Exercise 1

An ASCII file is organized as a sequence of lines. Each line specifies a shell command followed by all its parameters.

The following is a correct example of such a file:

```
ls -laR
wc --byte --words --lines
find . -name "*.c" -exec grep -H "foo" \{} \;
...
```

Write the function

```
void my_exec (char *name);
```

which receives the name of the file as a parameter, and it runs all commands specified in the file appropriately using the system calls **fork** and **exec**.

Please, remind the prototype of the following system calls:

```
pid_t fork (void);
int execv (char *path, car *argv[]);
int execvp (cahr *name, char +argvè]9;
int execve (char *path, char *argv[], char *envp[]);
```

Solution

Exercise 2

Two UNIX processes must transfer information between them. Depending on

- The quantity of information the two processes need to transfer (a few bytes/sec or Gbytes/seconds)
- The type of information to transfer (formatted or byte-oriented)
- The relationship between the processes (related or unrelated)

describe the advantages and disadvantages of using pipes, FIFOs, message queues, and shared memory.

Solution

- Pipe: P with relationship, limited amunt of data, byte oriented, R/W blocking
- FIFO: Simular to pipe, P do not have to have a relationhip, byte-oriented
- Message Queues: must have a key that the OS tranform into an id (ftok key from pathname and process id), oriented to structured data type (msgget, msgctl, msgsnd, msgrcv); fetching order may be establiehd run time
- Shared Memory: faster than other methods, do not require kernel intervention, need synch among P, uses ftok, shmget, shmat, shmdt, shmctl

Exercise 3

In a Windows system, a **binary** file stores an undefined number of **fixed length records**. Each line includes the following fields:

- The identifier current_id of the current record, i.e., an integer value increasing at each line, and starting from 0 on the first line.
- Two strings, s1 and s2, each one of exactly 30 characters.
- A real (float) value f.
- The identifier next_id of the next record.

Overall, the structure of each record is the following one:

```
current_id     s1     s2     f     next_id
```

Essentially, each line of identified **current_id** points to (that is, refers or indicates) the line **logically** following it, i.e., the one with the identifier **next_id**. The following is the ASCII version of a correct binary file:

```
0  Harry  Potter  9.5  4
1  Hermione  Granger  9.0  7
2  Ron  Weasley  8.5  1
3  Albus  Silente  8.6  0
4  Severus  Piton  6.5  -2
5 ...
```

where, for example, the line of **current_id=0** (i.e., "Harry Potter 9.5") indicates the line with identifier **next_id=4** as next line (i.e., the line "Severus Piton 6.5"), which in turns does not have any line following it, as its **next_id** is equal to **-2.**

Write the procedure

```
void my_read (LPCTSTR file_name, DWORD current_id);
```

which opens the binary file of name **file_name**, and then it reads it, following the chain of identifier **current_id -> next_id**, with the following logic:

1. It reads and displays (on the standard output) the line with the identifier equal to **current_id**.
2. It moves on the record whose identifier is specified by the identifier **next_id** of the current line until such an identifier is a negative value or it identifies a record that does not exist in the file.
3. It repeats the entire process from step 1.

For example, with the previous file, starting from **current_id=3**, the function must display (on standard output) the following values:

```
Albus Silente 8.6
Harry Potter  9.5
Severus Piton 6.5
```

Note: if you do not remember the exact syntax of a specific system call, please write down a version that likely resembles what you remember together with some comments, briefly summarizing what you were willing to use.

Solution

```
typedef struct data_s{
DWORD id;
TCHAR s1[30+1];
TCHAR s2[30+1];
FLOAT f;
DWORD nextId;
} data_t

void my_read(LPCTSTR file_name, DWORD current_id){
HANDLE hin;
data_t row;
DWORD nIn, id = 1;
OVERLAPPED ov = {0, 0, 0, 0, NULL};
LARGE_INTEGER filepos;

hin = CreateFile(file_name, GENERIC_READ, 0, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, NULL);
id = current_id
while(id >= 0 && it-exist){
  filepos.Quadpart = id * sizeof(data_t);
  ov.Offset = filepos.LowPart;
  ov.OffsetHigh = filepos.HighPart;
  ReadFile(hin, &row, sizeof(data_t), &nIn, &ov);
  // Check if nIn is equal to the size of data_t, if not error
  _tprintf(_T("%s %s %f\n"), row.s1, row.s2, row.f);
  id = row.nextId;
}
CloseHandle(hin);
return;
}
```

Solution 2

```
typedef struct record {
  DWORD current_id;
  LPCTSTR name[30];
  LPCTSTR surname[30];
  FLOAT f;
  DWORD next_id; } record_t

void my_read (LPCTSTR file_name, DWORD current_id) {

  HANDLE fileH;
  record_t rec;
  DWORD size, rec_number, actual_read;
  LARGE_INTEGER distance;
  PLARGE_INTEGER newPointer;


  // OPEN FILE
  fileH = CreateFile (file_name; GENERIC_READ, 0, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
  // GET FILE SIZE -> compute record number
  size = GetFileSizeEx(fileH, NULL);
  rec_number = (DWORD) size / sizeof(record_t);
  rec.next_id = current_id;
  while (rec.next_id < max_rec && rec.next_id > 0)  {
     // MOVE FILE POINTER
     SetFilePointerEx(fileH, distance, newPointer, FILE_BEGIN);

    // READ
    FileRead(fileH, &rec, sizeof(record_t),  &actual_read, NULL);
    // PRINT
    _tprinf("%s %s %f \n", rec.name, rec.surname, rec.f)

  }

  CloseHandle(fileH);

}
```

Exercise 4

Implement the POSIX system call

```
void pthread_barrier_wait (&barrier);
```

in the Windows system, using only **semaphores** and **mutexes**, in a situation in which **the barrier is inserted within a cycle which is repeated more than once**. Motivate and describe your solution with short comments.

