Hochschule Esslingen – University of Applied Sciences

Wintersemester 2010/2011		**Zahl der Seiten: 14; Seite 1**	
Fakultät:	Informationstechnik	**Semester:**	IT3A, IT3B
Prüfungsfach:	Betriebssysteme (1) KTB/TIB/SWB 3071 (2) KTB/TIB/SWB 3072		
Dozent:	Seiffert	**Fachnummer:**	
Hilfsmittel:	keine	**Zeit:**	90 Minuten
Name:		**Matrikelnummer:**	

Vorbemerkung: *der freigelassene Platz sollte in der Regel zur Beantwortung der Fragen ausreichen und ist vorrangig zu nutzen. Bei Bedarf verwenden Sie bitte die Rückseiten und vermerken Sie dies auf der Vorderseite. Bitte tragen Sie auf jeder Seite Ihre Matrikelnummer ein und benutzen Sie keine roten Farbstifte!*

Viel Erfolg!

Maximal erreichbare Gesamtpunktzahl: 104

Aufgabe 1 *Grundlagen* (15 Punkte)

(a) Was sind die zentralen Aufgaben eines Betriebssystems? Nennen Sie diese drei Aufgabenbereiche und geben Sie je ein Beispiel, was das Betriebssystem in diesem Bereich leistet

Abstraktion der Hardware • Lies einen Block von Floppy • Darstellung eines Geräts als „special file“	3
Verwaltung der Betriebsmittel • CPU-Zuteilung an Prozesse (Scheduling) • Druckerverwaltung	3
Betriebssystemdienste / Services • Sende eine Nachricht an einen (anderen) Prozess • Kommandozeileninterpreter	3

(b) Nennen Sie drei zentrale Abstraktionen, die ein Betriebssystem dem Programmierer zur Verfügung stellt:

Prozesse Adressräume / Virtueller Speicher Dateien / Dateisysteme	3

(c) Wie kann ein Benutzerprozess (user mode) sicher(*) auf privilegierte Funktionen des Prozessors (supervisor mode) zugreifen, z.B. zur Ausführung von System Calls? Erklären Sie in Stichworten!
(*)Ein Benutzerprozess darf nicht selbst Privilegien erlangen können

„Trap“ ist eine Instruktion des Prozessors, die den Prozessor in den privilegierten Zustand (supervisor mode) umschaltet, dann an einer festen physischen Speicheradresse in einer Tabelle (vom Betriebssystem dort hinterlegt) eine Adresse ausliest und zum dort befindlichen Code (trap handler) verzweigt. Diese Instruktion wird zur Implementierung von Systemaufrufen benutzt. So ist es nicht möglich, dass eine nicht-privilegierte Anwendung (im user mode) eigenen Code im supervisor mode ausführt.	3

Aufgabe 2 *Prozesse* (28 Punkte)

(a) Erklären Sie die folgenden Begriffe in Stichworten:

Prozess	Ein Prozess ist ein in der Ausführung befindliches Programm	2
Scheduler	Der Scheduler (zentrale BS-Komponente) teilt die physische CPU nach verschiedenen Kriterien abwechselnd den Prozessen zu	2
Context Switch	Ein Context Switch speichert den aktuellen Ausführungszustand eines Prozesses von der physischen CPU (Register etc), lädt den gespeicherten Ausführungszustand eines anderen Prozesses und setzt diesen fort	2

Labor 2 Aufgabe 1

(b) Im Win32 API werden neue Prozesse mit **CreateProcess** erzeugt: „The **CreateProcess** function creates a new process and its primary thread. The new process executes the specified executable file.“ In UNIX benötigt man zwei Systemaufrufe. Welche? Was machen die beiden Systemaufrufe jeweils?

fork() - erzeugt einen neuen Prozess, als identische Kopie des aufrufenden Prozesses	2
exec() - lädt ein Programm in den laufenden Prozess beginnt dessen Ausführung	2

(c) Mit welchem Kommando können Sie einem Prozess ein Signal senden?

