Разработка программного обеспечения ОС UNIX

Процессы и сигналы

Процесс. Атрибуты процесса

Процесс – программа в стадии выполнения.
Процесс = адресное пространство + машинный код + атрибуты.

Атрибуты:
PID, PPID, UID, EUID, GID, …
Класс, приоритет, значение nice, …
Текущий и корневой каталог, переменные окружения, umask, …
Параметры обработки сигналов, …
Открытые файлы, …

Адресное пространство = TEXT + DATA (BSS + DYN) + STACK

Создание процесса

Новый процесс: fork() + exec().

fork – клонирование процесса

exec – замена существующего образом процесса из файла.

```
#include <unistd.h>
pid_t fork(void);
```

Пример:

```
pid_t p;
switch ( p = fork() ) {
        case -1:  ошибка…break;
        case 0:   код дочернего процесса; break;
        default:  printf("PID дочернего = %d\n", p );
}
```

Проблемы fork() и альтернативы

pid_t fork(void);

создаются копии страниц памяти для дочернего процесса

1) Предложение BSD

pid_t vfork(void);

страницы памяти родительского процесса связываются с дочерним

2) Предложение AT&T

pid_t fork(void);

страницы памяти родительского процесса помечаются как copy-on-write и связываются с дочерним.

Что меняется при fork()

PID,
PPID,
tms_utime, tms_stime, tms_cutime, and tms_cstime устанавливаются в 0.
Блокировки на файл,
    память,
    таймеры
    ожидаемые сигналы
                сбрасываются,
таймеры,
асинхронные операции ввода/вывода
                не наследуются.

Счетчики времени CPU устанавливаются в 0.

Процесс создается с одним потоком.

Дескрипторы открытых файлов,
текущий каталог копируются.

Системный вызов exec

```
#include <unistd.h>

    extern char **environ;
    int main (int argc, char *argv[]);

int execl( const char *path, const char *arg0, ... /*, (char *)0 */);

int execle( const char *path, const char *arg0, ... /*,(char *)0, char *const envp[]*/);

int execlp( const char *file, const char *arg0, ... /*, (char *)0 */ );

int execv( const char *path, char *const argv[] );

int execve( const char *path, char *const argv[], char *const envp[] );

int execvp( const char *file, char *const argv[] );

int fexecve( int fd, char *const argv[], char *const envp[] );
```

Что наследуется

Nice value
semadj values
Process ID
Parent process ID
Process group ID
Session membership
Real user ID
Real group ID
Supplementary group IDs
Time left until an alarm clock signal (см. alarm())
Current working directory
Root directory
File mode creation mask (см. umask())
File size limit (см. getrlimit() and setrlimit())
Process signal mask (см. pthread_sigmask())
Pending signal (см. sigpending())
tms_utime, tms_stime, tms_cutime, and tms_cstime (см. times())
Resource limits
Controlling terminal
Interval timers

Примеры exec

1.

```
int ret;
...
ret = execl ("/bin/ls", "ls", "-l", (char *)0);
```

2.

```
int ret;
char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
...
ret = execle ("/bin/ls", "ls", "-l", (char *)0, env);
```

3.

```
int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
...
ret = execve ("/bin/ls", cmd, env);
```

Совместно fork+exec

```
int fd1;
int fd2;
pid_t pid = fork();
if ( pid == -1 ) … Ошибка!
if ( pid == 0 ) {
    fd1 = open( “somefile.txt”, O_RDONLY );
    fd2 = open( “desiredfile.txt”, O_WRONLY|O_CREAT|O_TRUNC, 0644 );
    if ( fd1 == -1 ) … Ошибка!
    if ( fd2 == -1 ) … Ошибка!
    if ( dup2( fd1, STDIN_FILENO ) == -1 ) … Ошибка!
    if ( dup2( fd2, STDOUT_FILENO ) == -1 ) … Ошибка!
    close( fd1);
    close( fd2 );
    execlp( “sort”, “sort”, 0 );
    perror( … );
    _exit(1);
} else {
    …
```

