uc3m | Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática

SYSTEMS ARCHITECTURE **January 16th, 2023**

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

Complete exam **Time: 235 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, 50 minutes)

Instructions:

There is only one correct answer. Unanswered questions do not add or subtract, correct answers add 0.15 points and incorrect answers subtract 0.03 points.

1. Given the files q1.h and q1.c, what is the output of the following code?

q1.h

```
#ifndef Q1
#define Q1
#define TAM 5
struct point{
      int x;
      int y;
};
#endif
```

q1.c

```
#include <stdio.h>
#define Q1 5
#include "q1.h"
#define TAM 4

int main(){
      struct point array[TAM];
      int i = 0;
      for(i=0; i<TAM; i++){
            array[i].x = i;
            array[i].y = 2*i;
      }
      printf("(x,y)=(%d,%d)\n",array[TAM-1].x, array[TAM-1].y);
      return 0;
}
```

a) (x,y) = (3,6)
b) (x,y) = (4,8)
c) The code does not compile
d) Segmentation Fault

2. Which of the following is the correct way for the code to print 12345?

```
#include <stdio.h>
#define SIZE 5

int main(){
      int array[SIZE] = {1,2,3,4,5};
```

```
      int *ptr_1 = array;
      int **ptr_2 = &ptr_1;
      for(int i=0; i<SIZE;i++){
            // YOUR ANSWER
      }
      printf("\n");
      return 0;
}
```

a) printf("%d", *(*ptr_2+i));
b) printf("%d", **ptr_2[i]);
c) printf("%d", *ptr_1[i]);
d) printf("%d", **ptr_2->i);

3. What is the output of the following code?

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

int main(){
      char text[5]="123a";
      int base = 10;
      char *endptr;
      int val = strtol(text, &endptr, base);
      printf("Val: %d | Endptr: %s\n", val, endptr);
      return 0;
}
```

a) Val: 123 | Endptr: a
b) Val: 123 | Endptr: (the value of endptr is \0 and nothing is printed since \0 is a non-printable character)
c) Val: -1 | Endptr: 0
d) The process terminates when the strtol line is reached since the text is not a number.

4. Given the following code, where you can assume that the print() function prints the current list with printf() and the free_list function clears the list correctly (no memory leaks), we can say that:

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

struct node{
      int value;
      struct node *next;
};

struct node *add(struct node *mylist){
      int value = 1;
      struct node *new_node = (struct node *) malloc(sizeof(struct
node));
      new_node -> value = value;
      new_node->next = mylist;
      mylist = new_node;
      return mylist;
}

struct node *delete(struct node *mylist){
      mylist = mylist->next;
      return mylist;
}

void print(struct node *mylist){...}
void free_list(struct node *mylist){...}

int main(){
```

```
        struct node *mylist = NULL;
        struct node *(*operation)(struct node*);
        operation = add;
        mylist = operation(mylist);
        mylist = operation(mylist);
        operation = delete;
        mylist = operation(mylist);
        print(mylist);
        free(mylist);
}
```

a) The code works perfectly, no memory leaks and the output is a 1.
b) The code works perfectly, the output is a 1, but there is a memory leak (3 allocs, 2 frees)
c) The code works perfectly, the output is a 1, but there are two memory leaks (3 allocs, 1 free)
d) The code does not work because it is not possible to perform operations of the type operation = functionName

5. Given the following code in C and the text file q5.txt, where you can assume that q5.txt is in the program directory and has read permissions, what happens when executing the code?

q5.txt
```
a1
a2
b1
b2
c1
c2
```

q5.c
```
#include <stdio.h>
#include <string.h>

int main(){
        FILE *fd;
        char *buffer;
        ssize_t readc=0;
        size_t n=0;
        fd = fopen("q5.txt", "r");
        while((readc=getline(&buffer,&n,fd)!=-1)){
                if(strcmp(buffer,"c1") == 0){
                        printf("A");
                }
        }
        free(buffer);
        fclose(fd);
        printf("B\n");
        return 0;
}
```

a) Only A is printed
b) Only B is printed
c) AB is printed
d) Neither A nor B is printed because an error occurs before the while is reached.

6. The content in a local git repository is as follows:

```
monitor01:~/Desktop/AS_2022/git_main/prueba> ls
example.txt  README.md  test_file.txt
```

After using a command, the content is:

```
monitor01:~/Desktop/AS_2022/git_main/prueba> ls
example1_created.c  example.txt  README.md  test_file.txt
```

What command may I have executed?
a) git push
b) git pull
c) git commit -m “example1_created.c”
d) git commit -m “example1_created.c” and afterwards, git push

