Fundamentos de los Sistemas Operativos

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

fSO

EJERCICIOS DEL BT2
Gestión de Procesos
UT3, UT4, UT5 y UT6
Versión 1.1

Contenido

1 Cuestiones y Problemas sobre Procesos y Planificación (UT3 y UT4)

1.1 Cuestiones sobre conceptos teóricos

A continuación se enumeran un conjunto de cuestiones que profundizan en los conceptos estudiados en las clases de teoría, y son idóneas para conocer si se han comprendido o asumido la materia, además de practicar la capacidad de expresarnos sobre los nuevos conceptos aprendidos.

Intente responder de manera razonada a cada una de las siguientes cuestiones.

1. ¿Qué es un proceso?
2. ¿Qué diferencias existen entre los procesos orientados a E/S y los procesos orientados a CPU?
3. ¿Qué es la planificación de CPU?, ¿Cómo puede ejecutar múltiples procesos un sistema de tiempo compartido en un computador con una única CPU?
4. Describa los algoritmos de planificación que conoce e indique a qué tipo de procesos, procesos con ráfagas cortas de CPU o ráfagas largas de CPU, favorece
5. ¿En qué se diferencian la ejecución de procesos en primer y segundo plano (background) de UNIX ? ¿Cómo se pueden ejecutar órdenes del shell como procesos de primer plano y de segundo plano? De un ejemplo de cada una de ellas.
6. Describa cuáles son los estados principales en que puede encontrarse un proceso.
7. ¿Qué ventajas aporta la multiprogramación al rendimiento del sistema?
8. Describa las diferencias entre planificación a corto, medio y largo plazo.
9. ¿Cuántos procesos UNIX crearía la ejecución de la siguiente línea de ordenes del shell?

```
$cat f1 f2 f3 | grep comienza | wc -l >traza
```

10. ¿ Qué utilidad aporta almacenar en el bloque de control de proceso (PCB) el contador de programa?

1.2 Cuestiones de Verdadero y Falso

A continuación, se proponen un conjunto de cuestiones que ayudan al alumno a discernir sobre aspectos concretos de los conceptos estudiados. A partir de un enunciado el alumno tendrá que elegir la(s) opción(es) más idónea(s) o verdadera

11. Considere la ejecución del siguiente código:

```
#include <stdio.h>
int main()
{if (fork() == 0) { sleep(10);}
  else { sleep(5);}
  return 0;
}
```

Indique si las siguientes afirmaciones son verdaderas (V) o falsas (F)

V/F		V/F	
	Se genera un proceso hijo zombie		Se genera un proceso padre zombie
	Se genera un proceso hijo huérfano		Se genera un proceso padre huérfano

12. Indique si las siguientes afirmaciones son verdaderas o falsas para un sistema Unix:

a) La única forma de crear un proceso nuevo (con distinto PID) es mediante la llamada al sistema "exec".

b) El mandato "ps -la" se utiliza para visualizar los atributos de todos los ficheros del directorio actual, incluidos los ocultos.

1.3 Cuestiones sobre código

13. Dado el código de la figura, indique cuántos procesos se crean durante su ejecución. Justifíquelo adecuadamente.

```
#include <unistd.h>
int main(void) {
   int i;
     for (i=0; i < 5; i++)
     if (!fork()) break;
     while ( wait(NULL) != -1 );
}
```

14. Dado el código en C y POSIX de la figura, indique que mensaje mostrará por salida estándar cada uno de los procesos que se generan. Considere también el caso en que la ejecución de la llamada fork() no fuese posible.

```
#include <stdio.h>
#include <unistd.h>
main(){
int  pid, i, status;

 i=0;
 printf("Message 1: value %d\n",i);

 switch(pid=fork()){
   case -1: i++;
            printf("Message 2: value %d\n",i);
            exit(-1);
   case 0:  i++;
            printf("Message 3: value %d \n",i);
            exit(3);
   default: i++;
            printf("Message 4: value %d \n",i);
            while (wait(&status)>0);
 }
 printf("Message 5: value %d \n",i);
}
```

15. Indique cuántos procesos serán generados por la ejecución de este programa (incluyendo al original). Indique cuál de ellos será el último en terminar. Suponga que ninguna de las llamadas al sistema utilizadas provoca errores y que los procesos generados no reciben ninguna señal.