Ожидание завершения процесса

```
#include <sys/wait.h>
pid_t wait(int *stat_loc);
pid_t waitpid(pid_t pid, int *stat_loc, int options);
options:
    WCONTINUED          WNOHANG        WUNTRACED

stat_loc:
0…6     номер сигнала, завершившего процесс
7       есть ли core
8…15    код завершения
WIFEXITED(stat_val)       WEXITSTATUS(stat_val)
WIFSIGNALED(stat_val)     WTERMSIG(stat_val)
WIFSTOPPED(stat_val)      WSTOPSIG(stat_val)
WIFCONTINUED(stat_val)

pid = wait( &status );
if ( WIFEXITED(status) ) printf("Код завершения процесса %d равен %d\n,
     pid, WEXITSTATUS(stat_val) );
```

Получение/установка ID-параметров процесса

```
#include <unistd.h>
pid_t getpid(void);
pid_t getppid(void);
pid_t getpgrp(void);                    get the process group ID of the calling process
int setpgid(pid_t pid, pid_t pgid);     set process group ID for job control
pid_t getsid(pid_t pid);        get the process group ID of a session leader
pid_t setsid(void);             create session and set process group ID

uid_t getuid(void);             get a real user ID
uid_t geteuid(void);            get the effective user ID
gid_t getgid(void);             get the real group ID
gid_t getegid(void);            get the effective group ID
int seteuid(uid_t uid);         set effective user ID
int setreuid(uid_t ruid, uid_t euid);   set real and effective user IDs
```

Ресурсы процесса

```
#include <sys/resource.h>
int getrlimit(int resource, struct rlimit *rlp);
int setrlimit(int resource, const struct rlimit *rlp);

struct rlimit {
    rlim_t rlim_cur;    The current (soft) limit.
    rlim_t rlim_max;  The hard limit.
};
```

RLIMIT_CORE	размер core
RLIMIT_CPU	время в с
RLIMIT_DATA	размер в байтах (malloc)
RLIMIT_FSIZE	размер файла в байтах
RLIMIT_NOFILE	число файлов
RLIMIT_STACK	размер стека
RLIMIT_AS	размер адресного пространства процесса

rlim_t: RLIM_INFINITY RLIM_SAVED_MAX RLIM_SAVED_CUR

Обработка сигналов

Способы реакции на сигнал:

- выполнить действие по умолчанию; SIG_DFL
- проигнорировать; SIG_IGN
- вызвать функцию, заданную пользователем;
- задержать, отложить реагирование.

Сигналы SIGKILL, SIGSTOP перехватить нельзя!

Для большинства действие по умолчанию – завершение процесса
(искл.: SIGUSR, SIGCHR, SIGPWR)

Послать сигнал:

```
#include <signal.h>
int kill(pid_t pid, int sig);
int raise(int sig);
```

Стандартные сигналы

Signal	Default Action	Description
SIGABRT	A	Process abort signal.
SIGALRM	T	Alarm clock.
SIGBUS	A	Access to an undefined portion of a memory object.
SIGCHLD	I	Child process terminated, stopped,
SIGCONT	C	Continue executing, if stopped.
SIGFPE	A	Erroneous arithmetic operation.
SIGHUP	T	Hangup.
SIGILL	A	Illegal instruction.
SIGINT	T	Terminal interrupt signal.
SIGKILL	T	Kill (cannot be caught or ignored).
SIGPIPE	T	Write on a pipe with no one to read it.
SIGQUIT	A	Terminal quit signal.
SIGSEGV	A	Invalid memory reference.
SIGSTOP	S	Stop executing (cannot be caught or ignored).
SIGTERM	T	Termination signal.
SIGTSTP	S	Terminal stop signal.
SIGTTIN	S	Background process attempting read.
SIGTTOU	S	Background process attempting write.

Стандартные сигналы

SIGUSR1	T	User-defined signal 1.
SIGUSR2	T	User-defined signal 2.
SIGPOLL	T	Pollable event.
SIGPROF	T	Profiling timer expired.
SIGSYS	A	Bad system call.
SIGTRAP	A	Trace/breakpoint trap.
SIGURG	I	High bandwidth data is available at a socket.
SIGVTALRM	T	Virtual timer expired.
SIGXCPU	A	CPU time limit exceeded.
SIGXFSZ	A	File size limit exceeded.

T Abnormal termination of the process.
A Abnormal termination of the process with additional actions.
I Ignore the signal.
S Stop the process.
C Continue the process, if it is stopped; otherwise, ignore the signal.