7. Given the following code:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main(){
      pid_t childpid,childpid2,childpid3;
      childpid = fork();
      childpid2 = fork();
      childpid3 = fork();
      if (childpid2 == 0){
            if(childpid3 == 0){
                  printf("4 ");
                  exit(EXIT_SUCCESS); // Added in the exam
            }
            printf("2 ");
            exit(EXIT_SUCCESS);
      }
      if (childpid3 == 0){
            printf("3 ");
            exit(EXIT_SUCCESS);
      }
      if (childpid == 0){
            printf("1 ");
            exit(EXIT_SUCCESS);
      }
      printf("root ");
}
```

a) A possible output of the terminal is “root 1 2 3 4”.
b) A possible output of the terminal is “root 1 1 2 2 3 3 4”.
c) A possible output of the terminal is “root 1 2 2 2 3 3 4”.
d) A possible output of the terminal is “root 1 2 2 3 3 4 4”.

8. Given the following C code, what happens when you execute it?

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <signal.h>

pid_t pidex = 0;

int main(){
      int i=0;
      if ((pidex = fork())<0){
            exit(EXIT_FAILURE);
      }else if (pidex == 0){
            while(1){
                  printf("final exam\n");
                  i++;
                  if(i == 5){
                        kill(getppid(), SIGINT);
                  }
            }
      }else{
            while(1){
                  pause();
            }
      }
```

```
      return 0;
}
```

a) The child process becomes a zombie once i==5
b) The child process becomes an orphan once i==5
c) The child process dies once i==5, because it is killed by the father
d) The father and the son die once i==5, since the father kills the son and then the father ends

9. Given the following code, what happens when you execute it?

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <signal.h>

pid_t pidex = 0;
pid_t pid = 0;
int status;

void handler(int signal){
      kill(pidex, SIGINT);
      pid = wait(&status);
      exit(EXIT_SUCCESS);
}

int main(){
      if ((pidex = fork())<0){
            exit(EXIT_FAILURE);
      }else if (pidex == 0){
            while(1){
                  printf("question 9\n");
            }
      }else{
            signal(SIGALRM, handler);
            alarm(3);
            while(1){
                  pause();
            }
      }
      return 0;
}
```

a) The parent process enters the while(1) loop after 3 seconds and then waits indefinitely while the child process prints “question 9” until there is a SIGINT signal.
b) The child process becomes a zombie once the SIGALRM signal is triggered.
c) The child process prints “question 9” for 3 seconds and then terminates successfully, just before the parent.
d) The parent process terminates after 3 seconds and then the child process becomes orphan and continues to display the text “question 9”.

10. What happens after executing the following “echo” command in Linux?

```
monitor01:~/Desktop/AS_2022/ordinario> ls -l q10
prw-r--r--. 1 0291481 0291481 0 ene 10 00:21 q10
monitor01:~/Desktop/AS_2022/ordinario> echo "question 10" > q10
```

a) Write the text “question 10” in the file q10, overwriting the current content of the file.
b) Write the text “question 10” in the file q10, adding the content at the end of the file.
c) A permissions error occurs if it is executed by the user himself
d) A blockage occurs in the terminal

11. Two threads of a process share:
    a) Memory space
    b) Id
    c) Stack
    d) Registers

12. What is the meaning of mutex?
    a) None of the others
    b) Multiple extensión
    c) Mutua from Texas
    d) Multiple exclusion

13. Select the incorrect statement concerning a lock in Java.
    a) It can encapsulate an integer type variable
    b) It can limit the use of a static method
    c) It can be used in an instance of a class as a lock
    d) A class can be used as a lock

14. What is the result of the following program?

```
public class P1
    {
    public static int c = 0;

    public static void main(String[] args)
        {
        T[] ts = new T[3];
        for (int i=0; i<3; i++)
            ts[i] = new T();
        for (int i=0; i<3; i++)
            ts[i].start();
        for (int i=0; i<3; i++)
            {
            try {ts[i].join();}
            catch(Exception e){System.exit(666);}
            }
        System.out.println(c);
        }
    }
class T extends Thread
    {
    public void run()
        {
        synchronized(P1.c) {P1.c++;}

        try{Thread.sleep(100);}
        catch(Exception e){System.exit(666);}

        synchronized(P1.c) {P1.c++;}
        }
    }
```

    a) Compilation error
    b) 6
    c) 8
    d) 4

15. Select the correct option related to wait();
    a) It wait for a notify
    b) It waits for the end of the process
    c) It waits for the end of the program
    d) It waits for the end of its descendant processes

16. To implement a mutex with semaphores:
    a) A semaphore light initialized to 1 is used
    b) A semaphore initialized to 0 is used
    c) A semaphore initialized to N, with n>1, is used
    d) It is not possible to build a mutex with semaphores

17. What must be the content of line 16 for the execution to be correct and the result to be 3?