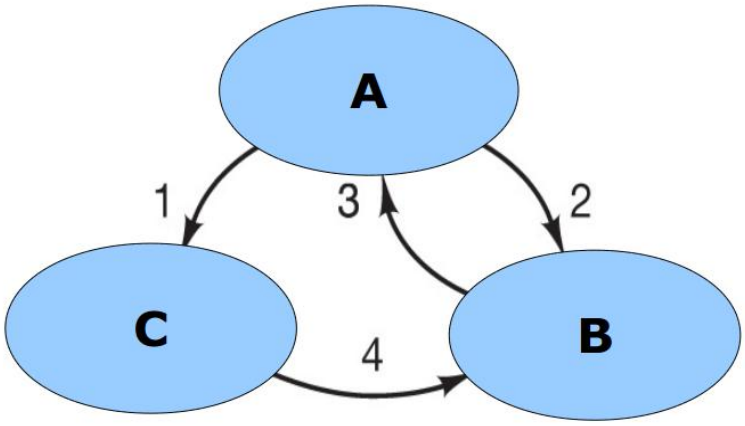
kill	1

(d) Wie können Sie in einem Prozess ein Signal „fangen“?

Labor 2 Aufgabe 1

signal handler installieren mit signal() Systemaufruf	1

(e) Zur Verwaltung von Prozessen betrachtet das Betriebssystem verschiedene Zustände, in denen sich ein Prozess jeweils befinden kann. Die Graphik stellt die drei zentralen Prozesszustände (A,B,C) und die möglichen Zustandsübergänge (1,2,3,4) dar.



(e1) Benennen und erläutern (Stichworte) Sie die drei Zustände:

A	rechnend (running) – der Prozess wird im Moment auf der physischen CPU ausgeführt	2
B	rechenbereit (ready) – der Prozess könnte rechnen, wurde aber gestoppt, um einem anderen Prozess die physische CPU zu geben	2
C	blockiert (blocked) – der Prozess ist nicht ablauffähig bis ein bestimmtes externes Ereignis eintritt, z.B. eine Eingabe	2

(e2) Erklären Sie die vier Zustandsübergänge in Stichworten:

1	Laufender Prozess blockiert, weil er z.B. eine Eingabe anfordert (blocking system call)	2
2	Scheduler stoppt den laufenden Prozess, um einen anderen Prozess ablaufen zu lassen	2
3	Scheduler lässt den Prozess nun weiterlaufen	2
4	Der Prozess wird wieder ablauffähig, da z.B. die erwartete Eingabe erfolgt ist	2

(e3) Warum gibt es keinen Zustandsübergang von B nach C? Erklären Sie in Stichworten:

Im Zustand B „steht“ der Prozess (führt keine Befehle aus) und kann daher auch keine Eingabe anfordern oder andere Aktionen auslösen, die ihn blockieren würden	2

Labor 3

Aufgabe 3 *Interprozesskommunikation* (10 Punkte)

(a) Wenn mehrere Prozesse auf gemeinsame Ressourcen zugreifen, müssen sie sich synchronisieren. Eine Möglichkeit dazu ist die Verwendung von Semaphoren. Ein Semaphor ist im wesentlichen eine spezielle, systemweite Variable mit zwei Zugriffsoperation. Nennen Sie diese Operationen und beschreiben Sie ihre Funktion stichwortartig:

down(S) – falls S>0, dekrementiere Semaphor S, falls S=0 blockiere (warte bis wieder S>0)	2
up(S) – inkrementiere S; falls Prozesse auf diesen Semaphor warten, wecke einen davon auf	2

(b) Wie heißt ein Semaphor, der nur zwei Werte („locked“ oder „unlocked“) annehmen kann? Wozu werden solche Semaphore meist verwendet?

[..] Mutexe dienen zur Verwaltung des wechselseitigen Ausschlusses [..] Ein Mutex ist eine Variable, die zwei Zustände annehmen kann: nicht gesperrt und gesperrt. [..] Zwei Prozeduren, mutex_lock() und mutex_unlock(), werden für das Arbeiten mit Mutexen verwendet. [..]	2

(c) Prozesse können auch Speicherbereiche gemeinsam benutzen. Um das IPC-Elements „shared memory“ in einem Prozess verwenden zu können, gibt es eine Reihe von Systemaufrufen. Erläutern Sie in Stichworten den Unterschied zwischen den beiden Systemaufrufen shmget() und shmat():

shmget() gibt den Identifikator des gemeinsamen Speichersegments zurück, der dem angegebenen Schlüssel entspricht Die Funktion shmat() bindet den durch den Identifikator bezeichneten Speicher in den Adressraum des Prozesses ein - aus Programmsicht ähnlich zu malloc()	2

(d) Mit welchem Linux-Kommando kann man die im System verwendeten IPC-Elemente anzeigen? Mit welchem Kommando kann man sie löschen?

ipcs – anzeigen ipcrm – löschen	1 1

Aufgabe 4 *Informationssicherheit* (10 Punkte)

In UNIX wird die Autorisierung zum Zugriff auf Dateien durch das Zusammenspiel von Benutzerinformation (UID, GID) und zu den Dateien gehörigen Zugriffsrechten verwaltet.

