Fonaments dels Sistemes Operatius

Departament d'Informàtica de Sistemes i Computadores (DISCA)
Universitat Politècnica de València

fSO

EXERCICIS DEL BT2
Gestió de Processos
UT3, UT4, UT5 i UT6
Versió 1.1

Contingut

1 Qüestions i problemes sobre processos i planificació (UT3 i UT4)

Qüestions sobre conceptes teòrics

A continuació s'enumeren un conjunt de qüestions que profunditzen en els conceptes estudiats en les clastes de teoria, i són idònies per a conèixer si s'ha comprès o assumit la matèria, a més de practicar la capacitat d'expressar-nos sobre els nous conceptes apressos.

Intenta respondre de manera raonada a cadascuna de les següents qüestions.

1. Què és un procés?
2. Quines diferències existeixen entre els processos orientats a E/S i els processos orientats a CPU?
3. Què és la planificació de CPU?, Com pot executar múltiples processos un sistema de temps compartit en un computador amb una única CPU?
4. Descriu els algorismes de planificació que coneixes e indica a quin tipus de processos, amb ràfegues curtes de CPU o ràfegues llargues de CPU, afavoreix
5. En què es diferencien l'execució de processos en primer i segon pla (background) de UNIX ? Com se poden executar ordres del shell com processos de primer pla i de segon plano? Dona un exemple de cadascuna d'elles.
6. Descriu quins són els estats principals en que pot trobar-se un procés.
7. Quins avantatges aporta la multiprogramació al rendiment del sistema?
8. Descriu les diferències entre planificació a curt termini i a llarg termini.
9. Quants processos UNIX crearia l'execució de la següent línia d'ordres del shell?

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

10. Quina utilitat aporta emmagatzemar en el bloc de control de procés (PCB) el comptador de programa?

Qüestions de veritable i fals

A continuació es proposen un conjunt de qüestions que ajuden a l'alumn@ a discernir sobre aspectes concrets dels conceptes estudiats. A partir d'un enunciat l'alumn@ haurà de triar la(es) opció(s) més idònia(s) o veritable

11. Considera l'execució del següent codi:

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

Indica si les següents afirmacions son veritables (V) o falss (F)

V/F		V/F	
	Es genera un procés fill zombi		Es genera un procés pare zombi
	Es genera un procés fill orfe		Es genera un procés pare orfe

12. Indica si les següents afirmacions són veritables o falses per a un sistema Unix:

a) L'única forma de crear un procés nou (amb distint PID) es mitjançant la crida al sistema "exec".

b) El comandament "ps -la" s'utilitza per a visualitzar els atributs de tots els fitxers del directori actual, incloent els ocults.

Qüestions sobre codi

13. Donat el codi de la figura, indica quants processos es creen durant la seua execució. Justifica-ho adequadament.

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

14. Donat el codi en C i POSIX de la figura, indica quin missatge mostrarà per sortida estàndard cadascun dels processos que es generen. Considera també el cas en que l'execució de la crida fork() no fora possible.

```
#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. Indica quants processos seran generats per l'execució d'aquest programa (incloent a l'original). Indica quin d'ells serà l'últim en acabar. Suposa que cap de les crides al sistema utilitzades provoca errors i que els processos generats no reben cap senyal.

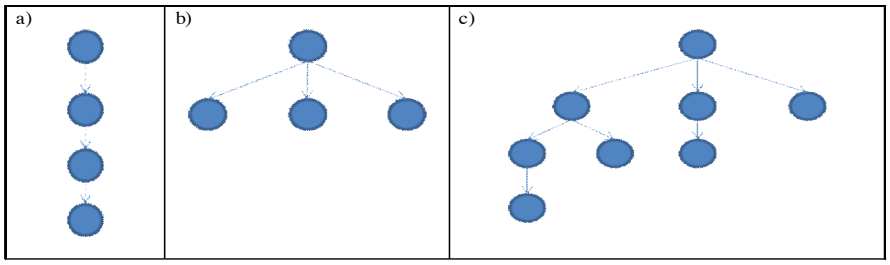
```
#include <stdlib.h>
int main(void) {
 int i, pid, pid2;

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

16. Donat el següent codi:

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

Indica quina condició ha d'acomplir-se per a arribar a crear un grup de processos amb la relació de parentiu reflectida en cadascun dels següents esquemes:



17. Donat el següent codi el fitxer executable del qual ha sigut generat amb el nom "Exemple1".

```
1   /*** Exemple1***/
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);
    }
```

Indica de forma justificada:
a) Quants processos se poden arribar a crear durant la seua execució, en quin nombre de línies es duu a terme aquesta creació i quin missatge imprimeix cadascun d'ells.
b) Quan apareixerà per la sortida estàndard el missatge "Message 3: before exit()".
Nota: examen novembre 2011

18. Donat el següent codi el fitxer executable del qual ha sigut generat amb el nombre “Exemple2”.

```
1   /*** Exemple2***/
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  }
```

Indica de forma justificada:

a) El nombre de processos que se generen a l’executar-lo i el parentiu existent entre ells.

b) Indica què missatges se mostraran en pantalla, si la crida fork() sempre te èxit.

