6.12. Простой интерпретатор команд.

Данный раздел просто приводит исходный текст простого интерпретатора команд. Функция *match* описана в главе "Текстовая обработка".

```
    /* Примитивный интерпретатор команд. Распознает построчно
     * команды вида: CMD ARG1 ... ARGn <FILE >FILE >>FILE >&FILE >>&FILE
     * Сборка: cc -U42 -DCWDONLY sh.c match.c pwd.c -o sh
     */

    #include <sys/types.h>/* определение типов, используемых системой */
    #include <stdio.h>      /* описание библиотеки ввода/вывода  */
    #include <signal.h>     /* описание сигналов                 */
    #include <fcntl.h>      /* определение O_RDONLY              */
    #include <errno.h>      /* коды системных ошибок             */
    #include <ctype.h>      /* макросы для работы с символами    */
    #include <dirent.h>     /* эмуляция файловой системы BSD 4.2 */
    #include <pwd.h>        /* работа с /etc/passwd              */
    #include <sys/wait.h>   /* описание формата wait()           */

    char cmd[256];          /* буфер для считывания команды */
    #define   MAXARGS 256   /* макс. количество аргументов  */
    char *arg[MAXARGS];     /* аргументы команды */
    char *fin, *fout;       /* имена для перенаправления ввода/вывода */
    int  rout;              /* флаги перенаправления вывода */

    char *firstfound;       /* имя найденной, но невыполняемой программы */
    #define LIM ':'         /* разделитель имен каталогов в path */
    extern char *malloc(), *getenv(), *strcpy(), *getwd();
    extern char *strchr(), *execat();
    extern void callshell(), printenv(), setenv(), dowait(), setcwd();
    extern struct passwd   *getpwuid();
            /* Предопределенные переменные */
    extern char **environ;  /* окружение: изначально смотрит на тот же
                             * массив, что и ev из main() */
    extern int errno;       /* код ошибки системного вызова   */

    char *strdup(s)char *s;
    { char *p; return(p=malloc(strlen(s)+1), strcpy(p,s)); }
            /* strcpy() возвращает свой первый аргумент */
    char *str3spl(s, p, q) char *s, *p, *q;
    { char *n = malloc(strlen(s)+strlen(p)+strlen(q)+1);
      strcpy(n, s); strcat(n, p); strcat(n, q); return n;
    }

    int cmps(s1, s2) char **s1, **s2;
    { return strcmp(*s1, *s2); }
    /* Перенаправить вывод */
    #define APPEND 0x01
    #define ERRTOO 0x02
    int output (name, append, err_too, created) char *name; int *created;
    {
        int     fd;
        *created = 0;     /* Создан ли файл ? */

        if( append ){                   /* >>file */
            /* Файл name существует? Пробуем открыть на запись */
            if((fd = open (name, O_WRONLY)) < 0) {
                if (errno == ENOENT) /* Файл еще не существовал */
                    goto CREATE;
                else
                    return 0;    /* Не имеем права писать в этот файл */
            }
            /* иначе fd == открытый файл, *created == 0 */
        }else{
    CREATE: /* Пытаемся создать (либо опустошить) файл "name" */
            if((fd = creat (name, 0666)) < 0 )
                        return 0;     /* Не могу создать файл  */
            else        *created = 1; /* Был создан новый файл */
```

```
        }
        if (append)
            lseek (fd, 0l, 2);      /* на конец файла */
     /* перенаправить стандартный вывод */
        dup2(fd, 1);
        if( err_too ) dup2(fd, 2);  /* err_too=1 для >& */
        close(fd); return 1;
    }

    /* Перенаправить ввод */
    int input (name) char *name;
    {
        int     fd;
        if((fd = open (name, O_RDONLY)) < 0 ) return 0;/* Не могу читать */
     /* перенаправить стандартный ввод */
        dup2(fd, 0); close(fd); return 1;
    }
    /* запуск команды */
    int cmdExec(progr, av, envp, inp, outp, outflg)
            char *progr;       /* имя программы */
            char **av;         /* список аргументов */
            char **envp;       /* окружение */
            char *inp, *outp;  /* файлы ввода-вывода (перенаправления) */
            int outflg;        /* режимы перенаправления вывода */