(a) Welche Zugriffsrechte für Dateien kennt UNIX?

Lesen [r] – Schreiben [w] – Ausführen [x]	2

(b) Betrachten Sie die Ausgabe des folgenden `ls` Kommandos. Welchem Benutzer gehört die Datei `test`? Welcher Gruppe?

```
> ls -al test
-rwxr-xr-- 1 verwalter admin 119 2010-06-04 14:25 test
```

Eigentümer: verwalter Gruppe: admin	2

(c) Sie sind angemeldet als Benutzer `user1`. Informationen zur UserID entnehmen Sie der Ausgabe des Kommandos `id`. Was dürfen Sie mit der Datei `test` tun? Warum?

```
> id
uid=501(user1) gid=500(users)
groups=24(cdrom),46(plugdev),106(lpadmin),121(admin),500(users)
```

`user1` gehört zur Gruppe `admin` und darf daher die Datei `test` lesen und ausführen, aber nicht verändern	2

(d) Mit welchem Kommando kann man den Benutzer `user1` zum Eigentümer der Datei `test` machen? Können Sie als Benutzer `user1` das selbst tun?

`chown user1 test` Nein, `user1` hat nicht die Berechtigung – das kann nur der Eigentümer oder `root`, auch mittels `sudo`	2

(e) Betrachten Sie das folgende ls Kommando und seine Ausgabe. Was bedeuten die Zugriffsrechte für /usr/bin/passwd?

```
> ls -al /usr/bin/passwd
-rwsr-xr-x 1 root root 37140 2010-01-26 18:09 /usr/bin/passwd
```

Jeder darf das Programm ausführen. Zusätzlich darf jede/r das Programm "lesen" - sprich irgendwo anders hinkopieren. Der User root darf es verändern. Da das „setuid“-Bit ist gesetzt ist, läuft das Programm aber immer mit der effektiven UID „root“, ist also privilegiert.	2

Aufgabe 5 *Shell-Programmierung* (15 Punkte)

Labor 1 Aufgabe 2

Gegeben ist das untenstehende Shell-Skript, das einige Lücken aufweist und in dem einige Stellen markiert sind. Beantworten Sie dazu die nachfolgenden Fragen.

Hinweis: Es sind 20 Punkte möglich, die Maximalpunktzahl für die Aufgabe ist aber 15. Sie müssen nicht also nicht alle Fragen beantworten können.

```
#     Lücke L1
#
# convert_images.sh SOURCE TARGET
#    konvertiere alle Bilddateien im Verzeichnis
#    SOURCE in JPEG-Dateien der maximalen Auflösung
#    1200x900 in das (neu angelegte) Verzeichnis
#    TARGET. Die Verzeichnisstruktur wird kopiert.

# ist die Anzahl der Aufrufparameter korrekt?
if     Lücke L2     then
    echo USAGE: $0 source-directory target-directory
    exit 1
fi

InDir=$1
OutDir=$2

# existiert das Eingabeverzeichnis?
if      Lücke L3      then
    echo ERROR: source directory $InDir not found
    exit -1
fi
# das Ausgabeverzeichnis darf *nicht* existieren!
if      Lücke L4      then
    echo ERROR: target directory $OutDir already exists
    exit -1
fi

# erzeuge die komplette Verzeichnisstruktur in der Ausgabe
find $InDir -type d | sed s:$InDir:$OutDir: | xargs mkdir

# definiere eine Funktion do_convert() zur Konvertierung
# einer einzelnen Datei
      Lücke L5      {
   INFILE=$1
   OUTFILE=$(echo $1 | sed s:$2:$3: | sed s:'\.[A-Za-z]*$':'.jpg':)
   echo converting $INFILE to $OUTFILE
   convert $INFILE -resize 1200x900 -quality 75 $OUTFILE
}

# Konvertiere alle Bilddateien (JPG JPEG BMP jpg jpeg bmp)
# im Eingabeverzeichnis
EXT='.\(JPG\|JPEG\|BMP\|jpg\|jpeg\|bmp\)$'
find $InDir -type f | grep -e $EXT | \
    Lücke L6 \
      do do_convert $FILENAME $InDir $OutDir; \
      done

# so, das war's... Skript beenden als „fehlerfrei gelaufen“ beenden
    Lücke L7
```

