CS606 Spring 2021
Programming Project
Due: Wednesday Feb 10 @ 3:30p

You will complete a multithreaded two process system that communicates via System V message queues. The goal is to find the longest word that begins with the supplied prefix in a series of text passages.

Search Manager Logic

Search manager reads one or more prefixes from the command line, creates and sends prefix request messages (contains prefix string and prefix ID) via System V ipc queues and waits for the *passage processor* to return a series of responses. The search manager will print the results for each prefix as once all responses for that prefix have been received. Use the passage count on the first response to determine how many responses should be received. The *search manager* will send a message to *passage processor* letting the processor know that all the requests have been sent and the passage processor can terminate once all responses have been sent. Once all the responses are received, the *search manager* should terminate. *Search manager* will print a status of the requests when a SIGINT is received.

Format:./searchmanager <secs between sending prefix requests> <prefix1> <prefix2> …

```
./searchmanager 3 con pre wor

Message(1): "con" Sent (8 bytes)

Report "con"
Passage 0 - Sense_And_Sensibility.txt - constant
Passage 1 - Mansfield_Park.txt - contemptible
Passage 2 - The_Call_Of_The_Wild.txt! - no word found
Passage 3 - Tale_Of_Two_Cities.txt -  no word found
Passage 4 - Peter_Pan.txt - conspicuous

Message(2): "pre" Sent (8 bytes)

Report "pre"
Passage 0 - Sense_And_Sensibility.txt - no word found
Passage 1 - Mansfield_Park.txt - predict
Passage 2 - The_Call_Of_The_Wild.txt! - no word found
Passage 3 - Tale_Of_Two_Cities.txt -  preserves
Passage 4 - Peter_Pan.txt - no word found

Message(3): "wor" Sent (8 bytes)

Report "wor"
Passage 0 - Sense_And_Sensibility.txt - no word found
Passage 1 - Mansfield_Park.txt - world
Passage 2 - The_Call_Of_The_Wild.txt1 - no word found
Passage 3 - Tale_Of_Two_Cities.txt -  worst
Passage 4 - Peter_Pan.txt - no word found

Message(0): "   " Sent (8 bytes)

Exiting ...
```

SIGINT captures Ctl-C to print status

```
^C con – pending
pre – pending
wor – pending
```

** status before responses

```
^C con – done
pre – 3 of 5
wor – pending
```

** status after **con** completes and during **pre** responses

Passage Processor logic

The passage processor will read a series of passage file names from passages.txt. A thread will be created for each passage that builds Trie with words in the passage text, receives requests for longest word searches, searches the trie for the longest word and asynchronously returns the longest word. The Passage Processor will read each prefix request from the System V ipc queue using a Java Native Call, sends the requests to each worker, retrieves responses from each worker and sends them back to the search manager via the system V queue. For each prefix request, the trie should be searched for the longest word that starts with that prefix. In the case that a word is found, the longest word will be sent to the SearchManager via the System V ipc queue via a Java Native call. The response will include the prefix id, the passage id, the passage name, the number of passages, present = 1, and the longest word. In the case that the prefix is not found, the response will include the prefix id and present = 0.