Nota: examen novembre 2011

19. Donat el següent codi el fitxer executable del qual ha sigut generat amb el nom “Exemple3”.

```
    /*** Exemple3***/
    #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);
    }
```

Indica de forma justificada:

a) Quants processos es creen durant la seua execució, nombre de línia on se crea cada procés i línies de codi que executa cadascun dels processos creats.

b) Quins processos mostren en pantalla el missatge de la línia 15 (“Message 4:..”)

Nota: examen gener 2012

20. Donat el següent codi en C i POSIX corresponent a un procés que denominarem Exemple4:

```
1   /***Exemple4***/
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) Representa l'arbre de processos generat a l'executar-lo i indica per a cada procés el valor de la variable "i" en l'instant de la seua creació.

b) Indica de forma justificada si existeix o no la possibilitat de que els fills creats queden orfes o zombis.

Nota: examen novembre 2013

21. Suposa que el següent codi en C i POSIX corresponent a un procés denominat Exemple5 que s'executa amb èxit:

```
1   /***Exemple5***/
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) Indica de forma justificada el nombre de processos que se creen al executar Exemple5 i el parentiu entre ells.

b) Indica de forma justificada que missatges e informació es mostra por pantalla com conseqüència d'executar Exemple5.

Nota: examen novembre 2013

22. Donat el següent codi del procés Exemple6

```
    /***Exemple6***/
1   #include " all necessary header .h"
2
3   int main(void)
4   { int val;
5     printf("Message 1\n");
6     val=fork();
7     /** Aquí deben ir sus modificaciones **/
8     sleep(5);
9     printf("Message 2\n");
10    return 0;
11  }
```

a) Indica quines modificacions serien necessàries introduir en Exemple6 per a que el procés fill es quede orfe i siga adoptat pel procés INIT(). (Nota: Utilitze la crida sleep() i instruccions en C).
b) Indica quines modificacions serien necessàries introduir en Exemple6 per a que el procés fill es quede zombi durant un temps. (Nota: Utilitze la crida sleep() i instruccions en C).
c) Indica de forma justificada en quines instruccions del codi Exemple6 pot assegurar-se que ocorrerà un canvi de context en la CPU i en quines es produirà un canvi d'estat en Exemple6.
Nota: examen novembre 2013

23. Considera els dos programes E i F, obtinguts al compilar els codis font següents:

/**Codi Programa E.c **/	/**Codi Programa F.c **/

#include <…
int pid;
main(){
  pid=fork();
  if (pid==0)
  {sleep(3);}
  execl("./F","F",NULL);
}
	#include <…
	int pid;
	main(){
	  pid=fork();
	  if (pid==0)
	  { sleep(1);}
	  execl("/bin/date","date","+%S",NULL);
	}

Nota: L'ordre "date +%S" imprimeix en pantalla els segons de l'hora actual. Per exemple a les 20:30:12, aquesta ordre imprimeix "12".

Suposa que els executables de E i F es troben en el directori de treball i que s'executen sense incidències. Indica raonadament:

a. Quants processos es creen a l'executar l'ordre "./F" i quin parentiu existeix entre ells.
b. Quina serà la sortida per pantalla, si s'executa l'ordre "./F" a les 09:10:25.
c. Quants processos es creen a l'executar l'ordre "./E" " i quin parentiu existeix entre ells.
d. Quina serà la sortida per pantalla, si s'executa l'ordre "./E" a les 09:10:25.

Nota: examen gener 2014

Problemes de planificació

24. Siguen els següents processos a executar en un sistema:

Procés	Arribada	Ordre	CPU	Prioritat
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

Representa el diagrama d'ocupació de la CPU i calcula els temps mitjans d'espera i retorno en el sistema per a els següents algorismes

a. FCFS
b. Round-Robin (q=1)
c. SJF
d. Prioritats no expulsives (a major número de prioritat, major prioritat)

25. Suposa que a un sistema arriben de forma simultània 5 processos (A, B, C, D,E), sent el perfil d'execució de cadascun d'ells el següent: 1000 unitats de temps de CPU, 5 de IMPRESSORA i altres 1000 de CPU. Si en aquest sistema s'aplica una política de planificació circular (Round-Robin) amb quantum q=5, ¿Quins seran els temps de retorn i espera per a cadascun dels 5 processos?. Justifica la resposta. (Sugerència: no es molt recomanable fer una traça completa).

