Task 02

General requirements

Please submit the assignment as a `*.zip` archive, where every problem is stored in its own subdirectory. All the code must be well-commented and must come with a Makefile which builds a program. Notes and explanations may come in a separate file in the problem's directory. For notes and explanations use text or pdf format.

1 Problem 1.1

Measure the time it takes to create a process in UNIX. Write a C program, that executes N forks, storing corresponding process ids in an array, and, after that, executes wait-calls for every child process. Each child process should call the `dummy` function shown below. Please be careful when measuring time. One of the ways to measure time accurately is by using the library `rt`. Consider the following template for your program:

```
#include <sys/time.h>
#include <time.h>

int dummy (void * arg)
{
    return 0;
}

int64_t xelapsed (struct timespec a, struct timespec b)
{
    return ((int64_t)a.tv_sec - b.tv_sec) * 1000000
           + ((int64_t)a.tv_nsec - b.tv_nsec) / 1000LL;
}

void measure_fork (unsigned N)
{
    struct timespec start, stop, finish;
```

Hand-out: 29/01/2013 Hand-in: 12/02/2013

```
    clock_gettime (CLOCK_REALTIME, &start);
    /* Make N forks, call dummy in every child.  */
    clock_gettime (CLOCK_REALTIME, &stop);
    /* Wait for the forks.  */
    clock_gettime (CLOCK_REALTIME, &finish);

    printf ("%d proc: fork=%d wait=%d sum=%d\n",
            N, xelapsed (stop, start), xelapsed (finish, stop),
            xelapsed (finish, start);
}
```

Execute for various N-s: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024.

2 Problem 1.2

Try to measure the overhead of thread creation and synchronisation. Apply the same approach as in problem 2.1, but use adequate library functions from the POSIX *pthread* library: In order to create a thread, use `pthread_create` function:

```
...
pthread_t pid;
int ret = pthread_create (&pid, NULL, dummy, NULL);
...
```

where `dummy` is the function from the previous task. An id of a thread will be stored in the `pid`. The function returns *0* in case of success.

In order to wait for the thread termination, use the `pthread_join` function:

```
...
void *result;
pthread_join (pid, &result);
...
```

where `pid` is the id of the thread you are waiting for. The result can be analysed in case a thread returns something, which is not happening in our case. When compiling a program, don't forget to add *-pthread* compilation option.

3 Problem 2

Write a sequential program in C that gets a string and a list of file names as arguments, searches for the given string in every file, and prints on stdout filename and position in the file, in case the string was found. Consider an example:

```
$ echo "a b c ddf g" > 1.txt
$ echo "ddaddg" > 2.txt
$ echo "da ddf qqq" > 3.txt
$ ./search "ddf " 1.txt 2.txt 3.txt
```

After the search is complete it should print on stdout:

```
1.txt   offset=7
3.txt   offset=4
```

A very straight forward implementation of the search function can be found in the appendix and on vision. You are more than welcome to use your own version or a faster substring algorithm like the Knuth-Morris-Pratt algorithm.

4 Problem 3.1

In order to parallelise the program from the previous task we want to scan all the files given as arguments at the same time by using either one process per file or one thread per file. Please sketch how each solution would look like in practice. Explain how to organise the communication in each case. Elaborate on which one would be faster, which would be easier to implement and why. Would it make sense to combine the approaches?

5 Problem 3.2

Implement your favourite design sketched in the previous problem.

A Search function

This is a straight forward implementation of substring search. The function looks at the file symbol-by-symbol and checks if a symbol matches the first symbol of the pattern. If so, it tries to match the rest of the pattern, and in case pattern does not match, it puts all except the first symbol it read from file back on the file stream.

```
int search (FILE *f, const char *pat)
{
    int i, j, c, pos = 1;
    const char *ptr;

    for (; EOF != (c = fgetc (f)); pos++)
      {
        if (c != *pat)
          continue;

        for (i = 1, ptr = pat + 1; *ptr != '\0'; i++, ptr++)
          if (*ptr != (c = fgetc (f)))
            {
              ungetc (c, f);
              break;
            }

        if (*ptr == '\0')
          return pos;
        else
          for (j = i - 1; j > 0; j--)
            ungetc (pat[j], f);
      }

    return -1;
}
```