Betriebssysteme

Kap. 3: Interprozesskommunikation (IPC)
3.3: IPC in UNIX

Stand: WS 08/09 (7.1.09)

Prof. Dr. Wolfgang Küchlin

Dipl.-Inform., Dr. sc. techn. (ETH)

Arbeitsbereich Symbolisches Rechnen
Wilhelm-Schickard-Institut für Informatik
Fakultät für Informations- und Kognitionswissenschaften

Universität Tübingen

Steinbeis Transferzentrum
Objekt- und Internet-Technologien (OIT)

Wolfgang.Kuechlin@uni-tuebingen.de
http://www-sr.informatik.uni-tuebingen.de

SR 

Teil III: IPC in UNIX

- Klassisches UNIX
  - Signale
  - Pipes
- System V
  - Shared Memory
  - Message Passing
  - Semaphore

SR 

Interprozesskommunikation (IPC)

- Zwei Kommunikationswege zwischen Prozessen
  - Direkter Zugriff über Shared-Memory
    - memory based locks
  - Vermittlung durch Betriebssystem
    - Signale
    - Pipes
    - POSIX mutexes / conditions
    - Shared memory (SV)
    - Message Passing (SV)
    - Semaphore (SV)

SR 

Übersicht über IPC-Konzepte in UNIX

- UNIX (BSD)
  - Signale, Pipes, Sockets
- UNIX System V
  - Shared Memory, Messages, Semaphore
- Solaris 2, Linux (ab 2.0)
  - zusätzlich Mutex, Conditions
  - (Signale, Mutex, Conditions
    - POSIX 4-Standard)

SR 

Synchronisation in UNIX

- einzig voll portable Synchronisationsoperation für Anwender:
  - Realisierung von mutex_lock durch das Erzeugen eines **Lock-Files**.
  - creat läuft atomar ab und liefert einenFehler, falls das File schon existiert, d.h. ein anderer Prozess das lock hat.
- Missbrauch des Filesystems ist problematisch:
  - lock-files bleiben liegen, wenn ein Prozess abstürzt
  - ein Prozess hängt, wenn zufällig ein anderes File mit dem gleichen Namen wie das lock-file existiert.

Signale (POSIX)

- UNIX Signale sind ein reines Software-Konzept
  - modelliert nach Vorbild Interrupt + Interrupt-Handler
  - Prozess schickt Signal durch Systemaufruf
  - BS liefert Signal an Empfänger aus
  - BS aktiviert Signal-Handler des Empfängers
- Signale informieren einen Prozess über asynchrone Ereignisse
  - Software-Ereignisse: Prozess verschickt Signal (via BS)
  - Hardware-Ereignisse: Interrupt-Handler verschickt Signal
    - Ctrl-C oder Ctrl-Z auf Tastatur
    - Hardwarefehler

Signale

Signal	Beschreibung	Aktion
SIGABRT	Abnormal Program Termination	T
SIGALRM	Ablaufen eines Timers	T
SIGFPE	Floating Point Exception	T
SIGHUP	Abbruch Terminal-Verbindung (hangup)	T
SIGILL	Illegale Instruktion	T
SIGINT	Interrupt-Taste	T
SIGKILL	Prozess-Termination	T
SIGPIPE	Pipe ohne Leseprozess	T
SIGSEGV	Ungült. Speicherzugr. (segment. viol.)	T
SIGTERM	Default-Signal, Termination	T
SIGUSR1	Benutzerdef. Signal 1	T
SIGUSR2	Benutzerdef. Signal 2	T
SIGCHLD	Term/Stopp eines Kindes	I
SIGCONT	Starten eines gestoppten Prozesses	I
SIGQUIT	Quit-Taste	T
SIGSTOP	Stoppen eines Prozesses	S
SIGSTP	Stopp-Taste	S
SIGTTIN	Backg.-Proz. Liest vom Terminal	S
SIGTTOU	Backg.-Proz. Schreibt auf Terminal	S

T=Termination, S=Stop, I=Ignorieren **man –s 5 signal**

Verschicken von Signalen

- int kill (pid_t pid, int sig)
  - kill(2) schickt Signal sig an Prozess mit ID pid (auch: Signal an Prozessgruppe)
- Verarbeitung von Signalen
  - Durch BS vorgegebene Aktion ausführen
  - Signal abfangen
    - Benutzerdefinierte Aktion (Signalhandler)
      - signal (int sig, handler) oder
      - int sigaction (int sig, sigaction *act)
  - Signal blockieren
    - int sigprocmask (int how, sigset_t *set)
- **SIGKILL** und **SIGSTOP** können nicht abgefangen werden.