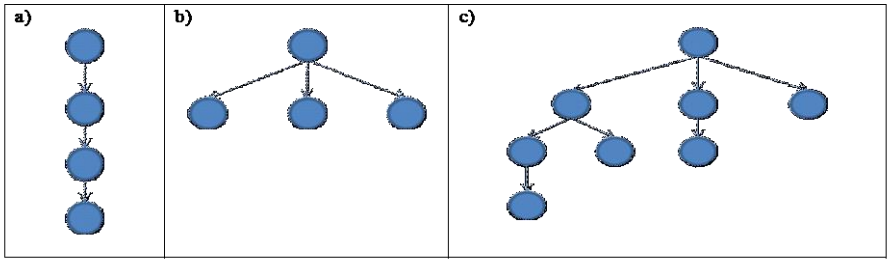
```
#include <stdlib.h>
int main(void) {
 int i, pid, pid2;

 for (i=0; i<4; i++) {
       pid=fork();
       if(pid==0) {
          pid2=fork();
          if (pid2!=0) wait(NULL);
          break; }
       wait(NULL);
    }
        return 0;
}
```

16. Dado el siguiente código:

```
int main (int argc, char **argv){
  int i;
  int pid;
  for (i=0; i<3; i++){
    pid = fork();
    if (/* CONDICIÓN*/) break;
  }
  while( wait(NULL)!=-1);
}
```

Indique que condición debe cumplirse para llegar a crear un grupo de procesos con la relación de parentesco reflejada en cada uno de los siguientes esquemas:



17. Dado el siguiente código cuyo fichero ejecutable ha sido generado con el nombre “Example1”.

```
1   /*** Example1***/
2   #include "all necessary header .h"
3   int main()
4   {
5     int status;
6     printf(“Message 1: before exec()\n”);
7     if (execl(“/bin/ps”,”ps”,”-la”, NULL)<0)
8        { printf(“Message 2: after  exec()\n”);
9          exit(1);}
10
11    printf(“Message 3: before  exit()\n);
12    exit(0);
    }
```

Indique de forma justificada:
a) Cuantos procesos se pueden llegar a crear durante su ejecución, en que número de líneas se lleva a cabo dicha creación y que mensaje imprime cada uno de ellos.
b) Cuando aparecerán por la salida estándar ”Message 3: before exit()”.
Nota: examen Noviembre 2011

18. Dado el siguiente código cuyo fichero ejecutable ha sido generado con el nombre “Example2”.

```
1   /*** Example2***/
2   #include  "all necessary header .h"
3   main()
4   { int i=0;
5     int pid;
6
7     while (i<2)
8     { switch((pid=fork()))
9       { case (-1): {printf("Error creating child\n");break;}
10        case (0): {printf("Child %i created\n",i);break;}
11      default: {printf("Father\n");}
12      }
13    i++;
14    }
15    exit(0);
16  }
```

Indique de forma justificada:

a) El número de procesos que se generan al ejecutarlo y el parentesco existente entre ellos.

b) Indique qué mensajes se mostrarán en pantalla, si la llamada fork() siempre tiene éxito.

Nota: examen Noviembre 2011

19. Dado el siguiente código cuyo fichero ejecutable ha sido generado con el nombre “Example3”.

```
    /*** Example3***/
    #include " all necessary header .h"
1
2   int main()
3   { int val, res;
4     int status;
5     printf("Message  1: before  exec()\n");
6     val=fork();
7     if (val==0)
8     {
9      if (execl("/bin/ps","ps","-la", NULL)<0)
10       { printf("Message 2: after  exec()\n");
11         exit(1);}
12    }
13    printf("Message 3: before  wait()\n);
14    while ((res=wait(&status))>0);
15     printf("Message 4: after wait\n");
16    exit(0);
    }
```

Indique de forma justificada:

a) Cuantos procesos se crean durante su ejecución, número de línea donde se crea cada proceso y líneas de código que ejecuta cada uno de los procesos creados.

b) Qué procesos muestran en pantalla el mensaje de la línea 15 (“Message 4:..”)

