uc3m | Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática

SYSTEMS ARCHITECTURE **June 16th, 2023**

LAST NAME: .....
NAME: .....

Complete exam **Time: 210 min**

General instructions:

The use of additional materials on paper or other devices, the use of headphones, communication with other people, leaving the seat during the exam, or the use of additional software is not allowed.

TEST (3 points, 30 minutes)

Instructions:

There is only one correct answer. Unanswered questions do not add or subtract, correct answers add 0.3 points and incorrect answers subtract 0.1 points.

ANSWER SHEET

Select your answer for each question. Questions unanswered here will NOT be considered

1	2	3	4	5	6	7	8	9	10

1. Given the following code, what is the output?

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

int main(){
      char *str = strdup("3.14 is pi");
      char *endptr;
      float val;
      val = strtof(str, &endptr);
      printf("val:%f%s\n" , val, endptr);
}
```

   a) val:3.140000
   b) val:3.140000 is pi
   c) The code does not work and produces an error.
   d) The code works but provides a different output compared to the other options, and it has a memory leak of 10 bytes.

2. Given the following hash table that stores country data using the continent as a key, what is the missing line to insert France into the table?

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

struct node {
      char country_code[3];
      int population;
      struct node* next;
};

int main(){
      int size = 5;
      struct node** hash_table = (struct node **) calloc(sizeof(struct
node*),size);
      struct node* spain = (struct node *) malloc(sizeof(struct node));
      strcpy(spain->country_code,"ESP");
      spain->population = 47;
      struct node* france = (struct node *) malloc(sizeof(struct node));
      strcpy(france->country_code,"FRA");
      france->population = 67;
      hash_table[0] = spain;
      xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
}
```

   a) hash_table[1] = france;
   b) hash_table[0]->next = france;
   c) hash_table[x] = france; Assuming x is a value that cannot be known beforehand due to the lack of information about key indices.
   d) None of the other options is valid.

3. In the terminal, three commands are executed as shown below. What will happen after executing the third command?

```
vit121:~/Desktop/AS_2022/extra> ls -lisa
total 52
17202056539  0 drwxr-xr-x.  2 0291481 0291481    66 jun 10 12:47 .
25770316010  4 drwxr-xr-x. 21 0291481 0291481  4096 jun 10 11:49 ..
17193483248 20 -rwxr-xr-x.  1 0291481 0291481 16712 jun 10 12:00 q1
17202537696  4 -rw-r--r--.  1 0291481 0291481   203 jun 10 12:00 q1.c
17202537695 20 -rwxr-xr-x.  1 0291481 0291481 16760 jun 10 12:47 q2
17202537697  4 -rw-r--r--.  1 0291481 0291481   743 jun 10 12:47 q2.c
vit121:~/Desktop/AS_2022/extra> ls
q1  q1.c  q2  q2.c
vit121:~/Desktop/AS_2022/extra> ls | grep .c > q3
```

a) The terminal will be blocked since q3 is a FIFO.
b) A file will be written containing q1.c and q2.c.
c) The pipe q3 (not FIFO) will be created, storing the concatenation of the outputs of ls and grep .c.
d) The pipe q3 (not FIFO) will be created, storing the result of grep .c applied to the output of ls.

4. Given the following code, Which of the following sentences is correct?

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>

pid_t child = 0;

void handler1(int s){
      printf("Hello %i (my father) \n", getppid());
      sleep(1);
      kill(getppid(), SIGTSTP);
}

void handler2(int s){
      printf("Hello %i (my child) \n", child);
      sleep(1);
      kill(child, SIGTSTP);
}

