Softwareentwicklung für eingebettete Systeme

Bachelorstudiengang Technische Informatik
Hochschule Pforzheim

Vorlesung 3
Sommersemester 2014

Dipl.-Ing.(FH) Marc Jüttner

IPC

- Interprozesskommunikation
  - Interaktion zwischen nebenläufigen Prozessen oder Threads
- Interprozessorkommunikation
  - Kommunikation zwischen Prozessoren
- Interprozessorinterrupts
  - Interrupts zwischen Prozessoren

Interprozesskommunikation

- Zwischen Prozessen auf demselben Prozessor
  - Strikte Speichertrennung
- Auch Kommunikation zwischen Prozessen auf entfernten Systemen/anderen Prozessoren
  - Verteilte Systeme
  - Heterogene SoCs

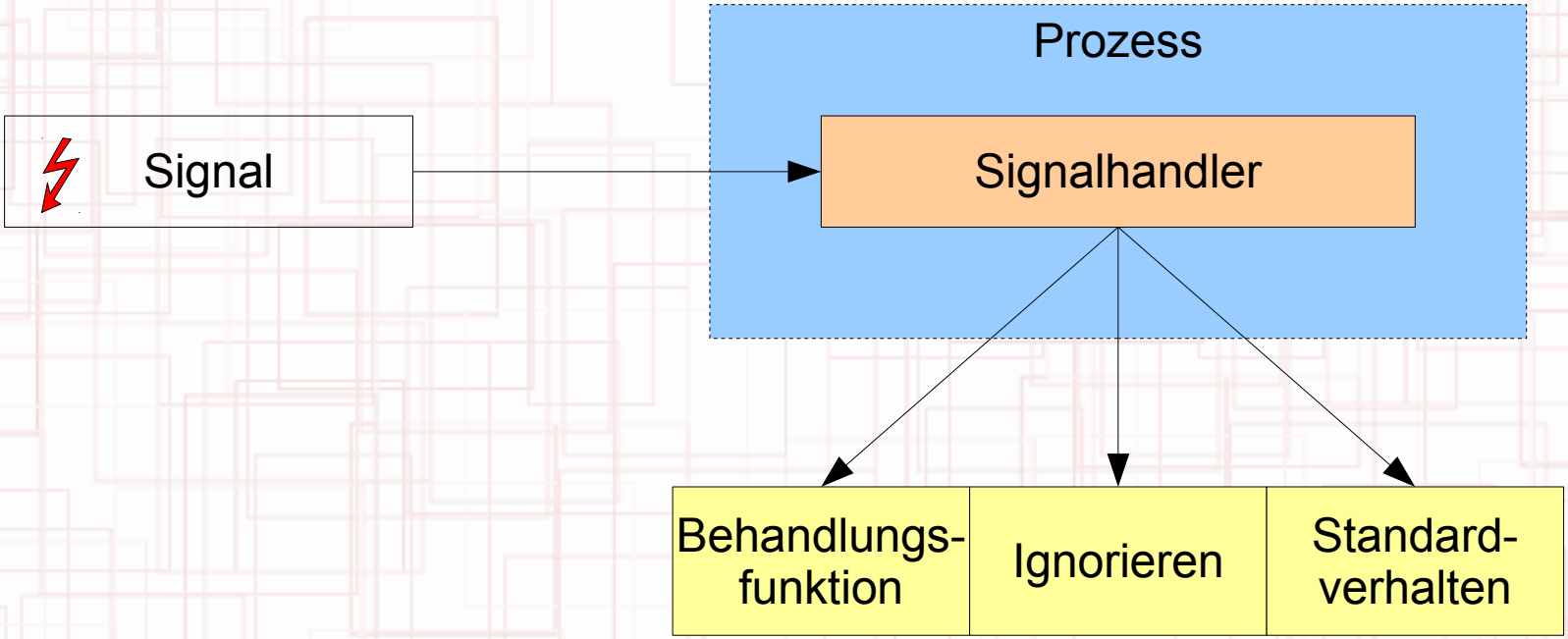
Voraussetzungen

- Impliziert Betriebssystem
- HLOS
  - Prozesse, Threads
  - Häufig nach POSIX-API
- RTOS
  - Tasks
  - Oft proprietäre API

Signale, Ereignisse

- Nachrichten mit bestimmtem Informationsgehalt
- Asynchrones Konzept
  - Erfordert Registrierung von Behandlungsfunktionen
- Warten auf Ereignis
  - Blockieren bis Ereignis eintritt

POSIX-Signale



Beispiel in C

```c
/* signal_simple.c */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>

#define ALARMTIME 1
#define SIGNALCOUNT 2

volatile sig_atomic_t signaled = 0;

void handler_usr(int signal) {
  printf("Caught signal %d.\n", signal);
  signaled++;
}

void handler_alarm(int signal) {
  printf("Caught signal %d.\n", signal);
  /* restart alarm */
  alarm(ALARMTIME);
}
```

```c
int main(int argc, char **argv) {
  struct sigaction sigusr1, sigusr2, sigalrm;

  sigfillset(&sigusr1.sa_mask);
  sigfillset(&sigusr2.sa_mask);
  sigfillset(&sigalrm.sa_mask);

  sigusr1.sa_handler = handler_usr;
  sigusr2.sa_handler = SIG_IGN;
  sigalrm.sa_handler = handler_alarm;

  if (0 > sigaction(SIGUSR1, &sigusr1, NULL))
    { perror("Failed to register handler for SIGUSR1.\n"); return 1; }
  if (0 > sigaction(SIGUSR2, &sigusr2, NULL))
    { perror("Failed to register handler for SIGUSR2.\n"); return 1; }
  if (0 > sigaction(SIGALRM, &sigalrm, NULL))
    { perror("Failed to register handler for SIGALRM.\n"); return 1; }

  alarm(ALARMTIME);
  while(signaled < SIGNALCOUNT) {
  }

  return 0;
}
```