Zeile Z1

Zeile Z2

Zeile Z3

(a) Es handelt sich um ein bash-Skript. Was steht in Lücke L1?

#!/bin/bash	1

(b) Überprüfung der Anzahl der Aufrufparameter. Was steht in Lücke L2?

[[$# -ne 2]];	2

(c) Das Eingabeverzeichnis muss existieren und es muss sich um ein Verzeichnis handeln. Was steht in Lücke L3?

[[! -d $InDir]];	2

(d) Das Ausgebeverzeichnis darf noch nicht existieren und auch keine zufällig gleichnamige andere Datei. Damit soll versehentliches Überschreiben vermieden werden. Was steht in Lücke L4?

[[-e $OutDir]];	2

(e) Die Zeile Z1 erzeugt die komplette Verzeichnisstruktur innerhalb des Eingabeverzeichnisses im neuen Ausgabeverzeichnis. Erklären Sie in Stichworten die Funktionweise von Zeile Z1.

find $InDir -type d gibt alle Verzeichnisse (-type d) beginnend im Eingabeverzeichnis in eine Pipe aus. sed s:$InDir:$OutDir: liest von der Pipe die Verzeichnisnamen und ersetzt in jedem dieser Strings den Namen des Eingabeverzeichnisses mit dem des Ausgabeverzeichnisses. Wieder Ausgabe auf Pipe. xargs mkdir liest von dier Pipe die neuen Verzeichnisnamen und führt für jeden Namen das Kommando mkdir aus	3

(f) Zur Konvertierung einer einzelnen Datei wird eine Funktion definiert. Was steht in Lücke L5?

function do_convert()	2

(g) Zur Erklärung der Zeile Z2 geben Sie bitte die jeweilige Ausgabe nach den Kommandos in jeder Zeile an:

`> date=DATUM` `> echo ${date}`	DATUM	1
`> echo date`	date	1
`> echo $(date)`	Sat Jan 8 13:41:28 CET 2011	1
`> DATE=$(date)` `> echo $DATE`	Sat Jan 8 13:42:43 CET 2011	1
`> echo $date`	DATUM	1

(h) In Zeile Z3 werden alle Bilddateien gesucht und in einer Schleife konvertiert. Was steht in Lücke L6?

while read FILENAME;	2

(i) So, das war's. Das Shell-Skript soll sich nun geordnet beenden und einen erfolgreichen Ablauf melden. Was steht in Lücke L7?

exit 0	1

Aufgabe 6 *Dateien und Dateisysteme* (16 Punkte)

(a) Gegeben ist das Programm `dup` mit dem abgebildeten Quelltext `dup.c`

```
#include <stdio.h>
main()
 char ch;
 ch=getchar();
 while ( ch > 0 ) {
 putchar(ch);
 putchar(ch);
 ch=getchar();
 }
}
```

Geben Sie zu den Eingaben in der linken Spalte jeweils die zu erwartende Ausgabe nach Drücken von <ENTER> an:

`> dup` `abcd^D`	aabbccdd	1

`> cat > xyz`
`abc^D`
`> cat xyz`
	abc	1

`> cat >> xyz`
`abc^D`
`> cat xyz`
	abcabc	1

`> cat xyz | dup`	aabbccaabbcc	1

(b) Einige Betriebssysteme stellen einen Systemaufruf `rename()` zur Verfügung, um Dateien einen neuen Namen zu geben. Beschreiben Sie den wesentlichen Unterschied zwischen diesem Aufruf und dem Kopieren der Datei auf eine neue Datei und anschließendes Löschen der alten Datei:

rename() überschreibt (oder erzeugt/löscht) lediglich den Namen der Datei im Verzeichniseintrag. Insbesondere die zur Datei gehörigen Datenblöcke werden nicht kopiert.	2

