CS300 Fall 2021
Report generator - Programming Project
Due Date: Monday Nov 8 @ 12pm

Project files: https://github.com/monicadelaine/f21_os_project

Overview
This project consists of two programs: **ReportingSystem** and **process_records**. **ReportingSystem** is a threaded java program that will read a report specification, send a search string to **process_records**, and use the received data records to write a formatted report to a file. **process_records** is a C program that reads records from stdin and sends them to the appropriate thread in the **ReportingSystem** based on received search string requests (one from each **ReportingSystem** thread). Each request includes the search string to use to identify records for this report and the queue number from which this report's **ReportingSystem** thread will read records. The C process will evaluate each record that is read from standard in to determine if the record matches any search strings. When a search string is found, the record is sent to the correct java thread by using the provided queue number.

ReportSystem.java is responsible for printing formatted reports to files.

1. **ReportSystem** opens and reads `report_list.txt` which contains the number of reports on line 1 and filenames (one per line until end of file) that contain the report specifications. The number on line 1 will agree with number of file names on subsequent lines (you do not need to check for this).
2. Each report specification file contains the report title (line 1), search string (line 2), output file name (line 3) and the column names and fields (one per line starting at line 4 until a blank line or end of file). Each report specification file in the `report_list.txt` will exist.
3. A separate thread will be created for each report. Each report will be assigned an id based on its index in the `report_list.txt` file (line 2 has index=1, line 3 has index=2, etc). This thread will parse the specification file, send the record request message, receive the matched records and format/write the report file to the output file name.
4. A report request should be sent to the process_records program via the System V IPC queue in the form of a record_request_buf message. A native C method is needed to access the System V IPC queue from java. Use the MessageJNI.java static native method writeRecordRequest(**int** reportIdx, **int** reportCount, String searchString) toz create and send a message that will be read by process_records. Every thread will send the record request to a single queue – ftok(FILE_IN_HOME_DIR,0xff)
5. Each report thread should use a unique System V IPC queue for receiving the records indicated by ftok(FILE_IN_HOME_DIR,index) where index is the number assigned based on order in the `report_list.txt`
6. When each thread receives a zero-length record, it should complete the writing of the formatted report to the output file and terminate.

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

```
anderson@cs-operatingsystems01.ua.edu: java -cp . -Djava.library.path=. edu.cs300.ReportingSystem
```

ReportingSystem programming requirements

- Written in Java with the main function in ReportingSystem.java

- Must send and receive data via System V message queues using the predefined message formats
- Use no command line parameters (report_list.txt is hardcoded and records will be read from stdin)
- Do not prompt for anything. Follow the assignment instructions
- Use no path (will run, read and write files in the project root directory)
- Lengths of fields defined in header file
- Check to make sure all files exist. If not, exit gracefully
- Do not modify existing prints to stderr
- Any diagnostic messages must go to stderr, not stdout
- Read report files from `report_list.txt` in java root directory (hardcode the name- "`report_list.txt`" with no path)
- Read contents of report specification file based on format. Assume it is correctly formatted.
- Column lines are formatted using delimiters: comma "," delimiting column positions from the column heading. No quotes. Assume column heading goes until the end of the ascii string without any trailing spaces. A dash "-" delimits beginning column position from ending column position
  Example: `12-24,Project formal name`
  - Beginning column is 12
  - End column is 24 inclusive
  - Column heading is "Project formal name"
- Report output files: separate column headings and report fields by a "\t". End each line with "\n". Do not use any additional formatting (changing case, changing spacing or justification)
  - Line 1: title
  - Line 2: column headings
  - Line 3 through end of file: one line for each received record
- Add appropriate synchronization to create a coherent report concurrently

process_records.c

1. The process_records process accepts all of the record_request messages via the System V IPC queue- ftok(FILE_IN_HOME_DIR,0xff). Each records_request contains the number of report requests to expect. This value can be used to set up the structures to hold report and record counts for the status report.
2. Start a thread that can print the status report when requested. Use synchronization primitives to safely print shared data. This thread starts after the report requests have been processed to avoid a segmentation fault when trying to read an uninitialized report record count structure.
3. The process_records program reads ASCII encoded record files (up to RECORD_MAX_LENGTH characters separated by "\n") from stdin. Each record will be tested against each received search string. It can match 0, 1 or many reports. Search strings can appear anywhere in the record to be a match. Maximum size of the search string is defined in the header file. After 10 records read from stdin, sleep 5 seconds.
4. Matching records will be sent to the **ReportGenerator** thread associated with the request via the System V queues. The queue will be determined by ftok(FILE_IN_HOME_DIR,index) where index is the order in the report_list.txt and the value in the index field on the incoming records_request message.
5. A signal handler will be added that will trigger a print the status of the records processing on SIGINT. It will print the number of report requests. It will also provide counts for the number of

records read and sent the ReportingSystem for each request. Hook signals after all report requests have been received.

```
***Report***
9 records read for 2 reports
Records sent for report index 0: 2
Records sent for report index 1: 2
```

6. When all records from the stdin have been processed, send a message with a zero length record to trigger final processing by the ReportingSystem threads. It will print the records report (same as the report generated by the signal (detailed in step 5).