Weitere Signalfunktionen

- alarm (int secs);
  - schickt SIGALARM nach **secs** Sekunden an aufrufenden Prozess
- pause();
  - Blockiert den aufrufenden Prozess bis zu einem Signal
- sleep (int secs);
  - Blockiert **secs** Sekunden bzw. bis zu Signal

Probleme mit Signalen

- Signalhandler asynchron zum normalen Programmfluss
  - Kann überall gestartet werden, wo Programm unterbrochen werden kann
    - MMU unterbricht mitten in Instruktion!
    - Datenstrukturen i.a. inkonsistent!
    - System-Calls durch Signale unterbrechbar
- Signale nur mit 1 Bit gespeichert
  - Eintreffen wird nicht gezählt → Sign. kann verloren gehen
  - Nur 1 Bit Information wird übertragen

→ Probleme werden mit POSIX 4 z.T. behoben

Pipes

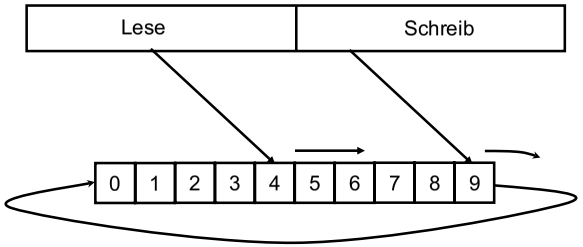
- Eine Pipe (Rohrleitung) ist eine FIFO Kommunikationsstruktur.
  - Pipes wurden traditionell mittels des Dateisystems realisiert.
  - Eine neue Realisierung benutzt sockets (→ „Verteilte Systeme“).
- Nach dem Öffnen können sie logisch wie eine Datei beschrieben bzw. gelesen werden, was für Transparenz und Überschaubarkeit sehr wichtig ist.

Pipes

Mittels einer Datei kann eine Pipe wie folgt realisiert werden:

- Der Pufferspeicher der Pipe wird durch eine feste Anzahl von Blöcken (in UNIX z.B. die 10 direkten Blöcke) der Datei realisiert, die logisch als Ringpuffer organisiert werden.
- Zudem gibt es einen Lesezeiger und einen Schreibzeiger, die auf die nächste zu lesende bzw. zu schreibende Position verweisen.
- Lese- und Schreibzeiger überholen sich nicht, sondern die entsprechenden Prozesse werden zuvor blockiert und dann wieder aufgeweckt, wenn wieder Platz ist.

Pipes



Implementierung einer pipe über inode

- Natürlich können mehrere Prozesse in die pipe schreiben oder aus ihr lesen.
- Es gibt benannte (named) und unbenannte (unnamed) pipes zur Kommunikation zwischen verwandten bzw. nicht verwandten Prozessen.

Pipes – Systemaufrufe

- unbenannte pipe mittels Filedeskriptoren
  - fds[0] für das Lesen
  - fds[1] für das Schreiben

```
int fds[2];
pipe(fds);
```

```
mkfifo(path, access-mode)
char* path, mode_t access-mode;
```

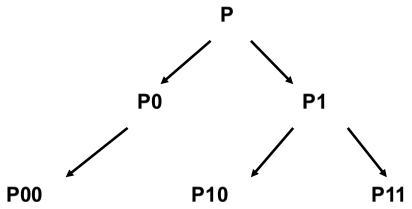
- pipe mit Namen `path`
  - Liefert eine pipe mit Namen path.
  - Die Lese- und Schreibdeskriptoren werden wie üblich durch nachfolgendes Öffnen der Datei zum Lesen bzw. Schreiben erhalten.
  - Da die pipe einen Namen hat und explizit zum Lesen/Schreiben geöffnet wird, eignet sie sich zur Kommunikation nicht verwandter Prozesse.

