Разработка программного обеспечения ОС UNIX

Потоки и их синхронизация

Потоки и легковесные процессы

Поток – thread – команды, выполняемые последовательно.

Преимущества потоков:

- использование многопроцессорных/многоядерных систем;
- исполнение программ при блокировке ресурсов (в др. потоке);
- структурирование кода по выполняемым задачам (ООП);
- экономия на переключении контекста.

Особенности:

- реализация с использованием библиотечных функций, меньше переключений контекста;
- для распределения процессорных ресурсов используется LWP (LWP – light weight process).
- общее адресное пространство (обмен данными проще, синхронизация);
- собственные стек, набор регистров, сигнальная маска, приоритет, специальная память;
- exit завершает весь процесс, а не только вызвавший поток.

Потоки POSIX. Создание потока

```
#include <pthread.h>

int pthread_create( pthread_t  *thread,
                    const pthread_attr_t  *attr,
                    void *(*start_routine)(void*),
                    void *arg);
```

Возвращает 0 или код ошибки (для всех функций семейства pthread_...).

Атрибуты attr. NULL – по умолчанию.

Создание/освобождение аттрибутного объекта:

```
int pthread_attr_init(pthread_attr_t *attr);  // Устанавливает атрибуты по
                                                 умолчанию.
int pthread_attr_destroy(pthread_attr_t *attr);
```

Установка особых атрибутов потока

Размер стека:

```
int pthread_attr_getstacksize( const pthread_attr_t *attr, size_t *stacksize);
int pthread_attr_setstacksize( pthread_attr_t *attr, size_t stacksize );
```

Атрибуты стека:

```
int pthread_attr_getstack( const pthread_attr_t *attr, void **stackaddr,
                                                        size_t *stacksize );
int pthread_attr_setstack( pthread_attr_t *attr, void *stackaddr, size_t stacksize );
```

Размер области контроля за переполнением стека:

```
int pthread_attr_getguardsize( const pthread_attr_t *attr, size_t *guardsize );
int pthread_attr_setguardsize( pthread_attr_t *attr, size_t guardsize);
```

Область видимости:

```
PTHREAD_SCOPE_SYSTEM,          PTHREAD_SCOPE_PROCESS
int pthread_attr_getscope( const pthread_attr_t *attr,  int *contentionscope );
int pthread_attr_setscope( pthread_attr_t *attr,  int contentionscope );
```

Атрибуты потоков – продолжение

```
#include <sched.h>
```

Алгоритм диспетчеризации:

```
    SCHED_FIFO,    SCHED_RR,  SCHED_OTHER,    [SCHED_SPORADIC]
    int pthread_attr_getschedpolicy( const pthread_attr_t *attr, int *policy );
    int pthread_attr_setschedpolicy( pthread_attr_t *attr, int policy );
```

Параметры диспетчеризации:

```
    int pthread_attr_getschedparam( const pthread_attr_t *attr,
                                    struct sched_param *param );
    int pthread_attr_setschedparam( pthread_attr_t *attr,
                                    const struct sched_param *param);
struct sched_param {
    int             sched_priority;         // Приоритет
    int             sched_ss_low_priority;  // Для SCHED_SPORADIC
    struct timespec sched_ss_repl_period;   // Необязательны!
    struct timespec sched_ss_init_budget;
    int             sched_ss_max_repl;
};
```

Атрибуты потоков – окончание

Параметры диспетчера:

```
#include <sched.h>
int sched_get_priority_max(int policy);
int sched_get_priority_min(int policy);
int sched_rr_get_interval(pid_t pid, struct timespec *interval);
```

Атрибуты наследования параметров диспетчеризации:

PTHREAD_INHERIT_SCHED – наследуется от потока-родителя;

PTHREAD_EXPLICIT_SCHED – задаётся точно.

```
int pthread_attr_getinheritsched( const pthread_attr_t *attr, int *inheritsched);
int pthread_attr_setinheritsched( pthread_attr_t *attr, int inheritsched );
```

Состояние присоединенности потока:

