

11. «Прочие» системные вызовы ОС GNU/Linux

Разделы:

- Библиотечные функции и системные вызовы
- Утилита *strace*
- Примеры системных вызовов



Функции и вызовы

- **Библиотечная функция** – это обычная функция, которая находится во внешней библиотеке, подключаемой к программе
- **Системный вызов** реализован в ядре ОС
 - Аргументы упаковываются и передаются ядру
 - Оно берет на себя управление программой, пока вызов не завершится
- В настоящее время в Linux свыше 300 СИСТЕМНЫХ ВЫЗОВОВ
- Их список находится в файле */usr/include/asm/unistd.h*



Утилита *strace*

- *\$ strace hostname*
- Вывод *hostname* смешивается с выводом *strace*
- Если запускаемая программа создает слишком много входных данных, то лучше перенаправить вывод в файл с помощью соответствующей опции

Очистка дисковых буферов

```
#include <fcntl.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

const char* g_JournalFilename = "journal.log";

void WriteJournalEntry(char* entry)
{
    int fd = open(g_JournalFilename
                  , O_WRONLY | O_CREAT | O_APPEND
                  , 0660
                  );
    write(fd, entry, strlen(entry));
    write(fd, "\n", 1);
    fsync(fd);
    close(fd);
}
```

Лимиты ресурсов

- Функции *getrlimit()* и *setrlimit()* позволяют процессу определять и задавать лимиты использования системных ресурсов
- Обе функции принимают два аргумента: код ограничения и указатель на структуру типа *rlimit*
- Функция *getrlimit()* заполняет поля этой структуры, а *setrlimit()* проверяет их и нужным образом изменяет лимиты
- У структуры *rlimit* два поля – *rlim_cur* со значением нежесткого лимита, в поле *rlim_max* – значение жесткого лимита



Лимиты ресурсов


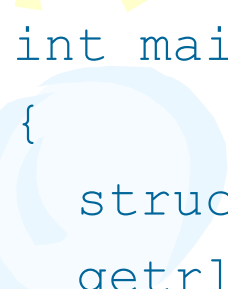
- *RLIMIT_CPU*. Это максимальный интервал времени процессора, задаваемый в секундах, занимаемый программой
- *RLIMIT_DATA*. Это максимальный объем памяти, который программа может запросить для данных
- *RLIMIT_NPROC*. Это максимальное число дочерних процессов, которые могут быть запущены пользователем
- *RLIMIT_NOFILE*. Это максимальное число файлов, которые могут быть одновременно открыты процессом



Лимиты ресурсов

```
#include <sys/resource.h>
#include <sys/time.h>
#include <unistd.h>

int main()
{
    struct rlimit rl;
    getrlimit(RLIMIT_CPU, &rl);
    rl.rlim_cur = 1;
    setrlimit(RLIMIT_CPU, &rl);
    while(!0);
    return 0;
}
```



Статистика процессов

- Функция *getrusage()* запрашивает у ОС статистику работы процессов
- Если первый аргумент этой функции равен *RUSAGE_SELF*, то процесс получит информацию о самом себе
- Если он равен *RUSAGE_CHILDREN*, то будет выдана информация обо всех завершенных дочерних процессах
- Вторым аргументом – это указатель на структуру *rusage*, в которую заносятся статистические данные

Статистика процессов

- Полезные поля структуры *rusage*:
 - *ru_utime*. Это структура типа *timeval*, в которой указано, сколько пользовательского времени в секундах ушло на выполнение процесса
 - *ru_stime*. Это структура типа *timeval*, в которой указано, сколько системного времени в секундах ушло на выполнение процесса
 - *ru_maxrss*. Это максимальный объем физической памяти, которую процесс занимал в любой момент времени своего выполнения



Статистика процессов

```
#include <stdio.h>
#include <sys/resource.h>
#include <sys/time.h>
#include <unistd.h>

void PrintCpuTime()
{
    struct rusage usage;
    getrusage(RUSAGE_SELF, &usage);
    printf("CPU time: %ld.%06ld sec user, %ld.%06ld sec system\n",
        usage.ru_utime.tv_sec, usage.ru_utime.tv_usec,
        usage.ru_stime.tv_sec, usage.ru_stime.tv_usec);
}
```

Системные часы

- Функция *gettimeofday()* определяет текущее системное время
- В качестве аргумента она принимает структура типа *timeval*, куда записывается значение времени в секундах, прошедшее с **начала эпохи**
- Это значение разделяется на два поля
 - В *tv_sec* хранится целое число секунд, а в поле *tv_usec* – дополнительное число микросекунд
- У функции есть второй аргумент, который всегда должен быть *NULL*
- Функция объявлена в *sys/time.h*