Ausgabe...

```
$ gcc -o signal_simple signal_simple.c

$ ./signal_simple

Caught signal 14.
Caught signal 14.
Caught signal 14.
Caught signal 14.
Caught signal 14.
Caught signal 10.
Caught signal 14.
Caught signal 14.
Caught signal 14.
Caught signal 14.
Caught signal 10.
```

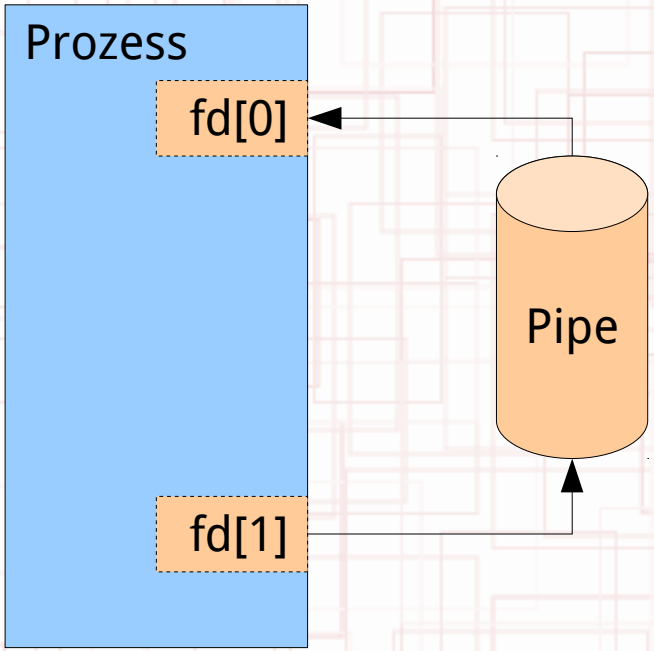
```
$ killall -SIGUSR2 signal_simple
$ killall -SIGUSR2 signal_simple
$ killall -SIGUSR1 signal_simple
$ killall -SIGUSR2 signal_simple
$ killall -SIGUSR1 signal_simple
```

Pipes

- Unidirektionaler Datenaustausch zwischen Prozessen
  - Bytestrom, Datenstrom
- *Content agnostic*
  - Keine definierten Datenformate
- Häufig mit Pufferspeicher
  - FIFO
- Richtungsgebunden: Nur eine Richtung

Funktion einer Pipe

- Erzeugung zweier Dateideskriptoren
  - Read: fd[0]
  - Write: fd[1]
- Daten werden von fd[1] nach fd[0] übertragen




```
#include <unistd.h>
int fd[2], ret;
...
ret = pipe(fd);
...
```

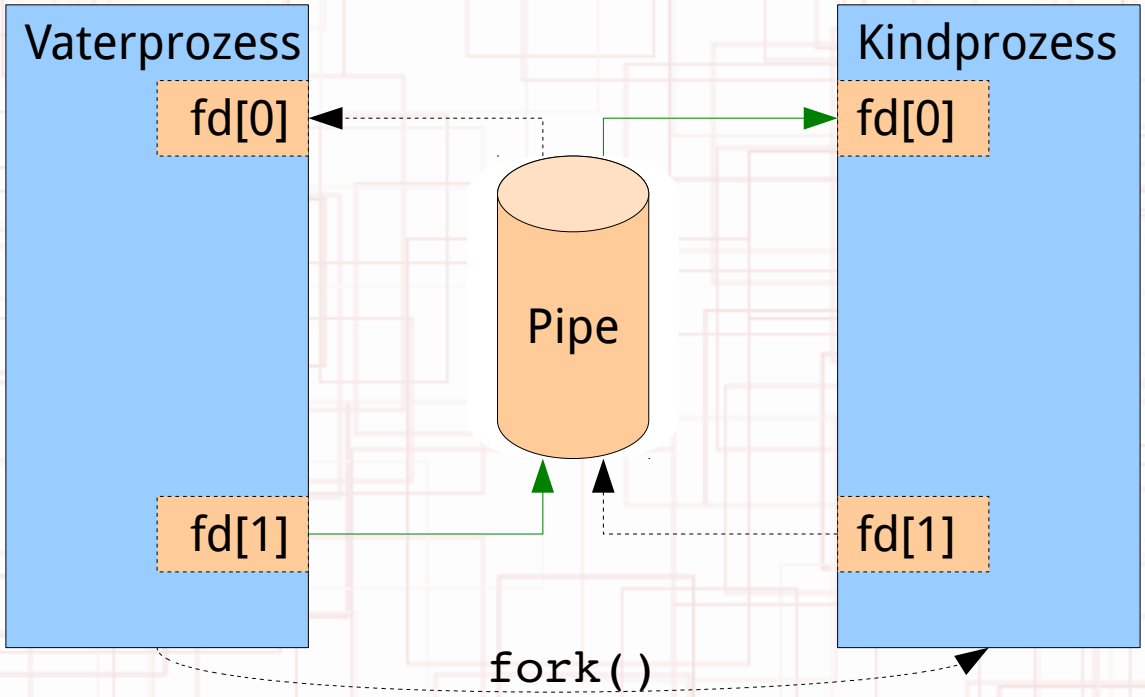
Beispiel in C

```c
#include <unistd.h>
#include <stdio.h>

#define BUFFER_SIZE 128
const char teststring[] = "This is a teststring.";

int main (int argc, char **argv) {
  int fd[2], n;
  char buffer[BUFFER_SIZE];

  /* Create a pipe */
  if (pipe(fd) < 0) { perror ("Failed to create pipe"); return 1; }
  printf("File descriptor fd[0]: 0x%08x\nFile descriptor fd[1]: 0x%08x\n", fd[0], fd[1]);

  /* Write to the pipe... */
  if (0 > (n = write(fd[1], teststring, sizeof(teststring))))
  { perror("Failed to write to pipe."); return 1; }
  printf("Wrote %d bytes to pipe: \"%s\"\n", n, teststring);

  /* Read from the pipe... */
  if (0 > (n = read(fd[0], buffer, BUFFER_SIZE)))
  { perror("Failed to read from pipe."); return 1; }

  printf("Read %d bytes from pipe: \"%s\"\n", n, buffer);

  close(fd[0]); close(fd[1]);

  return 0;
}
```