26. Siga un algorisme de planificació que es una versió modificada del Round-Robin tradicional per a donar millor servei als processos que ja s'han executat durant un cert període de temps que als que acaben d'arribar. La cua de preparats se divideix en dos: una de processos NOUS i una altra d'ACCEPTATS. Es tria sempre per a execució un procés de la cua de ACCEPTATS mitjançant una estratègia Round-Robin, i els processos que arriben al sistema esperen en la cua de NOUS fins que poden passar a la d'ACCEPTATS.

Quan un procés arriba al sistema la seua prioritat és 0, i en cada unitat de temps l'algorisme calcula les prioritats per a tots els processos de la forma següent:

- Si un procés està en la cua de NOUS, s'incrementa la seua prioritat en un factor a.
- Si un procés està en la cua de ACCEPTATS s'incrementa la seua prioritat en un factor b.

Quan la prioritat d'un procés NOU es fa major o igual a la de qualsevol procés de la cua d'ACCEPTATS, el procés NOU s'insereix en ella. En cas de que se buide la cua d'ACCEPTATS, s'introdueix en ella el procés més prioritari de la cua de NOUS. Un procés és més prioritari com major siga el valor numèric de la seua prioritat.

a. Suposant a=2, b=1 i q=1 i la següent situació, representa el diagrama d'ocupació de la CPU i calcula els temps d'espera i de retorno del sistema:

Procés	Instant arribada	Ràfega de CPU
A	0	5
B	1	4
C	3	2
D	9	6
E	11	3

b. Analitze el comportament en els casos a>>b, b>=a, sent a, b>0

27. Un sistema disposa d'un planificador a llarg termini (PLP) i de un altre a curt termini (PCP), que funcionen de la següent manera: el PCP utilitza un algorisme amb prioritats expulsives i el PLP utilitza una estratègia FCFS. Los processos nous entren pel PLP. S'admetien en memòria un màxim

de tres processos. El PLP passa un procés al PCP quan hi ha menys de tres processos ja admesos. Partint de la següent tabla de processos i tenint en compte que a menor nombre més prioritat.

Procés	Instant llegada	Temps CPU	Prioritat
A	0	2	4
B	1	4	3
C	3	4	2
D	5	1	1
E	6	2	3

Representa el diagrama d'ocupació de la CPU i calcula els temps mitjans d'espera i de retorno del sistema

28. Siga un algorisme de planificació multicua amb realimentació on cada cua es gestiona amb un Round-Robin de quantum $q_i = 2^i$. La planificació entre cues es de tipus prioritats no expulsives, la cua mes prioritària es la 0. Un procés passarà de la cua i a la cua i+1 quan s'esgote el seu quantum q_i sense finalitzar la seua execució. Els processos nous i els procedents de l'estat SUSPÈS entren per la cua 0.

Suposant que el S.Op. no consumeix temps i que les operacions d'E/S s'efectuen sobre el mateix dispositiu (gestionat de forma FCFS), dibuixar el diagrama temporal d'ocupació del processador i del dispositiu d'E/S per als processos A, B, C i D que s'il·lustren a continuació. Mostra també la situació de la cua de PREPARATS.

Calcula també els temps de retorn i espera (tant del processador com del disc) per a cada procés.

Procés	Instant llegada	Perfil Execució
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 Qüestions sobre fils d'execució i sincronització (UT5 i UT6)

Qüestions sobre conceptes teòrics

29. Contesta de forma concreta i concisa a les següents preguntes:
   a. ¿Què és un programa concurrent?
   b. ¿Què és una condició de carrera?
   c. ¿Què és una secció critica?
   d. ¿Quines condicions han d'acomplir els protocols de les seccions crítiques?

30. Cita almenys tres atributs que podríem trobar en el "bloc de control de fill".

31. Els semàfors no només serveixen per a resoldre el problema de la secció crítica. Enumera almenys altre dues aplicacions dels semàfors i fica un exemple simple de cadascuna d'aquestes aplicacions

32. Justifica de forma raonada que les operacions P(S) i V(S) s'han d'executar de forma atòmica per a que els semàfors funcionen correctament.