Установка пользовательской ловушки

```
#include <signal.h>
void (*signal(int sig, void (*func)(int)))(int);

void catcher( int signum ) {
    signal( signum, catcher );
    …
}
int main() {
    void (*old)(int));
    old = signal( SIGINT, catcher);
    signal( SIGTERM, SIG_IGN );
    signal( SIGSEGV, SIG_DFL );
    …
    signal( SIGINT, old );
}
```

```
void (*sigset(int sig, void (*disp)(int)))(int);
```

Управление сигналами

```
#include <signal.h>
int sighold(int sig);
int sigignore(int sig);
int sigpause(int sig);
int sigrelse(int sig);

int sigprocmask( int how, const sigset_t *set, sigset_t *oset);
                                  SIG_BLOCK       SIG_SETMASK   SIG_UNBLOCK
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signo);
int sigdelset(sigset_t *set, int signo);
int sigismember(const sigset_t *set, int signo);
int sigpending(sigset_t *set);          ожидают
int sigwait(const sigset_t *set, int *sig);
int sigsuspend(const sigset_t *sigmask);
```

Сигналы реального времени

SIGRTMIN … SIGRTMAX

Отличия сигналов реального времени от «обычных» сигналов:

1. Приоритет.
2. Счетчик количества в очереди.
3. Доставка данных вместе с сигналом.

```
#include <signal.h>

int sigqueue( pid_t pid, int signo, const union sigval value );

union sigval {
    int       sival_int;     // Integer signal value.
    void   *sival_ptr;    // Pointer signal value.
};
```

Обработка сигналов реального времени

```
#include <signal.h>

int sigaction( int sig, const struct sigaction *restrict act, struct sigaction *oldact);

struct sigaction {
    void   (*sa_handler)(int);          // Ловушка,  SIG_IGN или SIG_DFL.
    sigset_t        sa_mask;            // Какие сигналы блокировать.
    int             sa_flags;           // флаги
    void   (*sa_sigaction)(int, siginfo_t *, void *);      //Альтернативная ловушка
};
```

SA_RESETHAND
SA_RESTART
SA_SIGINFO

Обработчик сигнала реального времени

```
void   (*sa_sigaction)(int, siginfo_t *, void *);

struct siginfo_t {
    int         si_signo;       // Signal number.
    int         si_code;        // Signal code.
    int         si_errno;       // in <errno.h>.
    pid_t       si_pid;         // Sending process ID.
    uid_t       si_uid;         // Real user ID of sending process.
    void        *si_addr;       // Address of faulting instruction.
    int         si_status;      // Exit value or signal.
    long        si_band;        // Band event for SIGPOLL.
    union sigval  si_value;   // Signal value.
};
```

Code: SI_USER, SI_QUEUE, SI_TIMER, SI_ASYNCIO, SI_MESGQ

Таймеры с сигналом SIGALRM

```
#include <unistd.h>

unsigned alarm(unsigned seconds);

unsigned sleep(unsigned seconds);

#include <time.h>

int nanosleep( const struct timespec *rqtp,  struct timespec *rmtp );

struct timespec {
    time_t  tv_sec;    // Seconds.
    long    tv_nsec;   // Nanoseconds.
};
```

Таймеры SVR4

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

int getitimer( int which, struct itimerval *value );
int setitimer( int which, const struct itimerval *value,  struct itimerval *ovalue );

ITIMER_REAL             SIGALRM  - реальное время
ITIMER_VIRTUAL          SIGVTALRM – время в задаче пользователя
ITIMER_PROF             SIGPROF – пользов. время плюс время в ядре

struct itimerval {
    struct timeval    it_interval;        // Период срабатывания
    struct timeval    it_value;           // Время срабатывания
};
struct timeval {
    time_t            tv_sec;             // Секунды
    suseconds_t    tv_usec;               // Микросекунды
};
```

Таймеры POSIX

Всего TIMER_MAX