(c) Erläutern Sie den Unterschied zwischen den beiden foldenden Kommandos:

```
> ln xyz xyz1
> ln -s xyz xyz2
```

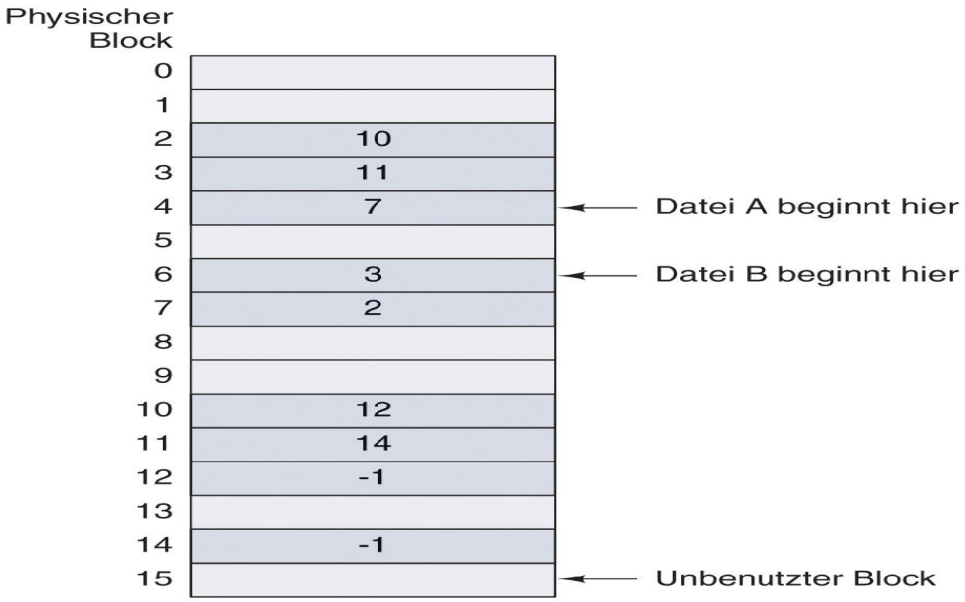
1) erzeugt einen „hard link“ - also einen weiteren Verzeichnis-eintrag (=Dateinamen) xyz1, der auf denselben i-Node verweist 2) erzeugt einen „soft link“ - also eine neue Datei mit Namen xyz2, deren Inhalt der Dateiname xyz (mit komplettem Pfad) ist	3

(d) Das immer noch weit verbreitete Dateisystem FAT32 benutzt zur Verwaltung der Abbildung von Datenblöcken in Dateien auf Festplattenblöcke eine sogenannte File Allocation Table (FAT).

1. Nehmen Sie an, ein FAT-basiertes Dateisystem soll eine Festplatte mit 1GB Kapazität mit einer Blockgröße von 1kB verwalten. Wie viele Einträge benötigt die FAT?

ca. 1 Million (1024x1024, 1G / 1k)	1

2. Gegeben eine (extrem) kleine FAT mit zwei Dateien A und B, wobei die Blockgröße wieder 1kB ist.



Welche Plattenblöcke gehören (in der richtigen Reihenfolge) zur Datei A?

4 – 7 – 2 – 10 – 12	1

In welchem Plattenblock befindet sich Byte 3000 der Datei B?

11 (Datei B hat die Plattenblöcke: 6-3-11-14)	1

Wie kann man wahlfreien Zugriff in einem FAT-System realisieren?

Man folgt den FAT-Einträgen – nicht den Plattenblöcken selbst – vom Beginn der Datei bis zur gewünschten Position und liest dann den dort angegebenen Block	2

Fragmentierung ist bei FAT theoretisch kein Problem. Warum sollten trotzdem nach Möglichkeit zusammenhängende Plattenblöcke gewählt werden?

Damit beim sequentiellen Lesen einer Datei auch aufeinanderfolgende Blöcke von Platte gelesen werden – was sehr viel höhere Effizienz bedeutet	2

Aufgabe 7 *Socket-Programmierung* (10 Punkte)

Gegeben ist die Server-Seite eines einfachen Client-Server-Programmes. Das Programm wartet darauf, dass Clients sich verbinden. Die eigentlich auszuführende Funktion (=Interaktion mit dem Client) findet in der Funktion `do_stuff()` statt, die hier nicht gezeigt ist. Das Programm weist einige Lücken auf und einige Stellen sind markiert sind. Beantworten Sie die sich darauf beziehenden nachfolgenden Fragen.

