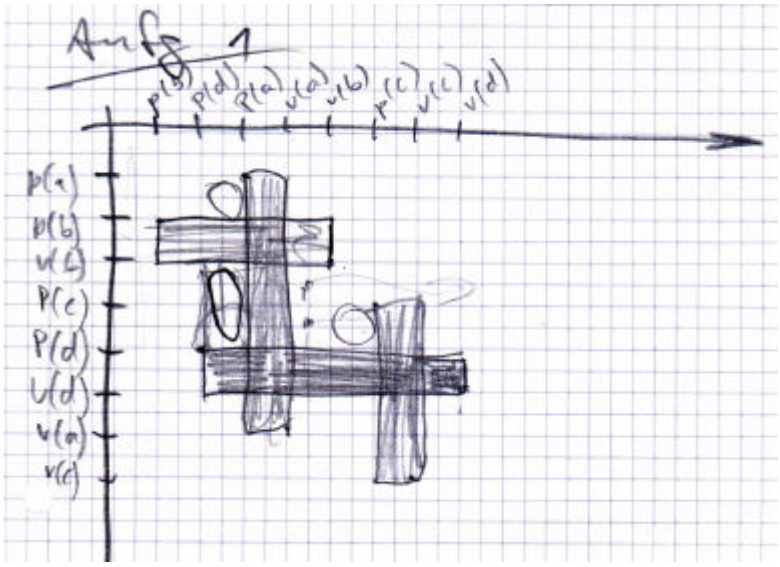
CS-Zusammenfassung für Prüfung 3 vom 24. 6. 2002

Deadlock



Im Beispiel gibt es 3 Deadlocks

Unterschied zwischen Blockieren, Verklemmen und Verhungenrn= -> Band 1 S. 203 unten & 204 oben.

Deadlock-sichere Systeme.
Wenn ein Computersystem sechs E/A-Kanäle besitzt und n Prozesse ausgeführt werden, wobei ein Prozess 2 Kanäle benötigt, die bei Bedarf reserviert werden. Dann darf n für ein sicheres System maximal 3 betragen.

Socket-Programmierung

Falls der Server von TCP auf UDP umzustellen ist, muss folgendes am Programm geändert werden:

- connect(), accept(), shutdown(), listen() werden weggelassen
- Client muss Aufruf nach einer gewissen Zeit wiederholen, da UDP verbindungslos ist

Multi-ThreadedApplikation wird auf einem Server implementiert. Es können so viele Clientverbindungen gleichzeitig bedient werden, wie das Betriebssystem Prozesse erlaubt, da bei jeder Verbindung ein neuer Prozess gestartet wird.
Es können aber keine Verbindungsaufnahmen pendent sein, da Elternprozess wenn er dafür bereit ist einen neuen Prozess erstellt, der die Verbindung entgegen nimmt.
Listen() hat zuweilen Parameter -> gibt an wie viele Verbindungen pendent sind

Port Nummer: Für eine Applikation sollte eine Portnummer über 1024 gewählt werden. Besser Portmapper verwenden.

Bei welchen Danten, die in einer heterogenen Umgebung ausgetauscht werden, ist dies problemlos möglich?

- `signed int a; unsignet int b; long c; short d;`
  // nicht ganz problemlos, da int verschieden gross ist, je nach Umgebung z.B. 2 bzw. 4 Byte
- `char* s = „Hello World!\n”;`
  // Ja, Zeichen sind alle 1 Byte lang. Es muss nnur angegeben werden, wie viel gelesen werden soll.
- `Struct f {char u; int v;};`
  // Nein, die einzelnen Parameter müssen eventuell auf Wort-Grenze ausgerichtet sein. -> char wäre dann 2 Byte lang und nicht nur 1

Pipe

```
Main() {
     int pipefd[2], n;
     char buff[100];
     pipe( pipefd ); // Erzeugen einer Pipe liefert 2
     //Deskriptoren für die beiden Enden der Pipe
     printf( „read fd = %d, write fd = %d\n”, pipefd[0],//read
     pipefd[1]);//write
     write( pipefd[1], “Hello world\n”, 12 );
     n = read(pipefd[0], buff, sizeof( buff ));
     write(1, buff, n );
     exit(0);
```

In obigem Beispiel findet eine Interprozess-Kommunikation statt, da das Programm ein Prozess ist, der mit sich selber über eine Pipe kommuniziert.
Die Variable **n** gibt die Anzahl der gelesenen Zeichen von read.
Close() fehlt noch im Beispiel. Mit close() kann das benutzte Ende der Pipe geschlossen werden.