```
#include <signal.h>
#include <time.h>

int timer_create( clockid_t clockid, struct sigevent *evp, timer_t *timerid );
                    CLOCK_REALTIME (+CLOCK_VIRTUAL, CLOCK_PROF)

int timer_delete(timer_t timerid);

struct sigevent {
    int     sigev_notify;       // SIGEV_NONE, SIGEV_SIGNAL, SIGEV_THREAD
    int     sigev_signo;                    //сигнал
    union sigval    sigev_value;        //информация, передаваемая с сигналом
    void          (*sigev_notify_function)(union sigval);   //Функция оповещения
    pthread_attr_t *sigev_notify_attributes;                //Атрибуты потока
};
```

Управление таймерами POSIX

```
int timer_gettime( timer_t timerid, struct itimerspec *value );

int timer_settime( timer_t timerid,
                   int flags,  // TIMER_ABSTIME, 0 (TIMER_RELTIME)
                   const struct itimerspec *value,
                   struct itimerspec *ovalue);

struct itimerspec {
    struct timespec  it_interval;          //  Период срабатывания
    struct timespec  it_value;             //  Время срабатывания
};
struct timespec {
    time_t  tv_sec;    // Секунды
    long    tv_nsec;   // Наносекунды
};

int timer_getoverrun(timer_t timerid);
```

Ещё о часах и таймерах

```
#include <time.h>

int clock_getres( clockid_t clock_id, struct timespec *res );
int clock_gettime( clockid_t clock_id, struct timespec *tp );
int clock_settime( clockid_t clock_id, const struct timespec *tp );

int clock_nanosleep(clockid_t clock_id, int flags, const struct timespec *rqtp,
     struct timespec *rmtp);

int clock_getcpuclockid(pid_t pid, clockid_t *clock_id);
```

Асинхронный ввод/вывод

```
#include <aio.h>
struct aiocb {
    int             aio_fildes;         //дескриптор файла
    off_t           aio_offset;         //позиция
    volatile void   *aio_buf;           //буфер
    size_t          aio_nbytes;         //сколько байт
    int             aio_reqprio;        //приоритет
    struct sigevent aio_sigevent;       //как узнать о завершении
    int             aio_lio_opcode;     //какую операцию выполняем (для lio)
};
struct sigevent {
    int     sigev_notify;       // SIGEV_NONE, SIGEV_SIGNAL, SIGEV_THREAD
    int     sigev_signo;                //сигнал
    union sigval    sigev_value;        //информация, передаваемая с сигналом
    void        (*sigev_notify_function)(union sigval);   //Функция оповещения
    pthread_attr_t *sigev_notify_attributes;              //Атрибуты потока
};
```

Функции асинхронного ввода/вывода

```
int  aio_read(struct aiocb *aiocbp);

int aio_write(struct aiocb *aiocbp);

int lio_listio( int mode,                    // LIO_WAIT или LIO_NOWAIT
            struct aiocb *const list[ ],
            int nent,
            struct sigevent *sig);           // LIO_READ, LIO_WRITE, LIO_NOP;

int      aio_cancel(int, struct aiocb *);
     // возвращает AIO_CANCELED, AIO_CANCELED, AIO_CANCELED
int aio_error(const struct aiocb *aiocbp); //0, код ошибки или EINPROGRESS

int aio_fsync(int op, struct aiocb *); //op: O_DSYNC, O_SYNC (целостность)

ssize_t  aio_return(struct aiocb *); //1 раз возвращает результат aio_read и др.