    {
            void (*del)(), (*quit)();
            int pid;
            int cr = 0;

            del = signal(SIGINT, SIG_IGN); quit = signal(SIGQUIT, SIG_IGN);
            if( ! (pid = fork())){     /* ветвление */
               /* порожденный процесс (сын) */
               signal(SIGINT, SIG_DFL); /* восстановить реакции */
               signal(SIGQUIT,SIG_DFL); /* по умолчанию         */
               /* getpid() выдает номер (идентификатор) данного процесса */
               printf( "Процесс pid=%d запущен\n", pid = getpid());

               /* Перенаправить ввод-вывод */
               if( inp ) if(!input( inp )){
                 fprintf(stderr, "Не могу <%s\n", inp ); goto Err;
               }
               if( outp )
                   if(!output (outp, outflg & APPEND, outflg & ERRTOO, &cr)){
                       fprintf(stderr, "Не могу >%s\n", outp ); goto Err;
                   }
     /* Заменить программу: при успехе
      * данная программа завершается, а вместо нее вызывается
      * функция main(ac, av, envp) программы, хранящейся в файле progr.
      * ac вычисляет система.
      */
               execvpe(progr, av, envp);

    Err:
      /* при неудаче печатаем причину и завершаем порожденный процесс */
               perror(firstfound ? firstfound: progr);
               /* Мы не делаем free(firstfound),firstfound = NULL
                * потому что данный процесс завершается (и тем ВСЯ его
                * память освобождается) :
                */
               if( cr && outp )  /* был создан новый файл     */
                   unlink(outp); /* но теперь он нам не нужен */

               exit(errno);
            }
            /* процесс - отец */

            /* Сейчас сигналы игнорируются, wait не может быть оборван
             * прерыванием с клавиатуры */
            dowait();       /* ожидать окончания сына */
            /* восстановить реакции на сигналы от клавиатуры */
            signal(SIGINT, del); signal(SIGQUIT, quit);
            return pid;     /* вернуть идентификатор сына */
    }
    /* Запуск программы с поиском по переменной среды PATH */
    int execvpe(progr, av, envp) char *progr, **av, **envp;
    {
            char *path, *cp;
            int try = 1;
            register eacces = 0;
            char fullpath[256];     /* полное имя программы */

            firstfound = NULL;
            if((path = getenv("PATH")) == NULL )
                path = ".:/bin:/usr/bin:/etc";
```

```
        /* имя: короткое или путь уже задан ? */
        cp = strchr(progr, '/') ? "" : path;
        do{     /* пробуем разные варианты */
                cp = execat(cp, progr, fullpath);
        retry:
            fprintf(stderr, "пробуем \"%s\"\n", fullpath );
                execve(fullpath, av, envp);
                /* если программа запустилась, то на этом месте данный
                 * процесс заменился новой программой. Иначе - ошибка. */
                switch( errno ){  /* какова причина неудачи ? */
                case ENOEXEC:  /* это командный файл */
                        callshell(fullpath, av, envp);
                        return (-1);
                case ETXTBSY:   /* файл записывается */
                        if( ++try > 5 ) return (-1);
                        sleep(try); goto retry;
                case EACCES:    /* не имеете права */
                        if(firstfound == NULL)
                           firstfound = strdup(fullpath);
                        eacces++; break;
                case ENOMEM:    /* программа не лезет в память */
                case E2BIG:
                        return (-1);
                }
        }while( cp );
        if( eacces ) errno = EACCES;
        return (-1);
}

/* Склейка очередной компоненты path и имени программы name */
static char *execat(path, name, buf)
        register char *path, *name;
        char *buf;      /* где будет результат */
{
        register char *s = buf;
        while(*path && *path != LIM )
                *s++ = *path++;         /* имя каталога */
        if( s != buf ) *s++ = '/';