Alarm(3) erstell Alarmsignal, welches nach 3 Sekunden erscheint, alarm(0) deaktiviert alarm. Man benutzt in Echtzeitsystemen alarm(x), um zu sagen, wie lange eine Funktion/Methode laufen darf.
Wenn im Programmablauf ctrl-c gedrückt wird, wird die Catcher-Funktoin d.h. signal(SIGINT, catcher) aufgerufen. => ctrl-c kann durch catcher-Methode abgefangen werden.
Bei ctrl-z wird gestoppt
Siehe auch Anhang E.22

Prozessvergabelung

Im Quellcode 3 mal `fork()` -> 3 Kindprozesse maximal

RPC

Wechsel von TCP auf UDP: UDP verbindungslos: Falls ein Packet verloren oder braucht es übermässig lange zum Server weiss der Client nicht, ob er es nochmals schicken muss (ev. Mehrfacher aufruf)

Wenn eine fernaufrufbare Prozedur überarbeitet wird und die Aufrufdefinition (Parameter, Rückgabewert) jedoch gleich bleibt, muss an der XDR-Definitionsdatei nichts geändert werden, da alle Parameter noch gleich bleiben. Ausser wenn die alte Version der Prozedur immer noch gebraucht wird und die neue Prozedur etwas anderes berechnet als der alte Client will muss die Versionsnummer angepasst werden.

Probleme in der Rechnerkommunikation, die von XDR vermieden werden können: Die verschiedenen HLL-Datentypen werdenauf allen Systemen gleich verwendet (Little-/Bigendian). Anpassung auf Wordgrenzen (argument rule) wird überall richtig verwendet. -> Zeichensatz, String-Darstellung

Muss Programmnummer, Versionsnummer und Prozedurnummer geändert werden, von TCP nach UDP? – Nein.

Server bietet eine Reihe von Diensten via RPC an. Was muss Client-Applikation alles Wissen, um diese nutzen zu können -> Siehe Buch S. 201

Falls der Server nicht läuft, erscheint beim Aufruf des Clients `remote System error – connection refused`. Falls im Client ein ungültiger Rechnername angegeben wurde erscheint die Meldung `unknown host`

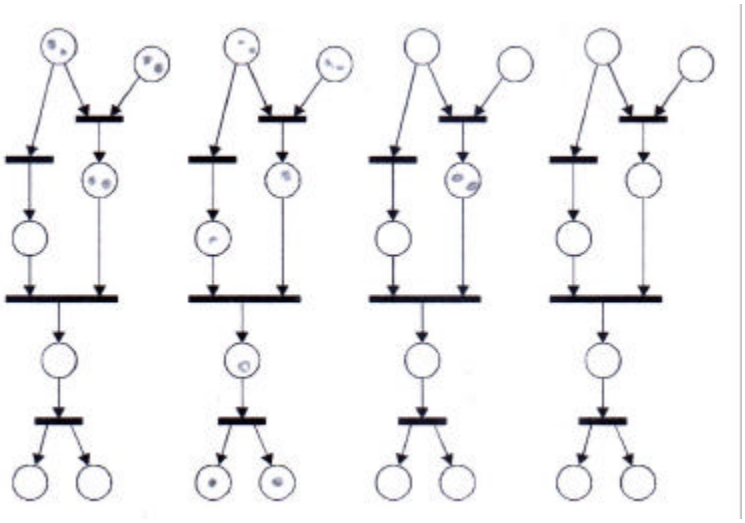
Multitasking unter UNIX

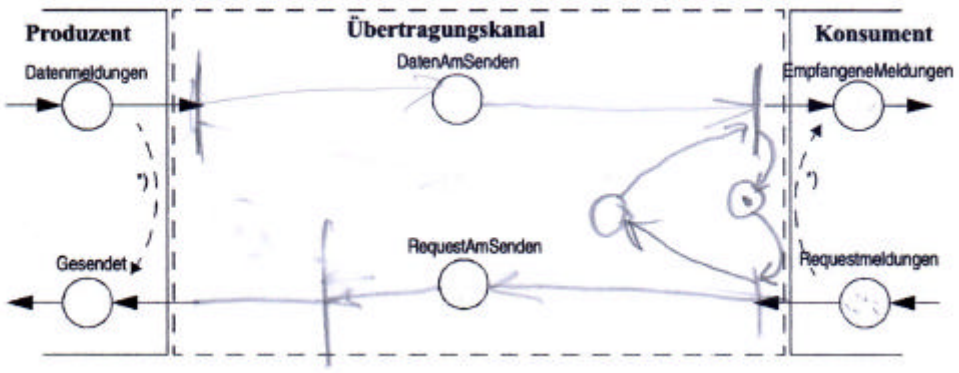
```
#include <stdio.h>
main() {
     static char* cmd[]={“who”, “ls”, “date”};
     int I;
     while(1) {
          printf( “0=who, 1=ls, 2=date : “);
          scanf(“%d”, &I);
          if( i<3) execlp( cmd[i], cmd[i], 0 );
          else printf(“command not found\n”);
     }
}
```

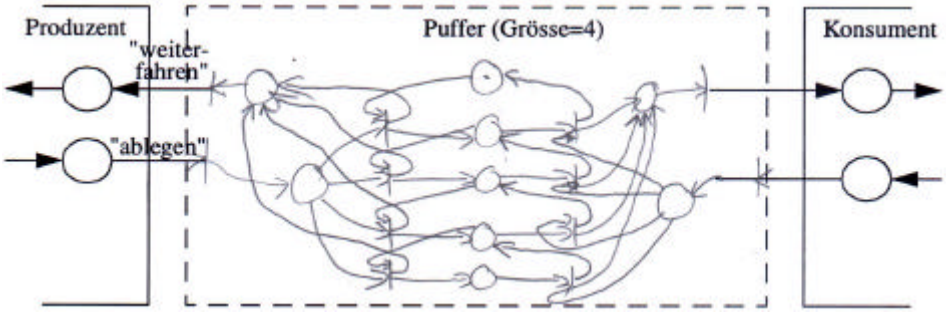
Das Programm läuft nicht wie vorgesehen, da `fork()` Befehl fehlt, um die Kindprozesse aufzurufen, welche die Programme ausführen, während der Elternprozess der nächsten Kindprozesse starten kann.
Zudem weiss man nicht, was passiert, wenn **–1** eingegeben wird.