```
 1 import java.util.concurrent.Semaphore;
 2
 3 public class P1
 4     {
 5     public static Semaphore semaforo = new Semaphore(0);
 6     public static int c = 0;
 7
 8     public static void main(String[] args)
 9         {
10         T[] ts = new T[3];
11         for (int i=0; i<3; i++)
12             ts[i] = new T();
13         for (int i=0; i<3; i++)
14             ts[i].start();
15
16
17
18         for (int i=0; i<3; i++)
19             {
20             try {ts[i].join();}
21             catch(Exception e){System.exit(666);}
22             }
23
24         System.out.println(c);
25         }
26     }
27
28 class T extends Thread
29     {
30     public void run()
31         {
32         try {P1.semaforo.acquire();}
33         catch(Exception e){System.exit(666);}
34
35         P1.c++;
36         P1.semaforo.release();
37         }
38     }
```

    a) semaforo.relase();
    b) semaforo.acquire();

c) semaforo.notify();
d) synchronized(semaforo) {notify();}

18. Java, the signaling strategy with monitors is:
    a) Signal-and-continue
    b) Signal-and-wait
    c) Signal-and-exit
    d) Signal-and-release

19. In reference to monitors, which statement is **NOT** correct?
    a) All their methods must be public
    b) They encapsulate critical resources
    c) They contain condition variables
    d) They must be defined in a class

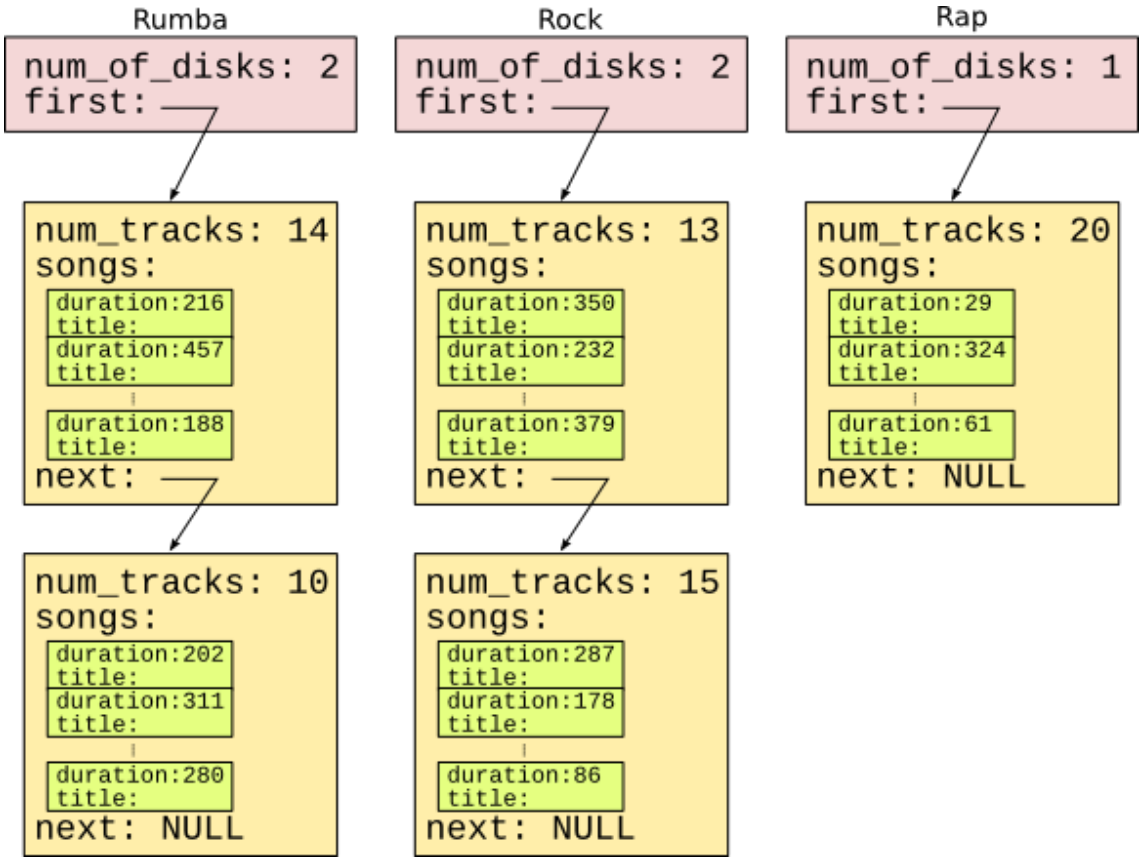
20. What is the output of the execution of the following program?

```
1  public class P1
2      {
3      public static M m = new M();
4      public static void main(String[] args)
5          {
6          T[] ts = new T[4];
7          for (int i=0; i<4; i++)
8              ts[i] = new T(i);
9
10         for (int i=0; i<4; i++)
11             ts[i].start();
12
13         for (int i=0; i<4; i++)
14             {
15             try {ts[i].join();}
16             catch(Exception e){System.exit(666);}
17             }
18         System.out.println("");
19         }
20     }
21
22 class T extends Thread
23     {
24     int n;
25     public T(int n)
26         {
27         this.n = n;
28         }
29     public void run()
30         {
31         P1.m.Get_in(n);
32         System.out.print(n);
33         P1.m.Get_out(n);
34         }
35     }
36
37 class M
38     {
39     private int n;
40
41     public synchronized void Get_in(int i)
42         {
43         int k;
44
45         while (true)
46             {
47             k = 2*(n%2) - (n/2) +1;
48             if (k==i) return;
49             try {wait();}
50             catch(Exception e) {System.exit(666);}
51             }
52         }
53     public synchronized void Get_out(int i)
54         {
55         n = i;
56         notifyAll();
57         }
58     }
```

a) 1320
b) 0123
c) 3210
d) 3120

PROBLEMS (7 points, 185 minutes)
PROBLEM 1 (3.5 points)

The program `my_music.c` stores information about a set of records, according to their category (Rumba, Rock and Rap) and calculates some duration statistics (see figure):



The `my_music.c` program has two processes, a parent and a child. The child process is in charge of loading the data structures with the disc information from a file, calculating the total duration for each of the 3 categories and sending a single message with the 3 values to the parent through a pipe. The parent must receive the message sent by the child through the pipe. In addition, during the execution of the process, its interruption through the SIGTSTP signal must be prevented.

As a guide, a program outline is provided, where some already implemented functions are provided, which may be useful for the implementation of the program.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

#define MUSIC_DATA_FILE "music_data.txt"
#define LINE_SIZE 120
#define MAXIMUM_TRACKS 20

typedef struct song_info song;
typedef struct disk_info disk, *disk_ptr;
typedef struct category_info category;

struct song_info
{
```

```
  unsigned duration;
  char *title;
};

struct disk_info
{
  char *disk_title;
  int num_tracks; /* Number of tracks in the disk */
  song songs[MAXIMUM_TRACKS];
  disk_ptr next;
};

struct category_info
{
  int num_of_disks;
  disk_ptr first;
};

enum categories {RAP, ROCK, RUMBA};
category rap, rock, rumba;
pid_t childpid, pid;
int status;

static int   initialize_data(char *filename);
static void destroy_data();
static void destroy_category(category *category_ptr);

// Calculates the duration of all the songs in a category
int calculate_duration(category *category_ptr){ ... }
int main(int argc, char **argv){ // SECTION 1.3 }
void add_item(song new_song, category *category_ptr, char *disk_title){ //
SECTION 1.1
}
static int initialize_data(char *filename) { // SECTION 1.2 }

// Destroy the data created
static void destroy_data() {...}
// Destroy the data for one category
static void destroy_category(category *category_ptr) {...}
```