void handler3(int s){
      kill(child,SIGTSTP);
}

int main(){
      signal(SIGTSTP, SIG_IGN);
      signal(SIGINT, SIG_IGN);
      if((child = fork())<0){ exit(EXIT_FAILURE); }
      else if(child == 0){
            signal(SIGTSTP, handler1);
            while(1) { pause(); }
      } else {
          signal(SIGTSTP,handler2);
            signal(SIGINT,handler3);
            while(1) { pause(); }
      }
}
```

a) After performing a CTRL-C, the parent sends a greeting message to the child, then the child replies, and both processes then terminate.
b) After performing a CTRL-C, the parent sends a greeting message to the child, then the child sends a greeting message back to the parent, and these greetings continue to be sent indefinitely in turns.
c) After performing a CTRL-C, the child sends a greeting message to the parent, then the parent sends a greeting message back to the child, and these greetings continue to be sent indefinitely in turns.
d) After performing a CTRL-C, the child dies because it is not associated with handler3 and, as there is no wait, it becomes a zombie.

5. What is the command `git checkout` for?
a) To create a new branch.
b) To update the local repository copy to have the content of the remote.
c) To upload all new content from the local repository to the remote repository.
d) To configure the user and email to be used in the Git account.

6. Given the following code, what happens when it is executed?

```
class T1 extends Thread{
      public void run() {
            for(int i=0; i<10; i++)
                  System.out.println("tic");
      }
}

class T2 implements Runnable{
      public void run() {
            for(int i=0; i<10; i++)
                  System.out.println("toc");
      }
}

public class Q6 {
      public static void main(String[] args) {
            T1 t1 = new T1();
            T2 t2 = new T2();
            t1.start();
            t2.start();

      }
}
```

a) Tic and toc will be displayed 10 times each, appearing alternately (first tic, then toc, and so on).
b) Tic and toc will be displayed 10 times each, but they will not be synchronized.
c) The code does not work because T2 does not inherit from Thread.
d) The code does not work because T2 is not started correctly.

7. Given the following code in which there are waiters serving one dish each and diners take a dish and leave, what can we say about the synchronization of this code?

```
public class Q7 extends Thread {
      private String profile;
      private int dishes_available;

      public Q7(String profile) {
            this.profile = profile;
      }

      public void run() {
            if(profile.equals("waiter")) { // Waiter
                  synchronized(Q7.class) {
                  dishes_available++;
                  System.out.println("A new dish is available. Total:
" + dishes_available);
                  Q7.class.notify();
                  }
            }
            if(profile.equals("guest")) { // Guest
                  synchronized(Q7.class) {
                        while(dishes_available==0) {
                              System.out.println("Wait");
                              try       {Q7.class.wait();}       catch
(InterruptedException e) {}
                        }
                        dishes_available--;
                        System.out.println("A  dish  has  been  eaten.
Total: " + dishes_available);
                  }
            }
      }

      public static void main(String args[]) {
            Q7[] waiters = new Q7[10];
            Q7[] guests = new Q7[10];
            for(int i=0; i<10;i++) {
                  waiters[i] = new Q7("waiter");
                  guests[i] = new Q7("guest");
                  guests[i].start();
                  waiters[i].start();
            }
      }
}
```

a) It does not synchronize properly because the shared variable is not actually shared among all threads.
b) It does not synchronize properly because in this case, the Q7 class should use locks with "this" instead of class locks, as it is currently programmed.
c) It does not synchronize properly because the code in the main should be enclosed in synchronized.
d) The code works perfectly and synchronizes properly.

8. The previous problem is transformed into using semaphores, but now it is restricted so that each waiter can only serve 10 dishes, and each diner eats 10 times. Considering this, we have the following code. What is the initialization value of the semaphores?

```
public class Q8 extends Thread {
      private String profile;
      private int ident;
      HIDDEN Semaphore semA = new Semaphore(__________);
      HIDDEN Semaphore semB = new Semaphore(__________);
      HIDDEN int dishes_available;

      public Q8(String profile, int ident) {
            this.profile = profile;
            this.ident=ident;
      }

      public void run() {
            if(profile.equals("waiter")) { // Waiter
                  for(int i=0;i<10;i++) {
                        try{semA.acquire();}                    catch
      (InterruptedException e) {}
                        ____________________ (1)
                        ____________________ (2)
                        ____________________ (3)
                  }
            }
            if(profile.equals("guest")) { // Guest
                    int eaten_times = 0;
                        while(eaten_times<10) {
                              try       {semA.acquire();}       catch
      (InterruptedException e) {}
                              if(_________________ (4) > 0) {
                                    dishes_available--;
                                    eaten_times++;
                                    _________________ (5)
                              } else {
                                    _________________ (6)
                                    _________________ (7)
                              }
                  }
            }
      }

