МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ»

Кафедра автоматизированных систем управления (АСУ)

УТВЕРЖДАЮ

Зав. кафедрой АСУ, профессор

 А.М. Кориков

АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНЫХ КОМПЛЕКСОВ

Методические рекомендации для выполнения лабораторной работы №4

Тема: «POSIX. Обмен сообщениями»

Учебно-методическое пособие

для студентов уровеня основной образовательной программы магистратура
направления подготовки 010400.68 «Прикладная математика и информатика»
профиля Математическое и программное обеспечение вычислительных комплексов и компьютерных сетей

Разработчик

доцент кафедры АСУ

В.Г. Резник

2012

Резник В.Г.

Архитектура вычислительных комплексов. Лабораторная работа №4: POSIX. Обмен сообщениями. Учебно-методическое пособие. – Томск, ТУСУР, 2012. – 20 с.

Учебно-методическое пособие предназначено для выполнения лабораторной работы №4 по дисциплине «Архитектура вычислительных комплексов» для студентов уровеня основной образовательной программы магистратура направления подготовки 010400.68 «Прикладная математика и информатика» профиля «Математическое и программное обеспечение вычислительных комплексов и компьютерных сетей».

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Рассматриваемое учебно-методическое пособие содержит методические рекомендации по написанию программного обеспечения (ПО) при выполнении лабораторных работ по дисциплине «Архитектура вычислительных комплексов» (АВК). Тема лабораторной работы: «POSIX. Обмен сообщениями».

Знание основ стандарта POSIX входит в набор общекультурных компетенций студента. Умение студента разрабатывать программы, удовлетворяющие стандарту POSIX, составляет основу профессиональных компетенций по дисциплине АВК. В целом, лабораторная работа №4 продолжает формирование базовых знаний по разработке ПО на основе вычисительного комплекса кафедры АСУ. При изложении материала использовалась информация с сайта [1].

Последовательность и изложение материала данного учебно-методичес-кого пособия предполагает, что студент:

- успешно выполнил задания по лабораторным работам №1, №2 и №3 по темам: «Технология поготовки сетевого вычислительного комплекса (СВК) кафедры АСУ. Тестовый пример на СВК», «POSIX. Сигналы процессов», «POSIX. Разделяемая память»;
- владеет теоретическими знаниями и практическим умением, полученными при изучении дисциплины «Современные операционные системы»;
- имеет практические навыки разработки прикладного ПО на языке программирования С.

Первый раздел учебно-методического пособия дает описание средств локального межпроцессного взаимодействия в стандарте POSIX. Здесь вводится краткое описание утилит, функций и информационных структур средств обмена сообщениями, сигналами и работы с разделяемой памятью.

Во втором разделе дано описание функций, необходимых как для создания очередей сообщений, так и передачи самих сообщений.

В третьем разделе приведены конкретные примеры программ для работы с очередями сообщений.

В четвертом разделе ставится задача по проведению лабораторной работы и дается описание этапов, обеспечивающих упешное выполнение учебного задания.

Пятый раздел учебно-методического пособия содержит рекомендаци по подведению итогов лабораторной работы №4.

1. Средства локального межпроцессного взаимодействия POSIX

Средства локального межпроцессного взаимодействия являются необязательной части стандарта POSIX-2001, которая именуется " X/Open-расширение системного интерфейса " (XSI). Эти средства включают в себя очереди сообщений, семафоры и разделяемые сегменты памяти. Необходимость изучения этих средств обусловлена высокой эффективностью их применения, что, при правильном их применении, приводит к увеличению быстродействия разрабатываемых приложений.

Каждое из перечисленных выше средств для своей работы использует целочисленный идентификатор, однозначно определяющее это средство:

- **msqid** — идентификатор очереди сообщений, который возвращается в качестве результатов функций **msgget()**;
- **semid** — идентификатор семафора, который возвращается в качестве результатов функций **semget()**;
- **shmid** — идентификатор сегмента разделяемой памяти, который возвращается в качестве результатов функций **shmget()**.