        while( *name )
                *s++ = *name++;         /* имя программы */
        *s = '\0';
        return ( *path ? ++path /* пропустив LIM */ : NULL );
}

/* Запуск командного файла при помощи вызова интерпретатора */
void callshell(progr, av, envp) char *progr, **av, **envp;
{
        register i; char *sh; char *newav[MAXARGS+2];
        int fd; char first = 0;

        if((fd = open(progr, O_RDONLY)) < 0 )
                sh = "/bin/sh";
        else{
                read(fd, &first, 1); close(fd);
                sh = (first == '#') ? "/bin/csh" : "/bin/sh";
        }
        newav[0] = "Shellscript"; newav[1] = progr;
        for(i=1; av[i]; i++)
                newav[i+1] = av[i];
        newav[i+1] = NULL;
        printf( "Вызываем %s\n", sh );
        execve(sh, newav, envp);
}

/* Ожидать окончания всех процессов, выдать причины смерти. */
void dowait(){
        int ws; int pid;

        while((pid = wait( &ws)) > 0 ){
          if( WIFEXITED(ws)){
            printf( "Процесс %d умер с кодом %d\n",
                            pid,            WEXITSTATUS(ws));
          }else if( WIFSIGNALED(ws)){
            printf( "Процесс %d убит сигналом %d\n",
                            pid,             WTERMSIG(ws));
            if(WCOREDUMP(ws)) printf( "Образовался core\n" );
            /* core - образ памяти процесса для отладчика adb */
          }else if( WIFSTOPPED(ws)){
            printf( "Процесс %d остановлен сигналом %d\n",
                            pid,            WSTOPSIG(ws));
          }
       }
}
/* Расширение шаблонов имен. Это упрощенная версия, которая
```

```
 * расширяет имена только в текущем каталоге.
 */
void glob(dir, args, indx, str /* что расширять */, quote )
     char *args[], *dir; int *indx; char *str;
     char quote; /* кавычки, в которые заключена строка str */
{
        static char globchars[] = "*?[";
        char *p; char **start = &args[ *indx ];
        short nglobbed = 0;

        register struct dirent *dirbuf;
        DIR *fd; extern DIR *opendir();

        /* Затычка для отмены глоббинга: */
        if( *str == '\\' ){ str++;    goto noGlob; }

        /* Обработка переменных $NAME    */
        if( *str == '$' && quote != '\'' ){
                char *s = getenv(str+1);
                if( s ) str = s;
        }
        /* Анализ: требуется ли глоббинг */
        if( quote ) goto noGlob;
        for( p=str; *p; p++ )  /* Есть ли символы шаблона? */
                if( strchr(globchars, *p))
                        goto doGlobbing;
noGlob:
        args[ (*indx)++ ] = strdup(str);
        return;

doGlobbing:
       if((fd = opendir (dir)) == NULL){
              fprintf(stderr, "Can't read %s\n", dir); return;
       }
       while ((dirbuf = readdir (fd)) != NULL ) {
              if (dirbuf->d_ino == 0) continue;
              if (strcmp (dirbuf->d_name, ".") == 0 ||
                  strcmp (dirbuf->d_name, "..") == 0) continue;
              if( match( dirbuf->d_name, str)){
                  args[ (*indx)++ ] = strdup(dirbuf->d_name);
                  nglobbed++;
              }
       }
       closedir(fd);
       if( !nglobbed){
           printf( "%s: no match\n", str);
           goto noGlob;
       }else{   /* отсортировать */
           qsort(start, nglobbed, sizeof (char *), cmps);
       }
}

/* Разбор командной строки */
int parse(s) register char *s;
{
        int i; register char *p;
        char tmp[80];   /* очередной аргумент */
        char c;

        /* очистка старых аргументов */
        for(i=0; arg[i]; i++) free(arg[i]), arg[i] = NULL;
        if( fin  ) free(fin ), fin  = NULL;
        if( fout ) free(fout), fout = NULL;
        rout = 0;

        /* разбор строки */
        for( i=0 ;; ){
                char quote = '\0';

                /* пропуск пробелов - разделителей слов */
                while((c = *s) && isspace(c)) s++;
                if( !c ) break;
                /* очередное слово */
                p = tmp;
                if(*s == '\'' || *s == '"' ){
                /* аргумент в кавычках */
