Exercise 1

Using the following data structure and system calls:

```
struct dirent {
  inot_t d_no;
  char d_name[NAM_MAX+1];
  . . .
};
DIR *opendir (const char *filename);
struct dirent *readdir (DIR *dp);
int closedir (DIR *dp);
```

write the function

```
void visit (char *pathname, char *prefix, int level);
```

which displays the content of all directories within the directory called **pathname** whose name start with the string **prefix** and have a level (within the directory tree) equal to **level**.
For example, if **pathname is "mydir"**, **prefix is "foo"** and **l=2**, the function must display the content of all subdirectories of directory **"mydir"** such as **"foo1"**, **"foobar", "fooABC"**, etc., at level 2 of the directory tree rooted at **"mydir"**.

Solution

```
void visitDirRecur (char *fullnameR, int level) {
  DIR *dp;
  struct stat statbuf;
  struct dirent *dirp;
  char nameR[N];

  if (lstat(fullnameR, &statbuf) < 0 ) {
    fprintf (stderr, "Error Running lstat.\n");
    exit (1);
  }
  if (S_ISDIR(statbuf.st_mode) == 0) {
    return (1);
  }
  if ( (dp = opendir(fullnameR)) == NULL) {
    fprintf (stderr, "Error Opening Dir.\n");
    exit (1);
  }
  while ( (dirp = readdir(dp)) != NULL) {
    sprintf (nameR, "%s/%s", fullnameR, dirp->d_name);
    if (lstat(nameR, &statbuf) < 0 ) {
      fprintf (stderr, "Error Running lstat.\n");
      exit (1);
    }
    if (S_ISDIR(statbuf.st_mode) == 0) {
      if (level==2 && strstr (dirp->d_name, prefix)) {
        fprintf (stdout, "Level=%d File=%s\n", level+1, dirp->d_name);
      }
    } else {
      /* Directory */
      if (strcmp(dirp->d_name, ".") == 0 ||
        strcmp(dirp->d_name, "..") == 0)
        continue;
      visitDirRecur (nameR, level+1);
    }
  }
```

```
  if (closedir(dp) < 0) {
    fprintf (stderr, "Error.\n");
    exit (1);
  }

  return;
}
```

Exercise 2

A program runs N threads of type A, N threads of type B, and N threads of type C.
N is a positive integer passed on the command line (e.g., 10) and all threads are identified by their category (i.e., A, B or C) and a creation index (i.e., 1, 2, ..., N).
Coordinate the effort of all threads such that for each set of 3 threads running consecutively, there are only two possible sequences: A B C or C B A.
The following is a correct example of execution with N=10:

```
A3 B1 C6
A2 B4 C1
C8 B7 A8
etc.
```

Write the code using the C language but, or the sake fo simplicity, suppose semaphores can be managed by pseudo-code functions such as: init(sem,k), wait(sem), signal (sem), and destroy (sem).

Solution

```
num_threads = atoi(argv[1]);
sa = (sem_t *) malloc (sizeof(sem_t));
sb = (sem_t *) malloc (sizeof(sem_t));
sc = (sem_t *) malloc (sizeof(sem_t));
sac = (sem_t *) malloc (sizeof(sem_t));
pi = (int *) malloc (num_threads * sizeof(int));
sem_init(sa,0,1); sem_init(sc,0,1); sem_init(sac,0,1); sem_init(sb,0,0);
n = 0;
for (i=0; i<num_threads; i++){
  pi[i] = i;
  pthread_create (&th, NULL, TB, pi);
  pthread_create (&th, NULL, TA, pi);
  pthread_create (&th, NULL, TC, pi);
}
free (sa); free (sb); free (sc); free (sac); free (pi);
pthread_exit(0);

static void *TA (void *arg) {
  int id;
  int *pi = (int *) arg;
  id = *pi;
  pthread_detach (pthread_self ());
  sem_wait (sa);
  sem_wait (sac);
  printf ( "A%d ", id);
  n++;
  if (n==1) {
    sem_post (sb);
  } else {
    n=0; sem_post(sa); sem_post(sc); sem_post(sac);
  }
  return 0;
}
```

```
static void *TC (void *arg) {
  int id;
  int *pi = (int *) arg;
  id = *pi;
  pthread_detach (pthread_self ());
  sem_wait (sc);
  sem_wait (sac);
  printf ( "C%d ", id);
  n++;
  if (n==1) {
    sem_post (sb);
  } else {
    n=0; sem_post(sa); sem_post(sc); sem_post(sac);
  }
  return 0;
}

static void *TB (void *arg) {
  int id;
  int *pi = (int *) arg;
  id = *pi;
  pthread_detach (pthread_self ());
  sem_wait (sb);
  printf ("B%d ", id);
  sem_post (sac);
  return 0;
}
```