Ausgabe...

```
$ gcc -o pipe_simple pipe_simple.c

$ ./pipe_simple

File descriptor fd[0]: 0x00000003
File descriptor fd[1]: 0x00000004
Wrote 22 bytes to pipe: "This is a teststring."
Read 22 bytes from pipe: "This is a teststring."
```

Pipe zwischen Prozessen



Beispiel in C

```c
/* Create a pipe */
if (pipe(fd) < 0) { perror("Failed to create pipe"); return 1; }
printf("File descriptor fd[0]: 0x%08x\nFile descriptor fd[1]: 0x%08x\n", fd[0], fd[1]);

if (0 > (pid = fork())) { perror("Failed to fork."); return 1; }
else if (pid > 0) {
  /* parent process */
  printf("[parent] My pid is %d.\n", getpid());
  close(fd[0]); /* close read side - we don't need it. */
  sleep(5);
  /* Write to the pipe... */
  if (0 > (n = write(fd[1], teststring, sizeof(teststring))))
  { perror("[parent] Failed to write to pipe."); return 1; }
  printf("[parent] Wrote %d bytes to pipe: \"%s\"\n", n, teststring);
  /* Wait for child process termination */
  if (0 > (waitpid (pid, NULL, 0))) { perror("[parent] Failed to wait for child process."); return 1; }
  close(fd[1]);
} else {
  /* child process */
  printf("[child ] My pid is %d.\n[child ] Waiting...\n", getpid());
  close(fd[1]); /* close write side - we don't need it. */
  /* Read from the pipe... */
  if (0 > (n = read(fd[0], buffer, BUFFER_SIZE))) { perror("[child ] Failed to read from pipe."); return 1; }
  printf("[child ] Read %d bytes from pipe: \"%s\"\n", n, buffer);
  close(fd[0]);
}
```

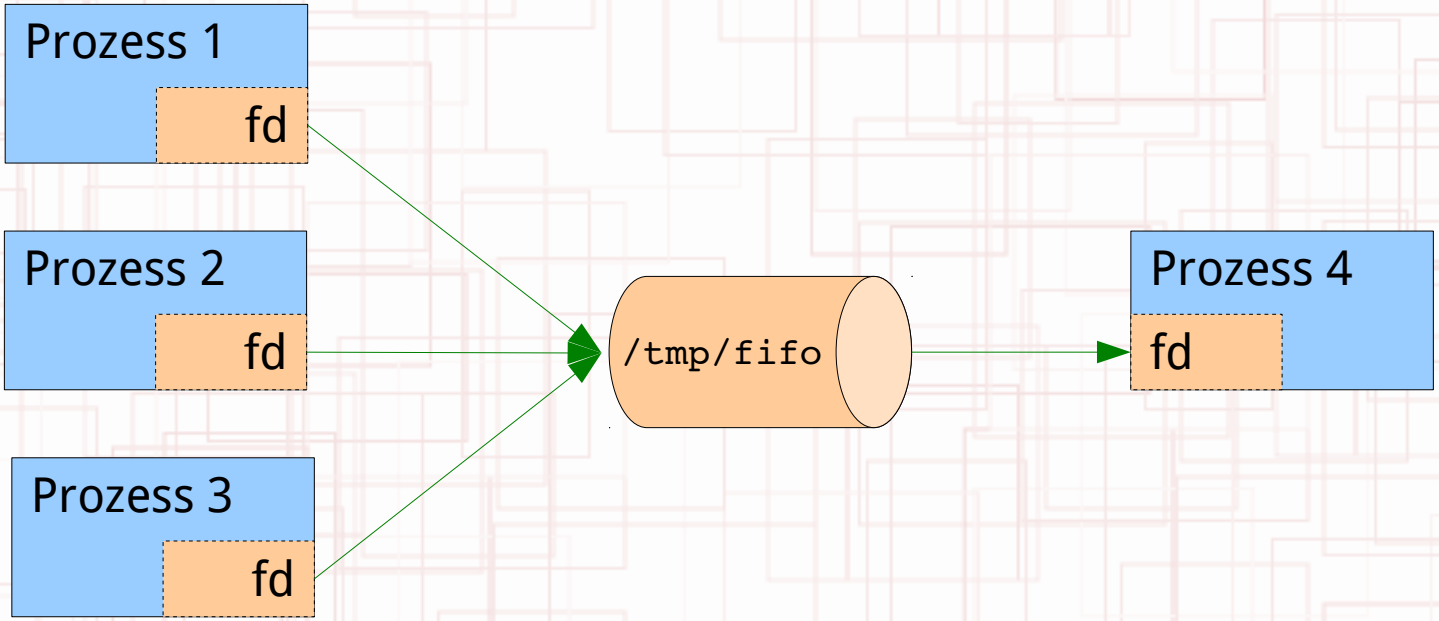
Ausgabe...

```
$ gcc -o pipe_fork pipe_fork.c

$ ./pipe_fork

[parent] My pid is 14519.
[child ] My pid is 14520.
[child ] Waiting...
[parent] Wrote 22 bytes to pipe: "This is a teststring."
[child ] Read 22 bytes from pipe: "This is a teststring."
```

