6.2.10. Второй пример использования таймера – это таймер, отсчитывающий текущее время суток (а также дату). Чтобы получить значение этого таймера используется вызов функции *gettimeofday*

```
#include <time.h>

void main(){
        struct timeval timenow;

        gettimeofday(&timenow, NULL);
        printf("%u sec, %u msec\n",
                timenow.tv_sec,
                timenow.tv_usec
        );
        printf("%s", ctime(&timenow.tv_sec));
        exit(0);
}
```

Поле *tv_sec* содержит число секунд, прошедшее с полуночи 1 января 1970 года до данного момента; в чем полностью соответствует системному вызову *time*. Однако плюс к тому поле *tv_usec* содержит число миллионных долей текущей секунды (значение этого поля всегда меньше 1000000).

6.2.11. К данному параграфу вернитесь, изучив раздел про *fork*() и *exit*(). Каждый процесс может пребывать в двух фазах: системной (внутри тела системного вызова – его выполняет для нас ядро операционной системы) и пользовательской (внутри кода самой программы). Время, затраченное процессом в каждой фазе, может быть измеряно системным вызовом *times*(). Кроме того, этот вызов позволяет узнать суммарное время, затраченное порожденными процессами (порожденными при помощи *fork*). Системный вызов заполняет структуру

```
struct tms {
        clock_t tms_utime;
        clock_t tms_stime;
        clock_t tms_cutime;
        clock_t tms_cstime;
};
```

и возвращает значение

```
#include <sys/times.h>

struct tms time_buf;
clock_t real_time = times(&time_buf);
```

Все времена измеряются в "тиках" – некоторых долях секунды. Число тиков в секунде можно узнать таким системным вызовом (в системе *Solaris*):

```
#include <unistd.h>
clock_t HZ = sysconf(_SC_CLK_TCK);
```

В старых системах, где таймер работал от сети переменного тока, это число получалось равным 60 (60 Герц – частота сети переменного тока). В современных системах это 100. Поля структуры содержат:

tms_utime

время, затраченное вызывающим процессом в пользовательской фазе.

tms_stime

время, затраченное вызывающим процессом в системной фазе.

tms_cutime

время, затраченное порожденными процессами в пользовательской фазе: оно равно сумме всех **tms_utime** и **tms_cutime** порожденных процессов (рекурсивное суммирование).

tms_cstime

время, затраченное порожденными процессами в системной фазе: оно равно сумме всех **tms_stime** и **tms_cstime** порожденных процессов (рекурсивное суммирование).

real_time

время, соответствующее астрономическому времени системы. Имеет смысл мерять только их разность.

Вот пример программы:

```
#include <stdio.h>
#include <unistd.h>     /* _SC_CLK_TCK */
#include <signal.h>     /* SIGALRM */
#include <sys/time.h>   /* не используется */
#include <sys/times.h>  /* struct tms */

struct tms tms_stop,  tms_start;
clock_t    real_stop, real_start;

clock_t HZ;     /* число ticks в секунде */

/* Засечь время момента старта процесса */
void hello(void){
        real_start = times(&tms_start);
}
/* Засечь время окончания процесса */
void bye(int n){
        real_stop = times(&tms_stop);
#ifdef CRONO
        /* Разность времен */
        tms_stop.tms_utime -= tms_start.tms_utime;
        tms_stop.tms_stime -= tms_start.tms_stime;
#endif

        /* Распечатать времена */
        printf("User   time          = %g seconds [%lu ticks]\n",
          tms_stop.tms_utime / (double)HZ, tms_stop.tms_utime);
        printf("System time          = %g seconds [%lu ticks]\n",
          tms_stop.tms_stime / (double)HZ, tms_stop.tms_stime);
        printf("Children user   time = %g seconds [%lu ticks]\n",
          tms_stop.tms_cutime / (double)HZ, tms_stop.tms_cutime);
        printf("Children system time = %g seconds [%lu ticks]\n",
          tms_stop.tms_cstime / (double)HZ, tms_stop.tms_cstime);
        printf("Real time            = %g seconds [%lu ticks]\n",
          (real_stop - real_start) / (double)HZ, real_stop - real_start);
        exit(n);
}

/* По сигналу SIGALRM - завершить процесс */
void onalarm(int nsig){
        printf("Выход #%d ================\n", getpid());
        bye(0);
}
/* Порожденный процесс */
void dochild(int n){
        hello();
        printf("Старт #%d ================\n", getpid());
        signal(SIGALRM, onalarm);

        /* Заказать сигнал SIGALRM через 1 + n*3 секунд */
        alarm(1 + n*3);

        for(;;){}       /* зациклиться в user mode */
}

#define NCHLD 4
int main(int ac, char *av[]){
        int i;

        /* Узнать число тиков в секунде */
        HZ = sysconf(_SC_CLK_TCK);
        setbuf(stdout, NULL);

        hello();
        for(i=0; i < NCHLD; i++)
                if(fork() == 0)
                        dochild(i);
        while(wait(NULL) > 0);
        printf("Выход MAIN ================\n");
        bye(0);
        return 0;
}
```