      public static void main(String args[]) {
            Q8[] waiters = new Q8[10];
            Q8[] guests = new Q8[10];
            for(int i=0; i<10;i++) {
                  waiters[i] = new Q8("waiter",i);
                  guests[i] = new Q8("guest",i);
                  guests[i].start();
                  waiters[i].start();
            }
      }
}
```

a) semA = 1, semB = 0
b) semA = 0, semB = 1
c) semA = 1, semB = 10
d) semA = 10, semB = 0

9. Following the code above, in which of the available slots is it possible that there is a semB.acquire()?
   a) Slot 2
   b) Slot 4
   c) Slot 6
   d) Another slot

10. Which of the following statements about monitors is false?
   a) Condition variables control the progress of threads in the critical section.
   b) Monitors in Java are implemented with Signal-and-Continue.
   c) All variables that are part of the critical resource must be private.
   d) All methods in the monitor must be synchronized.

PROBLEMS (7 points, 180 minutes)
PROBLEM 1 (4 points, 90 minutes)

The PT-10L20-XA router from the company PocoTráfico S.A., for the purpose of this exercise, is simply dedicated to receiving messages and forwarding them to their destination. To do this, it examines the destination IP address of the message and determines which physical device it needs to be transmitted to, for which it needs to know its MAC address.

This is solved by implementing a list of (IP, MAC) pairs in the Nodes.dat file (which is a binary file).

There is a program written in the C language called Add that adds pairs to this file.

An example of how this program works is as follows:

```
[PT-10L20-XA]$ ./Add
usage: ./Add <ip> <mac>
[PT-10L20-XA]$ ./Add 147.6.170.251 FB:7D:30:22:30:3C
--------Node list----------
Empty
------------------------
--------Node list----------
ip="147.6.170.251" mac="FB:7D:30:22:30:3C"
------------------------
[PT-10L20-XA]$ ./Add 198.62.208.208 08:D0:96:1C:D1:9C
--------Node list----------
ip="147.6.170.251" mac="FB:7D:30:22:30:3C"
------------------------
--------Node list----------
ip="147.6.170.251" mac="FB:7D:30:22:30:3C"
ip="198.62.208.208" mac="08:D0:96:1C:D1:9C"
------------------------
[PT-10L20-XA]$ ./Add 161.151.224.179 57:FC:4D:3D:88:3A
--------Node list----------
ip="147.6.170.251" mac="FB:7D:30:22:30:3C"
ip="198.62.208.208" mac="08:D0:96:1C:D1:9C"
------------------------
--------Node list----------
ip="147.6.170.251" mac="FB:7D:30:22:30:3C"
ip="198.62.208.208" mac="08:D0:96:1C:D1:9C"
ip="161.151.224.179" mac="57:FC:4D:3D:88:3A"
------------------------
[PT-10L20-XA]$ ./Add 182.101.216.94 4C:10:83:A4:1B:E6
--------Node list----------
ip="147.6.170.251" mac="FB:7D:30:22:30:3C"
ip="198.62.208.208" mac="08:D0:96:1C:D1:9C"
ip="161.151.224.179" mac="57:FC:4D:3D:88:3A"
------------------------
--------Node list----------
ip="147.6.170.251" mac="FB:7D:30:22:30:3C"
ip="198.62.208.208" mac="08:D0:96:1C:D1:9C"
ip="161.151.224.179" mac="57:FC:4D:3D:88:3A"
ip="182.101.216.94" mac="4C:10:83:A4:1B:E6"
------------------------
[PT-10L20-XA]$
```

PataPalo S.L. has obtained fragments of the Add program's code and needs an expert like you to complete the program. They have committed to paying you 10 gold doubloons if you succeed.

However, it won't be easy, as there are some mandatory restrictions that must be followed, or else you won't get a single coin.

- Respect the original code and use the functions, variables, types, constants, etc. that have been pirated whenever possible. Use the functions, even if you haven't answered the questions where they are implemented.
- Respecting the original program means, among other things, reading the data file into a linked list of Nodes, adding the new node to the end of the list, and saving the list to the file.
- You can use the functions from ASlib.

The first pirated piece is the header of the Add.c file:

