

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	13/01/2016	12:00

75.097 13 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	13/01/2016	12:00

1. Teoría

Contestad justificadamente las siguientes preguntes.

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

Este cambio de estado se produce cuando el planificador de la cpu selecciona un proceso de la cola de preparados para asignarle la CPU.

b) Sea un sistema de gestión de memoria basado en paginación bajo demanda. ¿Es posible que el espacio lógico de un proceso sea mayor que la cantidad de memoria física instalada en la máquina?

Sí. Gracias a la paginación bajo demanda, el espacio lógico puede ser más grande que la memoria física porque el área de swap puede almacenar las páginas del espacio lógico que no caben en memoria física.

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

El primer creará un proceso hijo. El padre y el hijo hará el exec con lo que cambiarán de imagen y ejecutarán la orden ls, por tanto por stdout aparecerá dos veces la lista de ficheros del directorio actual. El resto de código no se ejecutará porque, al hacer un exec, el código ha sido substituido por el de ls.

d) Definid el concepto de “lista de control de acceso” (LCA).

Es una lista de parejas (derecho, domino) asociada a un objeto. Para cada dominio, indica qué derechos tiene sobre el objeto.

e) ¿En qué consiste la condición de “Circular Wait” (Espera circula) necesaria para la existencia de deadlocks?

Consiste en que existe un conjunto de 2 o más threads donde cada uno tiene concedido, como mínimo, un recurso en exclusión mútua y todos ellos están bloqueados esperando obtener un recurso que está concedido a otro de los threads del conjunto.

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	13/01/2016	12:00

2. Gestión Memoria

Tenemos un sistema que tiene las siguientes características:

- Tamaño dirección lógica: 20 bits.
- Tamaño dirección física: 18 bits.
- Tamaño memoria física: 256KB.
- Tamaño página/frame: 4KB

Asumiendo un sistema de gestión de memoria basado en paginación bajo demanda y la siguiente tabla de páginas para el proceso A:

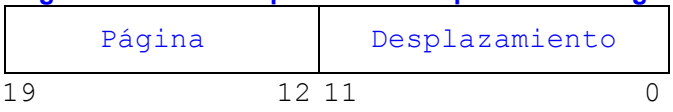
Tabla de páginas (proceso A)

	V	P	Frame		V	P	Frame
00h	0	0	22h	0Eh	0	0	17h
01h	0	0	32h	0Fh	0	0	28h
02h	1	1	21h	10h	1	1	26h
03h	1	1	31h	11h	1	1	3Eh
04h	1	0	12h	12h	0	0	3Bh
05h	1	0	23h	13h	0	0	0Ah
06h	1	0	34h	14h	0	0	1Ch
07h	1	0	13h	15h	0	0	2Dh
08h	1	0	01h				
09h	1	1	3Ah	FBh	0	0	3Fh
0Ah	1	0	3Dh	FCh	0	0	00h
0Bh	0	0	24h	FDh	1	0	11h
0Ch	0	0	35h	FEh	1	1	02h
0Dh	0	0	06h	FFh	1	0	09h

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 20 bits que se descomponen de la siguiente forma:



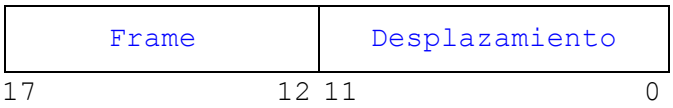
Desplazamiento página = $\log_2$(Tam_pag) = $\log_2$(4096) = 12 bits

Página = $\log_2$(Número_pags) = $\log_2$(2^20/4096) = 8 bits

Para acceder a toda la memoria física tenemos 18 bits:

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	13/01/2016	12:00



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

Frame = $\log_2$(Número_frames) = $\log_2$(2^18/4096) = 6 bits

b) Identificar el mapa de menoría lógica del proceso A.

Código: 02000h-07FFFh

Datos: 08000h-0AFFFh

Heap: 10000h-11FFFh

Pila: FD000h-FFFFFh

c) El sistema operativo quiere crear un nuevo proceso, B, que ejecute el mismo programa que el proceso A. Para ahora memoria física, interesa que ambos procesos compartan la región de código.

