BERGISCHE UNIVERSITÄT
GESAMTHOCHSCHULE WUPPERTAL
GAUSS-STRASSE 20
42097 WUPPERTAL
(Korrespondenzanschrift)
42119 WUPPERTAL
(Lieferanschrift)
TELEX 8 592 262 bughw
TELEFAX (0202) 439-2901
TELEFON (0202) 439-1

Fachbereich 7
MATHEMATIK

Prof. Dr. Hans-Jürgen Buhl
Praktische Informatik / Numerik

e-mail: Juergen.Buhl@math.uni-wuppertal.de

Betriebssysteme: Konzepte, Dienste, Schnittstellen (Betriebssysteme und betriebssystemnahe Programmierung)

SS 2003 – Übungsblatt 11

30. Juli 2003
Ausgabe: 23. Juli 2003

Aufgabe 1. *Pipes im Programm: dup*

Bringen Sie das folgende Programm zum Ablauf

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

    int main()
    {
        int pfds[2];

        pipe(pfds);

        if (!fork()) {
            close(1);       /* close normal stdout */
            dup(pfds[1]);   /* make stdout same as pfds[1] */
            close(pfds[0]); /* we don’t need this */
            execlp("ls", "ls", NULL);
        } else {
            close(0);       /* close normal stdin */
            dup(pfds[0]);   /* make stdin same as pfds[0] */
            close(pfds[1]); /* we don’t need this */
            execlp("wc", "wc", "-l", NULL);
        }
    }
```

und erklären Sie seine Wirkungsweise Zeile für Zeile.

Aufgabe 2. *FIFO*

Bringen Sie die folgenden Programme

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

#define FIFO_NAME "BUG_OS_SS2003"

    main()
    {
        char s[300];
        int num, fd;

        /* don't forget to error check this stuff!! */
        mknod(FIFO_NAME, S_IFIFO | 0666, 0);

        printf("waiting for writers...\n");
        fd = open(FIFO_NAME, O_RDONLY);
        printf("got a writer:\n");

        do {
            if ((num = read(fd, s, 300)) == -1)
                perror("read");
            else {
                s[num] = '\0';
                printf("tick: read %d bytes: \"%s\"\n", num, s);
            }
        } while (num > 0);
    }
```

sowie

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

#define FIFO_NAME "BUG_OS_SS2003"
```

```
    main()
    {
        char s[300];
        int num, fd;

        /* don't forget to error check this stuff!! */
        mknod(FIFO_NAME, S_IFIFO | 0666, 0);

        printf("waiting for readers...\n");
        fd = open(FIFO_NAME, O_WRONLY);
        printf("got a reader--type some stuff\n");

        while (gets(s), !feof(stdin)) {
            if ((num = write(fd, s, strlen(s))) == -1)
                perror("write");
            else
                printf("speak: wrote %d bytes\n", num);
        }
    }
```

(mehrfach) zum Ablauf und erklären Sie ihre jeweilige Wirkungsweise Zeile für Zeile.

Aufgabe 3. *socket*

Bringen Sie die folgenden Programme

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>

#define SOCK_PATH "echo_socket"

    int main(void)
    {
        int s, s2, t, len;
        struct sockaddr_un local, remote;
        char str[100];

        if ((s = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) {
            perror("socket");
            exit(1);
        }
        local.sun_family = AF_UNIX;
        strcpy(local.sun_path, SOCK_PATH);
        unlink(local.sun_path);
        len = strlen(local.sun_path) + sizeof(local.sun_family);
        if (bind(s, (struct sockaddr *)&local, len) == -1) {
            perror("bind");
            exit(1);
        }

        if (listen(s, 5) == -1) {
            perror("listen");
            exit(1);
        }

        for(;;) {
            int done, n;
            printf("Waiting for a connection...\n");
            t = sizeof(remote);
            if ((s2 = accept(s, (struct sockaddr *)&remote, &t)) == -1) {
                perror("accept");
                exit(1);
            }

            printf("Connected.\n");

            done = 0;
            do {
                n = recv(s2, str, 100, 0);
                if (n <= 0) {
                    if (n < 0) perror("recv");
                    done = 1;
                }

                if (!done)
                    if (send(s2, str, n, 0) < 0) {
                        perror("send");
                        done = 1;
                    }
            } while (!done);

            close(s2);
        }

        return 0;
    }
```

