CS342 Operating Systems - Spring 2016
Project 1: Processes, IPC, and Linux Modules

```
Assigned: Feb 11, 2016 Thursday
Due date: Feb 22, 2016 Monday, 23:55
```

You will do this project individually. You can do it on a virtual machine if you wish. You have to program in Linux using C programming language.

Part A: Concurrent Processes and IPC. [70 points]
Objective: Practicing process creation, process communication (by use of POSIX message queues), multi-process application development.

Develop an index generator application (program) that will use multiple processes to generate an index. The application will take a textfile as input and will generate an index for it. The textfile will be a sequence of English words (like a book). A line of the textfile contain multiple words.

The main process of your application will create n message queues and n child processes (worker processes): one message queue per worker process. A message queue for a worker process will be used to pass information from the main process (parent) to the worker (child). *Note that when a child is created, its address space (memory) is populated from parent's (hence **initial** one way information passing can be done **once** from parent to child), but since the address spaces are different, after a while later, they may contain different things in their memory and they can not access their variables. Therefore, in order to exchange information, they need to use an IPC mechanism like pipes or message queues. You will use POSIX message queues in this project [4,5,6].*

After creating the worker processes, the main process will open the input file and will read the words from it. Each word (after converting all letters to lowercase) and its line number, i..e., the number of the line where the word appears in the input file, will be sent through a message queue to a worker process using the following rule. There are 26 letters in English alphabet and there are n worker processes created (worker 1 though n). Let k = 26 div n (integer division). Worker 1 will be responsible for the words that start with the first k letters of the alphabet (a, b , c ..); worker 2 for the next k letters, and so on. Worker n will be responsible for the remaining letters and also for any word that is not starting with a letter. For example, if n = 5, then worker 1 will be responsible for words starting with {a,b,c,d,e} and worker 5 will be responsible for words starting with {u,v,w,x,y,z}. The minimum number of workers (MIN_WORKERS) is 1 and maximum (MAX_WORKERS) is 5. The maximum word length is 64 including the NULL character at the end of a word. The first line of an input file is line 1.

In this way, workers will receive their respective words. A worker will generate an index for the words that it is responsible for. An index is a sequence of entries in *sorted* order (according to words). To decide which word comes earlier, you can use the strcmp() available in string.h (#include <string.h>). An entry has a word and a sequence of line numbers (for lines where the word appears). If a word appears twice or more in a line, the respective line number is not repeated. When worker process

finishes generating the index, it will write the index into a temporary output file named "outfile-x", where x is worker number. The created file will have one word and res
pective line numbers separated by commas per line.

After all words are passed to workers, the parent will wait workers to finish their task and terminate. Each worker will generate a portion of the index. After all workers terminate, the main process will open the output files and will merge them into a single output file, containing the index information in sorted order. Then the parent will also terminate. Before terminating, the parent will delete the temporary output files and the message queues.

The program executable will be named as **indexgen** and will have the following parameters:

indexgen <n> <infile> <outfile>

Here <n> is the worker count, <infile> is input text file (ascii file) and <outfile> is output text file. An example invocation can be:

indexgen 3 in.txt out.txt

An example output can be

```
…
bilkent 98, 134, 245
school  7, 15, 256, 300
the  2, 5, 8, 14, 67, 345, 567
…
```

Part B: Kernel Module Programming. [30 points]
Objective: Learning how to develop a Linux loadable kernel module; touching to and interacting with Linux kernel; starting writing kernel code.

Develop a simple Linux kernel module and test it (insert and remove). First learn how to write a kernel module. Read the related parts from the 9th edition of our textbook [1]. There are additional references that you can benefit from [2]. The module will print out (using printk) the following information about the processes which are in waiting state: process name (command name), process id, parent process id, and state. The output will go to the kernel message buffer. You can see it using the dmesg command. Name your module as *modproclist*. Hence you module C file can be modproclist.c.

References:

[1]. *Operating System Concepts*, Silberschatz et al., 9th edition, Wiley, 2014, pages 94-98 and pages 156-158.

[2]. *The Linux Kernel Module Programming Guide*.
http://www.tldp.org/LDP/lkmpg/2.6/lkmpg.pdf. [Accessed: Feb 11, 2016].

[3]. *Linux Device Drivers*. Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. O'Reilly, 3rd Edition. https://lwn.net/Kernel/LDD3/. [Accessed: Feb 11, 2016].

[4] *Supplementary Notes*. CS342 webpage; in Lecture notes section. http://www.cs.bilkent.edu.tr/~korpe/courses/cs342spring2016/. [Accessed: Feb 11, 2106].

[5]. *Linux Message Queue Overview*. http://linux.die.net/man/7/mq_overview. [Accessed: Feb 11, 2016].

[6]. *The Linux Programming Interface*. http://man7.org/tlpi/download/TLPI-52-POSIX_Message_Queues.pdf. [Accessed: Feb 11, 2016].

Submission:

Put the following files into a project directory, tar the directory (using **tar xvf**), zip it (using **gzip**) and upload it to Moodle. You will also upload to PAGS (programming assignment grading system).

- indexgen.c: contains your C program in part A
- modproclist.c: contains the module code of part B
- Makefile: Compiles the program and the module.
- README: Your name and ID and any additional information that you want to put.

Additional Information and Clarifications:

- *Suggestion: work incrementally; step by step; implement something, test it, and when you are sure it is working move on with the next thing.*
- More **clarifications**, additional information and explanations that can be useful for you may be put to the **course website**, just near this project PDF. Check it regularly.
- Parent can create/open message queues and pass their descriptors (IDs) to the child processes automatically when creating child processes with fork() (that means fork already copies the address space of parent to the children and therefore the descriptors (IDs) are automatically passed to the children). This means, the children do not need to open the message queues again; They already have the message queue descriptors and can start accessing the queues with them.
- You can assume that the max line size (including newline character) is 4096 in an input file.
- Parent can send a special message to a child to indicate the end of messages it will send. This can be done very easily by defining a message to be a structure and having a field like msgs_done in the structure, which will indicate that no more messages will be sent.
- You can use command line utilities like ipcs, ipcrm, etc. to see the messages queues created and remove them outside of your program.
- In indexgen, considering all things done by parent and children, and index is generated. What each child essentially doing is sorting the words given to it. For each word, the child will have a list of line numbers. Sorting is done according to words, not line numbers.

- The following web page contains a project skelaton. You can clone it into your local machine, if you wish.
    - https://github.com/korpeoglu/cs342-spring2016-p1
- Correction in the Assignment: The example output should be like this: ...
  bilkent 98, 134, 245
  school 7, 15, 256, 300
  the 2, 8, 14, 67, 345, 567
- Parent can create the messages queues. Then their mq ids can be passed to the children or their mq names can be passed and children can open the message queues. It is OK if parent sends messages to a message queue before the respective child opens the queue. The messages will be buffered and if queue gets filled, the send operation in parent will block until the respective child starts retrieving messages.
- You can assume that the input file will contain words that consist of just English alphabet letters (no digits and other symbols).
- Instead of 256, please assume the maximum word length is 64 including the NULL character at the end of a string (word). This is better for you.
- Assume the input text file is ascii text file.
- You can implement and use any sorting algorithm you wish.