При получении идентификаторов средств межпроцессного взаимодействия используется еще одна сущность - ключ, для генерации котрого предназначена функция **ftok()**:

```
#include <sys/ipc.h>
key_t ftok (const char *path, int id);
```

здесь аргумент **path** должен задавать маршрутное имя существующего файла, к которому вызывающий процесс может применить функцию **stat()**.
В качестве значения аргумента **id**, по соображениям мобильности, рекомендуется использовать однобайтный символ.

В общем случае гарантируется, что функция **ftok()** сгенерирует один и тот же ключ для заданной пары (файл, символ) и разные ключи для разных пар.

С каждым идентификатором средств межпроцессного взаимодействия ассоциированы структуры данных, содержащие информацию о допустимых и выполненных операциях.

Соответствующие декларации сосредоточены в заголовочных файлах <sys/msg.h>, <sys/sem.h> и <sys/shm.h>.

В каждую упомянутую структуру входит подструктура **ipc_perm** с данными о владельцах и режимом доступа, описанная в файле <sys/ipc.h> и содержащая по крайней мере следующие поля:

```
uid_t  uid;
/* Идентификатор владельца */
gid_t  gid;
/* Идентификатор владеющей группы */
uid_t  cuid;
/* Идентификатор пользователя,
   создавшего данное средство
   межпроцессного взаимодействия */
gid_t  cgid;
/* Идентификатор создавшей группы */
mode_t mode;
/* Режим доступа на чтение/запись */
```

Управление доступом к этим средствам осуществляется аналогично файловому доступу. Отличие состоит в том, что наряду с владельцами (пользователем и группой) рассматриваются и те, кто эти средства создал (создатели).

Для опроса статуса присутствующих в данный момент в системе средств используется служебная программа **ipcs**:

```
ipcs [-qms] [-a | -bcopt]
```

По умолчанию выдается краткая информация обо всех средствах - очередях сообщений, семафорах и разделяемых сегментах памяти.

Если нужно ограничиться их отдельными видами, следует воспользоваться опциями **-q**, **-s** и/или **-m**, соответственно.

Для управления форматом выдачи используются опции:
-a - равносильно указанию всех опций формата;
-b - предписывает выдавать лимиты на размер (максимальное количество байт в сообщениях очереди и т.п.);
-c - имена пользователя и группы создателя средства;
-o - информацию об использовании (количество сообщений в очереди, их суммарный размер и т.п.),
-p - информацию о процессах (идентификаторы последнего отправителя, получателя и т.п.),
-t - информацию о времени (последняя управляющая операция, последняя отправка сообщения и т.п.).

Для удаления из системы активных средств межпроцессного взаимодействия предназначена служебная программа **ipcrm**, подверженная контролю прав доступа.

Удаляемые средства могут задаваться идентификаторами или ключами:

```
ipcrm [-q msgid | -Q msgkey | -s semid | -S semkey | -m shmid | -M shmkey ] …
```

2. Функции для работы с очередями сообщений

Механизм очередей сообщений позволяет процессам взаимодействовать, обмениваясь данными. Данные передаются между процессами дискретными порциями, называемыми сообщениями.

Процессы выполняют над сообщениями две основные операции - прием и отправку.

Процессы, отправляющие или принимающие сообщение, могут приостанавливаться, если требуемую операцию невозможно выполнить немедленно.

В частности, могут быть отложены попытки отправить сообщение в заполненную до отказа очередь, получить сообщение из пустой очереди и т.п. ("операции с блокировкой").

Если же указано, что приостанавливать процесс нельзя, то мы имеем "операции без блокировки". Такие операции: либо выполняются немедленно, либо завершаются неудачей.

Прежде чем процессы смогут обмениваться сообщениями, один из них должен создать очередь. Одновременно определяются первоначальные права на выполнение операций для различных процессов, в том числе и права соответствующих управляющих действий над очередями.

Работа с очередями сообщений в стандарте POSIX-2001 предусматривает использование следующих функций:

- **msgget()** - получение идентификатора очереди сообщений;
- **msgctl()** - управление очередью сообщений;
- **msgsnd()** - отправка сообщения;
- **msgrcv()** - прием сообщения.