Pipes (UNIX)

- FIFO-Buffer fester Größe
  - Früher über Files implementiert, jetzt auch über Sockets
- Unnamed Pipe

```
Int fdptr[2];
pipe (fdptr);     liefert 2 Filedescriptors
int read (fdptr[0], char *msg, int size);
int write (fdptr[1], char *msg, int size);
int close (int fdptr[ ]);
```

Berechtigte Prozesse:
Erzeugender Prozess und Nachkommen:
P_1 vererbt auf P_{10} und P_{11}



Named Pipe

- mkfifo (char *path, mode_t mode);
  - erzeugt mit Namen (path) versehene Pipe
- Verwendung wie File:
  - int open (char *path, int flags, int mode)
  - int write (int fdes, char *buf, int size)
  - int read (int fdes, char *buf, int size)
  - int close (int fdes)
- Entfernung mit unlink

IPC in UNIX System V (Übersicht)

- UNIX System V enthält drei Mechanismen für IPC:
  - Shared Memory
  - Messages
  - Semaphores
- Existenz einer Systemtabelle mit allen Bereichen, Queues, Semaphoren
- Zugriff über durch Benutzer gewählten Key: get (key, ...);
  - durchsucht Tabelle, gibt ID von bestehendem/neuen Eintrag zurück
  - die ID's werden nicht recycled (d.h. erst nach sehr langer Zeit).
- ... ctl setzt Zugriffsrechte, löscht, fragt Informationen ab.

Shared-Memory

- int shmget (key_t key, int size, int shmflg);
  - alloziert Speicher der Größe size
- char* shmat (int shmid, char *shmaddr, int flg)
  - fügt Speicher mit ID shmid an Stelle shmaddr in Speicheraddressraum ein
- int shmdt (char* shmaddr);
  - Gibt Bindung an gemeinsamen Speicher an Addresse shmaddr frei
- int shmctl (int shmid, int cmd, ...);
  - Verschiedene Operationen auf Shared Memory (ink. Destroy)

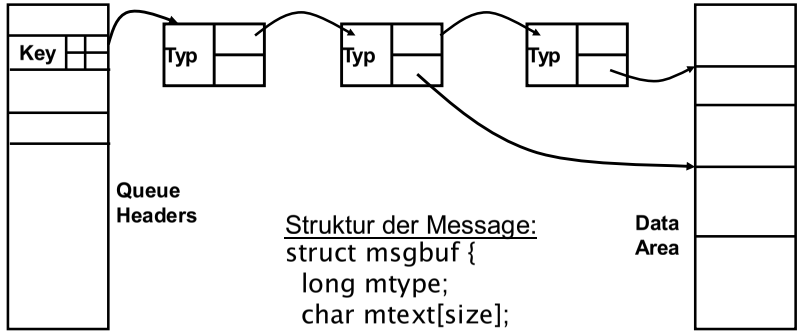
Shared Memory (UNIX)

Shared Memory wird bei fork vererbt
(d.h. nicht neu alloziert).
Bei exec und exit detached
(nicht vererbt, aber auch nicht freigegeben!)

UNIX System V messages

int msgget (key_t key, int flg);

- gibt ID auf Message Queue (mit User-Name key) zurück.



UNIX System V messages

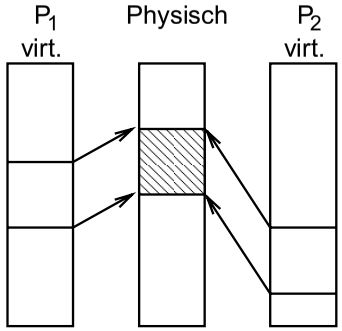
- int msgsnd (int id, msgbuf *p, int size, int flg);
  - schreibt size Bytes aus Buffer p auf Queue id
- int msgrecv (int id, msgbuf *p, int size, long type, int flag)
  - liest max. size Bytes aus Queue id in Buffer p falls Typ type kompatibel
    - type = 0: Erste Message der Queue id
    - type > 0: Erste Message mit type
    - type < 0: Erste Message mit minimalem Typ (max. - type).
      - Bsp: msgrcv (, , −7,) auf 7,17,5,2,7,2