Nota: examen Enero 2012

20. Dado el siguiente código en C y POSIX correspondiente a un proceso que denominaremos Example4:

```
1  /***Example4***/
2  #include " all necessary header .h"
3  #define N 3
4  main() {
5    int i = 0;
6    pid_t pid_a;
7
8    while (i<N)
9    { pid_a = fork();
10     switch (pid_a)
11     { case -1:
12         printf("Error creating child...\n");
13         break;
14       case 0:
15         printf("Message 1: i = %d \n", i);
16         if (i < N-1) break;
17         else exit(0);
18       default:
19         printf("Message 2: i = %d \n", i);
20         while (wait(NULL)!=-1);
21      }
22     i++;
23   }
24   printf("Message 3: i=%d\n",i);
25   exit(0);
26 }
```

a) Represente el árbol de procesos generado al ejecutarlo e indique para cada proceso el valor de la variable "i" en el instante de su creación.

b) Indique de forma justificada si existe o no la posibilidad de que los hijos creados queden huérfanos o zombies.

Nota: examen Noviembre 2013

21. Suponga que el siguiente código en C y POSIX correspondiente a un proceso denominado Example5 que se ejecuta con éxito:

```
1  /***Example5***/
2  #include " all necessary header .h"
3
4  int main()
5  { pid_t val;
6     printf("Message 1: before exec()\n");
7     fork();
8     execl("/bin/ls","ls","-la", NULL);
9     val = fork();
10    if (val==0)
11    {execl("/bin/ps","ps","-la", NULL);
12     printf("Message 2: after  exec()\n");
13        exit(1)
14     }
15    printf("Message 3: before exit()\n);
16   exit(0);
     }
```

a) Indique de forma justificada el número de procesos que se crean al ejecutar Example5 y el parentesco entre ellos.

b) Indique de forma justificada que mensajes e información ~~se muestra~~se muestra por pantalla como consecuencia de ejecutar Example5.

Nota: examen Noviembre 2013

22. Dado el siguiente código del proceso Example6

```
/***Example6***/
#include " all necessary header .h"

int main(void)
{ int val;
  printf("Message 1\n");
  val=fork();
  /** Aquí deben ir sus modificaciones **/
  sleep(5);
  printf("Message 2\n");
  return 0;
}
```

a) Indique qué modificaciones serían necesarias introducir en Example6 para que el proceso hijo se quede huérfano y sea adoptado por el proceso INIT(). (Nota: Utilice el sleep() e instrucciones en C).
b) Indique qué modificaciones serían necesarias introducir en Example6 para que el proceso hijo se quede zombie durante un tiempo. (Nota: Utilice el sleep() e instrucciones en C).
c) Indique de forma justificada en qué instrucciones del códigoExample6 puede asegurarse que ocurrirá un cambio de contexto en la CPU y en cuáles se producirá un cambio de estado en Example6.
Nota: examen Noviembre 2013

23. Considera los dos programas E y F, obtenidos al compilar los códigos fuente siguientes:

```
/**Código Programa E.c **/
#include <…
int pid;
main(){
  pid=fork();
  if (pid==0)
     {sleep(3);}
  execl("./F","F",NULL);
}
```

```
/**Código Programa F.c **/
#include <…
int pid;
main(){
  pid=fork();
  if (pid==0)
     { sleep(1);}
  execl("/bin/date","date","+%S",NULL);
}
```

Nota: La orden "date +%S" imprime en pantalla los segundos de la hora actual. Por ejemplo a las 20:30:12, esta orden imprime "12".

