Fundamentos de los Sistemas Operativos

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

fSO

PART 2 EXERCISES
Process management

Content

1 Questions and problems about processes and planning (UT3 UT4)

1.1 Questions about theoretical concepts

The following are a set of questions that delve into the concepts studied in theory classes, and are ideal to know if the material has been understood, furthermore they allow practicing the ability to express ourselves on the new concepts learned.
Try to respond in a reasoned way to each onw of the following issues.

1. What is a process?

2. Indicate what is meant when saying I/O-bound or CPU-bound processes.

3. What means CPU scheduling?, how can a time sharing system to run multiple processes on a computer with a single CPU?

4. Describe the scheduling algorithms that you know and indicate what kind of processes favours (processes with short CPU bursts or long CPU bursts).

5. What are processes running on foreground and background in UNIX? how can I run shell commands such as foreground and background processes? Give an example of everyone.

6. What are the main states of a process and describe what is the meaning of every state.

7. What is the main advantage of multiprogramming related to system performance?

8. Describe the differences between scheduling in the short, medium and long-term.

9. How many processes UNIX would execute the following command?

```
$cat f1 f2 f3 | grep begins | wc - l > trace
```

10. Why is it required that the process control block (PCB) includes a field that contains the program counter?

1.2 True/false questions

Below are a set of questions that help students to discern about particular aspects of the concepts studied. For every statement the student will have to indicate if is true or false

11. Considers the execution of the following code:

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

Indicate whether the following statements are true (V) or false (F)

	It generates a zombie child
	It generates an orphan child
	It generates a parent zombie
	It generates a parent orphan
	It doesn't generate any new process

12. Indicate whether the following statements are true or false for a Unix system

a) The only way to create a new process (with different PID) is using the "exec" system call.

b) The command "ps - la" is used to display the attributes of all files in the current directory, including hidden ones.

1.3 Questions about programming code

13. Indicate how many processes will be generated as a result of the execution of the following program and how many of them will end up. Explain your answer.

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

14. Given the following C and POSIX code, indicate what will be printed on the standard output by every generated processes. Consider all the possible cases of execution of fork().

```
#include <stdio.h>
#include <unistd.h>
main(){
int  pid;
int 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. Indicate how many processes will be generated by the execution of this program (including the original), as well as which of them will be the last ending. Suppose that none of the used system calls causes errors and that the generated processes do not receive any signal.

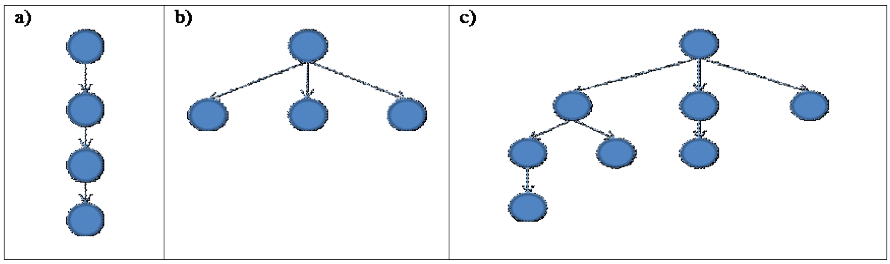
```
#include <stdlib.h>
int main(void) {
  int i, pid, pid2;

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

16. Given the following code:

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

Indicate which condition must be met to create a process group with the relationship reflected in each of the following schemes:



17. Given the following code on file "Example1.c":

```
1   /*** Example1.c ***/
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);
    }
```

Answer the following items:

a) How many processes can be created along its execution, in what lines are them created and what message prints everyone?

b) When will appear on the standard output the message "Message 3: before exit()".

Note. November 2011 exam

18. Given the following code on file "Example2.c":

```
1   /*** Example2.c ***/
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  }
```

Give an explained answer to:

a) The number of processes generated by its execution and the relationship between them.

b) What messages will appear on the screen if fork() call always succeeds.

