Министерство образования Республики Беларусь Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет: компьютерных систем и сетей

Кафедра: ЭВМ

Дисциплина: Операционные системы и системное программирование

Отчёт к лабораторной работе №5 на тему «Потоки исполнения, взаимодействие и синхронизация».

Выполнил студент гр.230501 Кочеров Р.С.

Проверил старший преподаватель кафедры ЭВМ Поденок Л.П.

Минск 2024

1 УСЛОВИЕ ЛАБОРАТОРНОЙ РАБОТЫ.

Задача производители-потребители для потоков. Аналогична лабораторной No 4, но только с потоками в рамках одного процесса.

Дополнительно обрабатывается еще две клавиши – увеличение и уменьшение размера очереди.

2 ОПИСАНИЕ АЛГОРИТМОВ И РЕШЕНИЙ.

Программа реализует многопоточное приложение на языке C для симуляции работы очереди сообщений, используя производителей и потребителей. Основные компоненты включают генерацию сообщений, добавление и извлечение сообщений из очереди, управление потоком сообщений и обработку сигналов.

Основные алгоритмы:

1) Генерация сообщений:

- generateRandomSize() возвращает случайный размер сообщения (от 1 до 256 байт).
- generateMessageData(size) создает и возвращает указатель на массив данных сообщения заданного размера, заполняя его случайными символами.
- generateNewMessage() создает новое сообщение с случайными данными, вычисляет его хэш и возвращает указатель на структуру сообщения.

2) Производители:

- producerTask() выполняет в цикле создание нового сообщения, ожидание доступного места в очереди, добавление сообщения в очередь, сигнализация потребителям и вывод информации о добавленном сообщении.
- createProducerThread(prodCount) создает новый поток производителя.

3) Потребители:

- consumerTask() выполняет в цикле ожидание доступности сообщений в очереди, извлечение сообщения из очереди, обработка сообщения и вывод информации о извлеченном сообщении.
- createConsumerThread(consCount) создает новый поток потребителя.

4) Управление очередью:

- initializeQueue() инициализирует структуру очереди, включая буфер, счетчики, мьютекс и условия.
- pushMessage(msg) добавляет сообщение в стек удаленных сообщений.
- popMessage() извлекает сообщение из стека удаленных сообщений.

5) Обработка сигналов:

- handleSIGUSR1(signum) устанавливает флаг для обработки сигнала завершения.
- setSignalHandlerSIGTERM() устанавливает обработчик сигнала SIGUSR1.

6) Изменение размера очереди:

- increaseQueueSize() увеличивает размер очереди, перемещая сообщения и перезагружая буфер.
- decreaseQueueSize() уменьшает размер очереди, перемещая сообщения и перезагружая буфер.

7) Завершение потоков:

- stopThread(childIdMas, idThread) завершает указанный поток и ждет его завершения.

3 ФУНКЦИОНАЛЬНАЯ СТРУКТУРА ПРОЕКТА.

Основные структурные компоненты:

1) Структуры данных:

- struct Message представляет сообщение с полями: тип, хэш, размер и данные.
- MessageQueue представляет очередь сообщений с буфером, головным и хвостовым указателями, счетчиками и синхронизационными объектами (мьютекс и условия).
- Stack представляет стек удаленных сообщений.

2) Глобальные переменные:

- pthread_t prodIdMas[] и pthread_t consIdMas[] массивы идентификаторов потоков производителей и потребителей.
- volatile sig_atomic_t sigTermSignalFlag флаг для обработки сигнала завершения.
- int sizeQueue размер очереди сообщений.
- Stack stack стек удаленных сообщений.
- MessageQueue queue очередь сообщений.

Основные функции и их роли:

1) Функции генерации и отображения сообщений:

- unsigned char generateRandomSize() генерирует случайный размер сообщения.
- char* generateMessageData(short int size) генерирует данные сообщения.
- void displayData(struct Message msg) отображает данные сообщения.
- struct Message* generateNewMessage() создает новое сообщение.

2) Функции обработки сигналов:

- void handleSIGUSR1(int signum) обработчик сигнала SIGUSR1.
- void setSignalHandlerSIGTERM() устанавливает обработчик сигнала SIGUSR1.