Section 1.1 (1.2 points)
Implement the function add_item. The function receives as parameters a song (following the struct defined in the code), a pointer to the struct of the category to which the song belongs (which can be rap, rock or rumba, although for the purposes of this section it is indifferent) and the name of the disc to which the song belongs. The function will first search if the disc is already defined in the category. If the disc is found, the song will be added to the disc (you can assume that there will be no problems with the song limit in a disc) and if it is not found, the disc will be created, and the song will be added to the disc in the first position.

Note 1: when looking at the next section, remember that in all cases memory is reserved for disk_title.
Note 2: read the information of the structs in the statement carefully to be consistent with them.

Section 1.2 (1.1 points)
Complete the initialize_data function. This function is in charge of reading the file with the information of the disks and initialize all the data structures. An example of a data file is the following:

```
rumba,disk1,song1,120
rumba,disk1,song2,135
rumba,disk2,song1,90
rock,disk1,song1,56
rap,disk1,song1,96
```

IMPORTANT NOTE: Do not use spaces in the solution, as this section will be automatically corrected and answers with spaces will be considered invalid. Note also that the semicolons are already inserted and are not part of the solution.
NOTE: In some cases, when the same structure is used several times, it is asked only once and, in order not to give clues, it appears as HIDDEN in the following times.

```
static int initialize_data(char *filename) {
	FILE *fd;
	char *buffer = NULL;
	ssize_t readc=0;
	size_t n=0;
	char* token;
	char* rest;
	int duration;
	enum categories music_category;

	fd = __________(1)__________;

	if(fd==NULL){ return -1;}

	// Read lines and initialize data
	while((readc=________(2)_____________!=-1)){
		buffer[strcspn(buffer, "\r\n")] = 0;
		// Get individual values with strtok
		rest = buffer;

		// Get category
		token = strtok_r(rest, __(3)__, &rest);

		if(___(4)____(__(5)__,"rap") == 0) ___(6)__ = RAP;
		else if(HIDDEN(HIDDEN,"rock") == 0) HIDDEN = ROCK;
		else if(HIDDEN(HIDDEN,"rumba") == 0) HIDDEN = RUMBA;

		// Get disk
		token = strtok_r(rest, ",", &rest);
		char *disk_title = ____(7)___; // 1-argument function to copy the String

		// Get song title
		token = strtok_r(rest, ",", &rest);
		char *title = HIDDEN;

		// Get duration
		token = strtok_r(rest, ",", &rest);
		duration = ___(8)___(token);

		// Append to data structures
		song new_song;
		new_song.duration = duration;
		new_song.title = title;
		if(music_category==RAP){_________(9)__________; }
		if(music_category == ROCK) { HIDDEN; }
		if(music_category == RUMBA) { HIDDEN; }
	}
	______(10)_________;

	if(______(11)_______!=0){
		perror("Error when closing file");
		return -1;
	}
	return 0;
}
```

Section 1.3 (1.2 points)
Implement the main function. This function will create a child process, so that there is a parent process and a child process. The child process is in charge of loading the data structures with the disk information from a file, calculating the total duration for each of the 3 categories and sending a single message with the 3 values to the parent through a pipe. To initialize the data structure, you can use the functions already implemented and to calculate the total duration, you can use the function `calculate_duration()`, already implemented, which calculates the total duration of the songs in a category.
The output to be obtained for the above file is:

```
Duration of rumba songs: 345
Duration of rock songs: 56
Duration of rap songs: 96
```

To build a single string with all the above content, you can use the snprintf function, which puts in a string the content that would be printed following a format similar to printf. Its arguments are:

```
int snprintf( char *buffer, size_t count, const char *format [, argument] ...);
```

where buffer is the output String, count is the number of characters to store (you can use LINE_SIZE) and the following parameters are the same as printf.

As for the parent process, it, it must receive the message sent by the child through the pipe. In addition, during the execution of the process, its interruption through the SIGTSTP signal must be prevented.

Note: The solution must work without memory leaks and zombie processes. You can use already implemented functions as a help in this part.

PROBLEM 2 (1.4 points)
There is a store that sells pizzas. The threads will be instances of the classes Oven, Cashier and Waiter. The number of threads is predefined. Inside the store there are 3 ovens, a cashier and a waiter. The threads must be synchronized with semaphores.