Format: java -cp . -Djava.library.path=. edu.cs300.TextSamples

```
anderson@cs426: java -cp . -Djava.library.path=. edu.cs300.TextSamples 2>/dev/null
Worker-0 (Sense_And_Sensibility.txt) thread started ...
Worker-1 (Mansfield_Park.txt) thread started ...
Worker-2 (The_Call_Of_The_Wild.txt) thread started ...
Worker-4 (Peter_Pan.txt) thread started ...
**prefix(1) con received
Worker-2 1:con ==> not found
Worker-3 (Tale_Of_Two_Cities.txt) thread started ...
Worker-3 1:con ==> not found
msgsnd Reply 2 of 5 on 1:con from The_Call_Of_The_Wild.txt present=0 lw=----(len=4) msglen=144
msgsnd Reply 3 of 5 on 1:con from Tale_Of_Two_Cities.txt present=0 lw=----(len=4) msglen=144
Worker-1 1:con ==> contemptible
Worker-4 1:con ==> conspicuous
msgsnd Reply 1 of 5 on 1:con from Mansfield_Park.txt present=1 lw=contemptible(len=12) msglen=144
msgsnd Reply 4 of 5 on 1:con from Peter_Pan.txt present=1 lw=conspicuous(len=11) msglen=144
Worker-0 1:con ==> constant
msgsnd Reply 0 of 5 on 1:con from Sense_And_Sensibility.txt present=1 lw=constant(len=8)
msglen=144
**prefix(2) pre received
Worker-2 2:pre ==> not found
Worker-0 2:pre ==> not found
msgsnd Reply 2 of 5 on 2:pre from The_Call_Of_The_Wild.txt present=0 lw=----(len=4) msglen=144
msgsnd Reply 0 of 5 on 2:pre from Sense_And_Sensibility.txt present=0 lw=----(len=4) msglen=144
Worker-4 2:pre ==> not found
msgsnd Reply 4 of 5 on 2:pre from Peter_Pan.txt present=0 lw=----(len=4) msglen=144
Worker-1 2:pre ==> predict
msgsnd Reply 1 of 5 on 2:pre from Mansfield_Park.txt present=1 lw=predict(len=7) msglen=144
Worker-3 2:pre ==> preserves
msgsnd Reply 3 of 5 on 2:pre from Tale_Of_Two_Cities.txt present=1 lw=preserves(len=9) msglen=144
**prefix(3) wor received
Worker-1 3:wor ==> world
Worker-3 3:wor ==> worst
Worker-0 3:wor ==> not found
Worker-2 3:wor ==> not found
msgsnd Reply 1 of 5 on 3:wor from Mansfield_Park.txt present=1 lw=world(len=5) msglen=144
Worker-4 3:wor ==> not found
msgsnd Reply 3 of 5 on 3:wor from Tale_Of_Two_Cities.txt present=1 lw=worst(len=5) msglen=144
msgsnd Reply 0 of 5 on 3:wor from Sense_And_Sensibility.txt present=0 lw=----(len=4) msglen=144
msgsnd Reply 2 of 5 on 3:wor from The_Call_Of_The_Wild.txt present=0 lw=----(len=4) msglen=144
msgsnd Reply 4 of 5 on 3:wor from Peter_Pan.txt present=0 lw=----(len=4) msglen=144
**prefix(0)     received
Terminating ...
```

Search Manager requirements

- Must be written in C
- Numeric parameter denoting delay will be present and an integer; if it is zero, then use no delay
- At least one prefix will be provided (may not be valid)
- Only process prefixes are at least 3 characters should be processed
- Only one prefix should be processed at a time. Once all the results on a prefix are returned, the next can be sent to the passage processor
- Send a prefix message with a zero id to notify the passage processor to complete

Passage processor requirements

- Written in Java with the main function in edu.cs300.PassageProcessor.java
- Read passage file names from *passages.txt* in root directory (hardcode the name)
- Read contents of each passages file in the root directory

General criteria

- Programs will only be graded on cs426.ua.edu
- Late submissions will not be accepted
- Input criteria:
    - Prefixes-at least three characters long and no longer that 20 characters
    - Text Passages-Unlimited number of passages
    - Longest word-Maximum length 100 characters
    - Prefix Request -Unlimited number of requests
    - Text Processing-Ignore punctuation, only store words. Words with punctuation in them should be ignored.
    - Passage Name should be the first 30 characters of the filename
- The System V message queue requires an existing file and integer to create a unique queue name. You should create a file using your crimson id in your home directory. Use queue_ids.h header file to create a constant string that holds the path to the queue and a constant integer to hold the day of your birthday. Use CRIMSON_ID and QUEUE_NUMBER in the ftok command to generate the identifier (see https://github.com/monicadelaine/Spring_cs300_project/blob/master/msgsnd_pr.c for an example)

```
#define CRIMSON_ID "/home/anderson/anderson"
#define QUEUE_NUMBER 12 //day of birth
```

- Place files in the directory structure below (matches sample github)

```
.
├── _passages.txt
├── _Pride_And_Prejudice.txt
├── _Mansfield_Park.txt.txt
├── _The_Call_Of_The_Wild.txt
├── _Tale_Of_Two_Cities.txt
├── _Peter_Pan.txt
├── _edu_cs300_MessageJNI.h
├── _queue_ids.h
├── _longest_word_search.h
├── _searchmanager.c
├── _system5msg.c
├── _<Additional supporting C files>.c
├── _<Additional supporting header files>.c
├── _edu
│   └── cs300
│           └── SearchRequest.java
│           └── MessageJNI.java
│           └── ParallelTextSearch.java
│           └── Worker.java
│           └── PassageProcessor.java
│           └── <Additional Supporting Java Source>.java
├── _ CtCILibrary
│   ├── Trie.java
│   └── TrieNode.java
└── _Makefile //if commands other than those provided are needed to build executables or dynamic library
└── _readme.md
```

Hints:

- Use blocking msgrcv
- Wrap search manager msgrcv in a while loop since SIGINT will cause the msgrcv to return with no message

