



5. Synchronization (4)

5. Synchronization

6 5.3 Synchr. methods

5.4 Linux

Messages (1)

- Message transport through two system calls
 - send (destination, &message);
 - receive (source, &message);
 - Synchronous communication:
threads block when *send* or *receive* cannot be executed immediately, e.g. because
 - the other part has not issued the corresponding call,
 - a buffer for the messages is full or empty.

Messages (2)

- **Advantage:** also works on system without shared memory (distributed systems, client server computing)
 - **Disadvantages:**
 - duplication of data is overhead
 - necessary to remember names of source and target
 - something must be done about possible loss of messages
 - Implementation uses pipes, mailslots (Windows) or RPCs.

Messages (3)

- **synchronous vs. asynchronous**
 - synchronous: *send / receive* block until corresponding operation on the other side has finished
 - asynchronous: *send* call returns immediately; success of sending might be checkable, e.g.:
 - other side sends an explicit ACK (new message)
 - messaging system sends signal on message delivery
 - **connection oriented vs. not conn. oriented**
 - connection oriented: permanent connection (TCP)
 - connectionless (cf. UDP)

Examples for messages (1)

Two processes toggle access rights

```
void function (int id) {
    int otherid = 1 - id;
    char message[10] = "";
    // one process may first, ID 0
    if (id==0) {
        message = "go";
    }
    // non-critical block
    while message != "go" {
        receive (otherid, &message);
    }
    // critical region
    send (otherid, "go");
    message = "";
    // further non-critical block
}
```

p0 calls *function(0)*,
p1 calls *function(1)*.

p0 can enter critical
region first

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-04

5. Synchronization (4) – Slide 5

Examples for messages (3)

Client requests access grant from the server

```
---- client -----
void function (int id) {
    reply = "";
    // non-critical block
    while (reply != "go") {
        send (server, "request");
        receive (server, &reply); // wait for grant
    }
    // critical section
    send (server, "release"); // release access
    // further non-critical block
}
```

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-04

5. Synchronization (4) – Slide 7

Examples for messages (2)

Server process grants resource access

```
---- server -----
int busy; // globale state

void main () {
    initialize ();
    while TRUE {
        receive (&id, &message);
        // message and also sender ID
        // (id) is known
        switch (message) {
            "request": enter_in_queue(id);
            "release": let_next_one();
        };
    };
}
```

```
void let_next_one () {
    if queue_is_empty () { busy = FALSE; }
    else {
        id = queue_head.id;
        send (&id, "go");
        queue_head = queue_head.next;
    }
}

void enter_in_queue (int id) {
    if queue_is_empty () and busy==FALSE {
        busy = TRUE;
        send (&id, "go")
    } else {
        allocate (&newentry);
        newentry.id = id;
        newentry.next = NULL;
        queue_last.next = newentry;
        queue_last = newentry;
    }
}
```

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-04

5. Synchronization (4) – Slide 6

Examples for messages (4)

Process 1

```
send(server,"request"); → receive(1,"request");
                                enter_in_queue(1);
                                /* busy==false, queue
                                 * is empty */
receive(server,"go"); ← send(1,"go");
busy = TRUE;
```

/* critical region */

```
send(server,"release"); → receive(1,"release");
...
```

```
let_next_one();
id = queue_head /* 2 */;
send(2,"go"); → receive(server,"go");
queue = [ ];
```

```
receive(2,"release"); ← send(server,"release");
let_next_one();
/* queue is empty */
busy = FALSE;
```

Server

```
enter_in_queue(1);
/* busy==false, queue
 * is empty */
send(1,"go");
busy = TRUE;
```

```
receive(2,"request"); ← send(server,"request");
enter_in_queue(2);
/* busy==true */
queue = [ 2 ];
```

Process 2

```
receive(1,"request");
enter_in_queue(1);
/* busy==true */
queue = [ 1 ];
send(1,"go"); → receive(server,"go");
/* critical region */

receive(2,"request"); ← send(server,"request");
enter_in_queue(2);
/* busy==true */
queue = [ 2 ];
```

5. Synchronization (4) – Slide 8

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-04

Messages

- broadcast to several (all) processes possible
- distributed systems: Voting algorithms (using broadcast) for drawing decisions

more about messages:

Chapter 6: IPC (Inter Process Communication)