PTHREAD_CREATE_DETACHED, PTHREAD_CREATE_JOINABLE

```
int pthread_attr_getdetachstate(const pthread_attr_t *attr, int *detachstate);
int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
```

Завершение потока

Завершение:

```
void pthread_exit(void *value_ptr);
```

Ожидание:

```
int pthread_join(pthread_t thread, void **value_ptr);
```

Передача управления другому потоку:

```
#include <sched.h>      int sched_yield(void);
```

Отмена потока:

```
int pthread_cancel(pthread_t thread);

int pthread_setcancelstate(int state, int *oldstate);
                                PTHREAD_CANCEL_ENABLE,
                                PTHREAD_CANCEL_DISABLE
int pthread_setcanceltype(int type, int *oldtype);
                                PTHREAD_CANCEL_DEFERRED,
                                PTHREAD_CANCEL_ASYNCHRONOUS
```

Проверить точку отмены:

```
void pthread_testcancel(void);
```

Обработчики потоков

Обработчики отмены:

```
void pthread_cleanup_pop( int execute ); //1 – выполнить, убирает с
                                                вершины стека
void pthread_cleanup_push( void (*routine)(void*), void *arg );
```

Обработчики вызова fork() (до и после вызова)

```
int pthread_atfork(
                void (*prepare)(void), void (*parent)(void), void (*child)(void));
```

Функция-инициализатор

```
int pthread_once(pthread_once_t *once_control, void (*init_routine)(void));
pthread_once_t once_control = PTHREAD_ONCE_INIT;
```

Уровень параллелизма:

```
int pthread_getconcurrency(void);
int pthread_setconcurrency(int new_level);
```

Идентификатор потока:

```
pthread_t pthread_self(void);
int pthread_equal( pthread_t t1, pthread_t t2 );
```

Данные потоков

Создание ключа:

```
int pthread_key_create( pthread_key_t *key, void (*destructor)(void*) );
```

Работа с данными:

```
void *pthread_getspecific(pthread_key_t key);
int pthread_setspecific(pthread_key_t key, const void *value);
```

Очистка:

```
int pthread_key_delete(pthread_key_t key);
```

Пример:

```
static pthread_key_t key;        static pthread_once_t key_once = PTHREAD_ONCE_INIT;
static void make_key() {     (void) pthread_key_create(&key, NULL); }
void func() {    void *ptr;
        (void) pthread_once( &key_once, make_key );
        if ((ptr = pthread_getspecific(key)) == NULL) {
                ptr = malloc(OBJECT_SIZE);        (void) pthread_setspecific(key, ptr);
        }
}
Thread_func() {
    int *ptr; func(); ptr = (int*)pthread_getspecific(key)); ptr[0] = …; …
}
```

Потоки и сигналы

#include <signal.h>

Отправка сигнала:

int pthread_kill(pthread_t thread, int sig);

Действие по умолчанию (остановка или завершение процесса)
распространяется на весь процесс!

Установка сигнальной маски для потока:

int pthread_sigmask(int how, const sigset_t *set, sigset_t *oset);
SIG_BLOCK SIG_SETMASK SIG_UNBLOCK

Потоки Sun Microsystems

```
#include <thread.h>
int thr_create( void *stack_base,  size_t stack_size,  void *(*start_func) (void*),
            void *arg, long flags,  thread_t *new_thread_ID);
```

THR_BOUND

THR_DETACHED

THR_NEW_LWP

THR_SUSPENDED

THR_DAEMON

```
int thr_suspend(thread_t target_thread);
int thr_continue(thread_t target_thread);
void thr_yield(void);
void thr_exit(void *status);
int thr_join(thread_t thread, thread_t *departed, void **status);
int thr_setprio(thread_t target_thread, int priority);
int thr_getprio(thread_t target_thread, int *priority);
```