Системные часы

- Функция *localtime()* принимает указатель на число секунд и возвращает указатель на структуру типа *tm*
- Поля этой структуры
 - *tm_hour, tm_min, tm_sec* – текущее время
 - *tm_year, tm_mon, tm_day* – текущая дата
 - *tm_wday* – день недели, начиная с 0
 - *tm_yday* – день года
 - *tm_isdst* – учет летнего времени

Системные часы

- Функция *strftime()* на основании структуры *tm* создает строку, отформатированную по заданному правилу
- Формат аналогичен тому, что используется в функции *printf()*
- Указывается строка с кодами, определяющими включаемые структуры
- Пример форматной строки:
"%Y-%m-%d %H:%M:%S"
- Ей соответствует результат:

2014-03-10 04:54:18

Системные часы

```
#include <stdio.h>
#include <sys/time.h>
#include <time.h>
#include <unistd.h>
void PrintTime()
{
    struct timeval tv;
    struct tm* ptm;
    char timeString[40];
    long milliseconds;
    gettimeofday(&tv, NULL);
    ptm = localtime(&tv.tv_sec);
    strftime(timeString, sizeof (timeString)
             , "%Y-%m-%d %H:%M:%S", ptm
             );
    milliseconds = tv.tv_usec / 1000;
    printf("%s.%03ld\n", timeString, milliseconds);
}
```

Блокирование физической памяти

```
/* Allocate and lock memory */  
const int allocSize = 48 * 1024 * 1024;  
char* g_memory = malloc(allocSize);  
mlock(g_memory, allocSize);  
  
/* Initialize memory */  
size_t i;  
size_t pageSize= getpagesize();  
for (i = 0; i < allocSize; i += pageSize)  
    g_memory[i] = 0;
```

Блокирование физической памяти

- Для разблокирования области памяти вызывается функция *munlock()*, которой передаются те же аргументы, что и функции *mlock()*
- Функция *mlockall()* блокирует все адресное пространство программы и принимает единственный аргумент – флаг
 - *MCL_CURRENT* приводит к блокированию всей выделенной к настоящему моменту памяти, но не будущей
 - *MCL_FUTURE* приводит к блокированию всех страниц, выделенных после вызова функции *mlockall()*
- Функция *munlockall()* разблокирует всю память текущего процесса
- Функции семейства *mlock()* объявлены в файле *sys/mman.h*

Задание прав доступа к памяти

```
/* Map file to memory */  
int fd = open("/dev/zero", O_RDONLY);  
char* memory = mmap(NULL, pageSize, PROT_READ |  
    PROT_WRITE, MAP_PRIVATE, fd, 0);  
close(fd);  
  
/* Write is denied */  
mprotect(memory, pageSize, PROT_READ);
```

Задание прав доступа к памяти

```
#include <fcntl.h>
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
static int g_AllocSize;
static char* g_memory;

void SegvHandler(int signalNumber)
{
    printf("memory accessed!\n");
    mprotect(g_memory, g_AllocSize, PROT_READ |
        PROT_WRITE);
}
```

Задание прав доступа к памяти

```
int main()
{
    int fd;
    struct sigaction sa;

    memset(&sa, 0, sizeof (sa));
    sa.sa_handler = &SegvHandler;
    sigaction(SIGSEGV, &sa, NULL);

    g_AllocSize = getpagesize();
    fd = open("/dev/zero", O_RDONLY);
    g_memory = mmap(NULL, g_AllocSize, PROT_WRITE, MAP_PRIVATE, fd,
0);
    close (fd);
    g_memory[0] = 0;
    mprotect(g_memory, g_AllocSize, PROT_NONE);

    g_memory[0] = 1;

    printf("all done\n");
    munmap(g_memory, g_AllocSize);
    return 0;
}
```



Высокоточная пауза

- Функция *nanosleep()* принимает указатель на структуру типа *timespec*, где время задается с точностью до наносекунды
- В структуре *timespec* имеется два поля:
 - *tv_sec* – целое число секунд
 - *tv_nsec* – дополнительное число наносекунд
- Во втором аргументе функция принимает еще один указатель на *timespec*, в которую заносится величина оставшегося интервала времени



Высокоточная пауза

```
#include <errno.h>
#include <time.h>
int BetterSleep(double sleepTime)
{
    struct timespec tv;
    tv.tv_sec = (time_t) sleepTime;
    tv.tv_nsec = (long) ((sleepTime - tv.tv_sec) * 1e+9);

    while (!0)
    {
        int rval = nanosleep(&tv, &tv);
        if (0 == rval)
            return 0;
        else if (errno == EINTR)
            continue;
        else
            return rval;
    }
    return 0;
}
```