Petri-Netze

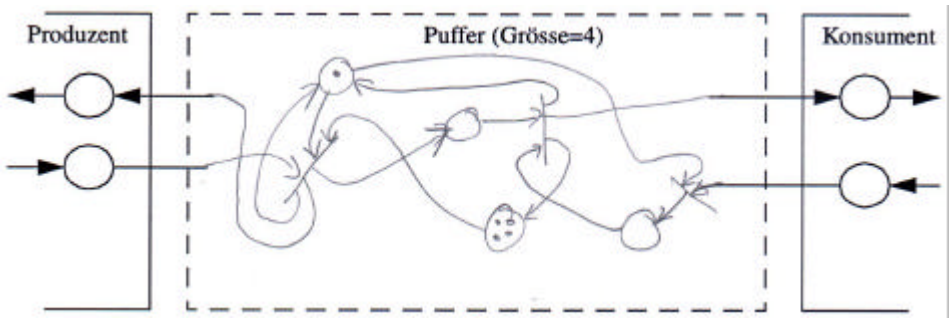
Beispiel 1:



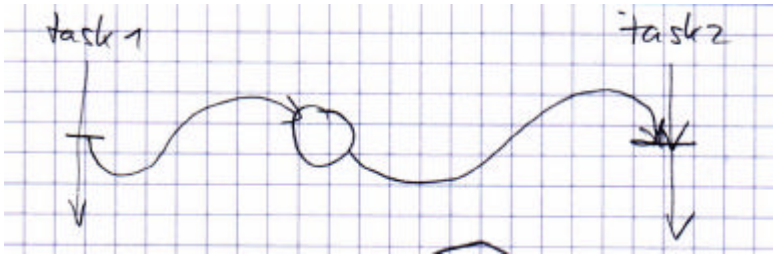




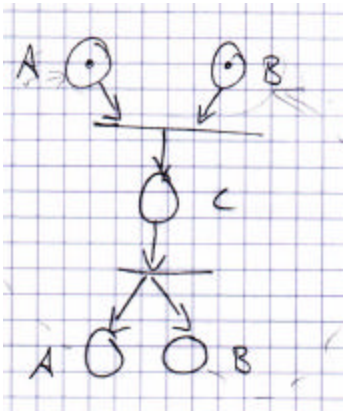
Grundlösung für Pufferung mit grösse 4



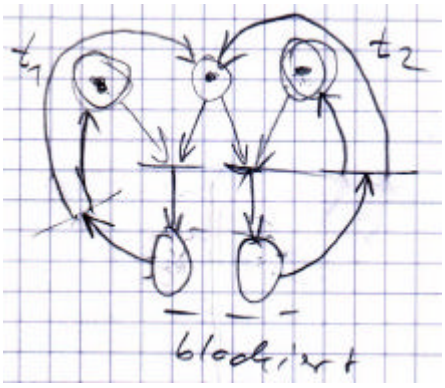
Vollständige Lösung mit wechselseitigem Ausschluss beim Zugriff auf den Pufferspeicher



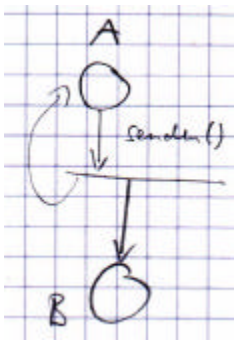
einseitige Ablauf-Synchronisation von 2 Tasks mit Hilfe eines Semaphors



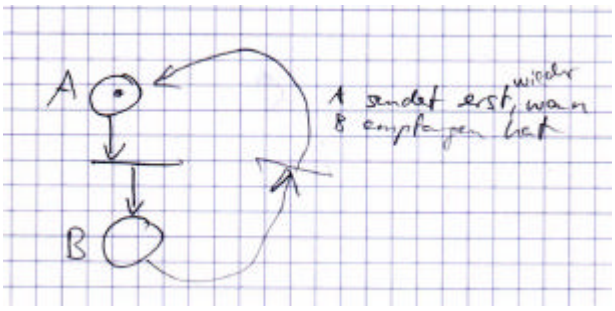
Rendevous einer Task bei einer anderen



Rendevous von 2 Tasks bei einer anderen. Dabei können Task A und B je für sich ein Rendevous mit Task C eingehen



Meldungsübermittlung ohne Warten beim Senden und Empfangen (A sendet Meldungen an B)



Meldungsübermittlung mit Warten beim Senden und Empfangen (A sendet Meldungen an B)

