1 Ablauf der Signalverarbeitung im Systemkern (2)

(2) Signal verarbeiten

- jeder Prozeß prüft vor Verlassen des Systemkerns (am Ende eines Systemaufrufs) und vor sowie nach einem Blockieren (*sleep( )*), ob Signale für ihn vorliegen
- blockierte Signale werden nicht bearbeitet, bleiben aber in der Signal-Bitmaske `p_sig` vorgemerkt
- für alle anderen Signale wird der Reihe nach die Signalbearbeitung veranlaßt
  - ◆ *core-dump* erzeugen (wenn nötig) und *exit( )* aufrufen, oder
  - ◆ Signalbearbeitungsfunktion aufsetzen
  - ◆ während des Ablaufs einer Signalbearbeitungsfunktion können andere Signale eintreffen, die (sofern nicht blockiert) die Bearbeitung des ersten Signals sofort unterbrechen

4 Beispiel für die Signalsemantik in System V (2)

```
void main()
{
    signal(SIGQUIT, trap_quit);
    setjmp(return_pt);
    signal(SIGINT, trap_int);

    for(i=1; i!=0; i++) {
    if(i%100000 == 0) {
        putchar('.');
        fflush(stdout);
        }
    }
    printf("Fertig!\n");
}
```

1 Communication Domain und Protokoll

- **Communication Domain** legt die **Protokoll-Familie**, in der die Kommunikation stattfindet, fest
- durch die Protokoll-Familie wird gleichzeitig auch die Adressierungsstruktur **(Adreß-Familie)** festgelegt
  (war unabhängig geplant, wurde aber nie getrennt)
- das **Protokoll**-Attribut wählt das Protokoll innerhalb der Familie aus
- ursprünglich (bis BSD 4.3) existierten nur zwei Communication Domains
  - ➤ UNIX-Domain (AF_UNIX)
  - ➤ Internet-Domain (AF_INET)
- nur AF_INET ist generell vorhanden
  daneben derzeit ca. 25 AF definiert
  (ISO-Protokolle, DECnet, SNA, Aplletalk, …)

3 Client-Server Modell

★ Ein ***Server*** ist ein Programm, das einen Dienst ***(Service)*** anbietet, der über einen Kommunikationsmechanismus erreichbar ist

■ Server

- **akzeptieren Anforderungen**, die von der Kommunikationsschnittstelle kommen
- **führen** ihren angebotenen **Dienst aus**
- **schicken** das **Ergebnis zurück** zum Sender der Anforderung
- *Server* sind normalerweise als normale Benutzerprozesse realisiert

■ Client

- ein Programm wird ein ***Client***, sobald es
  - ➤ eine **Anforderung an einen Server** schickt und
  - ➤ auf eine Antwort wartet

8 Stream-Sockets — Systemschnittstelle

- Verbindungsaufbau

listen(2) teilt dem System mit, daß man bereit ist, Verbindungen auf dem Socket entgegenzunehmen

accept(2) nimmt eine Verbindung an und generiert einen neuen Socket (***s_new***), der für die Kommunikation verwendet werden kann - steht kein Verbindungwunsch an, blockiert *accept*

connect(2) baut die Verbindung vom ***client*** zum ***server*** auf

- Kommunikation

read(2), write(2) verhalten sich wie gewohnt

send(2), recv(2) wie *write( )* bzw. *read(2)* + zusätzliche Funktionalität (Daten ansehen ohne zu lesen*, out-of-band-data, ...)*.

- Verbindungsabbau

close(2) Socket schließen

shutdown(2) Elimieren von bereitliegenden Daten vor einem *close( )*

1 IPC als Systemschnittstelle

- IPC wird in MACH auch für Systemaufrufe an Objekte des MACH-Kerns verwendet
  - ◆ Kern-Objekten (z. B. Tasks, Threads, E/A-Treiber) sind Ports zugeordnet
  - ◆ Operationen auf Kern-Objekten werden durch Mitteilungen an diese Ports ausgeführt

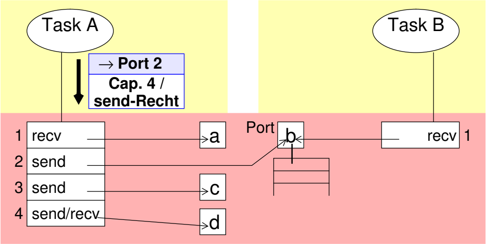
- Beispiel:
  Beim Erzeugen eines Threads erhält man ein Senderecht auf den *Thread-Port*.
  Durch Mitteilungen an diesen Port kann der Thread suspendiert, deblockiert, vernichtet, ... werden.

4 Übertragen großer Datenbereiche

- Task A sendet Mitteilung an Port von Task B
- der Datenbereich der Mitteilung wird temporär in den Adreßraum des MACH-Kerns abgebildet (= Senden an Port)
- im Adreßraum der Task A wird der Datenbereich als *copy on write* markiert
  (d. h. für Seiten, auf die nachfolgend schreibend durch A zugegriffen wird, wird eine Kopie erzeugt und diese wird in den virtuellen Adreßraum von A übernommen)
- Task B empfängt die Mitteilung, indem der Datenbereich in den virt. Adreßraum von B abgebildet wird
- Aus dem Adreßraum des MACH-Kerns wird der Datenbereich wieder entfernt
- Auch bei Task B ist der Datenbereich als *copy on write* markiert

3 Capabilities

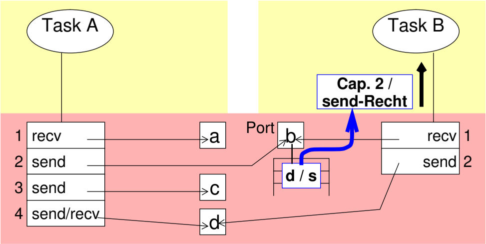
- MACH-Kern verwaltet pro Task eine Port-Capability-Liste
- auf Anwendungsebene werden Indizes in diese Liste verwendet (wie UNIX-Filedescriptoren)
- Übertragung einer Capability in einer Mitteilung



Task A sendet Mitteilung mit Capability 4 an Port 2 (= b)

3 Capabilities

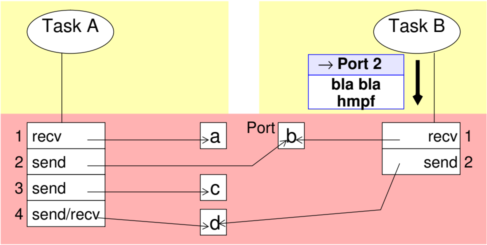
- MACH-Kern verwaltet pro Task eine Port-Capability-Liste
- auf Anwendungsebene werden Indizes in diese Liste verwendet (wie UNIX-Filedescriptoren)
- Übertragung einer Capability in einer Mitteilung



Task B empfängt send-Recht auf Port d als Capability 2

3 Capabilities

- MACH-Kern verwaltet pro Task eine Port-Capability-Liste
- auf Anwendungsebene werden Indizes in diese Liste verwendet (wie UNIX-Filedescriptoren)
- Übertragung einer Capability in einer Mitteilung



Task B kann jetzt über Capability 2 Mitteilung an Task A senden