Named Pipes

- Prozessunabhängige Pipes
- Verwendung wie Dateien
  - Lesen und Schreiben durch beliebig viele Prozesse
- Kein fork() zur Übergabe der Dateideskriptoren erforderlich

Named Pipe zwischen Prozessen




```
$ mkfifo /tmp/fifo
$ ls -l /tmp/fifo

prw-r--r-- 1 juemar juemar 0 Mai 10 10:20 /tmp/fifo
```

Beispiel in C: Reader

```c
#define BUFFER_SIZE 128
char defaultfifo[] = "/tmp/fifo";

int main (int argc, char **argv) {
  int fd, n;
  char *fifo = defaultfifo;
  char buffer[BUFFER_SIZE];

  /* open the fifo */
  printf("[reader] Opening fifo \"%s\"\n", fifo);
  if (argc > 1) fifo = argv[1];
  fd = open(fifo, O_RDONLY);
  printf("[reader] File descriptor fd: 0x%08x\n", fd);

  /* Write to the fifo... */
  n = read(fd, buffer, BUFFER_SIZE);
  printf("[reader] Read %d bytes from fifo: \"%s\"\n", n, buffer);

  close(fd);

  return 0;
}
```

Beispiel in C: Writer

```c
#define BUFFER_SIZE 128
const char teststring[] = "This is a teststring.";
char defaultfifo[] = "/tmp/fifo";

int main (int argc, char **argv) {
  int fd, n;
  char *fifo = defaultfifo;

  /* open the fifo */
  printf("[writer] Opening fifo \"%s\"\n", fifo);
  if (argc > 1) fifo = argv[1];
  fd = open(fifo, O_WRONLY);
  printf("[writer] File descriptor fd: 0x%08x\n", fd);

  /* Write to the fifo... */
  n = write(fd, teststring, sizeof(teststring));
  printf("[writer] Wrote %d bytes to fifo: \"%s\"\n", n, teststring);

  close(fd);

  return 0;
}
```

Ausgabe...

```
$ gcc -o fifo_reader fifo_reader.c
$ gcc -o fifo_writer fifo_writer.c

$ ./fifo_writer

[writer] Opening fifo "/tmp/fifo"
[writer] File descriptor fd: 0x00000003
[writer] Wrote 22 bytes to fifo: "This is a teststring."

$ ./fifo_reader

[reader] Opening fifo "/tmp/fifo"
[reader] File descriptor fd: 0x00000003
[reader] Read 22 bytes from fifo: "This is a teststring."
```

Besonderheiten

- Zugriff ist atomar
- read() blockiert bei leerer Pipe bis Daten zur Verfügung stehen
- write() blockiert bei voller Pipe bis wieder Daten geschrieben werden können
- Rückgabe von 0 bei leerer bzw. voller Pipe
- Parameter `O_NONBLOCK` oder `O_NDELAY` verhindern Blockieren

Kombinationen

- Events können durch Pipes/Fifos abgebildet werden
- Benötigt Listener-Funktion
  - Eigener Thread
  - Wartet auf Nachrichten
  - Ermitteln erforderlicher Aktionen anhand Nachrichtentyp
  - Aufruf von Behandlungsfunktionen

Einfaches Beispiel

- Definition eines Nachrichtentyps
- Erzeugen einer Pipe
- Fork()
  - Vaterprozess: Senden von Nachrichten
  - Kindprozess: Empfangen und bearbeiten von Nachrichten

Ausgabe (1)

```
File descriptor fd[0]: 0x00000003
File descriptor fd[1]: 0x00000004
[parent] My pid is 666.
[child ] My pid is 667.
[child ] Waiting...
[parent] Wrote 36 bytes to pipe.
[parent] Wrote 36 bytes to pipe.
[parent] Wrote 36 bytes to pipe.
[parent] Wrote 36 bytes to pipe.
[parent] Wrote 36 bytes to pipe.
[parent] Wrote 36 bytes to pipe.
[parent] Wrote 36 bytes to pipe.
-----------------------------
```