                        quote = *s++;  /* символ кавычки */
                        while((c = *s) != '\0' && c != quote){
                           if( c == '\\' ){ /* заэкранировано */
                               c = *++s;
                               if( !c ) break;
                           }
                           *p++ = c; ++s;
                        }
```

```
                        if(c == '\0')
                      fprintf(stderr, "Нет закрывающей кавычки %c\n", quote);
                        else s++; /* проигнорировать кавычку на конце */
                    } else
                        while((c = *s) && !isspace(c)){
                           if(c == '\\') /* заэкранировано */
                                if( !(c = *++s))
                                     break /* while */;
                           *p++ = c; s++;
                        }
                    *p = '\0';
                    /* Проверить, не есть ли это перенаправление
                     * ввода/вывода. В отличие от sh и csh
                     * здесь надо писать >ФАЙЛ <ФАЙЛ
                     * >< вплотную к имени файла.
                     */
                    p = tmp;        /* очередное слово */
                    if( *p == '>'){ /* перенаправлен вывод */
                        p++;
                        if( fout ) free(fout), rout = 0; /* уже было */
                        if( *p == '>' ){ rout |= APPEND; p++; }
                        if( *p == '&' ){ rout |= ERRTOO; p++; }
                        if( !*p ){
                                fprintf(stderr, "Нет имени для >\n");
                                fout = NULL; rout = 0;
                        } else  fout = strdup(p);
                    } else if( *p == '<' ){  /* перенаправлен ввод */
                        p++;
                        if( fin ) free(fin);        /* уже было */
                        if( !*p ){
                                fprintf(stderr, "Нет имени для <\n");
                                fin = NULL;
                        } else  fin = strdup(p);
                    } else /* добавить имена к аргументам */
                                glob( ".", arg, &i, p, quote );
            }
            arg[i] = NULL; return i;
    }

    /* Установить имя пользователя */
    void setuser(){
            int uid = getuid();     /* номер пользователя, запустившего Шелл */
            char *user = "mr. Nobody";      /* имя пользователя     */
            char *home = "/tmp";            /* его домашний каталог */
            struct passwd *pp = getpwuid( uid );
            if( pp != NULL ){
                    if(pp->pw_name && *pp->pw_name ) user = pp->pw_name;
                    if(               *pp->pw_dir  ) home = pp->pw_dir;
            }
            setenv("USER", user); setenv("HOME", home);
    }

    void setcwd(){ /* Установить имя текущего каталога */
            char cwd[512];
            getwd(cwd); setenv( "CWD", cwd );
    }
    void main(ac, av, ev) char *av[], *ev[]; {
            int argc;               /* количество аргументов */
            char *prompt;           /* приглашение           */

            setuser(); setcwd();
            signal(SIGINT, SIG_IGN);
            setbuf(stdout, NULL);  /* отменить буферизацию */
            for(;;){
                 prompt = getenv( "prompt" ); /* setenv prompt -->\  */
                 printf( prompt ? prompt : "@ ");/* приглашение */
                 if( gets(cmd) == NULL /* at EOF  */ ) exit(0);
                 argc = parse(cmd);
                 if( !argc) continue;
                 if( !strcmp(arg[0], "exit" )) exit(0);

                 if( !strcmp(arg[0], "cd" )){
                     char *d = (argc==1) ? getenv("HOME"):arg[1];
                     if(chdir(d) < 0)
                        printf( "Не могу войти в %s\n", d );
                     else setcwd();
                     continue;
                 }

                 if( !strcmp(arg[0], "echo" )){
                     register i; FILE *fp;
                     if( fout ){