и ее выдача:

```
Старт #3883 ================
Старт #3884 ================
Старт #3885 ================
Старт #3886 ================
Выход #3883 ================
User   time          = 0.72 seconds [72 ticks]
System time          = 0.01 seconds [1 ticks]
Children user   time = 0 seconds [0 ticks]
Children system time = 0 seconds [0 ticks]
Real time            = 1.01 seconds [101 ticks]
Выход #3884 ================
User   time          = 1.88 seconds [188 ticks]
System time          = 0.01 seconds [1 ticks]
Children user   time = 0 seconds [0 ticks]
Children system time = 0 seconds [0 ticks]
Real time            = 4.09 seconds [409 ticks]
Выход #3885 ================
User   time          = 4.41 seconds [441 ticks]
System time          = 0.01 seconds [1 ticks]
Children user   time = 0 seconds [0 ticks]
Children system time = 0 seconds [0 ticks]
Real time            = 7.01 seconds [701 ticks]
Выход #3886 ================
User   time          = 8.9 seconds [890 ticks]
System time          = 0 seconds [0 ticks]
Children user   time = 0 seconds [0 ticks]
Children system time = 0 seconds [0 ticks]
Real time            = 10.01 seconds [1001 ticks]
Выход MAIN =================
User   time          = 0.01 seconds [1 ticks]
System time          = 0.04 seconds [4 ticks]
Children user   time = 15.91 seconds [1591 ticks]
Children system time = 0.03 seconds [3 ticks]
Real time            = 10.41 seconds [1041 ticks]
```

Обратите внимание, что 72+188+441+890=1591 (поле tms_cutime для main).

6.2.12. Еще одна программа: хронометрирование выполнения другой программы. Пример:

timer ls -l

```
/* Хронометрирование выполнения программы */
#include <stdio.h>
#include <unistd.h>
#include <sys/times.h>

extern errno;

typedef struct _timeStamp {
        clock_t real_time;
        clock_t cpu_time;
        clock_t child_time;
        clock_t child_sys, child_user;
} TimeStamp;

TimeStamp TIME(){
        struct tms tms;
        TimeStamp  st;

        st.real_time  = times(&tms);
        st.cpu_time   = tms.tms_utime +
                        tms.tms_stime +
                        tms.tms_cutime +
                        tms.tms_cstime;
        st.child_time = tms.tms_cutime +
                        tms.tms_cstime;
        st.child_sys  = tms.tms_cstime;
        st.child_user = tms.tms_cutime;
        return st;
}

void PRTIME(TimeStamp start, TimeStamp stop){
        clock_t HZ = sysconf(_SC_CLK_TCK);
        clock_t real_time  = stop.real_time  - start.real_time;
        clock_t cpu_time   = stop.cpu_time   - start.cpu_time;
        clock_t child_time = stop.child_time - start.child_time;

        printf("%g real, %g cpu, %g child (%g user, %g sys), %ld%%\n",
                real_time       / (double)HZ,
                cpu_time        / (double)HZ,
                child_time      / (double)HZ,
```

```
                stop.child_user / (double)HZ,
                stop.child_sys  / (double)HZ,
                (child_time * 100L) / (real_time ? real_time : 1)
        );
}

TimeStamp start, stop;

int main(int ac, char *av[]){
        char *prog = *av++;
        if(*av == NULL){
                fprintf(stderr, "Usage: %s command [args...]\n", prog);
                return(1);
        }
        start = TIME();
        if(fork() == 0){
                execvp(av[0], av);
                perror(av[0]);
                exit(errno);
        }
        while(wait(NULL) > 0);
        stop = TIME();
        PRTIME(start, stop);
        return(0);
}
```