Определения этих функций на языке **C** имеют вид:

```
#include <sys/msg.h>
int msgget (key_t key, int msgflg);
int msgsnd (int msqid, const void *msgp, size_t msgsz, int msgflg);
ssize_t msgrcv (int msqid, void *msgp,
                size_t msgsz, long msgtyp, int msgflg);
int msgctl (int msqid, int cmd, struct msqid_ds *buf);
```

Структура **msqid_ds**, ассоциированная с идентификатором очереди сообщений, должна содержать по крайней мере следующие поля:

```
struct ipc_perm msg_perm;
/* Данные о правах доступа
к очереди сообщений */
msgqnum_t       msg_qnum;
/* Текущее количество сообщений в очереди */
```

```
msglen_t        msg_qbytes;
/* Максимально допустимый суммарный
размер сообщений в очереди */
pid_t           msg_lspid;
/* Идентификатор процесса, отправившего
последнее сообщение */
pid_t           msg_lrpid;
/* Идентификатор процесса, принявшего
последнее сообщение */
time_t          msg_stime;
/* Время последней отправки */
time_t          msg_rtime;
/* Время последнего приема */
time_t          msg_ctime;
/* Время последнего изменения
посредством msgctl() */
```

Проведем детальное рассмотрение функций работы с очередями сообщений.

Функция **msgget()** возвращает идентификатор очереди сообщений, ассоциированный с ключом **key**.

Новая очередь, ее идентификатор и соответствующая структура **msqid_ds** создаются для заданного ключа, если:

- значение аргумента **key** равно **IPC_PRIVATE**;
- очередь еще не ассоциирована с ключом, а в числе флагов **msgflg** задан **IPC_CREAT**.

Если необходима уверенность в том, что очередь с указанным ключом создается заново, то в дополнение к флагу **IPC_CREAT** следует установить **IPC_EXCL**. Тогда попытка получить идентификатор уже существующий очереди завершится неудачей.

Структура **msqid_ds** для новой очереди инициализируется следующим образом:

- Значения полей **msg_perm.cuid**, **msg_perm.uid**, **msg_perm.cgid** и **msg_perm.gid** устанавливаются равными действующим идентификаторам пользователя и группы вызывающего процесса.
- Младшие девять бит поля **msg_perm.mode** устанавливаются равными младшим девяти битам значения **msgflg**.
- Поля **msg_qnum, msg_lspid, msg_lrpid, msg_stime** и **msg_rtime** обнуляются.
- В поле **msg_ctime** помещается текущее время;
- В поле **msg_qbytes** - определенный в системе лимит.

Одим из важных вопросов, связанных с созданием очереди сообщений, заключается в выборе ключа: все процессы, которые намереваются работать с

общей очередью сообщений, для получения идентификатора **msqid** должны знать ключ очереди.

Задание ключа одинаковым константным значением во всех программах является небезопасным, поскольку может оказаться так, что тот же ключ будет случайно задействован и другими программами.

Наиболее приемлемым решением является использование функции **ftok()**, вычисляющей действительно "уникальный" ключ.

Операции отправки/приема сообщений выполняют функции **msgsnd()** и **msgrcv()**:

- **msgsnd()** - помещает сообщения в очередь;
- **msgrcv()** - читает и "достает" их оттуда.

В обоих случаях:

- первый аргумент задает идентификатор очереди;
- второй является указателем на содержащую сообщение структуру.

Каждое сообщение состоит из двух частей:

- **текста** - последовательности байт;
- **типа** - положительного целого числа.

Тип, указанный во время отправки, используется впоследствии при выборе сообщения из очереди.

Аргумент **msgsz** определяет длину сообщения;

Аргумент **msgflg** задает флаги.

В зависимости от значения, указанного в качестве аргумента **msgtyp** функции **msgrcv()**, из очереди выбирается то или иное сообщение:

- если значение аргумента равно нулю, запрашивается первое сообщение в очереди;
- если больше нуля - первое сообщение типа **msgtyp**;
- если меньше нуля - первое сообщение наименьшего из типов, не превышающих абсолютную величину аргумента **msgtyp**.