Suponga que los ejecutables de E y F se encuentran en el directorio de trabajo y que se ejecutan sin incidencias. Indique razonadamente:

a. Cuántos procesos se crean al ejecutar la orden "**.**/F" y qué parentesco existe entre ellos.
b. Cuál será la salida por pantalla, si se ejecuta la orden "**.**/F" a las 09:10:25.
c. Cuántos procesos se crean al ejecutar la orden "**.**/E" " y qué parentesco existe entre ellos.
d. Cuál será la salida por pantalla, si se ejecuta la orden "**.**/E" a las 09:10:25.

Nota: examen Enero 2014

1.4 Problemas de Planificación

24. Sean los siguientes procesos a ejecutar en un sistema:

Proceso	Instante de llegada (t)	Orden llegada	CPU	Prioridad
A	0	1	8	3
B	0	2	1	1
C	0	3	2	3
D	0	4	1	4
E	0	5	5	2

Represente el diagrama de ocupación de la CPU y calcule los tiempos medios de espera y retorno en el sistema para los siguientes algoritmos

a. FCFS
b. Round-Robin (q=1)
c. SJF
d. Prioridades no expulsivas (a mayor número de prioridad, mayor prioridad)

25. Suponga que a un sistema llegan de forma simultánea 5 procesos (A, B, C, D, E), siendo el perfil de ejecución de cada uno de ellos el siguiente: 1000 unidades de tiempo de CPU, 5 de IMPRESORA y otras 1000 de CPU. Si en dicho sistema se aplica una política de planificación circular (Round-Robin) con quantum q=5, ¿Cuáles serán los tiempos de retorno y espera para cada uno de los 5 procesos? Justifique la respuesta. (Sugerencia: no es muy recomendable hacer una traza completa).

1000 ut CPU	5 ut IMP	1000 ut CPU

25.

26. Sea un algoritmo de planificación que es una versión modificada del Round-Robin tradicional para dar mejor servicio a los procesos que ya se han ejecutado durante un cierto periodo de tiempo que a los recién llegados. La cola de preparados se divide en dos: una de procesos NUEVOS y otra de ACEPTADOS. Se escoge siempre para ejecución un proceso de la cola de ACEPTADOS mediante una estrategia Round-Robin, y los procesos que llegan al sistema esperan en la cola de NUEVOS hasta que pueden pasar a la de ACEPTADOS.

Cuando un proceso llega al sistema su prioridad es 0, y en cada unidad de tiempo el algoritmo calcula las prioridades para todos los procesos de la forma siguiente:

- Si un proceso está en la cola de NUEVOS, se incrementa su prioridad en un factor a.
- Si un proceso está en la cola de ACEPTADOS se incrementa su prioridad en un factor b.

Cuando la prioridad de un proceso NUEVO se hace mayor o igual a la de cualquier proceso de la cola de ACEPTADOS, el proceso NUEVO se inserta en ella. En caso de que se vacíe la cola de ACEPTADOS, se introduce en ella el proceso más prioritario de la cola de NUEVOS. Un proceso es más prioritario cuanto mayor sea el valor numérico de su prioridad.

a. Suponiendo a=2, b=1 y q=1 y la siguiente situación, represente el diagrama de ocupación de la CPU y calcule los tiempos de espera y de retorno del sistema:

Proceso	Instante llegada	Ráfaga de CPU
A	0	5
B	1	4
C	3	2
D	9	6
E	11	3

b. Analice el comportamiento en los casos a>>b, b>=a, siendo a, b>0

27. Un sistema dispone de un planificador a largo plazo (PLP) y de otro a corto plazo (PCP), que funcionan de la siguiente manera: el PCP utiliza un algoritmo con prioridades expulsivas y el PLP utiliza una estrategia FCFS. Los procesos nuevos entran por el PLP. Se admiten en memoria un máximo de tres procesos. El PLP pasa un proceso al PCP cuando hay menos de tres procesos ya admitidos. Partiendo de la siguiente tabla de procesos y teniendo en cuenta que a menor número más prioridad.