Please, remind and use the following system calls.

```
HANDLE CreateMutex(LPSECURITY_ATTRIBUTES lpsa,BOOL fInitialOwner,LPCTSTR
lpszMutexName);
BOOL ReleaseMutex (HANDLE hMutex);
HANDLE CreateSemaphore (LPSECURITY_ATTRIBUTES lpsa,LONG cSemInitial,LONG
cSemMax,LPCTSTR lpszSemName);
BOOL ReleaseSemaphore (HANDLE hSemaphore,LONG cReleaseCount,LPLONG
lpPreviousCount);
DWORD WaitForSingleObject (HANDLE hObject,DWORD dwTimeOut);
```

Solution

```
LPHANDLE bar1, bar2, me;
DWORD count;
...
me = (LPHANDLE) malloc(sizeof(HANDLE));
bar1 = (LPHANDLE) malloc(sizeof(HANDLE));
bar2 = (LPHANDLE) malloc(sizeof(HANDLE));

me = CreateMutex(NULL, FALSE, NULL);
bar1 = CreateSemaphore(NULL, 0, n, NULL); // where n is the number of
thread that will run
bar2 = CreateSemaphore(NULL, 0, n, NULL);
...
WaitForSingleObject(me, INFINITE);
  count ++;
  if(count == n){ //where n is the number of running thraed
    ReleaseSemaphore(bar1, n, NULL); // n OR CYCLE
  }
```

```
ReleaseMutex(me);
WaitForSingleObject(bar1, INFINITE);
...
WaitForSingleObject(me, INFINITE);
  count --;
  if(count == 0){ // the last thread enter this if
    ReleaseSemaphore(bar2, n, NULL);
  }
ReleaseMutex(me);
WaitForSingleObject(bar2, INFINITE);
```

Exercise 5

Write a C++ multi-thread piece of code that is able to:

1. Run N threads, each one generating a random char and storing it into a vector.
2. Generate 1 thread evaluating how many chars in the vector are digits.
3. Generate 1 thread evaluating how many chars in the vector are punctuation marks.
4. Generate 1 thread evaluating how many chars in the vector are letters.

Let the synchronization be coherent with the task each thread has to carry.

Note: if you do not remember the exact syntax of a C++ class, write down a version that likely resemble what you remember together some "// comment" briefly summarizing what you were willing to use.

Solution

```
// FUNCTIONS LETTERS AND MARKS ARE EQUAL TO DIGITS

void digits () {
  l1.shared_lock()
  int i = 0;
  // count digits int the array

  l1.unlock();

}

void rand_func (std::vectr<int>& arr) {
    int n = rand();

    l2.lock()
    arr.emplace_back(int);
    l2.unlock()
}


int main() {
  std::vector<thread> ts;
  std::vector<int> arr;
```

```
 std::mutex mut1;
 std::shared_mutex shmut2;
 std::unique_lock l1{mut1}, l2{shmut2}
 int m, d, l;

 std::thread tdig(digits, &d);
 std::thread tmark(marks, &m);
 std::thread tlet(letters, &l);

 l2.unlock()

 for (int i=0; i<N; i++) {
   ts.emplace_back(std::thread(rand_func));
   i++
 }

 for(int i=0; i < N; i++)
   ts[i].join();

 l1.unlock()
 tdig.join() ;
 tmark.join();
 tlet.join()

}
```

Exercise 6

Write a small C++ program that computes the factorial of several numbers using asynchronous tasks and prints them altogether in the main program once all computations are completed. Each task will get its input from a queue container that is going to be filled by the main program as soon as the user enters the numbers.

Note: if you do not remember the exact syntax of a C++ class, write down a version that likely resemble what you remember together some "// comment" briefly summarizing what you were willing to use.

Solution

```
#include <iostream>
#include <thread>
#include <future>
#include <queue>
using namespace std;

#define N 10

queue q[N]; //queue containing N values

int main(){
 promise<int> vettP[N];
 future<int> vettF[N];
 future<int> vettAsync[N];
 int i, num;

 for(i=0; i<N; i++){
  vettF[i] = vettP[i].get_future();
  vettAsync[i] = async(factorial, ref(vettF[i])); //call to the async function for each task
 }

 for(i=0; i<N; i++){
  cout << "Insert value: ";
  cin >> num;
  q.push(num); //insertion inside the queue of the number obtained as input
  vettP[i].set_value(num); //passing to the async task its index, used to access the value in the queue
 }

 for(i=0; i<N; i++){
  cout << "Factorial of number in " << i << "position: " << q[i]; //printing the result inside the queue
 }
 return 0;
}

void factorial(ref<future> &fut){
 int i, index, res;
 index = fut.get(); //the index of the task is obtained, the async task can now start computing
 res = q[index]; //the value the task has to calculate the factorial of is obtained from the queue
 for(i=res-1; i>0; i--){
  res = res*i; //calculation of the factorial value
 }
 q[index] = res; //saving the result in the queue so that the main can print it
 return;
}
```