- At each table there are 5 customers, each of whom orders a pizza. At the counter of the pizzeria there is room for only 6 pizzas.
- There are 3 ovens producing the pizzas. Each oven can bake two pizzas at the same time. The pizzas can only be placed on the counter if there is room. If not, the ovens wait until there is room. When the pizzas are ready, they are placed on the counter. If there is no room for new pizzas on the counter, the ovens wait. However, exceptionally, if there are 5 pizzas on the counter, as this value is less than 6, it is allowed to bake two pizzas, although at some point there may be 7 pizzas on the counter.
- After the ovens produce the pizzas, there is a cashier and a waiter. They are independent, but both must wait for the pizzas to cook in the oven. Thus, once there are 5 pizzas to serve a table, both the cashier can go to collect the money and the waiter can go to serve the 5 pizzas. As long as the pizzas are not served, the ovens cannot produce more if the counter is full.
    - The cashier collects the money for the 5 pizzas from a table. He can do it before or after the waiter serves the 5 pizzas. But he cannot collect the money if the ovens have not finished the pizzas.
    - Each waiter waits until he has the 5 pizzas of his order available (not taking some of the pizzas in his hands until all of them are available) before picking them up and taking them to the table he is serving.

The number of total tables can be variable, although to ensure that the system works correctly without adding more complexity, you can assume that the number of customers (5*number of tables) is going to be a multiple of 6, so that all ovens are turned on the same number of times and the oven does not have to be turned on for any single pizza because there is an odd value of pizzas to be prepared.

As a guide, part of the code is already implemented:

- Class Oven is responsible for preparing the pizzas. The preparePizzas() method receives the oven identifier (0, 1, 2) as an argument. Its goal is to prepare 2 pizzas in each oven. If there are more than 6 pizzas on the counter, it waits before preparing more.
- Class Cashier has one method, chargeBill(), which charges the bill of a table. This method must wait until 5 pizzas have been baked before charging them and is independent of whether they are served or not (they can be charged before or after).

- Class Waiter has a method servePizzas(), whose aim is to bring the 5 pizzas from the counter to a table. If there are not 5 pizzas on the counter, it waits.
- Class PizzaSem. This class carries out the synchronization functions with semaphores. It has a first part of definition of variables, which represent the baked pizzas that have not yet been served (nPizzas), the number of customers that have been served (nServedCustomers), the number of tables available to be charged (nTablesToBeCharged) and the number of charged tables (nChargedTables).
- Class PizzaSemaforos, which contains the main method which creates and initializes the threads.

Below you can find an example of execution:

```
Oven 0: [2][0][0][0]
Oven 1: [4][0][0][0]
Oven 2: [6][0][1][0]
Cashier: [6][0][0][1] total cashed tables 1
Waiter: [1][5][0][1] total served customers 5
Oven 0: [3][5][0][1]
Oven 1: [5][5][1][1]
Oven 2: [7][5][1][1]
Cashier: [7][5][0][2] total cashed tables 2
Waiter: [2][10][0][2] total served customers 10
Oven 0: [4][10][0][2]
Oven 1: [6][10][1][2]
Waiter: [1][15][1][2] total served customers 15
Cashier: [1][15][0][3] total cashed tables 3
Oven 2: [3][15][0][3]
Oven 0: [5][15][1][3]
Waiter: [0][20][1][3] total served customers 20
Oven 1: [2][20][1][3]
Cashier: [2][20][0][4] total cashed tables 4
Oven 2: [4][20][0][4]
Oven 1: [6][20][1][4]
Waiter: [1][25][1][4] total served customers 25
Cashier: [1][25][0][5] total cashed tables 5
Oven 0: [3][25][0][5]
Oven 2: [5][25][1][5]
Cashier: [5][25][0][6] total cashed tables 6
Waiter: [0][30][0][6] total served customers 30
```

