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

**Цель работы**

Знакомство с возможностями очередей сообщений (Message Queues) – мощного и гибкого средства межпроцессного взаимодействия в ОС Linux.

**Пункт 1**

Скомпилируйте и выполните программу gener\_mq.cpp , создающую несколько очередей сообщений. После завершения программы выполните команду ipcs и поясните отличие результата оттого, что был при вызове подобной команды из программы.

|  |
| --- |
|  |
| Рис. 8-1 Результат выполнения gener и ipcs. |

При исполнении программы gener создаётся 5 очередей сообщений и после выводится информация о механизмах IPC, а после очереди сообщений удаляются. При вызове команды ipcs просто выводится информация о всех механизмах IPC в системе, но никаких очередей не создаётся.

**Пункт 2**

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

|  |
| --- |
| Изображение выглядит как текст  Автоматически созданное описание |
| Рис. 8-2 Передача сообщений между sender и receiver. |

Можно сделать вывод, что данные программы обмениваются данными по очереди сообщений.

|  |
| --- |
|  |
| Рис. 8-3 После завершения процесса. |

Если прекратить исполнение программы через CTRL+D, то очередь сообщений удалится.

При прерывании процессов при помощи отправки сигналов SIGQUIT и SIGINT очередь сообщений не удаляется и остаётся в системе.

|  |
| --- |
| Изображение выглядит как текст, снимок экрана, монитор, экран  Автоматически созданное описание  Изображение выглядит как текст  Автоматически созданное описание |
| Рис. 8-4 Завершения процессов через сигналы. |

**Пункт 4**

Ответьте на вопрос: что происходит, если процесс receiver запускается уже после того, как процесс sender отправил в очередь одно или множество сообщений?

Все сообщения будут ждать в очереди, и даже если завершить процесс (не удаляя очередь), то после запуска получателя все сообщения дойдут. Но при чтении сообщения сразу удаляются из очереди, и другой читатель их не получит.

**Пункт 5**

Запустите несколько процессов receiver на различных терминалах и, отправляя сообщения процессом sender , проанализируйте ситуацию.

|  |
| --- |
| Изображение выглядит как текст  Автоматически созданное описание |
| Рис. 8-5 Результат выполнения gener и ipcs. |

Приёмники получает лишь половину сообщений. Это связано с тем, что при получении сообщения его читают и сразу удаляют из очереди. Поэтому другой процесс не может прочитать те же данные.

**Пункт 6**

Модифицируйте 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 в поле mtype структуры my\_msgbuf на передающей стороне и параметром msgtyp в системном вызове msgrcv() на приемной стороне.

Добавим в код файла sender.cpp изменение поля mtype. Будем отправлять два сообщения, с разными типами. Таким образом получатели будут принимать по одному сообщению и всё будет работать как задумывалось.

|  |
| --- |
| Изображение выглядит как текст  Автоматически созданное описание |
| Рис. 8-6 Результат выполнения модифицированных программ. |

|  |
| --- |
| Листинг 8-1 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';    for(long i = 1; i < 3; 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 8-2 receiver.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;  } |

**Вывод**

В работе познакомились с возможностями инструмента межпроцессного взаимодействия в ОС Linux – очередь сообщений. С помощью него наглядно передавали сообщения между различными процессами. Также при помощи параметров настроили трансляцию очереди сообщений нексольким приемникам.