sowie

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>

#define SOCK_PATH "echo_socket"

    int main(void)
    {
        int s, t, len;
        struct sockaddr_un remote;
        char str[100];

        if ((s = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) {
            perror("socket");
            exit(1);
        }

        printf("Trying to connect...\n");

        remote.sun_family = AF_UNIX;
        strcpy(remote.sun_path, SOCK_PATH);
        len = strlen(remote.sun_path) + sizeof(remote.sun_family);
        if (connect(s, (struct sockaddr *)&remote, len) == -1) {
            perror("connect");
            exit(1);
        }

        printf("Connected.\n");

        while(printf("> "), fgets(str, 100, stdin), !feof(stdin)) {
            if (send(s, str, strlen(str), 0) == -1) {
                perror("send");
                exit(1);
            }

            if ((t=recv(s, str, 100, 0)) > 0) {
                str[t] = '\0';
                printf("echo> %s", str);
            } else {
                if (t < 0) perror("recv");
                else printf("Server closed connection\n");
                exit(1);
            }
        }

        close(s);
        return 0;
    }
```

zum Ablauf und erklären Sie ihre Wirkungsweise Zeile für Zeile.

Aufgabe 4. *socketpair*

Bringen Sie das folgende Programm zum Ablauf

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>

    int main(void)
    {
        int sv[2]; /* the pair of socket descriptors */
        char buf; /* for data exchange between processes */

        socketpair(AF_UNIX, SOCK_STREAM, 0, sv);

        if (!fork()) {  /* child */
            read(sv[1], &buf, 1);
            printf("child: read '%c'\n", buf);
            buf = toupper(buf);  /* make it uppercase */
            write(sv[1], &buf, 1);
            printf("child: sent '%c'\n", buf);

        } else { /* parent */
            write(sv[0], "b", 1);
            printf("parent: sent 'b'\n");
            read(sv[0], &buf, 1);
            printf("parent: read '%c'\n", buf);
        }
        return 0;
    }
```

und erklären Sie seine Wirkungsweise Zeile für Zeile.

Aufgabe 5. *message queues*

Bringen Sie die folgenden Programme

```
/*
** kirk.c -- writes to a message queue
*/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

struct my_msgbuf {
    long mtype;
    char mtext[200];
};

int main(void)
{
    struct my_msgbuf buf;
    int msqid;
    key_t key;

    if ((key = ftok("kirk.c", 'B')) == -1) {
        perror("ftok");
        exit(1);
    }

    if ((msqid = msgget(key, 0644 | IPC_CREAT)) == -1) {
        perror("msgget");
        exit(1);
    }

    printf("Enter lines of text, ^D to quit:\n");

    buf.mtype = 1; /* we don't really care in this case */
    while(gets(buf.mtext), !feof(stdin)) {
        if (msgsnd(msqid, (struct msgbuf *)&buf, sizeof(buf), 0) == -1)
            perror("msgsnd");
    }

    if (msgctl(msqid, IPC_RMID, NULL) == -1) {
        perror("msgctl");
        exit(1);
    }

    return 0;
}
```

sowie

```
/*
** spock.c -- reads from a message queue
*/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

struct my_msgbuf {
    long mtype;
    char mtext[200];
};

int main(void)
{
    struct my_msgbuf buf;
    int msqid;
    key_t key;

    if ((key = ftok("kirk.c", 'B')) == -1) {  /* same key as kirk.c */
        perror("ftok");
        exit(1);
    }

    if ((msqid = msgget(key, 0644)) == -1) { /* connect to the queue */
        perror("msgget");
        exit(1);
    }

    printf("spock: ready to receive messages, captain.\n");

    for(;;) { /* Spock never quits! */
     if (msgrcv(msqid, (struct msgbuf *)&buf, sizeof(buf), 0, 0) == -1) {
            perror("msgrcv");
            exit(1);
     }
     printf("spock: \"%s\"\n", buf.mtext);
    }

    return 0;
}
```

zum Ablauf und erklären Sie ihre Wirkungsweise Zeile für Zeile.