Format:

```
./process_records
anderson@cs-operatingsystems01.ua.edu: ./process_records <datafile
```

process_records programming requirements

- Must be written in C. Source file containing main function is process_records.c. Any additional source files must be compiled and linked in the makefile
- Must send and receive data via System V message queues using the predefined message formats
- Indicate that all records have been processed by sending a zero length record to the ReportingSystem via each message queue. Program should complete after sending this message
- Do not put any blocking synchronization primitives in the signal handler.

Sample data files:

report_list.txt

```
2
report1.txt
report2.txt
```

Existing report1.txt

```
Customer report: H Peck
H Peck
customer-peck.rpt
1-8,Tran Date
11-21,Transaction
32-39,Customer
64-69,$Labor
52-56,$Part
71-75,Total
```

Existing report2.txt

```
Sales
Purchas
sales.rpt
1-8,Tran Date
11-21,Transaction
52-56,$Part
```

Input piped to stdin

```
000000000111111111122222222223333333333444444444455555555556666666666777777778
```

Last updated 10/21/21

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
09/22/21  Fix Flat     Repair  H Walter    SSmith  $0    0.5   $20    $20
09/22/21  Windshield   Repair  H Peck      JJDoe   $400  1.5   $60    $460
09/22/21  Oil Change   Repair  H Lovell    SSmith  $45   2     $80    $125
09/22/21  Wipers       Purchas H Peck      NA      $36   0     $0     $36
09/22/21  Rotate Tire  Maint   J Peck      SSmith  $0    1     $40    $40
09/22/21  End of Day Sales $496  Labor Hours 8        Labor $340
09/23/21  Air Freshnr  Purchas J Thorn     SSmith  $12   0     $0     $12
09/23/21  Rotate Tire  Maint   J Thorn     RRogers $0    1     $40    $40
09/23/21  End of Day Sales $822  Labor Hours 7.5      Labor $320
```

customer-peck.rpt

```
Customer report: H Peck
Tran Date   Transaction Customer    $Labor      $Part Total
09/22/21    Windshield  H Peck      $60         $400  $460
09/22/21    Wipers      H Peck      $0          $36   $36
```

sales.rpt

```
Sales
Tran Date   Transaction $Part
09/22/21    Wipers      $36
09/23/21    Air Freshnr $12
```

Additional project code specifications

- Any debug messages should print to stderr or be controlled by a DEBUG flag that is turned off by default
- You must use the header file and the predefined messages
- You must place and implement functionality as described in the description. Programs that do not follow the guidelines will receive a zero.
- 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 store a constant string that holds the path and name of that existing unique file and a constant integer 0xff for the shared queue for the record_request messages. Use FILE_IN_HOME_DIR and QUEUE_NUMBER of 0xff for the send of report requests and the report index for sending the report records in the ftok command to generate the identifier. ***The file in FILE_IN_HOME_DIR must exist for IPC queues to work***

```
#define FILE_IN_HOME_DIR "/home/anderson/anderson"
#define QUEUE_NUMBER 0xff
```

```
anderson@cs-operatingsystems01:~> ipcs

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x100329dd 688128     7335       666        0            0
0x05032d6d 67567617   21894      666        0            0
0xffffffff 33259522   63467      666        5217         58
0x180337a0 524291     99329      666        1352         13
0x02037d3e 52527108   80349      666        0            0
0x0103a08d 67600389   40016      666        0            0
0x17038545 53870598   11159      666        0            0
0xff03076e 67829767   anderson   666        0            0
0x0203076e 67764232   anderson   666        0            0
0x0303971b 65568777   100313     666        0            0
0x0103076e 67797002   anderson   666        0            0
0x010316fa 67862539   jmholmquis 666        160          2
0xff0316fa 67895308   jmholmquis 666        0            0
0x01038d2b 67993613   jclacount  666        0            0
0xff038d2b 68026382   jclacount  666        0            0

------ Shared Memory Segments --------
```

Error msg:
`Key cannot be 0xffffffff..fix queue_ids.h to link to existing file`
You must:
- set up the queue_ids.h correctly
- make clean
- make

- Place files in the directory structure below (matches sample github). Turn in a zip or tar file named `files.tar, files.tar.gz` or `files.zip` that contains only the edu/cs300 directory. All other files are

in the root. The project will be tested via a script. Not following this format breaks the script and will cause your project test to fail.