6.3. Свободное место на диске.

6.3.1. Системный вызов *ustat*() позволяет узнать количество свободного места в файловой системе, содержащей заданный файл (в примере ниже – текущий каталог):

```
#include <sys/types.h>
#include <sys/stat.h>
#include <ustat.h>
struct stat st; struct ustat ust;
void main(int ac, char *av[]){
     char *file = (ac==1 ? "." : av[1]);
     if( stat(file, &st) < 0) exit(1);
     ustat(st.st_dev, &ust);
     printf("На диске %*.*s\n"
       "%ld свободных блоков (%ld Кб)\n"
       "%d свободных I-узлов\n",
     sizeof ust.f_fname, sizeof ust.f_fname,
     ust.f_fname, /* название файловой системы (метка) */
     ust.f_tfree, /* блоки по 512 байт */
    (ust.f_tfree * 512L) / 1024,
     ust.f_tinode );
}
```

Обратите внимание на запись длинной строки в *printf*: строки, перечисленные последовательно, склеиваются *ANSI C* компилятором в одну длинную строку:

```
char s[] = "This is"  " a line "  "of words";
        совпадает с
char s[] = "This is a line of words";
```

6.3.2. Более правильно, однако, пользоваться сисвызовом *statvfs* – статистика по виртуальной файловой системе. Рассмотрим его в следующем примере: копирование файла с проверкой на наличие свободного места.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdarg.h>
#include <fcntl.h>              /* O_RDONLY */
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/statvfs.h>
#include <sys/param.h>          /* MAXPATHLEN */

char *progname;                 /* имя программы */

void error(char *fmt, ...){
        va_list args;

        va_start(args, fmt);
        fprintf(stderr, "%s: ", progname);
        vfprintf(stderr, fmt, args);
        fputc('\n', stderr);
        va_end(args);
}
```

```
int copyFile(char *to, char *from){       /* куда, откуда */
        char newname[MAXPATHLEN+1];
        char answer[20];
        struct stat stf, stt;
        int fdin, fdout;
        int n, code = 0;
        char iobuf[64 * 1024];
        char *dirname = NULL, *s;

        if((fdin = open(from, O_RDONLY)) < 0){
                error("Cannot read %s", from);
                return (-1);
        }
        fstat(fdin, &stf);
        if((stf.st_mode & S_IFMT) == S_IFDIR){
                close(fdin);
                error("%s is a directory", from);
                return (-2);
        }

        if(stat(to, &stt) >= 0){
                /* Файл уже существует */

                if((stt.st_mode & S_IFMT) == S_IFDIR){
                        /* И это каталог */

                        /* Выделить последнюю компоненту пути from */
                        if((s = strrchr(from, '/')) && s[1])
                                s++;
                        else    s = from;

                        dirname = to;

                        /* Целевой файл - файл в этом каталоге */
                        sprintf(newname, "%s/%s", to, s);
                        to = newname;

                        if(stat(to, &stt) < 0)
                                goto not_exist;
                }

                if(stt.st_dev == stf.st_dev && stt.st_ino == stf.st_ino){
                        error("%s: cannot copy file to itself", from);
                        return (-3);
                }
                switch(stt.st_mode & S_IFMT){
                case S_IFBLK:
                case S_IFCHR:
                case S_IFIFO:
                        break;

                default:
                        printf("%s already exists, overwrite ? ", to);
                        fflush(stdout);