Быстрая передача данных

- Функции `sendfile()` передаются дескрипторы для записи и чтения, указатель на переменную смещения и число копируемых данных
- Переменная смещения определяет позицию во входном файле, с которой начинается копирование
- После окончания копирования переменная будет содержать смещение конца блока
- Функция объявлена в файле `sys/sendfile.h`

Быстрая передача данных

```
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/sendfile.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main(int argc, char* argv[])
{
    int read_fd;
    int write_fd;
    struct stat statBuf;
    off_t offset = 0;
    read_fd = open(argv[1], O_RDONLY);
    fstat(read_fd, &statBuf);
    write_fd = open(argv[2], O_WRONLY | O_CREAT, statBuf.st_mode);
    sendfile(write_fd, read_fd, &offset, statBuf.st_size);
    close (read_fd);
    close (write_fd);
    return 0;
}
```

Интервальные таймеры

- Функция *setitimer()* планирует доставку сигнала по истечении заданного промежутка времени
- Можно создавать таймеры трех типов:
 - *ITIMER_REAL*
 - *ITIMER_VIRTUAL*
 - *ITIMER_PROF*
- Учитывается время выполнения самого процесса, а также запускаемых им системных вызовов
- Тип таймера - в первом аргументе функции
- Второй аргумент – это указатель на структуру типа *itimerval*
- Третий аргумент – либо *NULL*, либо указатель на другую структуру типа *itimerval*

Интервальные таймеры

- В структуре *itimerval* два поля:
 - *it_value*. Это структура типа *timeval*, куда записано время отправки сигнала
 - *it_interval*. Это еще одна структура типа *timeval*, где определяется то, что произойдет после отправки первого сигнала

Интервальные таймеры

```
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <sys/time.h>

void TimerHandler(int signum)
{
    static int count = 0;
    printf("timer expired %d times\n", ++count);
}

int main()
{
    struct sigaction sa;
    struct itimerval timer;
    memset(&sa, 0, sizeof (sa));
    sa.sa_handler = &TimerHandler;
    sigaction(SIGVTALRM, &sa, NULL);
    timer.it_value.tv_sec = 0;
    timer.it_value.tv_usec = 250000;
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 250000;
    setitimer(ITIMER_VIRTUAL, &timer, NULL);

    while (!0)
        ;
}
```

Получение системной статистики

- Функция *sysinfo()* возвращает системную статистику
- Ее единственным аргументом является указатель на структуру типа *sysinfo*
- Полезные поля этой структуры:
 - *uptime*
 - *totalram*
 - *freeram*
 - *procs*
- Для использования данной функции требуется включить в программу файлы *linux/kernel.h*, *linux/sys.h*, *sys/sysinfo.h*

Получение системной статистики

```
#include <linux/kernel.h>
#include <linux/sys.h>
#include <stdio.h>
#include <sys/sysinfo.h>

int main()
{
    const long minute = 60;
    const long hour = minute * 60;
    const long day = hour * 24;
    const double megabyte = 1024 * 1024;
    struct sysinfo si;
    sysinfo(&si);
    printf("system uptime : %ld days, %ld:%02ld:%02ld\n",
        si.uptime / day, (si.uptime % day) / hour,
        (si.uptime % hour) / minute, si.uptime % minute);
    printf("total RAM      : %5.1f MB\n", si.totalram / megabyte);
    printf("free RAM       : %5.1f MB\n", si.freeram / megabyte);
    printf("process count : %d\n", si.procs);
    return 0;
}
```

Информация о системе

- Функция *uname()* возвращает информацию о системе, в частности сетевое и доменное имя компьютера, а также версию ядра ОС
- Единственный аргумент – указатель на структуру типа *utsname* с полями:
 - *sysname*
 - *release, version*
 - *machine*
 - *nodename*
 - *domain_name*
- Функция *uname()* объявлена в файле *sys/utsname.h*



Информация о системе

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

int main()
{
    struct utsname u;
    uname(&u);
    printf("%s release %s (version %s) on %s\n", u.sysname, u.release,
          u.version, u.machine);
    return 0;
}
```

See also

- Лав, Р. Linux. Системное программирование/ Р.Лав. – СПб.: Питер, 2008. – 416 с.
- Руководство программиста для Linux - http://citforum.ru/operating_systems/linux_pg/lpg_03.shtml
- Linux Syscall Reference - <http://syscalls.kernelgrok.com/>
- Tracing Tools - <http://doc.opensuse.org/products/draft/SLES>