33. Raona adequadament si el següent codi és una bona solució al problema de la secció crítica, on S és un semàfor compartit per tots els processos o fils amb accés a aquesta secció crítica, amb valor inicial 1. A més, se sap que la política utilitzada pel sistema operatiu per a reactivar als processos suspesos és LIFO (Last In, First Out).

```
while(1) {
  P(S);
  Secció crítica
  V(S);
  Secció restant
}
```

Qüestions de veritable i fals

34. Indica per a cadascuna de les següents afirmacions si són veritables (V) o falses (F).

	Un procés pot suspendre's al realitzar una operació V sobre un semàfor
	Un procés sempre es suspèn al realitzar una operació P sobre un semàfor
	Un procés sempre es suspèn al realitzar una operació d'espera (wait) sobre una variable condició
	Un procés sempre desperta a un altre procés al realitzar una operació V sobre un semàfor

35. Indica per a cadascuna de les següents afirmacions si es veritable (V) o falsa (F).

	Mitjançant la funció pthread_self() es pot obtenir l'identificador d'un fill.
	Si els fils estan suportats pel sistema operatiu, sempre costa més un canvi de context entre processos que entre fils.
	Si el suport a fils es troba en el nucli, al suspendre's un dels fils, la resta podran seguir executant-se.

	El canvi de context entre fils d'execució suportats a nivell d'usuari és més ràpid que el de fils suportats a nivell de nucli.

Qüestions sobre codi

36. Indica les cadenes que imprimeix el programa en la Terminal rere la seua execució. Justifica la teua resposta. Nota: retras(n) realitza un retràs de n milisegons

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

```
void * funcion_hilo2(void * arg) {
  retras(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 següent programa implementa un algorisme que pretén resoldre el problema de la Secció Crítica de dos fils d'execució: fill_0 i fill_1. És correcta la solució proposada?
Nota: Analitza la implementació en termes d'exclusió mútua, progres i espera limitada.

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


int turno=0;


void *fill_0(void *p) {
  while(1) {
    while (turno != 0);
    /* Secció crítica */
    turno = 1;
    /* Secció restant */
 }
}
```

```
void *fill_1(void *p) {
  while(1) {
    while (turno != 1);
    /* Secció crítica */
    turno = 0;
    /* Secció restant */
 }
}

int main(int argc, char **argv){
   pthread_t h_0;
   pthread_t h_1;
   pthread_create(&h_0,NULL,fill_0,(void *)0);
   pthread_create(&h_1,NULL,fill_1,(void *)0);
   pthread_join(h_0, NULL);
   pthread_join(h_1, NULL);
}
```

38. Observa el següent fragment de codi corresponent a dos fils que pertanyen al mateix procés i s'executen concurrentment. Indica quins recursos compartits apareixen en el codi i quins mecanismes s'utilitzen per a evitar les condicions de carrera.
Nota: Les variables i funcions que no estan definides dins de les funcions agrega i resta són definides com globals.

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

  for (ct=0;ct<REPE;ct++)
```

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

  for (ct=0;ct<REPE;ct++)
```

```
  {
   while(test_and_set(&clau)==1);
   tmp=V;
   tmp++;
   V=tmp;
   clau=0;
  }

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

```
  {
   while(test_and_set(&clau)==1);
   tmp=V;
   tmp--;
   V=tmp;
   clau=0;
  }

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

39. Suposant que la variable global **`clau`** te valor inicial 0 i que la funció test_and_set està definida correctament, comenta les diferències entre les següents dues solucions al problema de la secció crítica atenent al temps d'execució observant en cadascuna d'elles.
Nota: la funció usleep() suspèn al fill que la invoca una quantitat de microsegons.

```
/* Solució a */
void *fill(void *p) {

  while(1) {
   while (test_and_set(&clau));
   /* Secció crítica */
   clau = 0;
   /* Secció restant */
  }
}
```

```
/* Solució b */
void *fill(void *p) {

  while(1) {
   while(test_and_set(&clau))
       usleep(1000);
   /* Secció crítica */
   clau = 0;
   /* Secció restant */
  }
}
```

40. Donat el següent codi, i assumint que:

a. Tots els semàfors tenen valor inicial zero,
b. Els fils presentats estan suportats pel nucli del sistema operatiu,
c. L'ordre en la cua de preparats és H1, H2, H3 (és a dir, H1 serà el primer en obtenir la CPU i H3 l'últim)
d. S'utilitza un algorisme de planificació 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);

Indica l'ordre de acabament dels fils. En el cas de que algun fill no acabe, indica en quina operació s'ha quedat suspès,