                        if((fp = fopen(fout, rout & APPEND ? "a":"w"))
                               == NULL) continue;
```

```
                    }  else fp = stdout;
                    for(i=1; i < argc; i++ )
                       fprintf( fp, "%s%s", arg[i], i == argc-1 ? "\n":"   ");
                    if( fp != stdout ) fclose(fp);
                    continue;
               }

               if( !strcmp(arg[0], "setenv" )){
                         if( argc == 1 ) printenv();
                    else if( argc == 2 ) setenv( arg[1], "" );
                    else                 setenv( arg[1], arg[2]);
                    continue;
               }
               cmdExec(arg[0], (char **) arg, environ, fin, fout, rout);
          }
     }

     /* ------------------------------------------------------------*/
     /* Отсортировать и напечатать окружение */
     void printenv(){
          char *e[40]; register i = 0; char *p, **q = e;

          do{
               p = e[i] = environ[i]; i++;
          } while( p );

     #ifdef SORT
          qsort( e, --i /* сколько */, sizeof(char *), cmps);
     #endif
          while( *q )
               printf( "%s\n", *q++ );
     }

     /* Сравнение имени переменной окружения с name */
     static char *envcmp(name, evstr) char *name, *evstr;
     {
          char *p; int code;
          if((p = strchr(evstr, '=')) == NULL ) return NULL; /* error ! */
          *p = '\0';      /* временно */
          code = strcmp(name, evstr);
          *p = '=';       /* восстановили */
          return code==0 ? p+1 : NULL;
     }

     /* Установить переменную окружения */
     void setenv( name, value ) char *name, *value;
     {
          static malloced = 0;    /* 1, если environ перемещен */
          char *s, **p, **newenv;
          int len, change_at = (-1), i;

          /* Есть ли переменная name в environ-е ? */
          for(p = environ; *p; p++ )
               if(s = envcmp(name, *p)){       /* уже есть */
                    if((len = strlen(s)) >= strlen(value)){
                        /* достаточно места */
                        strcpy(s, value); return;
                    }
                    /* Если это новый environ ... */
                    if( malloced ){
                        free( *p ); *p = str3spl(name, "=", value);
                        return;
                    }
                    /* иначе создаем копию environ-а */
                    change_at = p - environ;        /* индекс */
                    break;
               }
          /* Создаем копию environ-а. Если change_at == (-1), то
           * резервируем новую ячейку для еще не определенной переменной */
          for(p=environ, len=0; *p; p++, len++ );
          /* вычислили количество переменных */
          if( change_at < 0 ) len++;
          if((newenv = (char **) malloc( sizeof(char *) * (len+1)))
                  == (char **) NULL) return;
          for(i=0; i < len+1; i++ ) newenv[i] = NULL;     /* зачистка */
          /* Копируем старый environ в новый */
          if( !malloced ) /* исходный environ в стеке (дан системой) */
               for(i=0; environ[i]; i++ ) newenv[i] = strdup(environ[i]);
          else for(i=0; environ[i]; i++ ) newenv[i] =        environ[i];
          /* Во втором случае строки уже были спасены, копируем ссылки */

          /* Изменяем, если надо: */
          if( change_at >= 0 ){
                  free( newenv[change_at] );
```

```
                newenv[change_at] = str3spl(name, "=", value);
        } else {        /* добавить в конец новую переменную */
                newenv[len-1] = str3spl(name, "=", value);
        }
        /* подменить environ */
        if( malloced ) free( environ );
        environ = newenv; malloced++;
        qsort( environ, len, sizeof(char *), cmps);
}

/* Допишите команды:
   unsetenv имя_переменной - удаляет переменную среды;
   exit N                  - завершает интерпретатор с
                             кодом возврата N (это целое число);
*/
```


Назад | Содержание | Вперед

[Главная] [Гостевая]