IPC in System V

- UNIX System V enthält drei Mechanismen für IPC:
  - Shared Memory
  - Messages
  - Semaphores
- Diesen Mechanismen ist folgendes gemeinsam:
  - Eine Systemtabelle enthält alle Instanzen der Mechanismen.
  - Jeder Eintrag enthält einen Benutzer definierten (externen) numerischen Schlüssel key.
  - Ein get Systemaufruf durchsucht die Tabelle nach einem Eintrag mit Schlüssel key. Modifiziert durch flags können auch neue Einträge erzeugt oder Fehler geliefert werden, falls key schon existiert. get liefert einen eindeutigen internen Deskriptor ID, den die anderen Aufrufe benutzen.
  - Jeder Eintrag enthält Zugriffsrechte und andere Statusinformation (z.B. letzter zugreifender Prozess und Zeit des Zugriffs).
  - Jeder Mechanismus enthält einen control (ctl) Aufruf, um den Zustand eines Eintrags abzufragen oder zu verändern oder den Eintrag zu löschen.

IPC in System V – Shared Memory

- Prozesse können direkt kommunizieren, indem sie sich einen Teil ihres virtuellen Adressraums teilen und dort Daten schreiben und lesen.



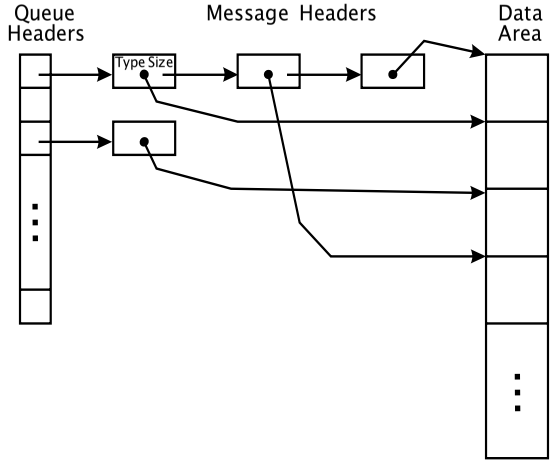
IPC in System V – Shared Memory

- Die gemeinsamen Speicherregionen werden in einer Tabelle gehalten.
- shmid = shmget (key,size,flag)
  - gibt Zugriff auf oder erzeugt eine neue gemeinsame Speicherregion
  - sucht zuerst nach einer Region key in der Tabelle
  - Falls keine da ist (und flag = IPC CREAT), wird ein neuer Eintrag erzeugt.
- virtaddr = shmat (id,addr,flags)
  - bildet eine Speicherregion in den virtuellen Adressraum des Aufrufenden ab (möglichst an der Stelle addr).
  - Der tatsächliche Ankerpunkt ist virtaddr. Ist addr = 0, ist die Wahl dem System überlassen; sonst muss der Aufrufer selbst Konflikte (z.B. mit dem Stack) vermeiden.
  - Seitentabellen für die Region werden erst beim ersten shmat erzeugt.

IPC in System V – Shared Memory

- virtaddr = shmat (id,addr,flags)
  - bildet eine Speicherregion in den virtuellen Adressraum des Aufrufenden ab (möglichst an der Stelle addr). Der tatsächliche Ankerpunkt ist virtaddr. Ist addr = 0, ist die Wahl dem System überlassen; sonst muss der Aufrufer selbst Konflikte (z.B. mit dem Stack) vermeiden.
  - Seitentabellen für die Region werden erst beim ersten shmat erzeugt.
- shmdt(addr)
  - löst die Bindung einer Speicherregion an die virtuelle Adresse addr.
- shmctl(id,cmd,shmstatbuf)
  - kann den Status (z.B. Zugriffsrechte) einer Region ändern. shmstatbuf enthält den neuen Status.
  - shmctl(id,IPC RMID,0) gibt die Region zum Auflösen frei (dies geschieht erst beim letzten shmdt).