Exercise 3

A UNICODE file stores a sequence of records, in which each record includes 3 fields: an integer value, a string, and a real number. The 3 fields have all variable size and are separated by a variable number of spaces. The following is a possible correct format of such a file:

```
123 ThisIsAString 45.67
687989 ABC -12345.879
. . .
```

Show how it is possible to use the system call

```
BOOL ReadFile (
  HANDLE hFile, LPVOID lpBuffer, DWORD nNumberOfBytesToRead,
  LPDWORD lpNumberOfBytesRead, LPOVERLAPPED lpOverlapped
);
```

To store the indicated records in an array of structures of type **record_t**, defined as follow:

```
#define N 1000

struct record_s {
  DWORD i;
  TCHAR s[N];
  FLOAT f;
} record_t;
```

Solution

```
// I suppose the file handle has already been opened and its handle is "fileHandle"

TCHAR c;
int n = 1;
BOOL not_end = true;
```

```
// iterate over all file until EOF is found or an error occurs
while(not_end && n){

TCHAR line[MAXIMUM_LINE_LENGTH];
int position = 0;

do {
   not_end = ReadFile(fileHandle, &c, sizeof(TCHAR), &n, NULL);
   if(not_end && n != 0)
   {
     line[position] = c;
     position++;
   }

// I suppose that newline are represented only by a '/n'
} while(c != _T('\n') && not_end && n != 0);

// once here the line array will contains the bytes relative to a line (also in case
end of file
// has been reached
// they can be parsed with _stscanf()

record_t r;

_stscanf(line, _T("%d %s %f"), &r.i, r.s, &r.f);

//
// process the current record
//
...
}
```

Exercise 4

Write two small segments of C code to explain how it is possible to use the two following system calls

```
DWORD WaitForSingleObject (HANDLE hObject, DWORD dwTimeOut);
DWORD WaitForMultipleObjects (DWORD nCount, LPHANDLE lpHandles, BOOL fWaitAll, DWORD
dwTimeOut);
```

to wait for N threads (with N>64).

Why "N>64"?

If the epilogue of a critical section includes (only) the following system call

```
ReleaseSemaphore (hSem, 3, &previousCount);
```

How would you write the corresponding prologue? Motivate the strategy adopted.

Solution

Parte I

```
for (i=0; i<N; i++) {
  WaitForSingleObject (hSem[i], INFINITE);
}
```

PARTE II

```
while (N > 0) {
  INDEX = WaitForMultipleObjects (min (MAXIMUM_WAIT_OBJECTS, N),
            hSem, FALSE, INFINITE);
  index = (int) INDEX - (int) WAIT_OBJECT_0;
```

```
  CloseHandle (hSem[index]);
  ...
  hSem[index] = hSem[N-1];
  ... Free threadData[index] ...
  threadData[index] = threadData[N-1];
  N--;
}
```

PARTE III

```
CRITICAL_SECTION cs;
...
InitializeCriticalSection (&cs);
...
...
EnterCriticalSection (&cs);
WaitForSingleObject (hSem, INFINITE);
WaitForSingleObject (hSem, INFINITE);
LeaveCriticalSection (&cs);
...

ReleaseSemaphore (hSem, 2, &previousCount);
```

Exercise 5

Verify the following C++ multi-thread code and propose a solution that avoids deadlocks without introducing or removing any mutex:

```
std::mutex read_m, write_m, verify_m;
void writer() {
    read_m.lock(); write_m.lock(); verify_m.lock();
    // body
    verify_m.lock(); write_m.lock(); read_m.lock();
 }

void reader() {
   write_m.lock(); read_m.lock();
    // body
    read_m.unlock(); write_m.unlock();
}

void verifier() {
    verify_m.lock(); write_m.lock();
    // body
    write_m.unlock(); verify_m.unlock();
}
```

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

```
void writer(){
  // same as above
}

void reader(){
  read_m.lock(); write_m.lock();
  // body
  write_m.lock(); read_m.unlock();
}

void verifier(){
```

```
  write_m.lock(); verify_m.lock();
  // body
   verify_m.lock(); write_m.lock();
}
```

Exercise 6

Write a small C++ example able to start N threads and let them all waiting for their input value to start the internal computation. The input value (for each thread) comes from the user input, and it must happen after the thread creation.

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

```
promise<int> p[N];
future<int> f[N];
thread ts[N];
int x;

for(int i=0; i < N; i++){  //start threds and pass them the future used for acquire the value

      f[N] = p[N].getFuture();
      ts=thread(thred_func, ref(f[N]));
}

for(int i=0; i<N; i++){   //take input and use promise to pass value
    cin >> x;
    p[N].setValue(x);
}

thread_func(future<int>& fut){
   int y;

  y=fut.getValue();

  \\start computing
}
```