Пусть, например, в очередь последовательно помещены сообщения с типами 5, 3 и 2. Тогда:

- вызов **msgrcv (msqid, msgp, size, 0, flags)** выберет из очереди сообщение с типом 5, поскольку оно отправлено первым;
- вызов **msgrcv (msqid, msgp, size, -4, flags)** - последнее сообщение, так как 2 - это наименьший из возможных типов в указанном диапазоне;
- вызов **msgrcv (msqid, msgp, size, 3, flags)** - сообщение с типом 3.

Многие приложения, взаимодействующие между собой посредством очередей сообщений, требуют синхронизации своего выполнения. Ряд примеров таких приложений:

- процесс-получатель, пытавшийся прочитать сообщение и обнаруживший, что очередь пуста, либо сообщение указанного типа отсутствует, должен иметь возможность подождать, пока процесс-отправитель не поместит в очередь требуемое сообщение;
- аналогичным образом, процесс, желающий отправить сообщение в очередь, в которой нет достаточного для него места, может ожидать его освобождения в результате чтения сообщений другими процессами.

Процесс, вызвавший блокировку подобного рода: "операцию с блокировкой", приостанавливается до тех пор, пока:

- либо станет возможным выполнение операции;
- либо будет ликвидирована очередь.

С другой стороны, имеются приложения, где подобные ситуации должны приводить к немедленному (неудачному) завершению вызова функции.

Если не указано противное, функции **msgsnd()** и **msgrcv()** выполняют операции с блокировкой. Например:

msgsnd (msqid, msgp, size, 0);
msgrcv (msqid, msgp, size, type, 0);

Чтобы выполнить операцию без блокировки, необходимо установить флаг **IPC_NOWAIT**:

msgsnd (msqid, msgp, size, IPC_NOWAIT);
msgrcv (msqid, msgp, size, type, IPC_NOWAIT);

Для хранения реальных принимаемых и передаваемых сообщений, в прикладной программе следует определить структуру, указав желаемый размер сообщения.

Аргумент **msgp**, в рассматриваемых функциях, указывает на значение структурного типа, в котором представлены тип и тело сообщения:

```
struct msgbuf {
 long mtype;        /* Тип сообщения */
 char mtext [1];    /* Текст сообщения */
};
```

На практике, буфер приема/передачи и его размер во многих случаях определяют так:

```
#define MAXSZTMSG 8192

struct mymsgbuf {
 long mtype;             /* Тип сообщения */
 char mtext [MAXSZTMSG]; /* Текст сообщения */
};
struct mymsgbuf msgbuf;
```

В качестве аргумента **msgsz** обычно указывается размер текстового буфера, например: **sizeof (msgbuf.text)**.

Если не указано противное, в случае, когда длина выбранного сообщения больше, чем msgsz, вызов msgrcv() завершается неудачей.

Если же установить флаг MSG_NOERROR, длинное сообщение обрезается до msgsz байт. Отброшенная часть пропадает, а вызывающий процесс не получает никакого уведомления о том, что сообщение обрезано.

При успешном завершении:

- **msgsnd()** возвращает **0**;
- **msgrcv()** - значение, равное числу реально полученных байт;
- при неудаче возвращается **-1**.

Процессы, обладающие достаточными правами доступа, посредством функции **msgctl()** могут получать информацию о состоянии очереди, изменять ряд характеристик, удалять очередь.

Управляющее действие определяется значением аргумента **cmd**. Допустимых значений три:

- **IPC_STAT** - получить информацию о состоянии очереди;
- **IPC_SET** - переустановить характеристики очереди;
- **IPC_RMID** - удалить очередь.

Для хранения информации об очереди, команды **IPC_STAT** и **IPC_SET** используют имеющуюся в прикладной программе структуру типа **msqid_ds**, указатель на которую содержит аргумент buf:

- **IPC_STAT** - копирует в нее ассоциированную с очередью структуру данных;
- **IPC_SET** - наоборот обновляет ассоциированную структуру.