Ausgabe (2)

```
[child ] Read 36 bytes from pipe
[child ]   msghandler()[37]: Received message 1, "Event1".
[child ] Message handler returned 0.
============================
[child ] Read 36 bytes from pipe
[child ]   msghandler()[37]: Received message 2, "Arbitrary".
[child ]   msghandler()[54]: Unknown message.
[child ] Message handler returned -1.
============================
[child ] Read 36 bytes from pipe
[child ]   msghandler()[37]: Received message 3, "Print status".
[child ]   msghandler()[44]: Received status command: Received 3 messages until now.
[child ] Message handler returned 0.
============================
[child ] Read 36 bytes from pipe
[child ]   msghandler()[37]: Received message 4, "Event2".
[child ] Message handler returned 0.
============================
[child ] Read 36 bytes from pipe
[child ]   msghandler()[37]: Received message 5, "Reset counter".
[child ] Message handler returned 0.
============================
[child ] Read 36 bytes from pipe
[child ]   msghandler()[37]: Received message 1, "Print status".
[child ]   msghandler()[44]: Received status command: Received 1 messages until now.
[child ] Message handler returned 0.
============================
[child ] Read 36 bytes from pipe
[child ]   msghandler()[37]: Received message 2, "Terminate".
[child ]   msghandler()[47]: Received terminate command, will terminate.
[child ] Message handler returned 0.
```

Semaphore

- Mechanismus zum wechselseitigen Ausschluss
  - „kritischer Bereich“
- Nach Dijkstra (1965)
- Kann >1 sein
  - Mehrere im kritischen Abschnitt
- Mutex = Semaphore mit Wert 1

POSIX-Userspace

- „Named semaphore“
- Verfügbar für mehrere Prozesse
- Linux, QNX, ...
- Abfolge
  - Initialisierung/Öffnen
  - Up/down, lock/unlock
  - Freigeben durch letzten Prozess

Beispiel in C

```c
static struct sembuf semaphore;

int main(int argc, char **argv) {
  int sem_id, my_waittime, i;

  sem_id = semget(SEM_KEY, 0, IPC_PRIVATE);
  if (0 > sem_id) {
    umask(0);
    if (0 > (sem_id = semget(SEM_KEY, 1, IPC_CREAT | IPC_EXCL | 0666))) { perror ("Failed to create semaphore"); exit(1); }
    if (0 > (semctl(sem_id, 0, SETVAL, (int)1))) { perror ("Failed to set initial value"); exit(1); }
    printf("[%d][%05d] Created semaphore 0x%08x.\n", time(NULL), getpid(), sem_id);
  } else {
    printf("[%d][%05d] Opened semaphore 0x%08x.\n", time(NULL), getpid(), sem_id);
  }

  my_waittime = (argc > 1)?atoi(argv[1]):1;
  printf("[%d][%05d] My waittime is %d seconds.\n", time(NULL), getpid(), my_waittime);

  for (i = 0; i < CYCLES; i++) {
    semaphore.sem_op = LOCK;
    semaphore.sem_flg = SEM_UNDO;
    if (0 > (semop(sem_id, &semaphore, 1))) { perror("Failed to lock"); exit(1); }
    printf("[%d][%05d] Locked semaphore 0x%08x.\n", time(NULL), getpid(), sem_id);
    sleep(my_waittime);
    semaphore.sem_op = UNLOCK;
    semaphore.sem_flg = SEM_UNDO;
    if (0 > (semop(sem_id, &semaphore, 1))) { perror("Failed to unlock"); exit(1); }
    printf("[%d][%05d] Unlocked semaphore 0x%08x.\n", time(NULL), getpid(), sem_id);
  }
  printf("[%d][%05d] Terminating.\n", time(NULL), getpid());

  return 0;
}
```

Ausgabe

```
[1399905431][03622] Locked semaphore 0x00048001.
[1399905435][03622] Unlocked semaphore 0x00048001.
[1399905435][03623] Locked semaphore 0x00048001.
[1399905438][03623] Unlocked semaphore 0x00048001.
[1399905438][03624] Locked semaphore 0x00048001.
[1399905441][03624] Unlocked semaphore 0x00048001.
[1399905441][03622] Locked semaphore 0x00048001.
[1399905445][03622] Unlocked semaphore 0x00048001.
[1399905445][03623] Locked semaphore 0x00048001.
[1399905448][03623] Unlocked semaphore 0x00048001.
[1399905448][03624] Locked semaphore 0x00048001.
[1399905451][03624] Unlocked semaphore 0x00048001.
[1399905451][03622] Locked semaphore 0x00048001.
[1399905455][03622] Unlocked semaphore 0x00048001.
[1399905455][03623] Locked semaphore 0x00048001.
[1399905458][03623] Unlocked semaphore 0x00048001.
[1399905458][03624] Locked semaphore 0x00048001.
[1399905461][03624] Unlocked semaphore 0x00048001.
[1399905461][03622] Locked semaphore 0x00048001.
[1399905465][03622] Unlocked semaphore 0x00048001.
...
[1399905475][03622] Terminating.
...
[1399905478][03623] Terminating.
...
[1399905481][03624] Terminating.
```

Semaphore > 1

- Anwendung von Semaphoren zur
  - Bandbreitenverwaltung
  - Begrenzung des Speicherverbrauchs
- Beispiel: USB-Treiber
  - Maximale Zahl gleichzeitig versandter URBs

R/W-Semaphore

- Vermeidung von Wartezeiten bei nur-lesen-Zugriff
- Wechselseitiger Ausschluss nur bei Schreibzugriffen
  - Anforderung mit Zusatzinformation: Lesen oder Schreiben
- Nur lesende Prozesse: Immer frei
- Ein schreibender Prozess: Zugriff schützen!