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-04

5. Synchronization (4) – Slide 9

```
Sep 19 14:27:41 amd64 syslog-ng(7653): STATS: dropped 0
Sep 20 01:00:01 amd64 /usr/sbin/cron(29278): (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 20 01:00:01 amd64 syslog-ng(7653): STATS: dropped 0
Sep 20 01:00:01 amd64 /usr/sbin/cron(29278): (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 20 02:00:01 amd64 syslog-ng(7653): STATS: dropped 0
Sep 20 12:46:44 amd64 sshd[6516]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62004
Sep 20 12:46:44 amd64 syslog-ng(7653): STATS: dropped 0
Sep 20 12:48:43 amd64 sshd[66091]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:48:43 amd64 syslog-ng(7653): STATS: dropped 0
Sep 20 15:27:35 amd64 sshd[90771]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 20 15:27:35 amd64 syslog-ng(7653): STATS: dropped 0
Sep 20 16:38:10 amd64 sshd[10140]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63546
Sep 21 01:00:01 amd64 /usr/sbin/cron(17055): (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 21 01:00:01 amd64 syslog-ng(7653): STATS: dropped 0
Sep 21 02:00:01 amd64 /usr/sbin/cron(17878): (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 21 02:00:01 amd64 syslog-ng(7653): STATS: dropped 0
Sep 21 17:43:26 amd64 sshd[31269]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63397
Sep 21 17:43:26 amd64 syslog-ng(7653): STATS: dropped 0
Sep 21 17:53:39 amd64 sshd[31269]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64391
Sep 21 18:43:26 amd64 syslog-ng(7653): STATS: dropped 0
Sep 22 01:00:01 amd64 /usr/sbin/cron(24739): (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 22 01:00:01 amd64 syslog-ng(7653): STATS: dropped 0
Sep 22 01:00:01 amd64 sshd[46741]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 22 01:00:01 amd64 syslog-ng(7653): STATS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron(54991): (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 22 02:00:01 amd64 syslog-ng(7653): STATS: dropped 0
Sep 23 01:00:01 amd64 /usr/sbin/cron(12436): (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 23 01:00:01 amd64 syslog-ng(7653): STATS: dropped 0
Sep 23 02:00:01 amd64 /usr/sbin/cron(25555): (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 23 02:00:01 amd64 syslog-ng(7653): STATS: dropped 0
Sep 23 18:04:05 amd64 sshd[65164]: Accepted pubkey for esser from ::ffff:87.234.201.207 port 61330
Sep 23 18:04:05 amd64 syslog-ng(7653): STATS: dropped 0
Sep 23 18:04:34 amd64 sshd[66061]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61330
Sep 24 01:00:01 amd64 /usr/sbin/cron(12436): (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 24 01:00:01 amd64 syslog-ng(7653): STATS: dropped 0
Sep 24 02:00:01 amd64 syslog-ng(7653): STATS: dropped 0
Sep 24 02:00:01 amd64 /usr/sbin/cron(13253): (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 24 02:00:01 amd64 syslog-ng(7653): STATS: dropped 0
Sep 24 11:15:48 amd64 sshd[20998]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64456
Sep 24 11:15:48 amd64 syslog-ng(7653): STATS: dropped 0
Sep 24 13:49:08 amd64 sshd[23191]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61330
Sep 24 13:49:08 amd64 syslog-ng(7653): STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_midi_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 syslog-ng(7653): STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_midi_event: unsupported module, tainting kernel.
Sep 24 20:25:31 amd64 sshd[29399]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62566
Sep 24 20:25:31 amd64 syslog-ng(7653): STATS: dropped 0
Sep 25 01:00:02 amd64 /usr/sbin/cron(6621): (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 25 01:00:02 amd64 syslog-ng(7653): STATS: dropped 0
Sep 25 01:00:02 amd64 sshd[11481]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 25 02:00:02 amd64 syslog-ng(7653): STATS: dropped 0
Sep 25 10:59:25 amd64 sshd[8899]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64188
Sep 25 10:59:25 amd64 syslog-ng(7653): STATS: dropped 0
Sep 25 10:59:25 amd64 sshd[11481]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:30:02 amd64 sshd[9372]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62029
Sep 25 11:39:25 amd64 syslog-ng(7653): STATS: dropped 0
Sep 25 14:05:37 amd64 sshd[11554]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62822
Sep 25 14:05:37 amd64 syslog-ng(7653): STATS: dropped 0
Sep 25 14:06:16 amd64 sshd[11608]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sshd[11608]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:33 amd64 sshd[11630]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63709
Sep 25 15:25:33 amd64 sshd[12930]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62778
```