- Create two SIGINT handlers: one that gives a starting status that does not include any shared information and a 2nd handler that returns shared (protected) information

longest_word_search.h notes

```
#define WORD_LENGTH 100+1
#define PASSAGE_NAME_LENGTH 30+1

// Declare the message structure

*** message struct for sending in msgsnd_pr.c and receive in
system5msg(Java_edu_cs300_MessageJNI_readPrefixRequestMsg)
*** type should be set 1 for sending and receiving
*** total 4 plus length of prefix plus 1
typedef struct prefixbuf {
  //set to 1 and corresponds to type in parm 4-msgrcv(msqid, &rbuf, WORD_LENGTH, 1, 0)
  // in system5msg- Java_edu_cs300_MessageJNI_readPrefixRequestMsg
  long mtype; //not in byte count
  int id; //4 bytes
  char prefix[WORD_LENGTH]; //4 bytes in the case of a three letter prefix and a null

} prefix_buf;

*** message struct for sending in msgsnd_pr.c and receive in
system5msg(Java_edu_cs300_MessageJNI_writeLongestWordResponseMsg)
*** type should be set 2 for sending and receiving
***total 144 bytes

typedef struct foundbuf {
        //set to 2 and corresponds to type in parm 4-msgrcv(msqid, &rbuf, WORD_LENGTH, 2, 0)
        // in msgrcv_lwr.c
        long mtype; //not in byte count
        int index; //index of response-4 bytes
        int count; //total excerpts available-4 bytes
        int present; //0 if not found; 1 if found-4 bytes
        char location_description[PASSAGE_NAME_LENGTH]; //31 bytes
        char longest_word[WORD_LENGTH]; //101 bytes
} response_buf;
```

Parameters

- Minimize resource usage (do not hardcode any values other than given above)
- Do not assume any ordering of the message retrieval
- Maximize parallel processing
- Appropriately protect data structures as needed
- Minimize use of global variables (don't use as a mechanism to avoid passing parameters)
- ~~Synchronize calls to msgsnd and msgrcv~~
- Free any allocated memory; join any threads

- Do not remove IPC queue when done
- Message queue key should be your crimson id
- Programs should be coded in C language (C99 standard) and will be compiled and tested on cs426.ua.edu. If you choose to program on another system, give yourself enough time verify it works on cs426. No other system will be used to test your code. May need _GNU_SOURCE switch.
- You should use the pthreads library for threading. You can use mutexes or condition variables from the pthreads library and/or semaphores from the posix library.
- Appropriate data structures should be selected based on your knowledge of data structures (CS201, etc).
- Algorithms should be efficient and appropriate. This program should demonstrate not only your understanding of process synchronization but your ability to design a program appropriately
- No sleeps other than sleep between sending prefix requests driven by command line argument
- Use #ifdef DEBUG to remove/add debug print statements based on compilation (-DDEBUG=0 or -DDEBUG=1)
- Use standard error to print error messages
- Use assert to check for unexpected conditions

Grading policy
Failure to follow directions will result in point deductions.

Late assignments will not be accepted. The source code and test results should be printed and brought to class on Feb. 10th. Make sure your printout is easy to read (line wrapping etc). The source code should also be turned in via Blackboard (not emailed to me or the TA). Test results (using your generated data) should also be printed and submitted via blackboard in pdf format. Test result submissions of any other type will not be graded.

This is an individual assignment. The program must represent your own work. You can discuss high-level concepts. Do not show your code to anyone. I reserve the right to ask you about your program to discern if you indeed wrote the code. If you cannot explain your code and choices verbally, you may be turned in for academic misconduct. All submissions will be analyzed to identify possible cases of cheating. Any cases of suspected collaboration will be referred to the College of Engineering Dean. A zero or low grade is always better than having an academic misconduct on your academic record.

** Programs will be evaluated based on many functional and design criteria **
Sample criteria include:

70% - functionality
- Program contains the correct trie code to produce the output (find longest words in variety of passages)
- Code for search manager contains correct functionality
- Code for passage processor contains correct functionality
- Hardcoding and lengths as specified
- Signal catch implemented and working
- Process sync correct (threads in PP and signals in SM)
- Maximizes concurrency
- Other functional or correctness features

25% - design
- Program exhibits defensible design choices in algorithms and data structures (if you add any)

- Program does not contain extra loops or any code that hurts efficiency
- Other design and efficiency features

5% - style

- Program must use appropriate and consistent style for naming of elements
- Program must include reasonable whitespace and appropriate indentation
- Program must include comments, especially in areas where you need to support your choices or where the purpose of the code is unclear.

** Clarifications on the assignment will be posted to blackboard.