int aio_suspend(const struct aiocb *const list[], // Приостанавливает поток до
     int nent,                                    // сигнала, или
     const struct timespec *timeout);             // завершения операции, или
                                                  // истечения времени
```

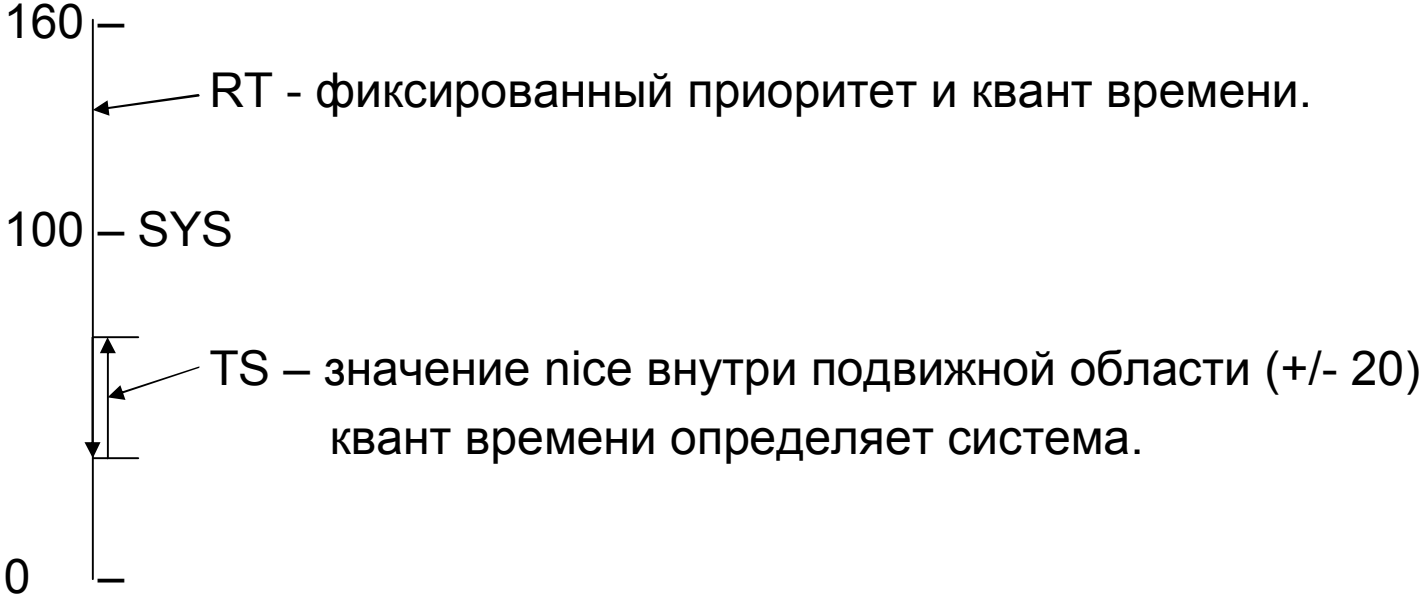
Поддержка систем реального времени

Система реального времени:

- контроль над порядком исполнения процессов;
- гарантированное время выполнения системного вызова.

Диспетчер.

Классы процессов и внутренние приоритеты.



Задание класса и параметров процесса

```
#include <sys/types.h>
#include <sys/priocntl.h>
#include <sys/rtpriocntl.h>
#include <sys/tspriocntl.h>
#include <sys/fsspriocntl.h>
#include <sys/fxpriocntl.h>

long priocntl(idtype_t idtype, id_t id, int cmd,/*arg*/ ...);
```

idtype: P_ALL, P_PID, P_UID, P_PPID, P_CID и др.

– как трактовать id.

cmd – команда, определяет тип аргумента.

PC_GETCID – запрос идентификатора класса

PC_GETCLINFO – запрос параметров (характеристик) класса

Параметры классов

<sys/priocntl.h>,

Для команд PC_GETCID и PC_GETCLINFO

```
struct pcinfo_t {
    id_t pc_cid;                        /* Class id */
    char pc_clname[PC_CLNMSZ];  /* Class name */
    int pc_clinfo[PC_CLINFOSZ];     /* Class information */
};
```

PC_GETCID → pc_clname содержит имя класса: “TS”, “RT”, “FX”,…

PC_GETCLINFO → pc_clinfo (для каждого класса своя структура)

RT → rtinfo_t { short rt_maxpri; } /* Maximum realtime priority */

TS → tsinfo_t { short ts_maxupri; } /* Limits of user priority range */

FSS → fssinfo_t {short fss_maxupri;} /* Limits of user priority range */

FX → fxinfo_t {pri_t fx_maxupri; } /* Maximum user priority */

Запрос/назначение характеристик процесса

Команды PC_GETPARMS / PC_SETPARMS

```
struct pc_parms_t {
    id_t pc_cid; /* LWP class, если PC_CLNULL – вернет о LWP */
    int pc_clparms[PC_CLPARMSZ]; /* Class-specific params */
};
```

RT → <sys/rtpriocntl.h>

