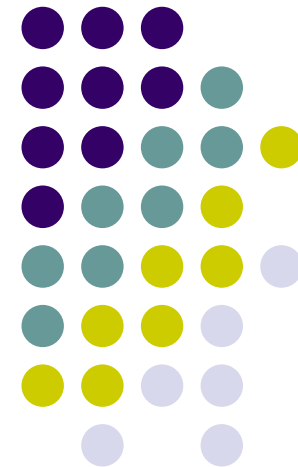


# Einführung in die Systemprogrammierung

3

Interprozesskommunikation

**Message-Queues**



# Kennungen und Schlüssel

- Message Queues und Semaphore verwenden Objekte zur Interprozesskommunikation
- Jedem Objekt wird beim Einrichten intern eine *Kennung (Identifier)* zugeteilt um es zu identifizieren
- Beim Einrichten eines Objekts muss vom Benutzer ein *Schlüssel (Key)* angegeben werden.
  - Datentyp des Schlüssels ist `key_t`
  - Schlüssel wird vom Kernel mit der entsprechenden Kennung verbunden
  - Auch *nicht verwandte Prozesse* können das Objekt durch Angabe des Schlüssels nutzen
  - Objekte, denen kein Schlüssel zugeordnet ist werden als *private Objekte* bezeichnet. Hier wird als Key `IPC_PRIVATE` angegeben.

# Kommunikationsmöglichkeiten 1

- Prozesse können auf verschiedene Arten ein gemeinsames Objekt zur Kommunikation verwenden:
  - Privates Objekt, verwandte Prozesse:
    - Elternprozess kreiert ein privates Objekt.
    - Zurückgegebene Kennung wird in Variable gespeichert
    - Beim Kreieren des Kindprozesses mit fork wird Kennung an Kindprozess vererbt
    - Kindprozess kann Kennung als Argument bei exec-Anweisung angeben und somit einem anderen Programm verfügbar machen
  - Privates Objekt, nicht verwandte Prozesse:
    - Prozess A kreiert ein privates Objekt.
    - Die zurückgegebene Kennung speichert er in einer Datei.
    - Prozess B liest Kennung aus Datei aus und kann unter Angabe der Kennung auf Objekt zugreifen

# Kommunikationsmöglichkeiten 2

- Prozesse können auf verschiedene Arten ein gemeinsames Objekt zur Kommunikation verwenden:
  - Nicht-privates Objekt, nicht verwandte Prozesse:
    - Prozess A und Prozess B können einen Schlüssel vereinbaren (z. B. in einer Header-Datei)
    - Prozess A kreiert ein neues Objekt mit diesem Schlüssel
    - Prozess B kann mit diesem Schlüssel auf das Objekt zugreifen
    - Problem: falls schon ein Objekt mit diesem Schlüssel existiert, schlägt das Erstellen des Objekts in Prozess A fehl. Prozess A muss den Fehler erkennen, altes Objekt löschen und kann dann ein neues Objekt mit diesem Schlüssel anlegen

# Message Queues

- Message Queues (Nachrichten-Warteschlangen) werden im Kernel mit einer verketteten Liste verwaltet
- Zu jeder Message-Queue existiert eine Kennung (Message Queue Identifier)
- Um eine Message Queue einzurichten oder eine existierende zu öffnen wird die Funktion *msgget* verwendet
- Zum Senden von Messages wird die Funktion *msgsnd* verwendet. Die Message wird dabei am Ende der Liste angehängt
- Um Messages zu empfangen wird die Funktion *msgrcv* verwendet

# msgget – Öffnen oder Kreieren einer Message-Queue

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget (key_t schlüssel, int flag);
```

- Rückgabewerte:
  - Bei Erfolg: Kennung der Message-Queue
  - Bei Fehler: -1
- Schlüssel:
  - IPC\_PRIVATE
  - Eine Wert vom Typ key\_t (meist long int)
- Flag:
  - Angabe mehrerer Flags, getrennt durch | möglich
  - Wenn Schlüssel nicht IPC\_PRIVATE:
    - IPC\_CREATE: zum kreieren eines neuen Objekts mit dem angegebenen Schlüssel
    - IPC\_EEXIST: Fehlermeldung beim Kreieren, wenn Objekt bereits existiert

# msgsnd – Senden von Messages

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgsnd (int kennung, const void *puffer,
            size_t mlaenge, int flag);
```

- Rückgabewerte: bei Erfolg: 0; bei Fehler: -1
- kennung: Message Queue an die die Message zu schicken ist
- flag:
  - bei voller Message-Queue wird normalerweise msgsnd blockiert bis ein Ereignis eintritt (Platz in Message-Queue, Löschen der Message-Queue, Signal bricht senden ab)
  - Soll nicht so lange blockiert werden, kann man als flag IPC\_NOWAIT setzen. Senden wird dann sofort abgebrochen, Rückgabewert =-1