3) Функции очереди и стека:

- void cleanupHandler() освобождает мьютекс очереди.
- void initializeQueue() инициализирует очередь.
- int checkIfEmpty() проверяет, пуст ли стек.
- int checkIfFull() проверяет, полон ли стек.
- void pushMessage(struct Message* msg) добавляет сообщение в стек.
- struct Message* popMessage() извлекает сообщение из стека.

4) Функции производителей и потребителей:

- void* producerTask() задача потока производителя.
- void createProducerThread(int prodCount) создает поток производителя.
- void* consumerTask() задача потока потребителя.
- void createConsumerThread(int consCount) создает поток потребителя.

5) Функции изменения размера очереди:

- void increaseQueueSize() увеличивает размер очереди.
- void decreaseQueueSize() уменьшает размер очереди.

6) Функции завершения потоков:

- bool stopThread(pthread_t* childIdMas, int idThread) завершает указанный поток.

Главная функция int main(): инициализирует очередь, создает и управляет потоками производителей и потребителей, обрабатывает пользовательские команды для создания/удаления потоков и изменения размера очереди, и завершает выполнение программы.

4 ПОРЯДОК СБОРКИ И ИСПОЛЬЗОВАНИЯ.

Для компиляции и сборки проекта используется makefile.

Порядок сборки:

1) Определение переменных:
   rwildcard - рекурсивно находит все файлы по заданному шаблону
   eq - определяет, равны ли два значения
   TARGET - целевой исполняемый файл
   BUILD_DIR - каталог для сборки
   DEBUG_SUFFIX - суффикс для отладочной сборки
   CFLAGS - флаги компиляции
   CC - компилятор с флагами
   VALGRIND и GDB - инструменты для отладки
   GDB_COMMANDS - файл с командами для GDB

2) Определение компиляторов и флагов компиляции:
   CFLAGS содержит флаги компиляции, в том числе для отладочной
сборки
   CC определяет компилятор с флагами

3) Определение зависимостей:
   SOURCES - список исходных файлов
   OBJECTS - список объектных файлов
   MAINS - список главных файлов
   EXECUTABLES - список исполняемых файлов

4) Определение целей:
   all - цель для сборки всех исполняемых файлов
   run - цель для запуска целевого исполняемого файла
   vrun - цель для запуска целевого исполняемого файла под Valgrind
   gdb - цель для запуска целевого исполняемого файла под GDB
   app - цель для сборки всех исполняемых файлов
   clean - цель для очистки директории сборки

Порядок использования.

1) Компиляция:

Для релизной сборки: make

2) Очистка:

make clean - удаляет все объектные файлы и исполняемый файл.

3) Запуск:

После успешной компиляции запустите исполняемый файл: ./build/main.

