**Содержание**

[**1. Цели работы** 2](#_Toc86619832)

[**2. Задачи** 2](#_Toc86619833)

[**3. Ход выполнения работы** 2](#_Toc86619834)

[**4. Вывод** 13](#_Toc86619835)

# **Лабораторная работа №8 – Обмен через очереди сообщений**

# **1. Цели работы**

Знакомство с возможностями очередей сообщений (Message Queues) – мощного и гибкого средства межпроцессорного взаимодействия в OS Linux

**2. Задачи**

1. Войти в систему и скопировать в свой HOME-каталог с разделяемого ресурса набор исходных файлов для восьмой лабораторной работы.

2. Скомпилировать и выполнить программу gener\_mq.cpp, которая создает несколько очередей сообщений. После завершения программы выполнить программу ipcs и привести отличия результата от того, что был при вызове подобной команды из программы.

3. Скомпилировать программы sender.cpp и receiver.cpp, задав соответствующим исполняемым файлам разные имена (g++ <имя.cpp файла> -o <имя файла>). Запустить процессы на разных терминалах и необходимо передать им текстовые сообщения от процесса sender процессу receiver. Проанализировать, что происходит с ресурсом Message Queue, после завершения каждого из процессов (командой ipcs). При этом выполнить различные виды завершения отправкой сигналов SIGQUIT и SIGINT (нажатием Ctrl+C).

4. Запустить несколько процессов receiver на различных терминалах и, отправляя сообщения процессом sender, проанализировать ситуацию.

5. Модифицировать программы sender.cpp и receiver.cpp так, чтобы организовать отправку сообщений двух типов через одну и ту же очередь для двух различных процессов получателей. Для этого необходимо управлять параметром поля mtype структуры my\_msgbuf на передающей стороне и параметром msgtyp в системном вызове msgrcv() на приемной стороне.

**3. Ход выполнения работы**

1. Войду в систему и скопирую в свой HOME-каталог с разделяемого ресурса набор исходных файлов для второй лабораторной работы. Содержимое файлов приведено в листингах 8.1 – 8.3, соответственно.

|  |
| --- |
| Листинг 8.1 Содержимое файла gener\_mq.cpp |
| /\* The program gener\_mq.cpp \*/  /\* Message queue generation \*/  #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <limits.h>  #include <sys/types.h>  #include <sys/ipc.h>  #include <sys/msg.h>  #define MAX 5  main(void){  FILE \*fin;  char buffer[PIPE\_BUF];  char u\_char = 'A';  int i, n, mid[MAX];  key\_t key;  for (i=0; i<MAX ; ++i, ++u\_char){  key = ftok(".", u\_char);  if ((mid[i] = msgget(key, IPC\_CREAT | 0660))==-1){  perror("Queue create");  exit(1);  }  }  fin = popen("ipcs", "r"); /\* Run the ipcs command \*/  while((n = read(fileno(fin), buffer, PIPE\_BUF))>0)  write(fileno(stdout), buffer, n);  /\*Display the ipcs output \*/  pclose(fin);  for (i=0; i<MAX; ++i)  msgctl(mid[i], IPC\_RMID, NULL);  /\* remove \*/  exit(0);  } |

|  |
| --- |
| Листинг 8.2 Содержимое файла reciever.cpp |
| /\*  \*\* receiver.cpp -- reads from a message queue  \*/  #include <stdio.h>  #include <stdlib.h>  #include <errno.h>  #include <sys/types.h>  #include <sys/ipc.h>  #include <sys/msg.h>  struct my\_msgbuf {  long mtype;  char mtext[200];  };  int main(void)  {  struct my\_msgbuf buf;  int msqid;  key\_t key;  if ((key = ftok(".", 'B')) == -1) { /\* same key as sender.cpp \*/  perror("ftok");  exit(1);  }  if ((msqid = msgget(key, 0644)) == -1) { /\* connect to the queue \*/  perror("msgget");  exit(1);  }  printf("spock: ready to receive messages, captain.\n");  for(;;) { /\* Spock never quits! \*/  if (msgrcv(msqid, &buf, sizeof(buf.mtext), 0, 0) == -1) {  perror("msgrcv");  exit(1);  }  printf("spock: \"%s\"\n", buf.mtext);  }  return 0;  } |

