Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работ №5 по курсу**

**«Операционные системы»**

**Работа с библиотеками в ОС**

Студент: Ефимов Александр Владимирович

Группа: М80 – 201Б-18

Вариант: 6

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2019

1. **Постановка задачи**

Требуется создать динамическую библиотеку, которая реализует определенный функционал. Далее использовать данную библиотеку 2-мя способами:

1. Во время компиляции (на этапе «линковки»/linking)

2. Во время исполнения программы, подгрузив библиотеку в память с помощью системных вызовов

В конечном итоге, программа должна состоять из следующих частей:

• Динамическая библиотека, реализующая заданных вариантом интерфейс;

• Тестовая программа, которая используют библиотеку, используя знания полученные на этапе компиляции;

• Тестовая программа, которая использует библиотеку, используя только местоположение динамической библиотеки и ее интерфейс.

Провести анализ между обоими типами использования библиотеки.

**Структура данных:** Очередь

**Тип данных:** MD5-суммы

1. **Описание программы**

Интерфейс очереди описан в **MD5Queue.h**, сами функции очереди описаны в **MD5Queue.c** (добавления и удаления из очереди, первый элемент, размер очереди, проверка на пустоту).

В исходном коде **queue.c** предоставляется пример программы без использования библиотеки динамической линковки. Его можно запустить после компиляции с объектным кодом очереди или после линковки при помощи флага **-L** (при этом необходимо добавить путь к динамической библиотеке используя команду в файле **set\_path.sh**).

В исходном коде **queue\_dyn.c** предоставляется пример программы с использованием библиотеки динамической линковки **dlfcn.h**. Указатели на функции загружаются функциями **dlsym**. При этом требуется использование дополнительного флага -**ldl** для **dlfcn.h**.

1. **Набор тестов**

*test.txt*

|  |
| --- |
| 3c738f8235cffa5b 4d24f14e1b9ab5cd  6ebfd5287bef41dd 7d0bf229a3054a5e  39d967bf8fa440b7 9071b59dacb38535 |

Данный тест является одинаковым для обоих программ, так как после загрузки библиотеки функции вызываются в одинаковом порядке.

1. **Результаты выполнения тестов.**

|  |
| --- |
| rookstar@Refrigerator:~/Git/os-labs/os\_lab\_05/src$ ./dynamic\_queue  1: 3c738f8235cffa5b 4d24f14e1b9ab5cd  2: 6ebfd5287bef41dd 7d0bf229a3054a5e  3: 39d967bf8fa440b7 9071b59dacb38535  3c738f8235cffa5b4d24f14e1b9ab5cd  6ebfd5287bef41dd7d0bf229a3054a5e  39d967bf8fa440b79071b59dacb38535  rookstar@Refrigerator:~/Git/os-labs/os\_lab\_05/src$ ldd dynamic\_queue  linux-vdso.so.1 (0x00007ffc03929000)  libMD5Queue.so => ./lib/libMD5Queue.so (0x00007fd51481a000)  libc.so.6 => /lib/x86\_64-linux-gnu/libc.so.6 (0x00007fd514429000)  /lib64/ld-linux-x86-64.so.2 (0x00007fd514c1e000)  rookstar@Refrigerator:~/Git/os-labs/os\_lab\_05/src$ ./dynamic\_queue\_sys  1: 3c738f8235cffa5b 4d24f14e1b9ab5cd  2: 6ebfd5287bef41dd 7d0bf229a3054a5e  3: 39d967bf8fa440b7 9071b59dacb38535  3c738f8235cffa5b4d24f14e1b9ab5cd  6ebfd5287bef41dd7d0bf229a3054a5e  39d967bf8fa440b79071b59dacb38535  rookstar@Refrigerator:~/Git/os-labs/os\_lab\_05/src$ ldd dynamic\_queue\_sys  linux-vdso.so.1 (0x00007ffd25bab000)  libdl.so.2 => /lib/x86\_64-linux-gnu/libdl.so.2 (0x00007f73b44a1000)  libc.so.6 => /lib/x86\_64-linux-gnu/libc.so.6 (0x00007f73b40b0000)  /lib64/ld-linux-x86-64.so.2 (0x00007f73b48a8000) |

Проверяется использование загруженных библиотек.

Команда **ldd** показывает зависимости бинарных файлов от динамических библиотек.

Как видно, dynamic\_queue напрямую зависит от динамической библиотеки с очередью, в то время как dynamic\_queue\_sys подгружает ее с помощью **dlfcn.h**.

1. **Листинг программы**

*MD5Queue.h*

|  |
| --- |
| #ifndef MD5\_QUEUE\_H  #define MD5\_QUEUE\_H  #define MAX 100  typedef struct {  unsigned long long a;  unsigned long long b;  } MD5Sum;  typedef struct SumQueue {  int size;  int first;  MD5Sum array[MAX];  } SumQueue;  void sumsInit (SumQueue\*);  int sumsSize (SumQueue\*);  int sumsEmpty(SumQueue\*);  int sumsFull (SumQueue\*);  MD5Sum sumsPeek (SumQueue\*);  MD5Sum sumsPop (SumQueue\*);  void sumsPush (SumQueue\*, MD5Sum);  #endif |