Потоки Sun – окончание

```
int thr_kill(thread_t thread, int sig);
int thr_sigsetmask(int how, const sigset_t *set, sigset_t *oset);

int thr_main(void); // 0/-1 первичный ли поток?
int thr_stksegment(stack_t *ss);

int thr_setconcurrency(int new_level);
int thr_getconcurrency(void);

int thr_keycreate(thread_key_t *keyp, void (*destructor)(void *));
int thr_setspecific(thread_key_t key, void *value);
int thr_getspecific(thread_key_t key, void **valuep);

size_t thr_min_stack(void);
thread_t thr_self(void);
```

Синхронизация потоков POSIX. Мьютексы

```
#include <pthread.h>
```

Создание:

```
int pthread_mutex_init(
                    pthread_mutex_t *mutex,  const pthread_mutexattr_t  *attr);
```

Создание со статическим инициализатором:

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

Разрушение мьютекса

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Атрибуты мьютекса:

```
int pthread_mutexattr_init(pthread_mutexattr_t *attr);
int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
```

Атрибуты мьютексов

Где действует мьютекс?

int pthread_mutexattr_getpshared(const pthread_mutexattr_t *attr, int *pshared);

int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr, int pshared);

PTHREAD_PROCESS_SHARED или PTHREAD_PROCESS_PRIVATE

Как работает мьютекс?

int pthread_mutexattr_gettype(const pthread_mutexattr_t *attr, int *type);

int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);

Типы:

PTHREAD_MUTEX_NORMAL не контролирует повторный захват

PTHREAD_MUTEX_ERRORCHECK повторный захват приводит к ошибке

PTHREAD_MUTEX_RECURSIVE считает число захватов

PTHREAD_MUTEX_DEFAULT повторно захватывать нельзя

int pthread_mutexattr_getrobust(const pthread_mutexattr_t *attr, int *robust);

int pthread_mutexattr_setrobust(pthread_mutexattr_t *attr, int robust);

PTHREAD_MUTEX_STALLED, PTHREAD_MUTEX_ROBUST

Работа с мьютексом.

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);

---------- Мьютексы Sun Microsystems ----------
#include <thread.h>
#include <synch.h>
int mutex_init(mutex_t *mp, int type, void * arg);
        USYNC_THREAD,  USYNC_PROCESS,
        LOCK_ROBUST  LOCK_RECURSIVE, LOCK_ERRORCHECK,
        LOCK_PRIO_INHERIT, LOCK_PRIO_PROTECT
или             mutex_t mp = DEFAULTMUTEX;
int mutex_lock(mutex_t *mp);
int mutex_trylock(mutex_t *mp);
int mutex_unlock(mutex_t *mp);
int mutex_consistent(mutex_t *mp);
int mutex_destroy(mutex_t *mp);
```

Условные переменные POSIX

```
#include <pthread.h>

int pthread_cond_init( pthread_cond_t *cond, const pthread_condattr_t *attr);
    или
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

int pthread_cond_destroy( pthread_cond_t *cond );
```

Атрибуты:

```
    int pthread_condattr_init( pthread_condattr_t *attr );
    int pthread_condattr_destroy( pthread_condattr_t *attr );
```

Область видимости:

```
int pthread_condattr_getpshared( const pthread_condattr_t *attr, int *pshared);
int pthread_condattr_setpshared(pthread_condattr_t *attr, int pshared);
    PTHREAD_PROCESS_PRIVATE,          PTHREAD_PROCESS_SHARED
```

Условные переменные POSIX - управление

```
int pthread_condattr_getclock( const pthread_condattr_t *attr, clockid_t *clock_id );
int pthread_condattr_setclock( pthread_condattr_t *attr, clockid_t clock_id );
```

Сигнал:

```
int pthread_cond_broadcast(pthread_cond_t *cond);
int pthread_cond_signal(pthread_cond_t *cond);
```

Ожидание:

```
int pthread_cond_wait( pthread_cond_t *cond,  pthread_mutex_t *mutex );
int pthread_cond_timedwait( pthread_cond_t *cond, pthread_mutex_t *mutex,
                                                  const struct timespec *abstime);
```

Задача мьютекса – обеспечить монопольный доступ к условию.

Когда вызывается …wait…, снимается блокировка с мьютекса на время ожидания, но при возврате из функции – восстанавливается!

Схема применения условных переменных