Note. November 2011 exam

19. Given the following code on file "Example3.c".

```
    /*** Example3.c ***/
    #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);
    }
```

Give an explained answer to the following questions:

a) How many processes are created during its execution? In what line numbers are they created? Which lines of code execute each created process?

b) What processes show on the screen the line 15 message ("("Menssage 4: after wait\n"")?

Note. January 2012 exam

20. Given the following C and POSIX code on file "Example4.c":

```
1  /***Example4.c ***/
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) Draw the process tree generated when it is running and indicate for every process at what value of "i" it has been created.
b) Explain if there is a possibility of appearing orphan and/or zombie children.
Note. November 2013 exam

21. Given the following C POSIX code on file "Example5.c" that runs successfully:

```
1  /***Example5.c ***/
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) Explain the number of processes that are creating when executing Test and the parent/child relationship.
b) Indicate and explain the messages and information shown on the screen when executing Test.
Note. November 2013 exam

22. Given the followint code on file "Example6.c"

```
    /***Example6.c***/
1   #include " all necessary header .h"
2
3   int main(void)
4   { int val;
5     printf("Message 1\n");
6     val=fork();
7     /** Write here your changes **/
8     sleep(5);
9     printf("Message 2\n");
10    return 0;
11  }
```

a) Indicate what changes are required in the previous code in order to make the child process orphan and to be adopted by `init`. (**Note**: Use sleep() and C sentences).
b) Indicate what changes are required in the previous code in order to make the child process zombie for a while. (**Note**: Use sleep() and C sentences).
c) Explain in what instructions in the proposed code we can be sure that a CPU context switch will happen and in what instructions the will be an state change in the Example process
Note. November 2013 exam

23. Let's consider two executable files E and F, obtained after compiling the following C programs:

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

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

Info. The command "`date +%S`" prints on the screen the seconds portion of the current time. For example if the current time is 20:30:12, then this command will print "12".

Suppose that the executables E and F are in the working directory and that they run without errors. Answer the following items:
a) How many processes are created when running the command ". / F" and what is their parent ship.
b) What will be the output on the screen when running the command "`./F`" at 09:10:25.
c) How many processes are created when running the command ". / E" and what is their parent ship.
d) What will be the output on the screen when running the command "`./E`" at 09:10:25.

Note. January 2014 exam

1.4 Scheduling problems

24. Consider the following processes to run on a system:

Process	Arrival	Order	CPU	Priority
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

Obtain the scheduling diagram and compute the average turnaround and waiting times in the system for the following algorithms

a. FCFS

b. Round-Robin (q=1)

c. SJF

d. Not preemptive priorities (greater priority number, higher priority)

25. Suppose that to a system arrive simultaneously 5 processes (A, B, C, D, E), being the profile of each of them as follows: 1000 units of CPU time, 5 printer and other 1000 CPU. If that system applies a Round-Robin scheduling policy with quantum q=5, what will be the turnaround and waiting times for each of the 5 processes? Justify the answer. (Hint: it is not recommended to do a full trace).

26. Given a scheduling algorithm that is a modified version of the traditional Round-Robin to give better service to the processes that already have been running for a certain period of time, compared to the newcomers. The ready queue is divided into two queues: one for new processes and the other for running processes. Always choose to run a process in the queue of running through a Round-Robin approach, and processes that arrive to the system waiting in the queue for new until they can pass to the running one.

When a process arrives to the system its priority is 0, and in every unit of time the scheduler computes priorities for all the processes in the following way:

- If a process is in the new queue, its priority is increased by a factor **a**.
- If a process is in the running queue increases its priority by a factor of **b**.

When the priority of a a new process is greater than or equal to any process in the running queue, the new process is inserted in it. Where the running queue is empty, the process with higher priority of the new queue in inserted in the running queue. A process has more priority with the higher numerical value of its priority.

a. Assuming a = 2, b = 1 and q = 1 and the following situation, represents the CPU occupancy diagram and calculate the waiting time and turnaround timer of the system::

Process	Arrival time	CPU burst
A	0	5
B	1	4
C	3	2
D	9	6
E	11	3

b. Analyze the behavior in cases to > > b, b > = a, being a, b > 0

27. A system has a scheduler in the long term (PLP) and other short-term (PCP), that operates in the following way: the PCP uses an algorithm to process priorities and the PLP used a FCFS strategy. New processes enter the PLP. A maximum of three processes in memory is allowed. The PLP passes a process to the PCP when there are less than three already running processes. On the basis of the following table of processes and taking into account that smaller number means more priority.