Команда **IPC_SET** позволяет переустановить значения идентификаторов владельца (**msg_perm.uid**) и владеющей группы (**msg_perm.gid**), режима доступа (**msg_perm.mode**), максимально допустимый суммарный размер сообщений в очереди (**msg_qbytes**).

Увеличить значение **msg_qbytes** может только процесс, обладающий соответствующими привилегиями.

3. Примеры программ на С для работы с очередями сообщений

Непосредственное применение средств очередей передачи сообщений продемострируем конкретными примерами.

Пример 1. Программа создает очередь сообщений с правами доступа, указанными в командной строке. Для генерации ключа очереди сообщений используется функция **ftok()**.

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

/* Программа создает очередь сообщений.              */
/* В командной строке задаются имя файла для ftok() */
/* и режим доступа к очереди сообщений               */

#define FTOK_CHAR       'G'

int main (int argc, char *argv []) {
 key_t key;
 int msqid;
 int mode = 0;

 if (argc != 3) {
    fprintf (stderr, "Использование: %s маршрутное_имя режим_доступа\n", argv
[0]);
   return (1);
 }

 if ((key = ftok (argv [1], FTOK_CHAR)) == (key_t) (-1)) {
   perror ("FTOK");
   return (2);
 }
 (void) sscanf (argv [2], "%o", (unsigned int *) &mode);

 if ((msqid = msgget (key, IPC_CREAT | mode)) < 0) {
   perror ("MSGGET");
   return (3);
 }

 return 0;
}
```

Если после выполнения этой программы воспользоваться командой

ipcs -q

то результат может выглядеть:

```
------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x47034bac 163840     galat      644        0            0
```

Удалить созданную очередь из системы, соответствующей стандарту POSIX-2001, можно командой: **ipcrm -q 163840**.

Пример 2. Программа, изменяющая максимально допустимый суммарный размер сообщений в очереди.

Предполагается, что очередь сообщений уже создана, а ее идентификатор известен.

Эту программу можно выполнить с разными значениями максимально допустимого суммарного размера: как меньше, так и больше текущего.

Результат может меняться в зависимости от прав пользователя.

```
#include <stdio.h>
#include <sys/msg.h>

int main (int argc, char *argv []) {
 int msqid;
 struct msqid_ds msqid_ds;

 if (argc != 3) {
           fprintf    (stderr,    "Использование:    %s    идентификатор_очереди
максимальный_размер\n", argv [0]);
   return (1);
 }

 (void) sscanf (argv [1], "%d", &msqid);

 /* Получим исходное значение структуры данных */
 if (msgctl (msqid, IPC_STAT, &msqid_ds) == -1) {
   perror ("IPC_STAT-1");
   return (2);
 }
    printf    ("Максимальный    размер    очереди    до    изменения:    %ld\n",
msqid_ds.msg_qbytes);

 (void) sscanf (argv [2], "%d", (int *) &msqid_ds.msg_qbytes);

 /* Попробуем внести изменения */
 if (msgctl (msqid, IPC_SET, &msqid_ds) == -1) {
   perror ("IPC_SET");
 }

 /* Получим новое значение структуры данных */
 if (msgctl (msqid, IPC_STAT, &msqid_ds) == -1) {
   perror ("IPC_STAT-2");
   return (3);
 }
    printf    ("Максимальный    размер    очереди    после    изменения:    %ld\n",
msqid_ds.msg_qbytes);

 return 0;
}
```

Пример 3. Программы, демонстрирующие полный цикл работы с очередями сообщений: от создания до удаления.

Первая программа представляет собой родительский процесс, читающий строки со стандартного ввода и отправляющий их в виде сообщений процессу-потомку, который запускается из файла **msq_child** текущего каталога.

Следует обратить особое внимание на способ выработки согласованного ключа.