Hinweis: Es sind 14 Punkte möglich, die Maximalpunktzahl für die Aufgabe ist aber 10. Sie müssen nicht also nicht alle Fragen vollständig beantworten können.

```
#include <..hier müssten alle nötigen include statements stehen..>

void dostuff(int); /* function prototype */
void error(char *msg); /* function prototype */

int main(int argc, char *argv[]) {

     signal(SIGCHLD,SIG_IGN);

     int sockfd, newsockfd, portno, clilen, pid;
     struct sockaddr_in serv_addr, cli_addr;

     if (argc < 2) {
         fprintf(stderr,"ERROR, no port provided\n");
         exit(1);
     }
     sockfd =              Lücke L1              ;
     if (sockfd < 0) error("ERROR ...");
     bzero((char *) &serv_addr, sizeof(serv_addr));
     portno = atoi(argv[1]);
     serv_addr.sin_family = AF_INET;
     serv_addr.sin_addr.s_addr = INADDR_ANY;
     serv_addr.sin_port = htons(portno);
     if (    Lücke L2
               (struct sockaddr *) &serv_addr,
               sizeof(serv_addr)) < 0) error("ERROR ...");
          Lücke L3      ;
     clilen = sizeof(cli_addr);
     while (1) {
         newsockfd = accept(sockfd,
                (struct sockaddr *) &cli_addr, &clilen);
         if (newsockfd < 0)
             error("ERROR on accept");
         pid = fork();
         if (pid < 0)
             error("ERROR on fork");
         if (pid == 0)  {
             close(sockfd);
             dostuff(newsockfd);
             exit(0);
         }
         else close(newsockfd);
     } /* end of while */
     return 0; /* we never get here */
}
```

Zeile Z1

Zeile Z2

Zeile Z3

(a) Was bedeutet die Zeile Z1? (Stichworte)

Es wird ein signal handler (Funktion:SIG_IGN) installiert, der das Signal SIGCHLD fängt und ignoriert	2

(b) Hier wird ein Socket erzeugt, und zwar ein Stream-Socket in der Internet-Adressdomäne. Was steht in Lücke L2?

socket(AF_INET, SOCK_STREAM, 0);	2

(c) Was macht der Funktionsaufruf `htons(portno)` in Zeile Z2? Warum wird das in einem „guten Programm“ benötigt?

htons(portno) konvertiert den „short“-Wert portno von „host byte order“ in „netwerk byte order“. Da verschiedene Systeme verschiedene Byte-Reihenfolgen für binäre Daten verwenden, muss aus Portabilitätsgründen in die standardisierte Reihenfolge für Netzwerkübertragung konvertiert werden.	2

(d) Nach dem Erzeugen des Sockets (L1) muss nun in einem weiteren System Call dem Socket eine „lokale Adresse“, d.h. ein UNIX File Descriptor, zugeordnet werden. Was steht zu diesem Zweck in Lücke L2?

bind(sockfd,	2

(e) In einem weiteren System Call wird eine Request Queue der Länge 5 für eingehende Connection Requests (der Clients) erzeugt und der Socket „aktiviert“. Was steht hierzu in Lücke L3?

listen(sockfd,5);	2

(f) Was passiert beim letzten der System Calls, die Server-seitig zum Aufbau der Verbindung nötig sind? Erklären Sie in Zeile Z3 `newsockfd = accept(sockfd,` die Funktion des System Calls `accept()` und die Bedeutung von `newsockfd` und `sockfd` in Stichworten.

The `accept` function accepts the first connection on its queue of pending connections. If the queue has no pending connection requests, `accept` blocks the caller (unless **s** is in nonblocking mode). When connecting, the `accept` function creates a new socket descriptor `newsockfd` with the same properties as `sockfd` and returns it to the caller. Do not use the new socket to accept new connections. The original socket remains available to accept more connection requests.	2

(g) Mit welchen Systemaufrufen kann der Server Daten vom Client empfangen bzw zu ihm senden? (diese Aufrufe wären dann in `do_stuff()` zu finden...)

read() & write() --oder-- recv() & send()	2