Proceso	Instante llegada	Tiempo CPU	Prioridad
A	0	2	4
B	1	4	3
C	3	4	2
D	5	1	1
E	6	2	3

Represente el diagrama de ocupación de la CPU y calcule los tiempos medios de espera y de retorno del sistema

28. Sea un algoritmo de planificación multicolas con realimentación donde cada cola se gestiona con un Round-Robin de quantum q_i_=2*_i_. La planificación entre colas es de tipo prioridades no expulsivas, la cola más prioritaria es la 0. Un proceso pasará de la cola i a la cola i+1 cuando agote su quantum qi sin finalizar su ejecución. Los procesos nuevos y los procedentes del estado SUSPENDIDO entran por la cola 0.

Suponiendo que el s.o. no consume tiempo y que las operaciones de E/S se efectúan sobre el mismo dispositivo (gestionado de forma FCFS), dibujar el diagrama temporal de ocupación del procesador y del dispositivo de E/S para los procesos A, B, C y D que se ilustran a continuación. Muéstrese también la situación de la cola de PREPARADOS.

Calcule también los tiempos de retorno y espera (tanto del procesador como del disco) para cada proceso.

Proceso	Instante llegada	Perfil Ejecución
A	0	3 CPU + 2 E/S + 2 CPU
B	1	4 CPU + 1 E/S + 1 CPU
C	3	2 CPU + 1 E/S + 3 CPU
D	4	1 CPU + 2 E/S + 1 CPU

2 Cuestiones sobre Hilos de Ejecución y Sincronización (UT5 y UT6)

2.1 Cuestiones sobre Conceptos Teóricos

29. Conteste de forma concreta y concisa a las siguientes preguntas:
   a. ¿Qué es un programa concurrente?
   b. ¿Qué es una condición de carrera?
   c. ¿Qué es una sección critica?
   d. ¿Qué condiciones deben cumplir los protocolos de las secciones críticas?

30. Cite al menos tres atributos que podríamos encontrar en el "bloque de control de hilo".

31. Los semáforos no sólo sirven para resolver el problema de la sección crítica. Enumera al menos otras dos aplicaciones de los semáforos y ponga un ejemplo simple de cada una de estas aplicaciones

32. Justifique de forma razonada que las operaciones P(S) y V(S) se han de ejecutar de forma atómica para que los semáforos funcionen correctamente.

33. Razone adecuadamente si el siguiente código es una buena solución al problema de la sección crítica, donde S es un semáforo compartido por todos los procesos o hilos con acceso a dicha sección crítica, con valor inicial 1. Además, se sabe que la política utilizada por el sistema operativo para reactivar a los procesos suspendidos es LIFO (Last In, First Out).

```
while(1) {
   P(S);
   Sección crítica
   V(S);
   Sección restante
}
```

2.2 Cuestiones de Verdadero y Falso

34. Indique para cada una de las siguientes afirmaciones si es verdadera (V) o falsa (F).

	Un proceso puede suspenderse al realizar una operación V sobre un semáforo
	Un proceso siempre se suspende al realizar una operación P sobre un semáforo
	Un proceso siempre se suspende al realizar una operación espera (wait) sobre una variable condición
	Un proceso siempre despierta a otro proceso al realizar una operación V sobre un semáforo

35. Indique para cada una de las siguientes afirmaciones si es verdadera (V) o falsa (F).

	Mediante la función pthread_self() se puede obtener el identificador de un hilo.
	Si los hilos están soportados por el sistema operativo, siempre cuesta más un cambio de contexto entre procesos que entre hilos.

	Si el soporte a hilos se encuentra en el núcleo, al suspenderse uno de los hilos, los demás podrán seguir ejecutándose.
	El cambio de contexto entre hilos de ejecución soportados a nivel de usuario es más rápido que el de hilos soportados a nivel de núcleo.

