UNIVERSIDAD DE MÁLAGA
Dpto. Lenguajes y Ciencias de la Computación

Systems Programming and Concurrency

ANDALUCÍA TECH
Campus de Excelencia Internacional
Campus of International Excellence

Exam of C, March 2023

Description of the system

This exam is divided into two parts:

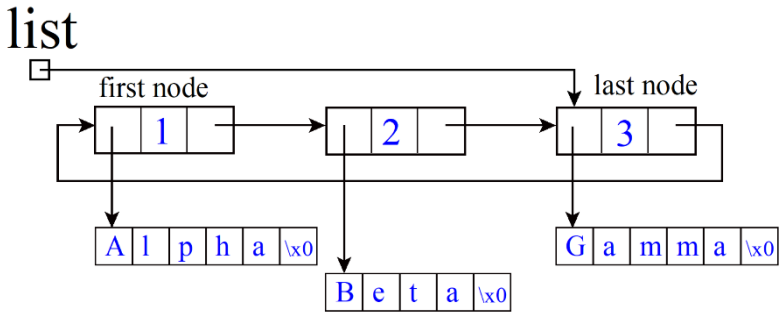
Part I: Circular Linked List (7,5 pts.)

In this part you have to develop a Circular linked List (CLL) whose nodes have the next structure:

```
typedef struct Node * TList;

struct Node {
	char * name; // This is, simply, a pointer.
	int value;
	struct Node * ptrNext;
};
```

The main pointer points to the last node of the CLL, whereas the last one points to the first. This way, it is easy to access both the last and the first nodes of the list. The next diagram shows this structure in a list with three nodes where the first is Alpha-1 and the last is Gamma-3:



When a CLL is empty, the pointer **list** points to NULL; and, when it holds a single node, such a node points to itself. Please, note that this is not a sorted list.

This CLL is different from the lists seen at class in three aspects:

1. They are circular, i.e., the last node points to the first.
2. The main pointer points to the last node.
3. Each node contains not only an **int**, but also a **char*** (note that **name** is not an array of char but a pointer to char).

To develop this part, you are given a header file **List.h** and source code file **MidtermC2023.c** to test your code. You have to implement the next functions in the file **List.c**:

```
// Makes the list to point to NULL
// 0,25 pts.
void createList(TList * ptrL);

// Insert a new node at the end of the list.
// Memory for the node must be allocated, as well as for the name.
```

```
// If no memory can be allocated (malloc returns NULL), then
// this function returns immediately.
// 1.50 pts.
void insertList(TList * ptrL, char * name, int value);

// Remove the first node of the list and frees all its memory.
// If the list is empty the stores 0 in ok, otherwise stores 1.
// 1,50 pts.
void deleteList(TList * ptrL, unsigned char *ok);

// Returns a pointer to the first node of the list.
// If the list is empty the stores 0 in ok, otherwise stores 1.
// 0,5 pts.
struct Node * firstList(TList list, unsigned char *ok);

// Prints the content of the list: from the first to the last.
// name and value are printed in different lines.
// 1,50 pts.
void showList(TList list);

// Return how many nodes the list has.
// 1,25 pt.
int lengthList(TList list);

// Frees all the memory used by the list and makes the list to point to NULL.
// Hint: Use deleteList as an auxiliary function.
// 1,00 pt.
void destroyList(TList * ptrL);
```

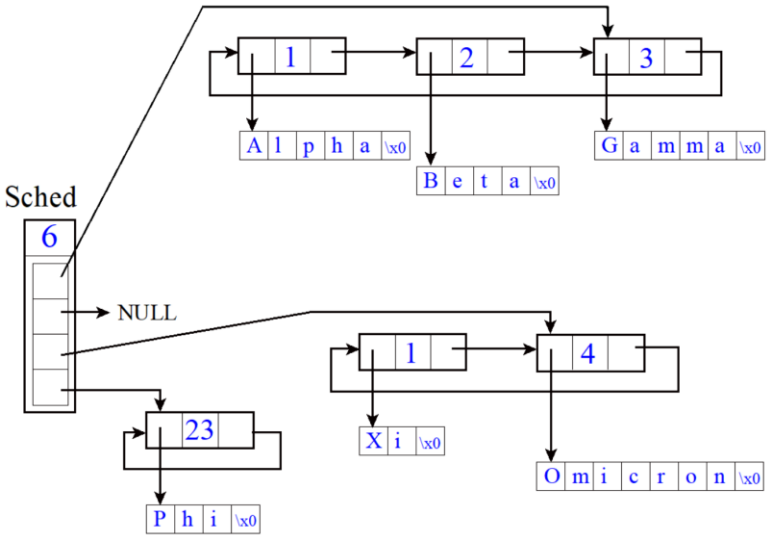
Part II: Scheduler (2,5 pts.)