```
struct {
      pthread_mutex_t   m;
      pthread_cjnd_t      c;
      int                       f;
      int                       data[DATASIZE];
} v = {PTHREAD_MUTEX_INITIALIZER, PTHREAD_COND_INITIALIZER, 0,1,2,…};
```

Производитель:

```
…
pthread_mutex_lock( &v.m );
v.data[0] = 123;
v.f = 1;
pthread_cond_signal( &v.c );
pthread_mutex_unlock( &v.m );
```

Потребитель:

```
…
pthread_mutex_lock( &v.m );
while ( !v.f ) pthread_cond_wait( &v.c, &v.m );
Process_data( &v.data[0] );     //Обработка данных
v.f=0;
pthread_mutex_unlock( &v.m );
```

Условные переменные Sun Microsystems

```
#include <thread.h>
#include <synch.h>

int cond_init( cond_t *cvp, int type, void *arg );
                        USYNC_THREAD (по умолчанию)
                        USYNC_PROCESS
            или
    cond_t cond = DEFAULTCV;

int cond_wait( cond_t *cvp, mutex_t *mp );
int cond_timedwait( cond_t *cvp, mutex_t *mp, timestruc_t *abstime );
int cond_reltimedwait( cond_t *cvp, mutex_t *mp, timestruc_t *reltime );

int cond_signal( cond_t *cvp );
int cond_broadcast( cond_t *cvp );

int cond_destroy( cond_t *cvp );
```

Блокировки чтения-записи POSIX

```
#include <pthread.h>

int pthread_rwlock_init( pthread_rwlock_t *rwlock, const pthread_rwlockattr_t *attr );
```

или

```
pthread_rwlock_t rwlock = PTHREAD_RWLOCK_INITIALIZER;
```

Завершение работы с блокировкой

```
int pthread_rwlock_destroy( pthread_rwlock_t *rwlock );
```

Атрибуты блокировки:

```
int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);

int pthread_rwlockattr_getpshared(const pthread_rwlockattr_t *attr, int *pshared);
int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr, int pshared);
```

PTHREAD_PROCESS_SHARED, PTHREAD_PROCESS_PRIVATE

```
int pthread_rwlockattr_destroy(pthread_rwlockattr_t *attr);
```

Установка/снятие блокировки

На чтение:

```
int pthread_rwlock_rdlock( pthread_rwlock_t *rwlock );
int pthread_rwlock_tryrdlock( pthread_rwlock_t *rwlock );
int pthread_rwlock_timedrdlock( pthread_rwlock_t *rwlock,
                                const struct timespec *abstime );
```

На запись:

```
int pthread_rwlock_timedwrlock( pthread_rwlock_t *rwlock,
                                const struct timespec *abstime );
int pthread_rwlock_trywrlock( pthread_rwlock_t *rwlock );
int pthread_rwlock_wrlock( pthread_rwlock_t *rwlock );
```

Снятие блокировки:

```
int pthread_rwlock_unlock( pthread_rwlock_t *rwlock );
```

Блокировки чтения-записи Sun

```
#include <synch.h>

int rwlock_init(rwlock_t *rwlp, int type, void * arg);
                                    USYNC_PROCESS
                                    USYNC_THREAD
    или
rwlock_t rwlp = DEFAULTRWLOCK;

int rw_rdlock(rwlock_t *rwlp);
int rw_wrlock(rwlock_t *rwlp);
int rw_tryrdlock(rwlock_t *rwlp);
int rw_trywrlock(rwlock_t *rwlp);

int rw_unlock(rwlock_t *rwlp);
int rwlock_destroy(rwlock_t *rwlp);
```

Семафоры Sun Microsystems для потоков Solaris

```
#include <synch.h>

int sema_init( sema_t *sp, unsigned int count, int type, void * arg);
                                                    USYNC_PROCESS
                                                    USYNC_THREAD

int sema_wait(sema_t *sp);
int sema_trywait(sema_t *sp);
int sema_post(sema_t *sp);

int sema_destroy(sema_t *sp);
```