5 МЕТОД ТЕСТИРОВАНИЯ И РЕЗУЛЬТАТ ТЕСТИРОВАНИЯ.

```
     rlinux@fedora:~/Kocherov/ОСиСП/lab05$ build/main
'p+' - create producer
'p-' - destroy producer
'c+' - create consumer
'c-' - destroy consumer
's+' - increase queue
's-' - decrease queue
'i' - info
'q' - end
p+
'p+' - create producer
'p-' - destroy producer
'c+' - create consumer
'c-' - destroy consumer
's+' - increase queue
's-' - decrease queue
'i' - info
'q' - end

MSG ADD: 18448 // 221
T,:fQvkp&oJw>bhsK"}%?^4dSv]$>_[qhsY;iD+o5RfQ5PD]ODak$sRVk/Z)m5zW*Soswxb,LJ
Z^x!=ID|7Eqfx^rTfbg`9n4)c+!E6Lmn+gnG02EFV8,Ps!%Z`k:vYM|?U|aiIPYS7IwGX?
j0Tt]Hr_"TJ;J
&fIB=E%&mS^Bi)<2^X|m,rKQf,R:VkcYQ,zkQ!q>SQ`=XyN8QL&\>N.&Y_=/J!ez,af
COUNTER ADD: 1
p+
'p+' - create producer
'p-' - destroy producer
'c+' - create consumer
'c-' - destroy consumer
's+' - increase queue
's-' - decrease queue
'i' - info
'q' - end
```

```
MSG ADD: 2962 // 33
eSL^&ve1/28,woYz,t@wCc-Mmjy$lWZQ,&10
COUNTER ADD: 2
p+
'p+' - create producer
'p-' - destroy producer
'c+' - create consumer
'c-' - destroy consumer
's+' - increase queue
's-' - decrease queue
'i' - info
'q' - end

MSG ADD: 16612 // 201
'T-Gp3J]nnovx8"`J!!1[f<\PZ-`RC6Xw@!gSJDA8680K7ot5p%nV@L(xWeLwx&n;
$WkK{,`3Ap]VaQjQSX*s%/kYr9Rl<C'?
yogv{I)<;cqz4^Mg6TZ:aGrU_GCxiH7b9}Y4HaPaEC]X!*A7\zP?CEs"i6yT
]28t/p*URY8vzrPzyo1Wiau-&j,n!&B[7YQDJ[x{42q0$A+
COUNTER ADD: 3

MSG ADD: 8375 // 97
<O4}h:>Y?oo$MQ.pg},tE+G{h._x$rv=A*=)AZa_IQ`u$mghkp_0{&+c4g]5\SQz\k&|
Ed[m7=d9+L#s>_#9e-yxtX/P-]LhKOdo
COUNTER ADD: 4

MSG ADD: 8529 // 105
*V%sEgYbg`y/)5rLa3k;@mONck:E0%@8[E+},c`qDY"LnquO'biFO:q3&-
W30tJj9TgC:I6[%5)p)}A/`*R/CEAFPuY_j#I%W2En[XK]mQOs
COUNTER ADD: 5

MSG ADD: 17720 // 219
6^Wk<mQmGPB\uvy_<Wd",MLQ+Q7qk>G!z{k8i>
%2lFkc=dBX<(WGR&wZT0MAKqAEm-]XJ_g8(SxD9<{TBTzqWqN.!zLLlkn[wM3A/yW4LQWekR;-
&6}])Kh*F4S4!Cnwp#;|yo0HBf--8G9=\7wcaal)sA<sc
*kS*&O$t^I8DUCY{YtXom;QO)WDGq9*z$[%),(|hP5,'Ue"0[W"HqPvw):@wQHt
COUNTER ADD: 6

MSG ADD: 12221 // 151
#G-etEa%bQhw{:@E6${F`Bk'`qcu',n'RxkF?MH${2xxK:@a>;'{\
o$=be2fp"mBxXf9'0:$@4|jN<KiVOf2@hL#O]hA]U`W0Ho6V*8u=6_hO+Q%Y86w"byP?
an{7NTFtDYLMpCg&"PR,#Vc8jZ9LSfj
COUNTER ADD: 7

MSG ADD: 20039 // 242
|Zc8ROZ>:sKaeEAQpKb;Hv$f;)+BcyaaVGv*uPG1Epp-53[([>@%4AjLJrm-
nOkFucMl5q{Wckcx}?}ZZ?\m_F;):)5*W}MNby:tk5NP"1H|
MGV)e4uEY1Mq9`xo_H=BAT6,ic\hr$gAK?H2Q?T*O$zecsVB;
```

```
pcYGye2\AyND`mn"7!PUTW%URi:G@Za2=<X6#grCc@eE/UDESqx*K|^|evC'P&7l?l
%@Vt`96G[Ez"gOpaV=]79
COUNTER ADD: 8

MSG ADD: 1813 // 23
?*<V<H/,_6IyQ1Yr"kG2[Q:A
COUNTER ADD: 9

MSG ADD: 6415 // 87
^dxX&)@J!|5i@A0Z`S/\d4>!;GLq4<>q}8K%>iO>ea((%5_dgk@M!^K;
%t.61I*0`R3!;_<#BC(FWf+>SJiQ(6i*
COUNTER ADD: 10
p-
'p+' - create producer
'p-' - destroy producer
'c+' - create consumer
'c-' - destroy consumer
's+' - increase queue