```
.
├── _report_list.txt
├── _report1.txt
├── _report2.txt
├── _edu_cs300_MessageJNI.h
├── _queue_ids.h
├── _report_record_formats.h
├── _gprocess_records.c
├── _system5_msg.c
├── _msgrcv_record_request.c
├── _msgsnd_report_record.c
├── _<Additional supporting C files>.c
├── _<Additional supporting header files>.h
├── _edu
│   └── cs300
│           └── ReportingSystem.java
│           └── MessageJNI.java
│           └── DebugLog.java
│           └── <Additional Supporting Java Source>.java
└── _makefile //update make file if needed with extra *.c or *.h in root or *java in edu/cs300
└── _README.md
```

report_record_formats.h notes

#define SEARCH_STRING_MAX_LENGTH 10
#define RECORD_MAX_LENGTH 80
#define SEARCH_STRING_FIELD_LENGTH SEARCH_STRING_MAX_LENGTH+1
#define RECORD_FIELD_LENGTH RECORD_MAX_LENGTH+1

// Declare the message structures

- Message struct for sending in `system5_msg.c:85` called from java edu/cs300/MessageJNI.writeReportRequest() and receiving in `msgrcv_report_request.c:33`
- Type should be set 1 for sending at `system5_msg.c:77`
- Receive message of type 1 using msgrcv(msqid, &rbuf, length, 1, 0) in `msgrcv_report_request.c:33`
- Record length: determined by length of search_string (see calculation on `system5_msg.c:74`)

```
//Report scan request
typedef struct reportrequestbuf {
   long mtype;
   int report_idx;
   int report_count;
   char search_string[SEARCH_STRING_FIELD_LENGTH];
} report_request_buf;
```

- Message struct for sending in `msgsnd_report_record.c:69` and received in `system5msg.c:125` called from called from `java_edu_cs300_MessageJNI.readReportRecord()`
- Type should be set 2 for sending `msgsnd_report_record.c:64`
- Receive message of type 2 using msgrcv(msqid, &rbuf, length, 2, 0) in `system5msg.c:125`
- Record length determined by length of record string (see calculation on `msgsnd_report_record.c:66`)

```
typedef struct reportrecordbuf {
     long mtype;
       char record[RECORD_FIELD_LENGTH];
} report_record_buf;
```

JNI Functions

- Defined in system5_msg.c
- Accessed via Java calls in MessageJNI.java
- Examples for using edu/cs300/MessageJNI.java:22 and edu/cs300/MessageJNI.java:23
- System generated header file for system5_msg is autogenerated from MessageJNI.java

Last updated 10/21/21

Other criteria

- Use constants in header for string handling lengths for search_string and data_records
- Minimize resource usage (do not hardcode any other values)
- Do not assume any ordering of the message retrieval
- Make no assumptions about format or names other than those provided. Examples do not exhaustively identify all combinations of acceptable input
- Maximize parallel processing
- Appropriately protect data structures as needed
- Minimize use of global variables (don't use as a mechanism to avoid passing parameters)
- Free any allocated memory; join any pthreads
- Do not remove IPC queue when done
- Message queue key should be your crimson id and use the macro defined in header file
- Programs should be coded in C language (C99 standard) and will be compiled and tested on cs-operatingsystems01.ua.edu. If you choose to program on another system, give yourself enough time verify it works on cs-operatingsystems01.ua.edu. No other system will be used to test your code. May need _GNU_SOURCE switch.
- You should use the pthreads library for threading. You should 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 required in process_records after record 10.
- 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. There are 60 students in this class. It is unreasonable to expect that any exceptions to the procedure will be made.

Late assignments will not be accepted unless you have a doctor's note covering the entire period from Oct 11-Nov 8. The source code and test results should be printed and brought to class on Nov 8. 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 code to process and save correctly formatted report files to file in a safe manner
- Code for process_records contains correct functionality
- Code for ReportingSystem contains correct functionality
- Hardcoding and lengths as specified
- Signal catch implemented and working
- Process sync correct (threads in ReportingSystem and signals in process_records)
- 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.

Last updated 10/21/21