# msgsnd – Senden von Messages

- puffer und mlaenge:
  - Message besteht aus Message-Typ, Message-String und Länge der Message
  - puffer enthält die Adresse des Message-Typs
  - Direkt auf Message-Type folgt der eigentliche Message-Text
  - mlaenge definiert die maximale Länge der Message
  - Bei leerem Message-Text muss für mlaenge 0 angegeben werden
  - Message-Typ ist nur von Interesse wenn Messages in einer anderen Reihenfolge empfangen werden sollen

```
struct meine_mesg
{
    long mtype;          /*Message Type */
    char mtext[256];    /* Message-Text */
}
```

# msgrcv – Empfangen von Messages

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgrcv (int kennung, void *puffer,
           size_t maxlaenge, long typ, int flag);
```

- Rückgabewerte: bei Erfolg: 0; bei Fehler: -1
- kennung: Message Queue von der Message zu empfangen ist
- puffer: gibt Adresse an, an die Message-Typ und dirket anschließend Message-Text zu schreiben ist
- maxlaenge: Maximale Länge des Message-Textes
  - ist die Message länger:
    - Fehler, es sei denn es
    - ist flag MSG\_NOERROR gesetzt: Abschneiden der Message

# msgrcv – Empfangen von Messages

- typ: Legt den Typ der zu empfangenden Message fest:
  - typ == 0: Erste Message aus der Queue
  - typ > 0:
    - Erste Message mit dem Typ typ, es sei denn es
    - ist flag MSG\_EXEPT gesetzt: erste Message die nicht Typ typ hat
  - typ < 0: Erste Message aus der Queue, deren Typ der kleinste Wert ist, der kleiner oder gleich dem absoluten Betrag von typ ist
  - Wird benutzt um Prioritäten zu vergeben
- flag:
  - Ist Queue leer wird msgrcv blockiert, bis ein Ereignis eintritt (Message in Queue, Message-Queue gelöscht, Signal zum Abbrechen von msgrcv)
  - flag IPC\_NOWAIT: msgrcv wird sofort unterbrochen, Fehlerstatus wird zurückgegeben

# msgctl –Löschen einer Message Queue

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgctl (int kennung, int kdo, struct msqid_ds *puffer);
```

- kdo: legt die durchzuführende Aktion fest:
  - IPC\_STAT: Abfragen der Statusinformationen an die Adresse puffer
  - IPC\_SET: Setzen des Eigentümers, der Zugriffsrechte und der maximalen Größe
  - IPC\_RMD: Löschen der Message-Queue mit allen ihren Daten
- puffer: wird benötigt für IPC\_STAT und IPC\_SET. Für IPC\_RMD ist hier NULL einzusetzen
- **Hinweis: Message-Queues werden nicht automatisch mit dem Beenden eines Prozesses gelöscht, sondern müssen explizit gelöscht werden!**

# Implementierungsbeispiele

```
/* Eine gemeinsame Headerdatei mq.h*/  
#ifndef MQ  
#define MQ  
  
/* Vereinbarer Schlüssel für gemeinsames Objekt */  
#define MEIN_KEY    10001  
  
/* Maximale Länge der Nachricht */  
#define MAX_LAENGE    200  
  
/* Datentyp für Nachricht A */  
typedef struct  
{  
    long mtyp;  
    char nachricht[MAX_LAENGE];  
} mymsgtyp;  
  
#endif
```

# Implementierungsbeispiele

```
/* Ein Prozess der eine Message-Queue einrichtet
   prozess_a.c */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/stat.h>
#include <stdio.h>
#include "mq.h"
int main (void)
{
    int myid;
    mymsgtyp nachricht1;

    if ( ( myid=msgget( MEIN_KEY, S_IRWXU|S_IWGRP|S_IWOTH
                      |IPC_CREAT) ) == -1)
        /* Error Handling */;
    / * Fortsetzung auf nächster Folie */
```

# Implementierungsbeispiele

```
/* Fortsetzung von vorheriger Folie */
while (1)
{
    if (msgrcv (myid, &nachricht1, MAX_LAENGE, 0, 0)
        == -1 )
        /* Error Handling */;

    if (nachricht1.mtyp == 1000)
    {
        if (msgctl(myid, IPC_RMID, NULL) == -1)
            /* Error Handling */;

        exit(0);
    }
}
}
```

# Implementierungsbeispiele

```
/* Ein Prozess der Nachrichten in eine Queue schreibt */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include "mq.h"
int main (void)
{
    int myid;
    mymsgtyp nachricht1;

    if ( ( myid=msgget( MEIN_KEY,0) == -1))
        /* Error Handling */;

    nachricht1.mtyp=0;
    sprintf (nachricht1.nachricht,"Nachricht");
    if (msgsnd (myid, &nachricht1, MAX_LAENGE, 0)== -1 )
        /* Error Handling */;

    exit (0);
}
```