2.3 Cuestiones sobre Código

36. Indique las cadenas que imprime el programa en la Terminal tras su ejecución. Justifique su respuesta. Nota: retraso(n) realiza un retraso de n milisegundos

```
void * funcion_hilo1(void * arg)
{
  retraso(4000+rand()%1000);
  printf("Hola Don Pepito\n");
  return null;
}
```

```
void * funcion_hilo2(void *
arg) {
  retraso(4000+rand()%1000);
  printf("Hola Don Jose\n");
  return null;
}
```

```
int main (void) {
  pthread_t th1,th2;
  pthread_attr_t atrib;

  pthread_attr_init( &atrib );

  printf("Eran dos tipos requeeeteeefinos...\n");
  pthread_create( &th1, &atrib, function_hilo1, null);
  pthread_create( &th2, &atrib, function_hilo2, null);
  exit(0);
}
```

37. El siguiente programa implementa un algoritmo que pretende resolver el problema de la Sección Crítica de dos hilos de ejecución: hilo_0 e hilo_1. ¿Es correcta la solución propuesta?
Nota: Analice la implementación en términos de exclusión mutua, progreso y espera limitada.

```
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>


int turno=0;


void *hilo_0(void *p) {
  while(1) {
    while (turno != 0);
    /* Sección crítica */
    turno = 1;
    /* Sección restante */
 }
}
```

```
void *hilo_1(void *p) {
  while(1) {
    while (turno != 1);
    /* Sección crítica */
    turno = 0;
    /* Sección restante */
 }
}

int main(int argc, char **argv){
   pthread_t h_0;
   pthread_t h_1;
   pthread_create(&h_0,NULL,hilo_0,(void *)0);
   pthread_create(&h_1,NULL,hilo_1,(void *)0);
   pthread_join(h_0, NULL);
   pthread_join(h_1, NULL);
}
```

38. Observe el siguiente fragmento de código correspondiente a dos hilos que pertenecen al mismo proceso y se ejecutan concurrentemente. Indique qué recursos compartidos aparecen en el código y qué mecanismos se emplean para evitar las condiciones de carrera.

Nota: Las variables y funciones que no están definidas dentro de las funciones agrega y resta son definidas como globales.

```
Void *agrega (void *argumento)
{
  int ct,tmp;

  for (ct=0;ct<REPE;ct++)
  {
   while(test_and_set(&llave)==1);
   tmp=V;
   tmp++;
   V=tmp;
   llave=0;
  }

  printf("->AGREGA (V=%ld)\n",V);
  pthread_exit(0);
}
```

```
void *resta (void *argumento)
{
  int ct,tmp;

  for (ct=0;ct<REPE;ct++)
  {
   while(test_and_set(&llave)==1);
   tmp=V;
   tmp--;
   V=tmp;
   llave=0;
  }

 printf("->RESTA (V=%ld)\n", V);
  pthread_exit(0);
}
```

39. Suponiendo que la variable global llave tiene valor inicial 0 y que la función test_and_set está definida correctamente, comenta las diferencias entre las siguientes dos soluciones al problema de la sección crítica atendiendo al tiempo de ejecución observado en cada una de ellas.
Nota: la función usleep() suspende al hilo que la invoca una cantidad de microsegundos.

```
/* Solución a */
void *hilo(void *p) {

  while(1) {
   while (test_and_set(&llave));
   /* Sección crítica */
   llave = 0;
   /* Sección restante */
  }
}
```

```
/* Solución b */
void *hilo(void *p) {

  while(1) {
   while(test_and_set(&llave))
        usleep(1000);
   /* Sección crítica */
   llave = 0;
   /* Sección restante */
  }
}
```

```
/* Solución b */
void *hilo(void *p) {

  while(1) {
   while(test_and_set(&llave))
        usleep(1000);
   /* Sección crítica */
   llave = 0;
   /* Sección restante */
  }
}
```

40. Dado el siguiente código, y asumiendo que todos los semáforos tenían valor inicial cero, que los hilos presentados están soportados por el núcleo del sistema operativo, se encontraban en la cola de preparados en el orden H1, H2, H3 (es decir, H1 será el primero en obtener la CPU y H3 el último) y que se utiliza un algoritmo de planificación FCFS:

H1	H2	H3
P(S3);	P(S2);	V(S2);
P(S1);	V(S4);	V(S1);
V(S1);	V(S3);	P(S1);
V(S3);	P(S3);	P(S4);

Indique el orden de terminación de dichos hilos. En caso de que algún hilo no termine, indique en qué operación se ha quedado suspendido.