```
 1
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <string.h>
 5 #include <sys/wait.h>
 6 #include <unistd.h>
 7
 8 #define IP_SIZE 20
 9 #define MAC_SIZE 20
10 #define TRACE printf("%s:%d\n",__FILE__,__LINE__)
11
12 #define FILE_NAME "Nodes.dat"
```

Also, we have the node type:

```
39
40 typedef struct Node
41     {
42     char ip[IP_SIZE];
43     char mac[MAC_SIZE];
44     struct Node *next;
45     } Node;
46
```

And the functions declarations:

```
48 void  Verify_args(int ac, char**av);
49
50 Node* New_node(const char*ip, const char *mac);
51 Node* Add(Node *p_first, Node *p_node);
52
53 Node* Read(const char *file_name);
54 void  Save(Node *p_first, const char*file_name);
55
56 void  Display(Node *p_first);
57 void  Destroy(Node *p_first);
```

- **Verify_args** debe comprobar que la invocación al programa es correcta.
- **New_node** debe construir un nuevo nodo a partir de la información de los argumentos
- **Add** debe añadir el nodo apuntado por p_node a la lista que comienza en p_first
- **Read** debe leer el archivo dado construyendo la cadena que devuelve con el puntero al primer nodo de la lista.
- **Save** debe salvar la lista que empieza en p_first en el archivo dado.
- **Display** presenta por pantalla la lista que comienza en p_first.
- **Destroy** debe liberar la memoria ocupada por la lista.

We are provided the following code as well:

```
59 int main (int ac, char**av)
60     {
61     Node *p_first = NULL;
62     Node *p_node;
63     Verify_args(ac, av);
64
65     p_first = Read (FILE_NAME);
66     Display (p_first);
67     p_node = New_node(av[1],av[2]);
68     p_first = Add(p_first, p_node);
69     Display (p_first);
70     Save(p_first,FILE_NAME);
71     Destroy(p_first);
72     }
```

```
75 void Verify_args(int ac, char**av)
76     {
77     if (ac != 3)
78         {
79         printf("usage: %s <ip> <mac>\n", av[0]);
80         exit(EXIT_SUCCESS);
81         }
82     }
```

```
84 Node* New_node(const char *ip, const char *mac)
85     {
86     Node *p_node = malloc(sizeof(Node));
87     strncpy(p_node->ip,ip,IP_SIZE);
88     strncpy(p_node->mac,mac,MAC_SIZE);
89     p_node->next = NULL;
90     return(p_node);
91     }
```

```
127 void  Display(Node *p_first)
128     {
129     puts("--------Node list----------");
130     if (p_first == NULL) puts("Empty");
131     else
132         {
133         for (Node *p = p_first; p; p=p->next)
134             printf("ip=\"%s\" mac=\"%s\" \n",p->ip,p->mac);
135         }
136     puts("-------------------------");
137     }
138
```

