System and device programming
29 June 2011

(Theory: no textbooks and/or course material allowed)

(15 marks) The final mark is the sum of the 1st > 8 and the 2nd part > 10 .

1. (3.0 marks) Describe the system data structures that are used when the system call **`open`** is performed.

2. (3.0 marks) Write the sequence of instructions that allow the bash process to interpret and execute the command
   `p1 | p2`
   where **`p1`** and **`p2`** are two executable files.

3. (2.0 marks) Explain the main features of multiple heaps in Win32. What are the advantages of using multiple heaps vs. one single heap? What can be the main benefit in multi-threaded programs? How can multiple heaps reduce memory fragmentation? Is it possible to free all the dynamically allocated memory of a heap at once, avoiding individual free of allocated data?

4. (2.0 marks) Describe the behaviour of events as synchronization data, with respect to the 4 possible cases given by Set/Pulse-Event and Auto/Manual-Reset? How are events used for asynchronous I/O?

5. (5.0 marks) The following incomplete routines (working in a win32 environment) have the purpose of modifying the content of selected records within an open file. The file is a large file (requiring 64 bit pointers), containing fixed length records (as defined by **`structure record_t`**). The purpose of the functions is to increment the **`num`** field of one record every 1000 records. The prototypes of some win32 file management functions are reported below for convenience:

```
BOOL ReadFile (HANDLE hFile, LPVOID lpBuffer, DWORD nNumberOfBytesToRead,
   LPDWORD lpNumberOfBytesRead, LPOVERLAPPED lpOverlapped);
BOOL WriteFile (HANDLE hFile, CONST VOID *lpBuffer, DWORD nNumberOfBytesToWrite,
   LPDWORD lpNumberOfBytesWritten, LPOVERLAPPED lpOverlapped);
DWORD SetFilePointer (HANDLE hFile, LONG lDistanceToMove,
   PLONG lpDistanceToMoveHigh, DWORD dwMoveMethod);
```

   Complete the following routines, according to the specifications.

```
typedef struct {
  /* data fields omitted */
  /* ... */
  int num;
} record_t;

/* this function requires file pointer update, using SetFilePointer */
int updateUsingFilePointers(HANDLE fh, DWORD period) {
  DWORD fsLow, fsHigh;
  LONGLONG n;
  record_t r;
  fsLow = GetFileSize (fh, &fsHigh);

  ..............
  while (.....................) /* for each record every period in file */
  {
    ..............
    SetFilePointer (.....................................)
    ReadFile (fh, &r, ........................);
    r.num++;
    SetFilePointer (.....................................)
    WriteFile (fh, &r, ........................);
    ..............
  }
}

/* this function exploits an overlapped structure for pointer management */
int updateUsingOverlapped(HANDLE fh, DWORD period)
{
  DWORD fsLow, fsHigh;
  LONGLONG n;
  record_t r;
  OVERLAPPED ov = { 0, 0, 0, 0, NULL};

  fsLow = GetFileSize (fh, &fsHigh);

  ..............
  while (.....................) /* for each record every period in file */
  {
    ..............
    ReadFileEx (fh, &r, ........................);
    r.num++;
    WriteFileEx (fh, &r, ........................);
    ..............
  }
}
```

System and device programming
29 Jun 2011

(C program: textbooks and/or course material allowed)

(18 marks) The final mark is the sum of the 1st > 8 and the 2nd part > 10 .

Write a C program (named **`file_concatenate`**) in the Unix environment using Pthreads, which behaves as follows.

The program creates **`K`** threads, where **`K`** is given as the first argument in the command line.

A variable number of text files are stored in the directory **`DIR`** given as the second argument of the command line. The main thread creates a subdirectory **`tmp`** in **`DIR.`**

Each thread cyclically reads from the directory two files and concatenates their content **creating a new file,** in the subdirectory **`tmp`**, with a filename composed by the concatenation of the filenames of the concatenated two files (if **`f1.txt`** and **`f2.c`** are concatenated the new filename in **`tmp`** will be **`f1.txtf2.c`**.

The two original files must be **immediately removed** from the directory after they have been opened to avoid conflicts among threads trying to read the same files.

When the new file created in **`tmp`** is complete and closed, a hard link to it must be set in **`DIR`**, and the file in **`tmp`** can be removed.

Care has to be taken to avoid deadlocks (for example, when two files remain **`DIR`**, and a thread read one of the files, and another one accesses the other).

The program terminates when a single file remains, which concatenates the content of all files originally included in the directory.

Line command example.

```
> file_concatenate 3 directory_name
```