The exercise

You are asked to complete the code to make the above system work. To do this, you must pay attention to the parts that are already filled that will ensure the solution is unique. In your solution, do not use spaces and note that the semicolons are already filled in, so do not use them. Except for the initialization of the semaphores, do not use numbers directly, and use the variables or constants defined throughout the program. If there is a defined variable, you must also use it as it is without using any operator.

```
class Oven extends Thread {
      int ident;
      PizzaSem f;

      public Oven(int id, PizzaSem f) {
            this.ident = id;
            this.f = f;
      }

      public void run() {
            int numPizzas = f.nTables * 5;
            int pizzas = numPizzas / 6;
            for (int i = 0; i < _____(1)______; i++) {
                  f.preparePizzas(____(2)_______);
                  try { Thread.sleep(200); } catch (Exception e) {
                  }
```

```
            }
      }
}

class Cashier extends Thread {
      PizzaSem f;

      public Cashier(PizzaSem f) {
            this.f = f;
      }

      public void run() {
            for (int i = 0; i < _____(3)_______; i++) {
                  f.chargeBill();
                  try { Thread.sleep(75); } catch (Exception e) {}
            }
      }
}

class Waiter extends Thread {
      PizzaSem f;

      public Waiter(PizzaSem f) {
            this.f = f;
      }

      public void run() {
            for (int i = 0; i < _____(4)_____; i++) {
                  f.servePizzas();
                  try { Thread.sleep(100); } catch (Exception e) {}
            }
      }
}

class PizzaSem {
      public static final int MAX_NUM_PIZZAS = 6;
      public static final int NUM_CLIENTS = 5;
      public int nTables;

      public PizzaSem(int nTables) {
            this.nTables = nTables;
      }

      private Semaphore semA = new Semaphore(__(5)___);
      private Semaphore semB = new Semaphore(__(6)___);
      private Semaphore semC = new Semaphore(__(7)___);
      private Semaphore semD = new Semaphore(__(8)___);

      private int nPizzas = 0;
      private int nServedCustomers = 0;
      private int nChargedTables = 0;
      private int nTablesToBeCharged = 0;

      public String toString() {
            return "[" + nPizzas + "][" + nServedCustomers + "][" +
nTablesToBeCharged + "][" + nChargedTables + "]";
      }

      public void preparePizzas(int id) {
            try {semA.acquire(); } catch (Exception e) {}
            while (___(9)____ >= MAX_NUM_PIZZAS) {
                  try { ______(10)______; } catch (Exception e) {}
```

```
                  try { ______(11)______; } catch (Exception e) {}
                  try { ______(12)______; } catch (Exception e) {}
            }

            nPizzas = nPizzas + 2;
            if ((nPizzas) / NUM_CLIENTS - (nPizzas - 2) / NUM_CLIENTS >
0)
                  ________(13)_________++;

            System.out.println("Oven " + id + ": " + toString());
            semC.release();
            _______(14)________;
            semA.release();
      }

      public void chargeBill() {
            try { semA.acquire(); } catch (Exception e) {}
            while (_____(15)______ < 1) {
                  try { ______(16)_______; } catch (Exception e) {}
                  try { semD.acquire(); } catch (Exception e) {}
                  try { ______(17)_______; } catch (Exception e) {}
            }

            nChargedTables = nChargedTables + 1;
            ______(18)________--;
            System.out.println("Cashier: " + toString() + " total cashed
tables " + nChargedTables);
            _______(19)________;
            semA.release();
      }

      public void servePizzas() {
            try { semA.acquire(); } catch (Exception e) {}
            while ((____(20)_____ < NUM_CLIENTS)) {
                  try { ____(21)________; } catch (Exception e) {}
                  try { _____(22)________; } catch (Exception e) {}
                  try { ______(23)_______; } catch (Exception e) {}
            }
            ____(24)_______ -= _____(25)_______;
            ______(26)_____ += ______(27)______;
            System.out.println("Waiter: " + toString() + " total served
customers " + nServedCustomers);
            _______(28)_________;
            semA.release();
      }
}

public class PizzaSemaforos {
      public static void main(String args[]) {
            int nMesas = 6;
            PizzaSem f = new PizzaSem(nMesas);
            for (int i = 0; i < 3; i++) {
                  new Oven(i, f).start();
            }
            new Cashier(f).start();
            new Waiter(f).start();
      }
}
```

PROBLEM 3 (2.1 points)

The nuclei of the atoms of ordinary matter are constituted by hadrons (protons [P] and neutrons [N]).

In turn, hadrons are constituted by 3 quarks; protons by two up-type quarks and one down-type quark, neutrons by one up and two down.

In a somewhat science fiction theory, the quantum fluctuations of the **Vacio** (**Empty)** *(this and other accents have been intentionally eliminated)* generate autonomously particle-antiparticle pairs, in our case quark-antiquark; the normal thing is that soon after these annihilate each other but, in some occasions, the antiparticle is attracted by a singularity before the annihilation obtaining as a result a net creation of matter.

In a state of sufficient energy, it is possible that the gluons of the strong interaction can reunite three quarks and form hadrons, let's call the **Agente** (**Agent)** that performs this action **Fuerte (Strong)**.

Once we have protons and neutrons, the W and Z bosons of the weak interaction can reunite protons and neutrons to form basic atomic nuclei; we will call the **Agente (Agent)** that performs this function **Debil (Weak)**.

In our case the **Weak Agents (threads of class Debil)** will be able to build nuclei of Hydrogen (1 proton), Helium (He = 2 protons + 2 neutrons) and Lithium (Li = 3 protons + 4 neutrons).

To simulate this theory, we will use threads that we will synchronize using monitors. One thread for each **Agente** (**Agent)**, including the **Vacio** (**Empty)** ones. The number of each type of agents is not predefined.

The threads will be instances of the classes **Empty, Weak** and **Strong (Vacio, Debil** and **Fuerte)**.

Consider the following statements:

Enumerations:

```
enum Quark  {UP,DOWN};
enum Hadron {PROTON,NEUTRON}
enum Nucleo {HIDROGENO,HELIO,LITIO}
```

The abstract class Agente (Agent):

```
abstract class Agente extends Thread
    {
    public void Join()
        {
        try {join();}
        catch(InterruptedException e)
            {
            System.err.println(e);
            System.exit(-1);
            }
        }
    public void Pausa(int min, int max)
        {
        int tiempo = min + (int)(Math.random()*max);
        Pausa(tiempo);
        }
    public void Pausa(int tiempo)
        {
        try
            {
            Thread.sleep(tiempo);
            }
        catch(InterruptedException e)
            {
            System.err.println(e);
            System.exit(-1);
            }
        }
    }
```