```
139 void  Save(Node *p_first, const char*file_name)
140     {
141     FILE *f = fopen(file_name,"wb");
142     if (p_first != NULL)
143         {
144         for (Node *p = p_first; p; p=p->next)
145             fwrite(p,sizeof(Node),1,f);
146         }
147     fclose(f);
148     }
149
```

Section 1.1 (0.7 points)

Node *Read(const char *filename)

Write this function that should read the list of nodes from the binary file.
A hint: As when saving the nodes, it has been made as is, when reading them, the last node will still have next==NULL, and the others will not.

Section 1.2 (0.7 points)

Node * Add(Node * p_first, Node*p_node)

Write this function that must add the node given by p_node to the list given by p_first.
Note that Add must return the pointer to the first node in the resulting list.

Section 1.3 (0.3 points)

void Destroy(Node *p_first)

Write this function that should free the memory occupied by the list given in the argument.

ADDENDA

The chief engineering officer at PocoTráfico is a cunning security engineer and has left a bug embedded in the code, which has cleverly been pirated.
Now it's time to undo it.
The problem is that, due to hardware limitations, it's not possible for the process triggered when executing Add to perform both the insertion operation in the file and displaying the status of the list.
The solution is to create a subprocess that will be responsible for listing the information on the screen. Communication between both processes will be done using signals (SIGUSR1 and SIGINT).
When the child process receives a SIGUSR1, it reads and prints the file, and when it finishes, it sends SIGUSR1 to the parent.
When the child process receives SIGINT, it terminates.
Let's make this modification by answering the following questions.
In each question, add the necessary variables as needed.
You can assume the existence of the remaining functions.
Remember that if a signal is sent and the receiver is not running or has not programmed that signal, it will be lost. A 1-second pause will be sufficient in this case.

Section 1.4 (0.3 points)

Child_handler()

Write this function that should capture the signals received by the child.

Section 1.5 (0.7 points)

Parent_handler()

Write this function that should capture the signals received by the parent

Section 1.6 (0.3 points)

Child_process()

Write here the code of the child process, to cover the described needs.

Section 1.7 (1 point)

Rewrite the main method according to the new needs, including orderly completion.
In case you want to copy-paste it the original code is:

```
int main (int ac, char**av)
   {
   Node *p_first = NULL;
   Node *p_node;
   Verify_args(ac, av);

   p_first = Read (FILE_NAME);
   Display (p_first);
   p_node = New_node(av[1],av[2]);
   p_first = Add(p_first, p_node);
   Display (p_first);
   Save(p_first,FILE_NAME);
   Destroy(p_first);
   }
```

PROBLEM 2 (1 point, 20 minutes)
Given the following software solution for the critical section problem, indicate which of the three fundamental requirements it satisfies and provide a reasoned justification for each answer. Assume that "turn" is a shared variable that is initialized at the beginning, and the code is executed by two processes, P0 and P1, as follows:

Process P0 code

```
do {
        while (turn == 1);
        critical section
        turn = 1;
        reminder section
} while (true);
```

Process P1 code

```
do {
        while (turn == 0);
        critical section
        turn = 0;
        reminder section
} while (true);
```

PROBLEM 3 (2 points, 70 minutes)
In a library, we have writers who are responsible for writing books, and readers who are responsible for reading the books written by these writers. However, there is only one computer with writing software, so only one writer can write at a time. To read, there are three separate reading stations that are independent of the writing station (in fact, a reader cannot use the writing station to read, even if it is available). Each reader who visits the library's mission is to read all the books written by the writers, although they cannot be read until they are written. Furthermore, since the books have a certain relationship, they must be read in the order of writing (i.e., the second book written is read after the first, the third after the second, and so on, regardless of the authors). Each book can only be read once, and to be fair, each time a reader requests a turn to read, they can only read one book.

Taking this into account, you are asked to implement this problem using semaphores. To do this, you should implement a class that models the writer threads (`Writer`), another class that models the reader threads (`Reader`), and a `Library` class that contains the `main` method. As additional information, each writer will write 5 books each time they go to the library, and there will be a simulated writing time of 200 ms and a reading time of 100 ms. For the `main` method, consider 3 writers and 10 readers, so that each reader has to read 15 books. Furthermore, it should be ensured that the program terminates correctly, meaning that all threads must be closed once each one completes its mission (the writer writes 3 books and the reader reads 15 books).

Note: To try to be fair in the turn of writing and reading, you can make the FIFO queues by adding true as a second parameter to the necessary semaphores.

SOLUTION

TEST

1	2	3	4	5	6	7	8	9	10
B	B	B	C	A	D	A	A	D	D

PROBLEM 1 (4 points)
Section 1.1