                        *answer = '\0';
                        gets(answer);

                        if(*answer != 'y'){     /* NO */
                                close(fdin);
                                return (-4);
                        }
                        break;
                }
        }

not_exist:
        printf("COPY %s TO %s\n", from, to);

        if((stf.st_mode & S_IFMT) == S_IFREG){
                /* Проверка наличия свободного места в каталоге dirname */
                struct statvfs fs;
                char tmpbuf[MAXPATHLEN+1];

                if(dirname == NULL){
                        /* To 'to' - это имя файла, а не каталога */
                        strcpy(tmpbuf, to);
                        if(s = strrchr(tmpbuf, '/')){
                                if(*tmpbuf != '/' || s != tmpbuf){
                                        /* Имена "../xxx"
                                         * и второй случай:
                                         * абсолютные имена не в корне,
                                         * то есть не "/" и не "/xxx"
                                         */
```

```
                                        *s = '\0';
                                }else{
                                        /* "/" или "/xxx" */
                                        if(s[1]) s[1] = '\0';
                                }
                                dirname = tmpbuf;
                        } else  dirname = ".";
                }

                if(statvfs(dirname, &fs) >= 0){
                        size_t size = (geteuid() == 0 ) ?
                                /* Доступно суперпользователю: байт */
                                fs.f_frsize * fs.f_bfree :
                                /* Доступно обычному пользователю: байт */
                                fs.f_frsize * fs.f_bavail;

                        if(size < stf.st_size){
                           error("Not enough free space on %s: have %lu, need %lu",
                                  dirname, size, stf.st_size);
                           close(fdin);
                           return (-5);
                        }
                }
        }

        if((fdout = creat(to, stf.st_mode)) < 0){
                error("Can't create %s", to);
                close(fdin);
                return (-6);
        } else {
                fchmod(fdout, stf.st_mode);
                fchown(fdout, stf.st_uid, stf.st_gid);
        }

        while (n = read (fdin, iobuf, sizeof iobuf)) {
                if(n < 0){
                        error ("read error");
                        code = (-7);
                        goto done;
                }
                if(write (fdout, iobuf, n) != n) {
                        error ("write error");
                        code = (-8);
                        goto done;
                }
        }

done:
        close (fdin);
        close (fdout);

        /* Проверить: соответствует ли результат ожиданиям */
        if(stat(to, &stt) >= 0 && (stt.st_mode & S_IFMT) == S_IFREG){
                if(stf.st_size < stt.st_size){
                        error("File has grown at the time of copying");
                } else if(stf.st_size > stt.st_size){
                        error("File too short, target %s removed", to);
                        unlink(to);
                        code = (-9);
                }
        }
        return code;
}

int main(int argc, char *argv[]){
        int i, code = 0;

        progname = argv[0];

        if(argc < 3){
                error("Usage: %s from... to", argv[0]);
                return 1;
        }
        for(i=1; i < argc-1; i++)
                code |= copyFile(argv[argc-1], argv[i]) < 0 ? 1 : 0;
        return code;
}
```

Возвращаемая структура *struct statvfs* содержит такие поля (в частности):

Типа *long*:

f_frsize	размер блока
f_blocks	размер файловой системы в блоках
f_bfree	свободных блоков (для суперпользователя)

```
    f_bavail            свободных блоков (для всех остальных)

    f_files             число I-nodes в файловой системе
    f_ffree             свободных I-nodes (для суперпользователя)
    f_favail            свободных I-nodes (для всех остальных)

    Типа char *
    f_basetype          тип файловой системы: ufs, nfs, ...
```

По два значения дано потому, что операционная система резервирует часть файловой системы для использования ТОЛЬКО суперпользователем (чтобы администратор смог распихать файлы в случае переполнения диска, и имел резерв на это). *ufs* - это UNIX file system из BSD 4.x

Назад | Содержание | Вперед

[Главная] [Гостевая]