*MD5Queue.c*

|  |
| --- |
| #include "MD5Queue.h"  void sumsInit(SumQueue \*Q)  {  Q->size = Q->first = 0;  }  int sumsSize(SumQueue \*Q)  {  return Q->size;  }  int sumsEmpty(SumQueue \*Q)  {  return Q->size == 0 ? 1 : 0;  }  int sumsFull(SumQueue \*Q)  {  return Q->size == MAX ? 1 : 0;  }  MD5Sum sumsPeek(SumQueue \*Q)  {  return Q->array[Q->first];  }  MD5Sum sumsPop(SumQueue \*Q)  {  int selected = Q->first;  --Q->size;  Q->first = (Q->first + 1) % MAX;  return Q->array[selected];  }  void sumsPush(SumQueue \*Q, MD5Sum add)  {  Q->array[(Q->first + Q->size) % MAX] = add;  ++Q->size;  } |

*queue.c*

|  |
| --- |
| #include <stdlib.h>  #include <stdio.h>  #include "MD5Queue.h"  int main()  {  SumQueue sumQ;  MD5Sum buff = {0, 0};  sumsInit(&sumQ);  printf("1: ");  scanf("%llx", &buff.a);  scanf("%llx", &buff.b);  sumsPush(&sumQ, buff);  printf("2: ");  scanf("%llx", &buff.a);  scanf("%llx", &buff.b);  sumsPush(&sumQ, buff);  printf("3: ");  scanf("%llx", &buff.a);  scanf("%llx", &buff.b);  sumsPush(&sumQ, buff);  printf("\n");  while(!sumsEmpty(&sumQ))  {  buff = sumsPop(&sumQ);  printf("%llx%llx\n", buff.a, buff.b);  }  } |

*queue\_sys.c*

|  |
| --- |
| #include <stdlib.h>  #include <stdio.h>  #include <dlfcn.h>  #include "MD5Queue.h"  int main()  {  void\* libHandle = NULL;  void (\*sumsInit) (SumQueue\*) = NULL;  void (\*sumsPush) (SumQueue\*, MD5Sum) = NULL;  int (\*sumsEmpty)(SumQueue\*) = NULL;  MD5Sum (\*sumsPop) (SumQueue\*) = NULL;  libHandle = dlopen("lib/libMD5Queue.so", RTLD\_LAZY);  if (!libHandle)  {  printf("%s\n", dlerror());  exit(EXIT\_FAILURE);  }  sumsInit = dlsym(libHandle, "sumsInit");  if (!sumsInit) {  printf("%s\n", dlerror());  exit(EXIT\_FAILURE);  }  sumsPush = dlsym(libHandle, "sumsPush");  if (!sumsPush) {  printf("%s\n", dlerror());  exit(EXIT\_FAILURE);  }  sumsEmpty = dlsym(libHandle, "sumsEmpty");  if (!sumsEmpty) {  printf("%s\n", dlerror());  exit(EXIT\_FAILURE);  }  sumsPop = dlsym(libHandle, "sumsPop");  if (!sumsPop) {  printf("%s\n", dlerror());  exit(EXIT\_FAILURE);  }    SumQueue sumQ;  MD5Sum buff = {0, 0};  sumsInit(&sumQ);  printf("1: ");  scanf("%llx", &buff.a);  scanf("%llx", &buff.b);  sumsPush(&sumQ, buff);  printf("2: ");  scanf("%llx", &buff.a);  scanf("%llx", &buff.b);  sumsPush(&sumQ, buff);  printf("3: ");  scanf("%llx", &buff.a);  scanf("%llx", &buff.b);  sumsPush(&sumQ, buff);  printf("\n");  while(!sumsEmpty(&sumQ))  {  buff = sumsPop(&sumQ);  printf("%llx%llx\n", buff.a, buff.b);  }  dlclose(libHandle);  } |

*Makefile*

|  |
| --- |
| LIBRARY\_NAME = MD5Queue  LIBRARY\_FLAGS = -Wall  BINARY\_FLAGS = -Wall  .PHONY: all  all: dynamic dynamic\_sys  dynamic: dynamic\_lib queue.c  gcc $(BINARY\_FLAGS) -Llib queue.c -l$(LIBRARY\_NAME) -o dynamic\_queue  dynamic\_sys: dynamic\_lib queue\_dyn.c  gcc $(BINARY\_FLAGS) queue\_dyn.c -ldl -o dynamic\_queue\_sys  #No dependencies since version control needs to be performed manually  dynamic\_lib:  gcc $(LIBRARY\_FLAGS) -fPIC -c $(LIBRARY\_NAME).c  gcc $(LIBRARY\_FLAGS) -shared -o lib/lib$(LIBRARY\_NAME).so $(LIBRARY\_NAME).o  static: static\_lib queue.c  gcc $(LIBRARY\_FLAGS) -Wall queue.c -Llib/ -l$(LIBRARY\_NAME) -o static\_queue  static\_lib: $(LIBRARY\_NAME).h $(LIBRARY\_NAME).c  gcc $(LIBRARY\_FLAGS) -Wall -c $(LIBRARY\_NAME).c  ar -cvq lib/lib$(LIBRARY\_NAME).a $(LIBRARY\_NAME).o  .PHONY: clean  clean:  rm -f \*.o dynamic\_queue dynamic\_queue\_sys static\_queue |

1. **Вывод**

Динамические библиотеки значительно сужают размер программ, использующих одни и те же библиотеки, так как они не компилируются не вместе с программой, а отдельно, затем подгружаясь в время работы программы (в то время как статические библиотеки будут многократно входить в разные программы одного проекта).