Locking im Kernel

- Locking = Sichern, *Mutual Exclusion*
- Kontext muss berücksichtigt werden
  - Semaphore legen einen Prozess schlafen
  - Im Kernel nur mit Prozesskontext!
- Sonderbehandlung für Interrupthandler
  - Spinlocks

Spinlocks

- Zugriff muss atomar sein!
- Implementierung häufig mittels Prozessorbefehlen
  - Fetch & add
  - Test & set
- Problem bei Prozessen mit unterschiedlichen Prioritäten

```
...
while (lock != 0);
lock = 1
...
Critical section...
...
lock = 0
...
```

Lockingübersicht

	IRQ-Handler 1	Kernelthread	Treiber im Prozesskontext	Anwendung
IRQ-Handler 1	-	Spinlock	Spinlock	x
Kernelthread	Spinlock	-	Spinlock	x
Treiber im Prozesskontext	Spinlock	Spinlock	-	Semaphore
Anwendung	x	x	Semaphore	Semaphore

- Lockingmechanismus ist abhängig von den beteiligten Softwaremodulen!

Sockets

- Verwandt mit Pipe-Kommunikation
  - Deskriptor, read(), write()
- Peer-to-peer
- Bidirektionale Kommunikation möglich
  - Antworten bei Pipes erfordert eine zweite Pipe…

Sockets

- Client/Server-Konzept
  - Ein Prozess wartet auf Anfragen
  - Clients verbinden und kommunizieren bei Bedarf
- Kommunikation wie bei IP
  - Localhost: 127.0.0.1 oder *unix sockets*
  - Leichte Skalierbarkeit für verteilte Implementierungen

Vergleich mit Pipes

```
--- pipe_message.c      2014-05-12 19:57:07.985448055 +0200
+++ socket_message.c    2014-05-12 20:04:20.823594384 +0200
@@ -16,2 +16,4 @@
 #include <stdio.h>
+#include <sys/types.h>
+#include <sys/socket.h>

@@ -111,5 +113,5 @@

-  /* Create a pipe */
-  pipe(fd);
-  printf("File descriptor fd[0]: 0x%08x\nFile descriptor fd[1]: 0x%08x\n", fd[0], fd[1]);
+  /* Create a socket pair */
+  socketpair(AF_LOCAL, SOCK_SEQPACKET, 0, fd);
+  printf("Socket descriptor fd[0]: 0x%08x\nSocket descriptor fd[1]: 0x%08x\n", fd[0], fd[1]);

@@ -124,3 +126,3 @@
       n = write(fd[1], (char *)&messages[i], sizeof(struct msg_t));
-      printf("[parent] Wrote %d bytes to pipe.\n", n);
+      printf("[parent] Wrote %d bytes to socket.\n", n);
     }
@@ -138,3 +140,3 @@
       n = read(fd[0], buffer, BUFFER_SIZE);
-      printf("[child ] Read %d bytes from pipe\n", n);
+      printf("[child ] Read %d bytes from socket\n", n);
       n = msghandler((struct msg_t *)buffer);
```