```
 94 Node* Read(const char *file_name)
 95     {
 96     Node *p_first = NULL;
 97     Node *p_node, *p;
 98
 99     if (access(file_name,F_OK) != 0) return(NULL);
100     FILE *f = fopen(file_name,"rb");
101
102     do
103         {
104         p_node = malloc(sizeof(Node));
105         fread(p_node, sizeof(Node), 1,f);
106         p = p_node->next;
107         p_node->next = 0;
108         p_first = Add(p_first,p_node);
109         }
110         while (p != NULL);
111
112     fclose(f);
113
114     return(p_first);
115     }
```

Section 1.2

```
117 Node* Add(Node*p_first, Node*p_node)
118     {
119     Node *p;
120     if (p_first == NULL) return (p_node);
121     for (p = p_first; p->next; p=p->next);
122     p->next = p_node;
123     p_node->next = NULL;
124     return (p_first);
125     }
```

Section 1.3

```
123 void  Destroy(Node *p_first)
124     {
125     Node *p, *p_next;
126     for (p=p_first; p; p = p_next)
127         {
128         p_next = p->next;
129         free(p);
130         }
131     }
```

Section 1.4

```
void Child_handler(int s)
    {
    Node *p_first = Read(FILE_NAME);
    Display(p_first);
    Destroy(p_first);
    kill(getppid(),SIGUSR1);
    }
```

Section 1.5

```
char *ip, *mac;
pid_t child_pid;
int estado=0;
```

```
 void Parent_handler(int s)
     {
     if (estado == 0)
         {
         Node *p_first = Read(FILE_NAME);
         Node *p_node = New_node(ip,mac);
         Add (p_first, p_node);
         Save(p_first, FILE_NAME);
         Destroy(p_first);
         estado = 1;
         kill(child_pid,SIGUSR1);
         }
     else
         {
         kill(child_pid,SIGINT);
         }
     }
```

Section 1.6

```
void Child_process()
    {
    void *p;
    p = signal (SIGUSR1,Child_handler);
    while(1);
    }
```

Section 1.7

```c
int main (int ac, char**av)
    {
    int status ;

    Verify_args(ac, av);
    ip = av[1];
    mac = av[2];

    child_pid = fork();
    if (child_pid <0)
        {
        puts("Error en el fork");
        exit (EXIT_FAILURE);
        }
    if (child_pid == 0) Child_process();

    //Parent process
    signal(SIGUSR1,Parent_handler);

    Pausa (1000,1000);
    kill(child_pid,SIGUSR1);

    wait (&status);
    exit(EXIT_SUCCESS);
    }
```

PROBLEM 2

1. Mutual Exclusion. It is satisfied since the variable "turn" indicates which process can enter the critical section, allowing only one thread to be inside. If we have processes P0 and P1, if "turn" equals 0, P1 waits and P0 enters the critical section. If later "turn" becomes 1, P0 would wait and P1 would enter the critical section.
2. Progress. It is not satisfied because, for example, if P0 enters the critical section, completes its execution, and gives the turn to P1, then even if P1 is not waiting to enter and is already in the critical section, P0 would not be able to execute. In that case, the critical section would be "free," but no one would be able to enter.
3. Bounded Waiting. It is satisfied since the processes take turns, ensuring that when one process enters, the other process knows it will be next. If it is P0's turn and it finishes, "turn" becomes 1, and then it will be P1's turn, and so on.

PROBLEM 3 (2 points)

```
import java.util.concurrent.Semaphore;

class Writer extends Thread {
      private int id;
      private static int currentBook;
      private static int numReaders;
      private Semaphore writerSemaphore;
      private Semaphore[] bookSemaphore;

      public Writer(int id, Semaphore writerSemaphore, Semaphore[]
bookSemaphore, int nReaders) {
      this.id = id;
      this.writerSemaphore = writerSemaphore;
      this.bookSemaphore = bookSemaphore;
      numReaders = nReaders;
```

