**Multi-processing and Multi-threading exercise**

# Multi-processing exercise

Implement a program with 2 processes, socket connection, shared memory:

## Process 1 (Parent process)

* SetupRead data from import files (around 5 files)

Import files with below structure, this is the emails content need to be collected.

|  |
| --- |
| From: sender 1  To: receiver 1  Content of email, write some thing here |

* Send Monitor message (“is\_alive”) to process 2 via socket (time out 1 second), this is a message check if process 2 still alive. Then wait for process 2 response alive message (“alive”).
* If process 2 still alive, save email data to shared memory.
* If process 2 is not alive, fork new process 2
* After process all files, send socket to Process 2 (“shutdown”), check alive of process 2, if not alive, shutdown process 1 also.

## Process 2 (Child process)

* After started, wait for (“is\_alive”) message and keep processing shared memory in case of data exist.
* After receive (“is\_alive”) message, response (“alive”) immediately.
* In case of shared memory having data, read it and log content to summary file (email\_archive.csv) with below structure.

|  |
| --- |
| sender 1;receiver 1;Content of email, write some thing here |

* If receive “shutdown” socket message, end process 2

## Sequence of program

* Process 1 read one file
* Process 1 check alive of process 2
* Process 2 response to process 1
* If process 2 is not alive, start again
* If process 2 is alive, send data to shared memory
* Process 2 detect shared memory is having data, read and process it
* Process 2 Read “email\_archive.csv”: if not exist->create one, if first line is not “From;To;Content”-> add into
* Process 2 process data from shared memory: convert to structure as example here “sender 1;receiver 1;Content of email, write some thing here” and append to “email\_archive.csv”
* Process 1 read all files, send shutdown message to Process 2 and terminate program.

## Require points

* Do not use multi-threading
* First line of email\_archive.csv is: “From;To;Content”

# Multi-threading exercise

Implement a program with 1 thread as main and many callback threads, mutex, semaphore:

## Main thread

* Manage status of 10 cencors. Main thread simulate value of these censors and send to.
* Simulate and send 10 values of random cencors (random choose 1 cencor, random choose number from 1->10). For each simulation of 10 values, sleep 1s.

Message structure: {

cencorName: string,

cencorValue: int

}

* Push each sencor name and value to message\_queue
* After simulate and send 100 values, stop program

## Callback threads

* Whenever message\_queue contains message, create thread to handle one message. 1 thread to one message.
* Maximum 3 threads at the same time.
* Callback thread pop 1 message, read data and append it to log file of that censor, log file is named as [cencor name].log. ex: cencor\_monitor1.log

There is 10 cencors: cencor\_monitor1…10.log

* If there is no log file of censor, create one
* Log value of cencor as following structure (append latest value to ending):

|  |
| --- |
| 1, 2, 4, 7, 1 |

## Require points

* Do not use multi-processing

# Coding convention

* Do not write function over 80 lines
* Do not keep line with over 80 characters.
* Indent: 4 spaces
* Variable name: camel case, ex: cencorName, inputValueName
* Function name: use verd for processing function. Ex: calculateDataValue (ok) dataValueConculation (NG), Sum (OK)
* Constant: UPPER CASE, ex: #define MAX\_THREADS 10
* Class name: camel case with upper the first character. Ex: CencorManager
* Meaning variable name: int a, b = 0 (NG).