The main method:

```
public static void main (String[] args)
    {
    Thread hilo;
    ArrayList<Agente> hilos = new ArrayList<>();
    Espacio espacio = new Espacio();

    for (int i=0; i<NV; i++) hilos.add(new Vacio(espacio));
    for (int i=0; i<NF; i++) hilos.add(new Fuerte(espacio));
    for (int i=0; i<ND; i++) hilos.add(new Debil(espacio));
    for (Agente h : hilos) h.start();
    for (Agente h : hilos) h.Join();
    }
```

The Vacio (Empty) class:

```
class Vacio extends Agente
    {
    Espacio espacio;

    public Vacio(Espacio espacio)
        {
        this.espacio = espacio;
        }

    private Quark Next()
        {
        double v = Math.random();
        Quark q = (v>0.5) ? Quark.UP : Quark.DOWN;
        return(q);
        }

    public void run()
        {
        Quark q;
        while (true)
            {
            Pausa (1000, 2000);
            q = Next();
            espacio.Add(q);
            }
        }
    }
```

The Fuerte (Strong) class:

```
class Fuerte extends Agente
    {
    Espacio espacio;

    public Fuerte(Espacio espacio)
        {
        this.espacio = espacio;
        }

    private Hadron Next()
        {
        double v = Math.random();
        Hadron h = (v>0.3)
            ? Hadron.PROTON : Hadron.NEUTRON;
        return(h);
        }

    public void run()
        {
        Hadron h;

        while (true)
            {
            h = Next();

            switch (h)
                {
                case PROTON:
                    espacio.Get_quarks(2,1);
                    break;
                case default:
                    espacio.Get_quarks(1,2);
                    break;
                }
            Pausa (100, 500);
            espacio.Add(h);
            }
        }
    }
```

The **Get_quarks** method receives as arguments the numbers of ups and downs needed to build the hadron; its mission is to remove the indicated quarks from **Espacio (Space)** waiting until there are them if necessary.

The Debil (Weak) class:

```
class Debil extends Agente
    {
    Espacio espacio;

    public Debil(Espacio espacio)
        {
        this.espacio = espacio;
        }

    public Nucleo Next()
        {
        double v = Math.random();
        Nucleo n = (v>0.90)
            ? Nucleo.LITIO
            : (v>0.60)
                ? Nucleo.HELIO
                : Nucleo.HIDROGENO;
        return(n);
        }

    public void run()
        {
        Nucleo n;
        while (true)
            {
            Pausa (1000, 2000);
            n = Next();

            switch (n)
                {
                case HIDROGENO:
                    espacio.Get_hadrones(1,0);
                    break;
                case HELIO:
                    espacio.Get_hadrones(2,2);
                    break;
                default:
                    espacio.Get_hadrones(3,4);
                    break;
                }
            Pausa(100,500);
            espacio.Add(n);
            }
        }
    }
```

The **Get_hadrons** method receives as arguments the numbers of protons and neutrons needed to build the nucleus; its mission is to remove the indicated hadrons from **Espacio** (**Space)**, waiting until there are them if necessary.

The exercise:

Implement the **Espacio (Space)** class, which performs the monitor functions. To do this, the methods invoked in the code of the previous classes must be implemented.

Special care must be taken with the overloads used.

SOLUTION

TEST

1	2	3	4	5	6	7	8	9	10
C	A	A	B	B	B	D	B	C	D
11	12	13	14	15	16	17	18	19	20
A	A	A	A	A	A	A	A	A	A

PROBLEM 1

Part 1.1 (1.2 points)

```
void add_item(song new_song, category *category_ptr, char *disk_title){
      disk_ptr dptr = category_ptr->first;
      if(dptr == NULL) { // Disk is the first
            disk_ptr new_disk = (disk_ptr) calloc(1,sizeof(disk));
            new_disk->disk_title = disk_title;
            new_disk->num_tracks = 1;
            new_disk->songs[0] = new_song;
            category_ptr ->first = new_disk;
            category_ptr ->num_of_disks = 1;
      } else {
            // Try to find the disk
            while(dptr->next != NULL){
                  if(strcmp(disk_title, dptr->disk_title) == 0){
                        // Disk exists, so I add the song
                        dptr->songs[dptr->num_tracks] = new_song;
                        dptr->num_tracks = dptr->num_tracks + 1; // Assume
that the limit will not be exceeded
                        free(disk_title);
                        return;
                  }
                  dptr = dptr->next;
            }
            // Last disk to check. Otherwise add new disk
            if(strcmp(disk_title, dptr->disk_title) == 0){
                        // Disk exists, so I add the song
                        dptr->songs[dptr->num_tracks] = new_song;
                        dptr->num_tracks = dptr->num_tracks + 1; // Assume
that the limit will not be exceeded
                        free(disk_title);
                        return;
            } else { // Add new disk
                  disk_ptr new_disk = (disk_ptr) calloc(1,sizeof(disk));
                  new_disk->disk_title = disk_title;
                  new_disk->num_tracks = 1;
                  new_disk->songs[0] = new_song;
                  dptr->next = new_disk;
                  category_ptr ->num_of_disks++;
            }
      }
}
```

Part 1.2 (1.1 points)

1 fopen(filename,"r")
2 getline(&buffer,&n,fd)
3 ","
4 strcmp
5 token
6 music_category
7 strdup(token)
8 atoi
9 add_item(new_song,&rap,disk_title)

10 free(buffer)
11 fclose(fd)