```
	}

	public void run() {
	for (int i = 0; i < 5; i++) { // Each Writer writes 5 books
		try {
			writerSemaphore.acquire();
			System.out.println("Writer " + id + " is writing book "
+ currentBook);
			Thread.sleep(200);
			bookSemaphore[currentBook].release(numReaders);
			currentBook++;
			writerSemaphore.release();
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}
	}
}

class Reader extends Thread {
	private Semaphore readerSemaphore;
	private Semaphore[] bookSemaphore;
	private int id;
	private int numBooks;

	public Reader(Semaphore readerSemaphore, Semaphore[] bookSemaphore,
int id, int numBooks) {
	this.readerSemaphore = readerSemaphore;
	this.bookSemaphore = bookSemaphore;
	this.id = id;
	this.numBooks = numBooks;
	}

	public void run() {
	for(int i=0; i<numBooks; i++) { // Finish after reading all the
books
		try {
			readerSemaphore.acquire();
			bookSemaphore[i].acquire();
			System.out.println("Reader " + id + " is reading book "
+ i);
			Thread.sleep(100);
			bookSemaphore[i].release();
			readerSemaphore.release();
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}
	}
}

class Library {

	public static void main(String[] args) {
		int nWriters= 3;
		int numBooks = 5*nWriters;
		int numReaders = 10;
		Semaphore writerSemaphore = new Semaphore(1, true); // Only 1
writer can write at a time
		Semaphore[] bookSemaphore = new Semaphore[numBooks];
		for(int i=0; i<numBooks;i++) { bookSemaphore[i] = new
Semaphore(0);}
```

```
            Semaphore readerSemaphore =  new Semaphore(3, true);

      System.out.println("Starting library");
      // Create and start writer threads
      for (int i = 0; i < nWriters; i++) {
            Writer writer = new Writer(i, writerSemaphore, bookSemaphore,
numReaders );
            writer.start();
      }