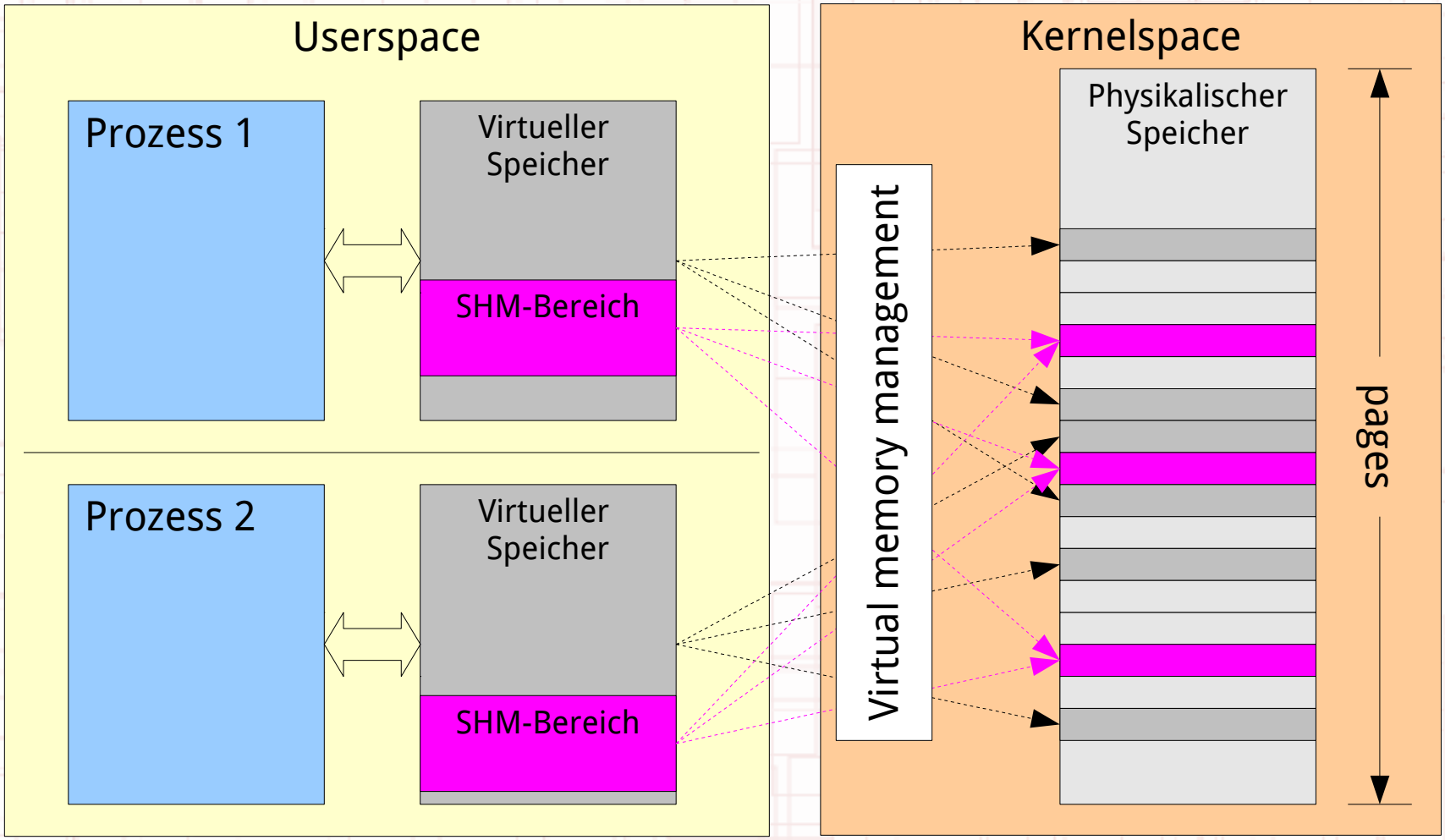
IPC-Zusammenfassung

- Signalisierung und Synchronisation
  - Signale, Events, Semaphore, Mutexe, Spinlocks
- Datenübertragung
  - Pipes, Sockets
  - Kopieren Daten zwischen zwei Prozessen
  - Problem bei großen Datenmengen!

Shared Memory

- Einfachere Lösung: Vermeiden von Kopien!
- Verwenden gemeinsamen Speichers zwischen Prozessen
- Problemstellung für das Betriebssystem!
  - Strikte Prozesstrennung!

SHM und virtueller Speicher



Fallstricke

- Bei Übergabe von Adressen im SHM dürfen nur Offsets übergeben werden
  - u.U. wird das SHM-Segment in jedem Prozess an unterschiedlichen virtuellen Adressen eingefügt
- SHM ist ein kritischer Bereich!

Beispiel in C

```c
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SHM_SIZE 30
#define SHM_KEY 0xdeadbabe

int main(int argc, char **argv)
{
  int shmId;
  char *shmPtr;
  int i;

  /* Create shared memory segment */
  shmId = shmget(SHM_KEY, SHM_SIZE, IPC_CREAT | 0666);
  if (shmId >= 0) {
    /* Map the SHM segment into our address space */
    shmPtr = shmat(shmId, 0, 0);
    if (shmPtr == (char *)-1) {
      perror("shmat");
    } else {
      printf("SHM area mapped to %p.\n", shmPtr);
      /* Fill the SHM segment */
      for (i=0; i<SHM_SIZE; i++) {
        shmPtr[i] = 'A' + i;
      }
      getchar(); /* Wait until <enter> is hit */
      /* detach SHM segment */
      shmdt(shmPtr);
    }
  } else {
    perror("shmget");
  }
}
```

```c
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SHM_SIZE 30
#define SHM_KEY 0xdeadbabe

int main(int argc, char **argv)
{
  int shmId;
  char *shmPtr;
  int i;

  /* Open an existing SHM segment */
  shmId = shmget(SHM_KEY, SHM_SIZE, 0666);
  if (shmId >= 0) {
    shmPtr = shmat(shmId, 0, 0);
    if (shmPtr == (char *)-1) {
      perror("shmat");
    } else {
      printf("SHM area mapped to %p.\n", shmPtr);
      for (i = 0; i < SHM_SIZE; i++) {
        printf("%c", shmPtr[i]);
      }
      printf("\n");
      shmdt(shmPtr);
    }
  } else {
    perror("shmget");
  }
}
```

Ausgabe...

```
$ ./shm_writer
SHM area mapped to 0x7f664d0c0000.

$ ./shm_reader
SHM area mapped to 0x7f2d34be3000.
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^

$
```

POSIX-Messagequeues

- Ähnlich Pipe, jedoch mit Nachrichtentyp
- Lesende Prozesse können einzelne Nachrichtentypen lesen