Parts 1.3 (1.2 points)

```
int main(int argc, char **argv)
{
    int result;
      char readbuffer[LINE_SIZE];
      char string[LINE_SIZE];

      int fd[2], nbytes;
      signal(SIGTSTP,SIG_IGN);

      // Pipe creation
      if(pipe(fd)<0){
            perror("pipe creation");
            exit(EXIT_FAILURE);
      }
      // Child creation
      childpid = fork();
      if((childpid < 0)){
            perror("fork");
            exit(EXIT_FAILURE);
      }
      if(childpid == 0){
            result = initialize_data(MUSIC_DATA_FILE);
            if (result != 0)
            {
              printf("\n *** Error while reading the music data file.\n");
              destroy_data();
              return 1;
            }
            close(fd[0]);
            snprintf(string,  LINE_SIZE,  "Duration  of  rumba  songs:
%d\nDuration  of  rock  songs:  %d\nDuration  of  rap  songs:  %d",
calculate_duration(&rumba),                    calculate_duration(&rock),
calculate_duration(&rap));
            nbytes = write(fd[1], string, (strlen(string)+1));
            if(nbytes == -1){
                  perror("Error writing pipe");
            }
            destroy_data();
            _exit(EXIT_SUCCESS);

      } else {
            close(fd[1]);
            nbytes = read(fd[0], readbuffer, sizeof(readbuffer));
            if(nbytes != -1){
                  printf("%s\n", readbuffer);
            } else {
                  perror("Error reading pipe");
            }
            pid = wait(&status);
            exit(EXIT_SUCCESS);
      }
    return 0;
}
```

PROBLEM 2

1. pizzas
2. ident
3. f.nTables
4. f.nTables
5. 1
6. 0
7. 0
8. 0
9. nPizzas
10. semA.release()
11. semB.acquire()
12. semA.acquire()
13. nTablesToBeCharged
14. semD.release()
15. nTablesToBeCharged
16. semA.release()
17. semA.acquire()
18. nTablesToBeCharged
19. semB.release()
20. nPizzas
21. semA.release()
22. semC.acquire()
23. semA.acquire()
24. nPizzas
25. NUM_CLIENTS
26. nServedCustomers
27. NUM_CLIENTS
28. semB.release()

PROBLEM 3

```
class Espacio
    {
    private int n_ups = 0;
    private int n_downs = 0;
    private int n_protones = 0;
    private int n_neutrones = 0;
    private int n_hidrogenos = 0;
    private int n_helios = 0;
    private int n_litios = 0;

    public String Estado()
        {
        return  ("["+n_ups+","+n_downs+"]  ["+n_protones+","+n_neutrones+"]
["+n_hidrogenos+","+n_helios+","+n_litios+"]");
        }

    public synchronized void Add(Quark q)
        {
        switch (q)
            {
            case UP: n_ups++; break;
            default: n_downs++; break;
            }
        notifyAll();
        }
    public synchronized void Add(Hadron h)
        {
        switch (h)
            {
```

```
            case PROTON: n_protones++; break;
            default: n_neutrones++; break;
            }
        notifyAll();
        }
    public synchronized void Add(Nucleo n)
        {
        switch (n)
            {
            case HIDROGENO: n_hidrogenos++; break;
            case HELIO: n_helios++; break;
            default: n_litios++; break;
            }
        }
    public synchronized void Get_quarks(int nu, int nd)
        {
        while (n_ups < nu || n_downs < nd) Wait();
        n_ups -= nu;
        n_downs -= nd;
        }
    public synchronized void Get_hadrones(int np, int nn)
        {
        while (n_protones < np || n_neutrones < nn) Wait();
        n_protones -= np;
        n_neutrones -= nn;
        }

    private void Wait()
        {
        try
            {
            wait();
            }
        catch (Exception e)
            {
            System.err.println(e.getMessage());
            System.exit(666);
            }
        }
    }
```

Assessment criteria

PROBLEM 1 (3.5 points)
Section 1.1 (1.2 points)

- 0.2: Case when the list is empty
- 0.2: Loop and correct movement throughout the list to search the disk
- 0.4: Case when the disk is found. Penalize 0.1 if the condition is wrong and 0.1 is free is missing.
- 0.2: Correct management of the pointers with the references to the list to make links work. You can stop in the last case and manage that case or use another approach (e.g., traverse the list first to determine if the disk is found) but that must be coherent to ensure information can be correctly added.
- 0.2: Case when the disk is not found after traversing the list, which must ensure that the new list is coherent
- Significant errors may be subject to additional penalties.

Section 1.2 (1.1 points)

- 0.1 each gap

Section 1.3 (1.2 points)

- 0.1: Declaration of the variables and the pipe. If the pipe is not correctly created, then 0.
- 0.1: Management of the signal
- 0.1: Creation of the child process
- 0.1: Correct call to initialize_data by the child process
- 0.1: Creation of the string to be sent
- 0.2: Correctly write the information in the pipe by the child process, including the close call
- 0.1: Correct management of the memory by calling destroy_data where necessary
- 0.1: Exit of the processes
- 0.2: Correct reception of the data throught the pipe by the parent process, including the close call
- 0.1: Wait to avoid the child becomes a zombie
- Significant errors may be subject to additional penalties.

PROBLEM 2 (1.4 points)

- 0.05 each gap

PROBLEM 3 (2.1 points)

- 0.35: Class declaration and definition of variables
- 0.35 each of the 5 methods (3 add, Get_quarks and Get_hadrones)
- Significant errors may be subject to additional penalties.