      // Create and start reader threads
      for (int i = 0; i < numReaders; i++) {
            Reader reader = new Reader(readerSemaphore, bookSemaphore, i,
numBooks);
            reader.start();
      }
      }
}
```

Evaluation criteria

PROBLEM 1 (4 puntos)
Section 1.1 (0.7 puntos)

Line	Concept	Value
Node* Read(const char *file_name)		
{		
Node *p_first = NULL;	List initialization	0,04
Node *p_node, *p;		
if (access(file_name,F_OK) != 0) return(NULL);	File exists, F_OK, F_READ or fopen ==null	0,08
FILE *f = fopen(file_name,"rb");	Open binary file using "rb" 0 points if not binary.	0,08
do		
{		
p_node = malloc(sizeof(Node));	Allocate memory using correct size	0,11
fread(p_node, sizeof(Node), 1,f);	Read node from file, no matter argument order.	0,11
p = p_node->next;	Anotate next	0,04
p_node->next = 0;	Terminate list	0,04
p_first = Add(p_first,p_node);	Add node to list	0,08
}		
while (p != NULL);	While not end of list	0,04
fclose(f);	Close file	0,04
return(p_first);	Devolver la lista	0,04
}		
	Global assessment	0,04

	TOTAL	0,70

- Significant errors may be subject to additional penalties.

Section 1.2 (0.7 puntos)

Line	Concept	Value
Node* Add(Node*p_first, Node*p_node)		
{		
Node *p;		
if (p_first == NULL) return (p_node);	Check empty list	0,07
for (p = p_first; p->next; p=p->next);	Search for last node	0,21
p->next = p_node;	Chain node	0,14
p_node->next = NULL;	Terminate list	0,21
return (p_first);	Return list	0,07
}		
	TOTAL	0,70

- Significant errors may be subject to additional penalties.

Section 1.3 (0.3 puntos)

Line	Concept	Value	
void Destroy(Node *p_first)			
{			
Node *p, *p_next;			
for (p=p_first; p; p = p_next)	Iterate list	0,1	
{			
p_next = p->next;	Annotate next	0,1	0 si se destruye antes de anotar el siguiente
free(p);	Destroy current	0,1	
}			
}			
	TOTAL	0,3	

- Significant errors may be subject to additional penalties.

Section 1.4 (0.3 puntos)

Line	Concept	Value
void Child_handler(int s)		
{		
Node *p_first = Read(FILE_NAME);	Read file (-0.03 not define used)	0,06
Display(p_first);	Display list	0,06
Destroy(p_first);	Destroy list	0,06
kill(getppid(),SIGUSR1);	Send signal to parent process	0,12
}		
	TOTAL	0,3

- Los errores significativos pueden verse sujetos a penalizaciones adicionales.

Apartado 1.5 (0.7 puntos)

Line	Concept	Value
int estado =0;	Some way to identify if we are before or after modifying the file. Correctly started.	0,04
char *ip, *mac; pid_t child_pid;	Needed variables	0,03
void Parent_handler(int s)		
{		
if (estado == 0)	Discriminate by current state	0,07
{		
Node *p_first = Read(FILE_NAME);	Read file (-0.15 if not define used)	0,03
Node *p_node = New_node(ip,mac);	Create new node	0,07
Add (p_first, p_node);	Add to list	0,07
Save(p_first, FILE_NAME);	Save list	0,07
Destroy(p_first);	Destroy list	0,07
estado = 1;	advance to next state	0,04
kill(child_pid,SIGUSR1);	Send Display signal to child	0,07
}		
else		
{		
kill(child_pid,SIGINT);	Terminate child process	0,07
}		0,07
}		
	TOTAL	0,7

- Significant errors may be subject to additional penalties.

.

Section 1.6 (0.3 puntos)

Line	Concept	Value
void Child_process()		
{		
void *p;		
p = signal (SIGUSR1,Child_handler);	Redirect signal	0,15
while(1);	Infinite wait	0,15
}		
	TOTAL	0,3

- Significant errors may be subject to additional penalties.

Apartado 1.7 (1 punto)

Línea	Objetivo	Puntuación
int main (int ac, char**av)		
{		

int status ;		
Verify_args(ac, av);	Verify args	0,07
ip = av[1];		0,07
mac = av[2];	annotate args for later use	
child_pid = fork();	Launch child	0,11
if (child_pid <0)	Verify fork result	0,11
{		
puts("Error en el fork");	Error message	0,04
exit (EXIT_FAILURE);	Terminate process with failure	0,07
}		
if (child_pid == 0) Child_process();	Invoque child process	0,07
//Parent process		
signal(SIGUSR1,Parent_handler);	Redirect signal	0,07
Pausa (1000,1000);	Wait for child start	0,07
kill(child_pid,SIGUSR1);	Send display command to child	0,07
wait (&status);	Wait for child end	0,11
exit(EXIT_SUCCESS);	Success end	0,07
}		0,07
	TOTAL	1

- Significant errors may be subject to additional penalties.

PROBLEM 2 (1 punto)

- 0.3: Mutual exclusion. Identify 0.1, Justify: 0.2.
- 0.4: Progreso. Identify 0.15. Justify: 0.25.
- 0.3: Limited wait. Identify: 0.1. Justify 0.2.

PROBLEMA 3 (2 puntos)

- 0.8: Writer class
  - 0.1: Class and attributes declaration.
  - 0.1: Constructor
  - 0.6: run method
    - 0.1: Correct loop
    - 0.2: Correct use of writerSemaphore & currentBook updating
    - 0.2: Correct use of bookSemaphore
    - 0.1: Correct sleep
    - The omission of the try/catch will not be penalized
- 0.8: class Reader
  - 0.1: Class and attributes declaration.
  - 0.1: Constructor
  - 0.6: run method
    - 0.1: Correct loop
    - 0.2: Correct use of writerSemaphore

        - 0.2: Correct use of bookSemaphore
        - 0.1: Correct sleep
        - The omission of the try/catch will not be penalized
- 0.4: class Library
    - 0.05: Variable definition
    - 0.20: Semaphore definition
    - 0.15: Thread initialization
- Significant errors may be subject to additional penalties.