IPC in System V – Messages



IPC in System V – Messages

- Messages dienen zum Austausch formatierter Daten.
- msgqid = msgget (key,flag)
  - liefert (und falls nötig erzeugt) einen Deskriptor auf eine Liste von Nachrichten (Briefkasten).
- msgsnd (msgqid,msg,count,flag)
  - sendet Nachricht msg der Länge count in die WS msgqid.
  - msg selbst ist eine Struktur aus einem Integer Typ und der Nachricht als char array.
  - Es werden Prozesse geweckt, die auf eine Nachricht von diesem Typ warten.

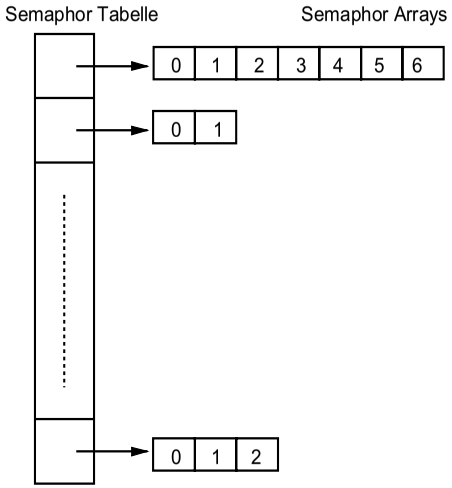
IPC in System V – Messages

- count = msgrcv (id,msg,maxcount,type,flag)
  - empfängt eine Nachricht des Typs type und der max. Länge maxcount von Schlange id, die in der Struktur msg abgelegt werden soll.
  - count ist die aktuelle Anzahl übermittelter Bytes.
  - Es wird die erste Nachricht übermittelt, deren Typ = type ist; falls type = 0, wird die erste Nachricht übermittelt.
- msgctl(id,cmd,mstatbuf)
  - liest oder setzt den Status eines Nachrichtendeskriptors.
- msgctl(msgid,IPC RMID,0)
  - löscht einen Deskriptor (eine NachrichtenWS).

IPC in System V – Semaphore

- Ein System V Semaphor besteht aus:
  - Dem Wert des Semaphors.
  - Der Prozess ID des letzten Prozesses, der das Semaphor manipuliert hat.
  - Der WS der Prozesse, die darauf warten, dass sich der Wert des Semaphors erhöht.
  - Der WS der Prozesse, die darauf warten, dass der Wert des Semaphors Null wird.

IPC in System V – Semaphore



IPC in System V – Semaphore

- Semget id = semget (key,count,flag);
  - gibt Zugriff auf einen Eintrag in der Semaphor-Tabelle mit einem Array aus Semaphoren mit count Elementen. key ist ein Benutzer definierter Name für den Eintrag und flag spezifiziert Zugriffsarten (z.B. ob Eintrag erzeugt werden soll, falls er noch nicht existiert).
  - Der Tabelleneintrag enthält auch die Zeiten der letzten semop und semctl Aufrufe sowie Zugriffsrechte für semop Aufrufe.
- semctl
  - Verschiedene Kontrolloperationen auf der Menge.
  - u.a. damit auch Löschen von Semaphoren möglich

IPC in System V – Semaphore

- Semop oldval = semop (id,oplist,count)
  - id ist der Deskriptor aus semget,
  - oplist ist ein Zeiger auf ein Array der Länge count aus Semaphor-Operationen.
  - oldval ist der Wert des letzten Semaphors in der Liste vor der Operation.
  - Jedes Element von oplist enthält:
    - den Index des Semaphors im Array
    - die Semaphor-Operation in Form einer Konstanten Op die zum Sem-Wert addiert wird
    - Flags

IPC in System V – Semaphore

- Als Resultat einer semop ändert der Kern die Werte der beteiligten Semaphore
  - Op > 0: Alle Prozesse werden geweckt, die darauf warten.
  - Op = 0: Der Prozess wartet, bis der Semaphor-Wert = 0 wird.
  - Op < 0: Wert = Wert + Op, falls Wert + Op > 0
    - Sonst warten bis Wert groß genug
- Wert + Op = 0: Die auf 0 wartenden Prozesse werden geweckt.
- In jedem Fall wird eine semop atomar durchgeführt.
  - Blockiert der Prozess bei einer Teiloperation → Originalwerte werden auf dem Semaphor-Array temporär wieder hergestellt
  - Dies macht System V Semaphore sehr kompliziert und aufwändig