¿Se podría realizar esta compartición en el sistema de gestión de memoria utilizado?

Si que se podría realizar.

En caso afirmativo, definir la tabla de páginas para el proceso B de forma que soporte dicha compartición. No es necesario especificar las entradas para las regiones no compartidas.

Tabla de páginas (Proceso B)

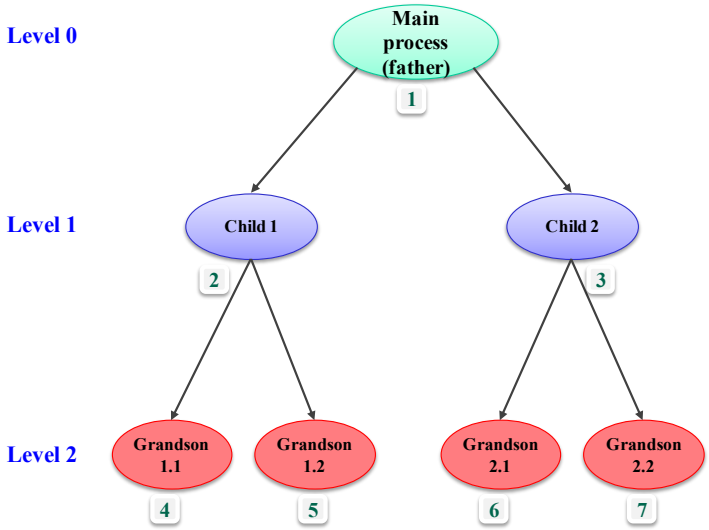
	V	P	Frame		V	P	Frame
00h				0Eh			
01h				0Fh			
02h	1	1	**21h**	10h			
03h	1	1	**31h**	11h			
04h	1	0	**12h**	12h			
05h	1	0	**23h**	13h			
06h	1	0	**34h**	14h			
07h	1	0	**13h**	15h			
08h	1	0	**01h**				
09h				78h			
0Ah				79h			
0Bh				7Ah			
0Ch				7Bh			
0Dh				7Ch			

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	13/01/2016	12:00

3. Gestión de Procesos

En la segunda práctica de la asignatura se pedía crear una jerarquía de procesos en árbol, en donde cada proceso imprimirá por pantalla su pid y el pid de su proceso padre.




```
int main(int argc, char *argv[])
{
    char cad[128];
    int result;

    result = Three_node(0,1);
   if (result>0)
    {
        sprintf(cad,"Result %d.\n", result);
        write(1,cad,strlen(cad));
        exit(0);
    }
    else Error("Error in Childs.\n");
}

int Three_node(int level, int pos)
{
    char cad[128];
    pid_t pid, pid_h1, pid_h2;
    int ret_h1, ret_h2;

    if (level<max_level)
   {
        int res_h1, res_h2;

        // Create left child.
        pid_h1=fork();
        if (pid_h1<0)
             Error("Fork process.");
        else
            if (pid_h1==0)
            {
                pos = (2*pos)+0;
```

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	13/01/2016	12:00

```
                sprintf(cad,"Process  Pid:%d  PPid:%d  (position:  %d).\n",  getpid(),
                getppid(),pos);
                write(1,cad,strlen(cad));
                res_h1=Three_node(level+1, pos);
                exit(res_h1);
             }

        pid_h2=fork();
        if (pid_h2<0)
            Error("Fork process.");
        else
            if (pid_h2==0)
            {
              pos = (2*pos)+1;
                sprintf(cad,"Process  Pid:%d  PPid:%d  (position:  %d).\n",  getpid(),
                      getppid(), pos);
                write(1,cad,strlen(cad));
                res_h2=Three_node(level+1, pos);
                exit(res_h2);
            }

        pid = wait(&ret_h1);
        if (pid<0)
            Error("Waiting child Process.");

        pid = wait(&ret_h2);
        if (pid<0)
            Error("Waiting child Process.");
    }

    if (level==max_level)
        return(pos);
    else
    {
        if (WIFEXITED(ret_h1)>0 && WIFEXITED(ret_h2)>0)
            return(WEXITSTATUS(ret_h1)+WEXITSTATUS(ret_h2));
        else
            return(-1);
    }
}
```

a) Explicar cómo pueden los procesos hijos calcular su posición. ¿Se podría utilizar el mismo método si lo procesos hijos ejecutasen un código diferente y realizasen un recubrimiento? Justificar las respuestas.