's-' - decrease queue
'i' - info
'q' - end
p-
'p+' - create producer
'p-' - destroy producer
'c+' - create consumer
'c-' - destroy consumer
's+' - increase queue
's-' - decrease queue
'i' - info
'q' - end
p-
'p+' - create producer
'p-' - destroy producer
'c+' - create consumer
'c-' - destroy consumer
's+' - increase queue
's-' - decrease queue
'i' - info
'q' - end
s+
'p+' - create producer
'p-' - destroy producer
'c+' - create consumer
'c-' - destroy consumer
's+' - increase queue
's-' - decrease queue
'i' - info
```

```
'q' - end
s-
'p+' - create producer
'p-' - destroy producer
'c+' - create consumer
'c-' - destroy consumer
's+' - increase queue
's-' - decrease queue
'i' - info
'q' - end
s-
Element added to stack
'p+' - create producer
'p-' - destroy producer
'c+' - create consumer
'c-' - destroy consumer
's+' - increase queue
's-' - decrease queue
'i' - info
'q' - end
+
'p+' - create producer
'p-' - destroy producer
'c+' - create consumer
'c-' - destroy consumer
's+' - increase queue
's-' - decrease queue
'i' - info
'q' - end
s+
Element removed from stack
'p+' - create producer
'p-' - destroy producer
'c+' - create consumer
'c-' - destroy consumer
's+' - increase queue
's-' - decrease queue
'i' - info
'q' - end
c+
'p+' - create producer
'p-' - destroy producer
'c+' - create consumer
'c-' - destroy consumer
's+' - increase queue
's-' - decrease queue
'i' - info
```

```
'q' - end

MSG READ: 18448 // 221
T,:fQvkp&oJw>bhsK"}%?^4dSv]$>_[qhsY;iD+o5RfQ5PD]ODak$sRVk/Z)m5zW*Soswxb,LJ
Z^x!=ID|7Eqfx^rTfbg`9n4)c+!E6Lmn+gnG02EFV8,Ps!%Z`k:vYM|?U|aiIPYS7IwGX?
j0Tt]Hr_"TJ;J
&fIB=E%&mS^Bi)<2^X|m,rKQf,R:VkcYQ,zkQ!q>SQ`=XyN8QL&\>N.&Y_=/J!ez,af

COUNTER GET: 1
c+
'p+' - create producer
'p-' - destroy producer
'c+' - create consumer
'c-' - destroy consumer
's+' - increase queue
's-' - decrease queue
'i' - info
'q' - end

MSG READ: 2962 // 33
eSL^&ve1/28,woYz,t@wCc-Mmjy$lWZQ,&10

COUNTER GET: 2

MSG READ: 16612 // 201
'T-Gp3J]nnovx8"`J!!1[f<\PZ-`RC6Xw@!gSJDA8680K7ot5p%nV@L(xWeLwx&n;
$WkK{,`3Ap]VaQjQSX*s%/kYr9Rl<C'?
yogv{I)<;cqz4^Mg6TZ:aGrU_GCxiH7b9}Y4HaPaEC]X!*A7\zP?CEs"i6yT
]28t/p*URY8vzrPzyo1Wiau-&j,n!&B[7YQDJ[x{42q0$A+

COUNTER GET: 3

MSG READ: 8375 // 97
<O4}h:>Y?oo$MQ.pg},tE+G{h._x$rv=A*=)AZa_IQ`u$mghkp_0{&+c4g]5\SQz\k&|
Ed[m7=d9+L#s>_#9e-yxtX/P-]LhKOdo

COUNTER GET: 4
c-
'p+' - create producer
'p-' - destroy producer
'c+' - create consumer
'c-' - destroy consumer
's+' - increase queue
's-' - decrease queue
'i' - info
'q' - end
c-
```

```
'p+' - create producer
'p-' - destroy producer
'c+' - create consumer
'c-' - destroy consumer
's+' - increase queue
's-' - decrease queue
'i' - info
'q' - end
q
rlinux@fedora:~/Kocherov/ОСиСП/lab05$
```