|  |
| --- |
| Листинг 8.3 Содержимое файла sender.cpp |
| /\*  \*\* sender.cpp -- writes to a message queue  \*/  #include <stdio.h>  #include <stdlib.h>  #include <errno.h>  #include <string.h>  #include <sys/types.h>  #include <sys/ipc.h>  #include <sys/msg.h>  struct my\_msgbuf {  long mtype;  char mtext[200];  };  int main(void)  {  struct my\_msgbuf buf;  int msqid;  key\_t key  if ((key = ftok(".", 'B')) == -1) {  perror("ftok");  exit(1);  }  if ((msqid = msgget(key, 0644 | IPC\_CREAT)) == -1) {  perror("msgget");  exit(1);  }  printf("Enter lines of text, ^D to quit:\n");  buf.mtype = 1; /\* we don't really care in this case \*/  while(fgets(buf.mtext, sizeof buf.mtext, stdin) != NULL) {  int len = strlen(buf.mtext);  /\* ditch newline at end, if it exists \*/  if (buf.mtext[len-1] == '\n') buf.mtext[len-1] = '\0';  if (msgsnd(msqid, &buf, len+1, 0) == -1) /\* +1 for '\0' \*/  perror("msgsnd");  }  if (msgctl(msqid, IPC\_RMID, NULL) == -1) {  perror("msgctl");  exit(1);  }  return 0;  } |

2. Скомпилирую и выполню программу gener\_mq.cpp, которая создает несколько очередей сообщений. С помощью опции g++ выполню компиляцию. Полученный результат программы gener\_mq.cpp записывается в файл gener\_mq. Выполню его в терминале. Вывод в терминал представлен на рисунке 8.1. После завершения программы выполню команду ipcs. Результат приведен на рисунке 8.2. Важно отметить, что при исполнении программы gener\_mq создается 5 очередей сообщений и после выводится информация о механизмах IPC, а далее очереди сообщений удаляются, но при вызове команды ipcs выводится информация о всех механизмах в системе IPC, но при этом очереди не создаются.

|  |
| --- |
| Рис. 8.1 – Выполнение программы gener\_mq |
| Рис. 8.2 – Выполнение команды ipcs |

3. Скомпилирую программы sender.cpp и receiver.cpp, задав соответствующим исполняемым файлам разные имена (g++ <имя.cpp файла> -o <имя файла>). Выполнение представлено на рисунке 8.3.

|  |
| --- |
| Рис. 8.3 – Компиляция файлов |

Далее запущу процессы на разных терминалах и передам текстовые сообщения от процесса sender процессу receiver. Иллюстрация выполненных действия представлена на рисунке 8.4.

|  |
| --- |
| Рис. 8.4 - Передача текстовых сообщений между процессами |

При завершении передачи Ctrl+D, очередь сообщений удаляется. Результаты приведены на рисунке 8.5.

|  |
| --- |
| Рис. 8.5 – Завершение передачи сообщений через Ctrl+D |

Важно отметить, что данные программы обмениваются сообщениями только используя очередь сообщений, что представлено на рисунке 8.4. Прерывание процессов с помощью SIGQUIT представлено на рисунке 8.6. В данном случаем очередь сообщений не удаляется и остается в системе. Аналогично и SIGINT. Результаты приведены на рисунке 8.7.

|  |
| --- |
| Рис. 8.6 – Завершение с помощью SIGQUIT |
| Рис. 8.7 – Завершение с помощью SIGINT |

4. Необходимо исследовать ситуацию, когда процесс receiver запускается уже после того, как процесс sender отправил в очередь одно или множество сообщений. Для этого выполню файл sender, отправлю несколько сообщений, а после уже создам процесс receiver. Результаты приведены на рисунке 8.8. Из результатов видно, что созданный позже процесс receiver получил сообщения, так как сообщения будут ждать в очереди, даже если и sender завершиться c помощью SIGINT или SIGQUIT. Моделирование ситуации, когда sender завершается раньше приведено на рисунке 8.9.

|  |
| --- |
| Рис. 8.8 – Запуск receiver позже отправки сообщений |
| Рис. 8.9 – Запуск receiver при завершенном sender |