In this part you have to develop a function to load from a file the content of a Scheduler. A Scheduler is a structure composed by **NUM_PROCESSORS** CLL and an integer **numProcesses**. This is defined in the file **Scheduler.h** as follows:

```
#define NUM_PROCESSORS 4

typedef struct Scheduler {
	unsigned int numProcesses;
	TList vector[NUM_PROCESSORS];
} TScheduler;
```

A graphical example of a Scheduler may be the next:



Note that a scheduler is not a pointer but a structure.

In this part, you have to develop the next function inside **Scheduler.c**:

```
// Creates a brand new Scheduler fulfilled with the textual data
// contained in a file with the next structure:
// numProcesses
// length of the list at position 0 of vector
// The list at position 0 of vector
// length of the list at position 1 of vector
// The list at position 2 of vector
// ...
// length of the list at position NUM_PROCESSORS-1 of vector
// The list at position NUM_PROCESSORS-1 of vector
// Each list is a sequence of name and value, from the first to the last node.
// 2,00 pts.
TScheduler readFile(char * filename, unsigned char * ok);
```

The file **Scheduler.txt** contains an example of a textual file to be read by this function:

```
12
3
Printer01
12
HDD03
22
USB02
32
0
1
Thunderbolt01
11
8
Printer11
11
Printer21
21
Printer31
31
USB11
111
USB-C101
101
```

```
TPM11
1011
Audio21
201
HDMI41
41
```

ADDENDUM.

The headers of the functions to manage strings (<**string.h**>) are:

char *strcpy(char *s1, const char *s2): copies the string pointed by s2 (including the final NULL character) into the string pointed by s1.

int strcmp(const char *s1, const char *s2): compares the string pointed by s1 against the string pointed by s2. And returns:

- Greater than zero when s1>s2 (lexicographical order)
- Zero if s1==s2 (lexicographical order)
- Lower than zero when s1<s2 (lexicographical order)

size_t strlen(const char *s): returns the number of characters of the string pointed to by s. The final character '\0' of the string is not counted.

Following are the headers of the functions to read and write to/from files provided by the library <**stdio.h**> (you should know by heart the headers of the required functions in <**stdlib.h**>, as **free** or **malloc**, and <**stdio.h**> as **scanf** and **printf**):

FILE *fopen(const char *path, const char *mode): opens the filename pointed to by path using the given mode ("rb"/"wb" to binary read/write, and "rt"/"wt" to text read/write). This function returns a FILE pointer; otherwise, NULL is returned.

int fclose(FILE *fp): saves the buffer content and closes the file. Returns 0 if success; otherwise -1 is returned.

unsigned fread(void *ptr, unsigned size, unsigned nmemb, FILE *stream): reads nmemb data blocks, each of them composed of size bytes, from the file stream, and stores them in the address given by ptr. Returns the number of blocks read.

unsigned fwrite(const void *ptr, unsigned size, unsigned nmemb, FILE *stream): writes nmemb data blocks, each of them composed of size bytes, to the file stream, taking them from the address pointed by ptr. Returns the number of blocks written.

int fprintf(FILE *stream, const char *format, ...): this function works the same than **printf** but, instead or printing a result on console, it sends the result to the text file represented by **stream**.

int fscanf(FILE *stream, const char *format, ...): this function works the same than **scanf** but, instead or reading the keyboard, it reads from the text file represented by **stream**.