Los procesos hijos calculan su posición debido a que heredan del proceso padre los valores de la variable pos y calculan su posición multiplicando el valor de esta variable por 2 y añadiéndole uno si se trata del segundo hijo (nodo izquierdo).

Si los procesos ejecutasen un código diferente no heredarían las variables del padre y tendrían que recibir por parámetro su posición (calculada previamente antes de realizar el recubrimiento).

b) En el caso de que ampliásemos el ejemplo y creásemos una árbol jerárquico de 8 niveles, ¿tendríamos algún problema a la hora de devolver el resultado al proceso padre? Justificar la respuesta.

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	13/01/2016	12:00

Si podríamos tener el problema de que el código de retorno se podría desbordar, ya que con 8 bits solo se puede representar 256 valores y no podríamos sumar todas las posiciones de todos los procesos hijos.

c) Modificar dicho código para que ahora se asegure que siempre se muestran los mensajes en post-orden (primero el mensaje del hijo izquierdo, después el hijo derecho y por último el mensaje del padre).

```
int Three_node(int level, int pos)
{
    char cad[128];
    pid_t pid, pid_h1, pid_h2;
    int ret_h1, ret_h2;

    if (level<max_level)
    {
       int res_h1, res_h2;

        // Create left child.
        pid_h1=fork();
        if (pid_h1<0)
             Error("Fork process.");
        else
            if (pid_h1==0)
            {
                pos = (2*pos)+0;
                res_h1=Three_node(level+1, pos);
                exit(res_h1);
             }

        pid = wait(&ret_h1);
        if (pid<0)
            Error("Waiting child Process.");

        pid_h2=fork();
        if (pid_h2<0)
            Error("Fork process.");
        else
            if (pid_h2==0)
            {
               pos = (2*pos)+1;
                res_h2=Three_node(level+1, pos);
                exit(res_h2);
            }

        pid = wait(&ret_h2);
        if (pid<0)
            Error("Waiting child Process.");
    }


    sprintf(cad,"Process    Pid:%d    PPid:%d    (position:    %d).\n",    getpid(),
                getppid(),pos);
    write(1,cad,strlen(cad));

    if (level==max_level)
        return(pos);
    else
    {
        if (WIFEXITED(ret_h1)>0 && WIFEXITED(ret_h2)>0)
```

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	13/01/2016	12:00

```
                return(WEXITSTATUS(ret_h1)+WEXITST.ATUS(ret_h2));
            else
                return(-1);
        }
    }
```

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	13/01/2016	12:00

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

  sem_wait(brr.sem_barrier);
}
```

El error consiste en que la rutina barrier realiza únicamente N-1 sem_signals sobre el semáforo sem_barrier. Por tanto, uno de los N threads no se desbloqueará semsem_wait sobre sem_barrier, con lo que no abandonará la barrera.

Examen 2015/16-1

Asignatura	Código	Fecha	Hora inicio
Sistemas operativos	75.097	13/01/2016	12:00

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) {
    sem_signal(brr.sem_barrier);
  }
  sem_signal(brr.sem_mutex);

  sem_wait(brr.sem_barrier);
  sem_signal(brr.sem_barrier);
}
```

Esta implementación se comporta como indica la especificación. El semáforo sem_mutex regula el acceso en mutex al campo count. Todos los threads, excepto el último, se bloquean en sem_barrier. El último thread también se bloquea pero antes de hacerlo realiza un sem_signal sobre el semáforo. Por tanto, uno de los threads bloqueados se desbloquea y realiza un sem_signal sobre sem_barrier, con lo que desbloquea otro thread, y así sucesivamente hasta que se desbloquean todos.
El estado final de la variable es count=n=3, el contador de sem_mutex a 1 y el de sem_barrier también a 1 porque el último thread, al abandonar la barrera, ha hecho un sem_signal sobre el semáforo que no ha sido consumido por nadie.