```
#include <unistd.h>
#include <stdio.h>
#include <limits.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/msg.h>

/* Программа копирует строки со стандартного ввода на стандартный вывод, */
/* "прокачивая" их через очередь сообщений                                */

#define FTOK_FILE       "/home/galat"
#define FTOK_CHAR       "G"

#define MSGQ_MODE       0644

#define MY_PROMPT       "Вводите строки\n"
#define MY_MSG          "Вы ввели: "

int main (void) {
 key_t key;
 int msqid;
 struct mymsgbuf {
   long mtype;
   char mtext [LINE_MAX];
 } line_buf, msgbuf;

 switch (fork ()) {
   case -1:
     perror ("FORK");
     return (1);
   case 0:
     /* Чтение из очереди и выдачу на стандартный вывод */
     /* реализуем в порожденном процессе.               */
      (void) execl ("./msq_child", "msq_child", FTOK_FILE, FTOK_CHAR, (char *)
0);
     perror ("EXEC");
     return (2);   /* execl() завершился неудачей */
 }

 /* Чтение со стандартного ввода и запись в очередь */
 /* возложим на родительский процесс                */

 /* Выработаем ключ для очереди сообщений */
 if ((key = ftok (FTOK_FILE, FTOK_CHAR [0])) == (key_t) (-1)) {
   perror ("FTOK");
   return (3);
 }

 /* Получим идентификатор очереди сообщений */
 if ((msqid = msgget (key, IPC_CREAT | MSGQ_MODE)) < 0) {
   perror ("MSGGET");
   return (4);
 }

 /* Приступим к отправке сообщений в очередь */
 msgbuf.mtype = line_buf.mtype = 1;
 strncpy (msgbuf.mtext, MY_PROMPT, sizeof (msgbuf.mtext));
 if (msgsnd (msqid, (void *) &msgbuf, strlen (msgbuf.mtext) + 1, 0) != 0) {
```

```
   perror ("MSGSND-1");
   return (5);
 }
 strncpy (msgbuf.mtext, MY_MSG, sizeof (msgbuf.mtext));

 while (fgets (line_buf.mtext, sizeof (line_buf.mtext), stdin) != NULL) {
   if (msgsnd (msqid, (void *) &msgbuf, strlen (msgbuf.mtext) + 1, 0) != 0) {
     perror ("MSGSND-2");
     break;
   }
   if (msgsnd (msqid, (void *) &line_buf, strlen (line_buf.mtext) + 1, 0) != 0)
{
     perror ("MSGSND-3");
     break;
   }
 }

 /* Удалим очередь */
 if (msgctl (msqid, IPC_RMID, NULL) == -1) {
   perror ("MSGCTL-IPC_RMID");
   return (6);
 }

 return (0);
}
```

Программа файла процесса-потомка, который принимает сообщения и выдает их тела на стандартный вывод. Предполагается, что файл этой программы имеет имя **msq_child** и находится в текущем каталоге.

```
#include <stdio.h>
#include <limits.h>
#include <sys/msg.h>

/* Программа получает сообщения из очереди */
/* и копирует их тела на стандартный вывод */

#define MSGQ_MODE       0644

int main (int argc, char *argv []) {
 key_t key;
 int msqid;
 struct mymsgbuf {
   long mtype;
   char mtext [LINE_MAX];
 } msgbuf;

 if (argc != 3) {
   fprintf (stderr, "Использование: %s имя_файла цепочка_символов\n", argv [0]);
   return (1);
 }

 /* Выработаем ключ для очереди сообщений */
 if ((key = ftok (argv [1], *argv [2])) == (key_t) (-1)) {
   perror ("CHILD FTOK");
   return (2);
 }

 /* Получим идентификатор очереди сообщений */
```

```
 if ((msqid = msgget (key, IPC_CREAT | MSGQ_MODE)) < 0) {
   perror ("CHILD MSGGET");
   return (3);
 }