Process	Arrival time	CPU time	Priority
A	0	2	4
B	1	4	3
C	3	4	2
D	5	1	1
E	6	2	3

Represents the CPU occupancy diagram and calculate average waiting and turnaround times.

28. In a scheduling multiqueue algorithm with feedback where each queue (i) is managed with a Round-Robin of quantum qi = 2*i. Scheduling between queues is non preemptive priorities, the higher priority queue is 0. A process will go from queue i to queue i+1 when consumes its qi quantum without ending its execution. New processes and those coming from the suspended state enter queue 0.

Assuming that the OS does not consume time and I/O operations are carried out on the same device (managed by FCFS), draw the temporary execution table of processes A, B, C and D that are described below.

Also calculate the turnaround time and (both the processor and the disk) waiting times for each process.

Process	Arrival time	Execution profile
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 Questions on threads of execution and synchronization (UT5 and UT6)

2.1 Questions about theoretical concepts

29. Answer the following questions as much precise and brief as possible:
   a. What is a concurrent program?
   b. What is a race condition?
   c. What is a critical critique?
   d. What conditions must comply the critical sections protocols?

30. Mention at least three attributes that we could find in the "thread control block".

31. Semaphores not only serve to solve the problem of the critical section. Lists at least two applications of semaphores and describe an example of each of these applications.

32. Justify in a reasoned way that operations P (S) and V (S) must be run atomically so semaphores work properly.

33. Explain if the following code is a good solution to the problem of the critical section, where S is a semaphore that is shared by all processes or threads with access to this critical section, with initial value 1. Also it is known that the policy used by the operating system to restart suspended processes is LIFO (Last In, First Out).

```
while(1) {
  P(S);
  Critical section
  V(S);
  Remaining section
}
```

2.2 True/false questions

34. Indicate for each of the following statements is true (T) or false (F).

	A process can be suspended when performing a V operation on a semaphore.
	A process is always suspended when performing a P operation on a semaphore.
	A process is always suspended when performing a P operation on a semaphore initialized to 0
	A process always awakens to another process when performing a V operation on a semaphore.

35. Indicate for each of the following statements if it is true (T) or false (F).

	Using function pthread_self() we can get the ID of a thread.
	If threads are supported by the operating system, always costs more a context switch between processes than between threads.
	If threads support is in the kernel, when suspending one of the threads of a process, the other threads of that process may continue running.
	Context switching between threads supported at user-level is faster than between threads supported at kernel level.

2.3 Question about programming code

36. Indicate the strings that prints the program in the terminal after its execution. Explain your answer. Note: delay (n) performs a delay of n milliseconds

```
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. The following program implements an algorithm that aims to solve the problem of the critical section of two threads of execution: hilo_0 e hilo_1. is the proposed solution correct?
Note : Analyze the implementation in terms of mutual exclusion, progress and limited waiting.

```
#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 the following fragment of code corresponding to two threads that belong to the same process and are run concurrently. Indicate what shared variables appear in code and what mechanisms are used to prevent race conditions.
Note : The variables and functions that are not defined within functions adds and subtracts are defined as global.

```
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. Assuming that the global variable llave has initial value 0 and that the test_and_set function is defined correctly; tell the differences between the following two solutions to the problem of the critical section relying on the execution time observed in each of them.
Nota: the usleep() function suspends the thread that calls it during a number of microseconds.

```
/* 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 */
  }
}
```

40. Given the following code, and assuming that: a) all the semaphores had initial value zero, b) threads are supported by the kernel, c) they were in the ready queue in the order H1, H2, H3 (i.e. H1 is the first to get the latest CPU and H3) and d) they are scheduled by 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);

Indicate the ending order of threads. When any thread does not finish, indicate which operation is suspended.