5.4 Synchronization on Unix / Linux

Content overview

5.4.1 Synchronization in applications

- POSIX Threads
- Synchroniz. between processes

5.4.2 Synchronization in the Linux kernel

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-04

5. Synchronization (4) – Slide 11

POSIX Threads

POSIX threads can use several standard synchronization primitives:

- Mutexes
- Semaphores
- Condition Variables

Hans-Georg Eßer, FH München

Operating Systems I, WS 2006/07 – 2006-12-04

5. Synchronization (4) – Slide 12

POSIX Mutexes (1)

pthread_mutex_init

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

- initializes a new mutex with specific attributes (attr can also be NULL)
- Initialization in the main program before the threads start and use the mutex
- „abbreviation“:

```
pthread_mutex_t fastmutex = PTHREAD_MUTEX_INITIALIZER;
```

POSIX Mutexes (3)

pthread_mutex_unlock

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

- remove lock

pthread_mutex_destroy

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

- discontinue using this mutex

POSIX Mutexes (2)

pthread_mutex_lock

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

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

- pthread_mutex_lock
 - tries to gain the lock
 - blocks if lock is already held by a different thread
- pthread_mutex_trylock
 - also tries to gain the lock
 - call returns even if lock could not be gained (with error code **EBUSY**)

POSIX Mutexes (4)

```
#include <stdio.h>  
#include <stdlib.h>  
#include <pthread.h>  
  
void *functionC();  
pthread_mutex_t mutex1 =  
    PTHREAD_MUTEX_INITIALIZER;  
int counter = 0;  
#define NO_THREADS 2  
  
main() {  
    int rc[NO_THREADS];  
    int i;  
    pthread_t thread[NO_THREADS];  
  
    /* create two threads which execute functionC() */  
    for (i=0; i<NO_THREADS; i++) {  
        if( (rc[i]=pthread_create( &thread[i], NULL, &functionC, NULL)) ) {  
            printf("Thread creation failed: %d\n", rc[i]);  
        };  
    };  
    /* wait for the two threads */  
    for (i=0; i<NO_THREADS; i++) pthread_join( thread[i], NULL);  
    printf("final result: %d\n", counter);  
    exit(0);  
}
```

POSIX Mutexes (5)

- ... and in *broken_threads.c* a version without mutexes (i.e. critical region is unprotected):

```
void *functionC()
{
    // pthread_mutex_lock( &mutex1 );
    int tmp=counter;           // read shared variable
    for (i=0; i<999999; i++) {};    // spend some time ...
    tmp++;
    counter=tmp;               // write shared variable
    printf("Counter value: %d\n",counter);
    // pthread_mutex_unlock( &mutex1 );
}
```

POSIX Mutexes (6)

Test with 20 threads

```
$ gcc -lpthread -o threads threads.c
$ gcc -lpthread -o broken_threads broken_threads.c

$ ./pthread
Counter value: 1
Counter value: 2
Counter value: 3
Counter value: 4
Counter value: 5
Counter value: 6
Counter value: 7
Counter value: 8
Counter value: 9
Counter value: 10
Counter value: 11
Counter value: 12
Counter value: 13
Counter value: 14
Counter value: 15
Counter value: 16
Counter value: 17
Counter value: 18
Counter value: 19
Counter value: 20
final result: 20

$ ./broken_threads
Counter value: 1
Counter value: 2
Counter value: 3
Counter value: 3
Counter value: 3
Counter value: 3
Counter value: 4
Counter value: 5
Counter value: 5
Counter value: 6
Counter value: 6
Counter value: 7
Counter value: 7
Counter value: 7
Counter value: 7
Counter value: 8
Counter value: 8
Counter value: 8
Counter value: 7
Counter value: 7
final result: 7
```

POSIX Mutexes (7)

Mutexes can be „recursive“:

normal	rekursiv
<pre>pthread_mutex_lock (mutex); /* Code */ pthread_mutex_lock (mutex); /* call blocks since mutex is already lockjed */ pthread_mutex_unlock (mutex); pthread_mutex_unlock (mutex);</pre>	<pre>pthread_mutex_lock (mutex); /* Code */ pthread_mutex_lock (mutex); /* successful because the same thread holds this lock */ pthread_mutex_unlock (mutex); pthread_mutex_unlock (mutex); pthread_mutex_unlock (mutex);</pre>

shortcut for initializing a recursive mutex:

```
pthread_mutex_t recmutex = PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP;
```

POSIX Semaphores (1)

sem_init

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
```

- initializes a semaphore with initial value *value*
- Initialization in main program before the threads start and use the semaphore
- pshared*: for shared usage by several processes (not possible on Linux systems)

POSIX Semaphores (2)

sem_wait, sem_trywait