 /* Цикл приема сообщений и выдачи строк */
 while (msgrcv (msqid, (void *) &msgbuf, sizeof (msgbuf.mtext), 0, 0) > 0) {
   if (fputs (msgbuf.mtext, stdout) == EOF) {
     break;
   }
 }

 return 0;
}
```

4. Задание на лабораторную работу №4

Среди имеющихся средств межпроцессного взаимодействия процессов, очереди сообщений обладают рядом преимуществ:

- возможность передачи сообщений произвольного формата;
- по сравнению с сигналами, не перегружены семантикой использования в операционной системе;
- по сравнению с разделяемыми сегментами памяти — менее чувствительны к взаимным блокировкам и возможностью разрушения данных.

Используя полученные в процессе обучения навыки программирования, теоретический и методический материал, изложенный в разделах 1 — 3 данного учебно-методического пособия, студент должен:

- написать, отладить и запустить две программы на языке С, которые обеспечивают обмен кодированными сообщениями;
- продемострировать преподавателю работу этих программ;
- подготовить и представить на проверку преподавателю письменный отчет о выполнении лабораторной работы, содержащий текст программ, результаты демострирующие исследовательскую часть работы и выводы о технологии и возможностях использования очередей сообщений при разработке прикладных программ.

Методические указания по проведению данной лабораторной работы рекомендуют студенту последовательное выполнение следующих этапов:

1. Изучение разделов 1 и 2 учебно-методического пособия с целью повышения теоретического уровня своей подготовки для целей разработки прикладных программ, использующих очереди сообщений.
2. Изучение и запуск примеров программ из раздела 3 данного учебно-методического пособия.
3. Завершив изучение методического и практического материала, студент должен подключиться к кластеру ЭВМ кафедры АСУ так, как изложено в методическом пособии по лабораторной работе №1.
4. Запустив на кластере инструментальную среду разработки EclipsePMP, студент должен создать необходимые проекты, написать, отладить запустить на выполнение прикладные программы, согласно указанному выше заданию.
5. Результаты выполения данных программ следует продемострировать преподавателю.
6. По результатам исследования студент подготавливает письменный отчет, при необходимости, иллюстрируя его результатами исследования.
7. Подготовив отчет, студент докладывает о выполнении задания преподавателю и следует его указаниям по сдаче выполненной работы.

5. Контроль навыков выполнения лабораторной работы №4

Обязательным требованием по контролю знаний и умений студента является наличие письменного отчета по выполненой лабораторной работе №4.

Отчет оформляется как четвертый раздел общего отчета по дисциплине «Архитектура вычислительных комплексов» и находится в определенном преподавателем месте учебного комплекса кафедры АСУ.

Отчет по лабораторной работе №4 должен содержать, как минимум три подраздела: постановка задачи, описание работы, выводы.

Цель подготовки и сдачи отчета — формирование у студентов общекультурных компетенций по оформлению и представлению научных и исследовательских документов.

Порядок контроля навыков студента и сдача отчета проводятся в следующей последовательности.

1. Студент сообщает преподавателю о завершении выполнения задания и готовности студента к контролю навыков и сдаче отчета.
2. Преподаватель убеждается в наличии отчета и необходимых элементов его оформления, а затем делает замечания по устранению недостаков отчета или уточняет время и условия контроля навыков и приема результатов работы.
3. В процессе сдачи отчета, студент демонстрирует результаты работы и отвечает на вопросы преподавателя, при необходимости, устраняет ошибки или недоработки, отмеченные преподавателем.
4. Приняв отчет студента, преподаватель сообщает об этом факте устно, при необходимости оценивает результаты работы, а также определяет дальнейший процесс обучения студента.

ЛИТЕРАТУРА

1. Галатенко В.А. Прогаммирование в стандарте POSIX. Интернет ресурс: http://www.intuit.ru/department/se/pposix/8/.

Учебное издание

Резник Виталий Григорьевич

АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНЫХ КОМПЛЕКСОВ

Методические рекомендации для выполнения лабораторной работы №4 по дисциплине «Архитектура вычислительных комплексов» для студентов уровеня основной образовательной программы магистратура направления подготовки 010400.68 «Прикладная математика и информатика» профиля «Математическое и программное обеспечение вычислительных комплексов и компьютерных сетей».

Учебно-методическое пособие

Усл. печ. л. . Тираж ___. Заказ .

Томский государственный университет

систем управления и радиоэлектроники

634050, г. Томск, пр. Ленина, 40