5. Запущу несколько процессов receiver на различных терминалах и, отправляя сообщения процессом sender, посмотрю, что будет при отправке сообщений. Моделирование ситуации представлено на рисунке 8.10.

|  |
| --- |
| Рис. 8.10 – Запуск двух процессов receiver в двух разных терминалах |

Чтобы протестировать ситуацию, введу четыре разных сообщения. Результаты приведены на рисунке 8.11. Как можно увидеть, сообщения доходят не все, а только половина. Данный факт обусловлен тем, что сообщение читается и сразу же удаляется из очереди – другой процесс не может его прочитать, а вот во время получения первого, второй успевает прочитать второе сообщение и так далее.

|  |
| --- |
| Рис. 8.11 – Результаты тестирования отправки на два процесса |

6. Необходимо модифицировать программы sender.cpp и receiver.cpp так, чтобы была организована отправка сообщений двух типов через одну и ту же очередь для двух различных процессов получателей. Для этого необходимо управлять параметром в поле mtype структуры my\_msgbuf на передающей стороне и параметром msgtyp в системном вызове msgrcv() на приемной стороне. Модифицированные программы представлены в листингах 8.4 и 8.5 соответственно.

|  |
| --- |
| Листинг 8.4 – Модифицированная программа sender\_m.cpp |
| /\*  \*\* sender.cpp -- writes to a message queue  \*/  #include <stdio.h>  #include <stdlib.h>  #include <errno.h>  #include <string.h>  #include <sys/types.h>  #include <sys/ipc.h>  #include <sys/msg.h>  struct my\_msgbuf {  long mtype;  char mtext[200];  };  int main(void)  {  struct my\_msgbuf buf;  int msqid;  key\_t key  if ((key = ftok(".", 'B')) == -1) {  perror("ftok");  exit(1);  }  if ((msqid = msgget(key, 0644 | IPC\_CREAT)) == -1) {  perror("msgget");  exit(1);  }  printf("Enter lines of text, ^D to quit:\n");  buf.mtype = 1; /\* we don't really care in this case \*/  while(fgets(buf.mtext, sizeof buf.mtext, stdin) != NULL) {  int len = strlen(buf.mtext);  /\* ditch newline at end, if it exists \*/  if (buf.mtext[len-1] == '\n') buf.mtext[len-1] = '\0';  for(long i = 1; i <= 2; i++){  buf.mtype = i;  if (msgsnd(msqid, &buf, len+1, 0) == -1) /\* +1 for '\0' \*/  perror("msgsnd");  }  }  if (msgctl(msqid, IPC\_RMID, NULL) == -1) {  perror("msgctl");  exit(1);  }  return 0;  } |

|  |
| --- |
| Листинг 8.5 – Модифицированная программа receiver\_m.cpp |
| /\*  \*\* receiver.cpp -- reads from a message queue  \*/  #include <stdio.h>  #include <stdlib.h>  #include <errno.h>  #include <sys/types.h>  #include <sys/ipc.h>  #include <sys/msg.h>  struct my\_msgbuf {  long mtype;  char mtext[200];  };  int main(void)  {  struct my\_msgbuf buf;  int msqid;  key\_t key;  if ((key = ftok(".", 'B')) == -1) { /\* same key as sender.cpp \*/  perror("ftok");  exit(1);  }  if ((msqid = msgget(key, 0644)) == -1) { /\* connect to the queue \*/  perror("msgget");  exit(1);  }  printf("spock: ready to receive messages, captain.\n");  for(;;) { /\* Spock never quits! \*/  if (msgrcv(msqid, &buf, sizeof(buf.mtext), buf.mtype, 0) == -1) {  perror("msgrcv");  exit(1);  }  printf("spock: \"%s\"\n", buf.mtext);  }  return 0;  } |

Результаты отправки сообщений на два процесса получателя приведены на рисунке 8.12. Как видно сообщений отправляются корректно. Важно отметить, чтобы изменять количество процессов-приемников нужно изменить for() в sender\_m.cpp.

|  |
| --- |
| Рис. 8.12 – Моделирование отправки на два процесса-получателя |

**4. Вывод**

В ходе выполнения работы были изучены основы межпроцессорного взаимодействия в ОС Linux при помощи очередей сообщений между процессами. При помощи параметров была настроена передача сообщений одновременно нескольким процессам-получателям.