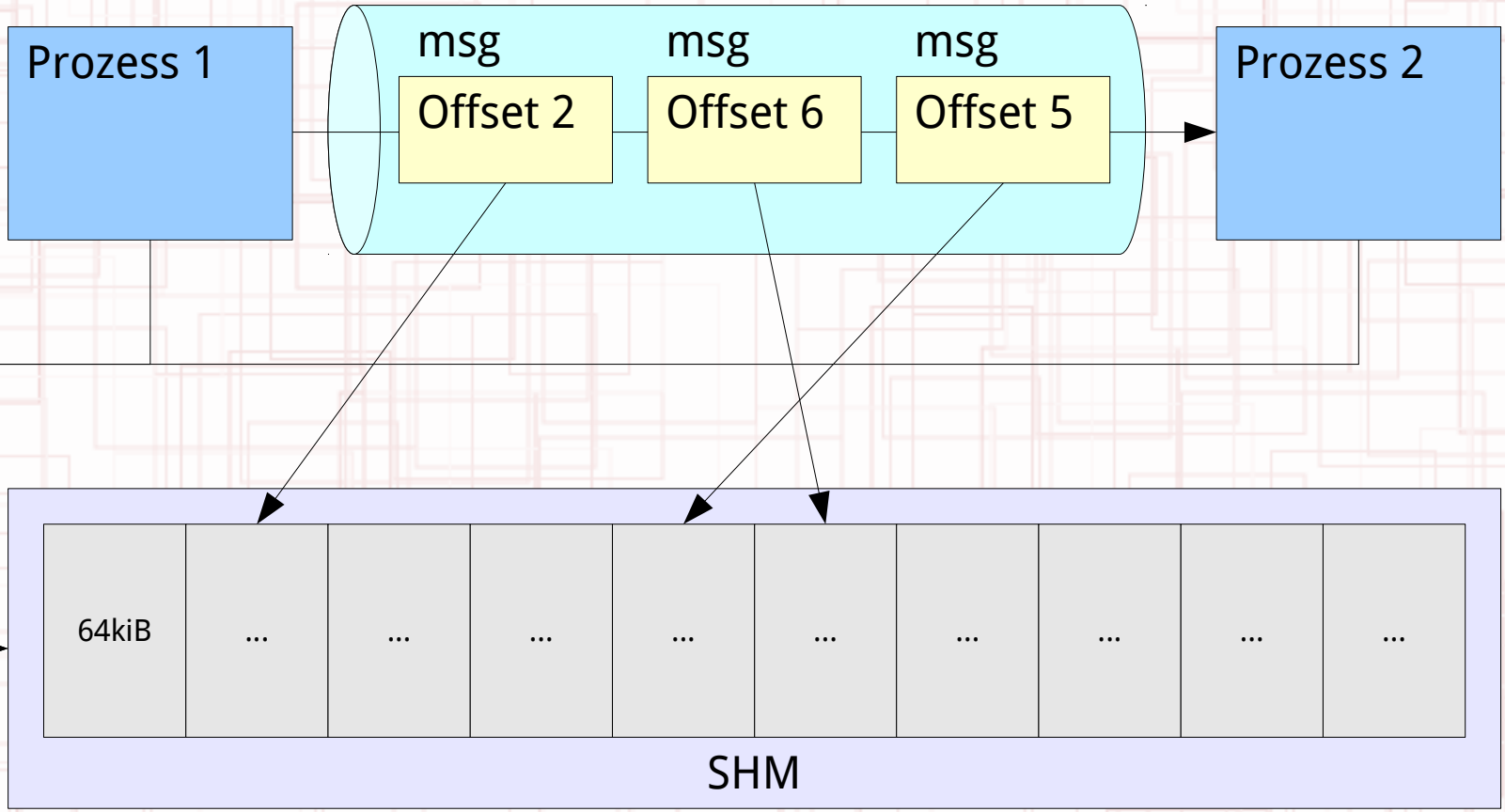
Beispiel in C

```c
#define MQ_MSGSIZE 20
#define MQ_KEY 2404

struct mqmsg {
  long mtype;
  char mtext[MQ_MSGSIZE];
};

int main(int argc, char **argv)
{
  int mqId;
  struct mqmsg message;
  long msgTyp = 0;

  if (argc > 1) {
    message.mtype = atol(argv[1]);
  }
  if (argc > 2) {
    strncpy(message.mtext, argv[2], MQ_MSGSIZE);
  } else {
    *message.mtext = 0;
  }

  mqId = msgget(MQ_KEY, IPC_CREAT | 0666);
  if (mqId >= 0) {
    printf("Sending msg type %ld\n", message.mtype);
    if (-1 == msgsnd(mqId, &message, sizeof(struct mqmsg), 0)) {
      perror("msgsnd");
    } else {
      printf("Sent data: %s\n", message.mtext);
    }
  } else {
    perror("msgget");
  }
}
```

Beispiel in C

```c
#define MQ_MSGSIZE 20
#define MQ_KEY 2404

struct mqmsg {
  long mtype;
  char mtext[MQ_MSGSIZE];
};

int main(int argc, char **argv)
{
  int mqId;
  struct mqmsg message;
  long msgType = 0;

  if (argc > 1) {
    msgType = atol(argv[1]);
  }

  mqId = msgget(MQ_KEY, IPC_CREAT | 0666);
  if (mqId >= 0) {
    printf("Waiting for message type %ld\n", msgType);
    if (-1 == msgrcv(mqId, &message, sizeof(struct mqmsg), msgType, 0)) {
      perror("msgrcv");
    } else {
      printf("Received data: %s\n", message.mtext);
    }
  } else {
    perror("msgget");
  }
}
```

Pipe + SHM


Prozess 1
msg
Offset 2
msg
Offset 6
msg
Offset 5
Prozess 2
64kiB
...
...
...
...
...
...
...
...
...
SHM