```
struct rtparms_t {
    short rt_pri;           /* Real-Time priority */
    uint_t rt_tqsecs;       /* Seconds in time quantum */
    int rt_tqnsecs;         /* Additional nanoseconds in quantum */
};
```

RT_TQINF RT_TQDEF RT_NOCHANGE

Запрос/назначение характеристик процесса

TS → <sys/tspriocntl.h>

```
struct tsparms_t {
    short ts_uprilim;       /* Time-Sharing user priority limit */
    short ts_upri;          /* Time-Sharing user priority */
};
```

FSS → <sys/fsspriocntl.h>

```
struct fssparms_t {
    short fss_uprilim;      /* Fair-share user priority limit */
    short fss_upri;         /* Fair-share user priority */
};
```

FX → <sys/fxpriocntl.h>

```
struct {
    pri_t fx_upri;          /* Fixed-priority user priority */
    pri_t fx_uprilim;       /* Fixed-priority user priority limit */
    uint_t fx_tqsecs;       /* seconds in time quantum */
    int fx_tqnsecs;         /* additional nanosecs in time quant */
};
```

Альтернативный интерфейс задания параметров

Команда PC_SETXPARMS затем ключ и параметр.

Примеры ключей:

RT_KY_PRI
RT_KY_TQSECS
RT_KY_TQNSECS

TS_KY_UPRILIM
TS_KY_UPRI

FX_KY_UPRILIM
FX_KY_UPRI
FX_KY_TQSECS
FX_KY_TQNSECS

Управление страницами памяти

```
#include <sys/types.h>
#include <sys/mman.h>
  int     memcntl(  caddr_t  addr,
                    size_t   len,
                    int      cmd,
                    caddr_t  arg,
                    int      attr,
                    int      mask);
```

cmd: MC_LOCK MC_LOCKAS
MC_UNLOCK MC_UNLOCKAS
MC_SYNC

arg: MS_ASYNC MS_SYNC MS_INVALIDATE
MCL_CURRENT MCL_FUTURE

attr: фильтр SHARED PRIVATE
PROT_READ PROT_WRITE PROT_EXEC
PROC_TEXT PROC_DATA или 0

Упрощенный интерфейс манипулирования памятью

```
#include <sys/mman.h>
int mlock(const void * addr, size_t len);
int munlock(const void * addr, size_t len);
int mlockall(int flags);
int munlockall(void);
```

Реализация:

```
memcntl( addr, len, MC_LOCK, 0, 0, 0 );
memcntl( addr, len, MC_UNLOCK, 0, 0, 0 );

memcntl( 0, 0, MC_LOCKAS, flags, 0, 0 );
memcntl( 0, 0, MC_UNLOCKAS, flags, 0, 0 );
```

Пример программы – стр. 1

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/priocntl.h>
#include <sys/mman.h>
#include <sys/time.h>
#include <signal.h>
#include <string.h>

static void trap_func( int sig ) {
     printf(“\a”);
}
int main() {
     pcinfo_t pcinfo;
     pcparms_t pcparms;
     struct sigaction sigact;
     struct itimerval itimerval;
     int count  = 0;
```

Пример программы – стр. 2

```
    strcpy( pcinfo.pc_clname, “RT” );
    priocntl( 0, 0, PC_GETCID, &pcinfo );
    pcparms.pc_cid = pcinfo.pc_cid;
    ((rtparms_t*)pcparms.pc_clparms)->rt_pri=((rtinfo_t*)pcinfo.pc_clinfo)->rt_maxpri;
    ((rtparms_t*)pcparms.pc_clparms)->rt_tqsecs = RT_TQINF;
    priocntl( P_PID, P_MYID, PC_SETPARMS, &pcparms );
    memcntl( 0, 0, MC_LOCKAS, MCL_CURRENT | MCL_FUTURE, 0, 0 );
    sigact.sa_handler = trap_func;
    sigfillset( &sigact.sa_mask );
    sigact.sa_flags = 0;
    sigaction( SIGPROF, &sigact, 0 );
    itimerval.it_interval.tv_sec = 5;    itimerval.it_interval.tv_usec = 0;
    itimerval.it_value.tv_sec = 0;       itimerval.it_value.tv_usec = 500;
    setitimer( ITIMER_PROF, &itimerval, NULL );
    while( 1 ) { ++count; if ( count > 1000000000 ) count = 0; }
}
```