```
int sem_wait(sem_t * sem);
int sem_trywait(sem_t * sem);
```

- **sem_wait** implements wait() operation
 - decrements counter c in the semaphore if c>0
 - otherwise blocks thread until c>0
- **sem_trywait**
 - decrements counter c in the semaphore if c>0
 - return error value EAGAIN if c<=0

POSIX Semaphores (4)

sem_getvalue

```
int sem_getvalue(sem_t * sem, int * sval);
```

- **sem_getvalue** reads the value of a semaphore and writes it into the given variable

POSIX Semaphores (3)

sem_post

```
int sem_post(sem_t * sem);
```

- **sem_post** implements signal() operation
 - increments counter c in the semaphore
 - never blocks

sem_destroy

```
int sem_destroy(sem_t * sem);
```

- stop using the semaphore

POSIX Semaphores (5)

```
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <pthread.h>

static sem_t sem;
void *functionC() {
    int i;
    sem_wait( &sem );
    int tmp=counter; // read shared variable
    for (i=0; i<999999; i++) {};
    // spend some time ...
    tmp++;
    counter=tmp;
    // write shared variable
    printf("Counter value: %d\n",counter);
    sem_post( &sem );
}

main() {
    int rc[NO_THREADS]; int i;
    pthread_t thread[NO_THREADS];
    /* initialize semaphore to 1 */
    sem_init(&sem, 0, 1);
    /* create two threads which execute functionC() */
    for (i=0; i<NO_THREADS; i++) {
        if( (rc[i]=pthread_create( &thread[i], NULL, &functionC, NULL)) ) {
            printf("Thread creation failed: %d\n", rc[i]);
        }
    };
    /* wait for the threads */
    for (i=0; i<NO_THREADS; i++) pthread_join( thread[i], NULL);
    printf("final result: %d\n",counter);
    exit(0);
}
```

POSIX Condition Variables (1)

- Idea: threads wait for a specific condition to be satisfied and meanwhile sleep (cf. *monitors*)
- two base functions:
 - **pthread_cond_signal** & **thread_cond_broadcast** signal (satisfaction of) condition
 - wakes up one thread / all threads which wait for the condition (non-sleeping threads are not signalled)
 - **pthread_cond_wait** wait until the condition is satisfied
- always protect condition variables with a mutex

POSIX Condition Variables (2)

pthread_cond_init

```
int pthread_cond_init(pthread_cond_t *cond,  
                      pthread_condattr_t *cond_attr);
```

- initializes a condition variable
- initialization happens in main program before the threads start and use the cond. variable
- „abbreviation“:
`pthread_cond_t cond = PTHREAD_COND_INITIALIZER;`

POSIX Condition Variables (3)

pthread_cond_wait

```
int pthread_cond_wait ( pthread_cond_t *cond,  
                      pthread_mutex_t *mutex );
```

- **pthread_cond_wait** unlocks the mutex and waits for the condition *cond* to be signalled (thread sleeps)
- when the condition is signalled, the function locks the mutex before transferring control back to the calling thread

POSIX Condition Variables (4)

pthread_cond_signal & **pthread_cond_broadcast**

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

- **pthread_cond_signal** wakes up **one of** the threads which wait for condition *cond*.
(If no thread waits, nothing happens.)
- **pthread_cond_broadcast** wakes up **all** threads which wait for condition *cond*.
(If no thread waits, nothing happens.)

POSIX Condition Variables (5)

```
int x,y;
pthread_mutex_t mut = PTHREAD_MUTEX_INITIALIZER;
// mutex protects accesses to x, y
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
// condition: x > y

thread_one () {
    /* wait for x > y */
    pthread_mutex_lock(&mut);
    while (x <= y) {
        pthread_cond_wait(&cond, &mut);
    }
    // use x and y
    pthread_mutex_unlock(&mut);
}

thread_two () {
    pthread_mutex_lock(&mut);
    // modify x and y
    if (x > y) pthread_cond_broadcast(&cond);
    pthread_mutex_unlock(&mut);
}
```

Overview of POSIX functions

	Mutexes	Semaphores	Condition Variables
wait or block	pthread_mutex_lock, pthread_mutex_trylock	sem_wait, sem_trywait	pthread_cond_wait
signal	pthread_mutex_unlock	sem_post	pthread_cond_signal, pthread_cond_broadcast
create	pthread_mutex_init	sem_init	pthread_cond_init
destroy	pthread_mutex_destroy	sem_destroy	pthread_cond_destroy