Aufgabe 6. *semaphores*

Bringen Sie die folgenden Programme nacheinander zum Ablauf

```
/*
** seminit.c -- sets up a semaphore for semdemo.c
*/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#if defined(__GNU_LIBRARY__) && !defined(_SEM_SEMUN_UNDEFINED)
       /* union semun is defined by including <sys/sem.h> */
#else
       /* according to X/OPEN we have to define it ourselves */
union semun {
             int val;                  /* value for SETVAL */
             struct semid_ds *buf;     /* buffer for IPC_STAT, IPC_SET */
             unsigned short *array;    /* array for GETALL, SETALL */
                                       /* Linux specific part: */
             struct seminfo *__buf;    /* buffer for IPC_INFO */
};
#endif

int main(void)
{
    key_t key;
    int semid;
    union semun arg;

    if ((key = ftok("semdemo.c", 'J')) == -1) {
        perror("ftok");
        exit(1);
    }

    /* create a semaphore set with 1 semaphore: */
    if ((semid = semget(key, 1, 0666 | IPC_CREAT)) == -1) {
        perror("semget");
        exit(1);
    }

    /* initialize semaphore #0 to 1: */
    arg.val = 1;
    if (semctl(semid, 0, SETVAL, arg) == -1) {
        perror("semctl");
        exit(1);
    }

    return 0;
}
```

sowie

```
/*
** semdemo.c -- demonstrates semaphore use as a file locking mechanism
*/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int main(void)
{
    key_t key;
    int semid;
    struct sembuf sb = {0, -1, 0};  /* set to allocate resource */

    if ((key = ftok("semdemo.c", 'J')) == -1) {
        perror("ftok");
        exit(1);
    }

    /* grab the semaphore set created by seminit.c: */
    if ((semid = semget(key, 1, 0)) == -1) {
        perror("semget");
        exit(1);
    }

    printf("Press return to lock: ");
    getchar();
    printf("Trying to lock...\n");

    if (semop(semid, &sb, 1) == -1) {
        perror("semop");
        exit(1);
    }

    printf("Locked.\n");
    printf("Press return to unlock: ");
    getchar();
    sb.sem_op = 1; /* free resource */
    if (semop(semid, &sb, 1) == -1) {
        perror("semop");
        exit(1);
    }

    printf("Unlocked\n");

    return 0;
}
```

und

```
/*
** semrm.c -- removes a semaphore
*/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#if defined(__GNU_LIBRARY__) && !defined(_SEM_SEMUN_UNDEFINED)
       /* union semun is defined by including <sys/sem.h> */
#else
       /* according to X/OPEN we have to define it ourselves */
union semun {
             int val;                    /* value for SETVAL */
             struct semid_ds *buf;       /* buffer for IPC_STAT, IPC_SET */
             unsigned short *array;      /* array for GETALL, SETALL */
                                         /* Linux specific part: */
             struct seminfo *__buf;      /* buffer for IPC_INFO */
};
#endif

int main(void)
{
    key_t key;
    int semid;
    union semun arg;

    if ((key = ftok("semdemo.c", 'J')) == -1) {
        perror("ftok");
        exit(1);
    }
    /* grab the semaphore set created by seminit.c: */
    if ((semid = semget(key, 1, 0)) == -1) {
        perror("semget");
        exit(1);
    }

    /* remove it: */
    if (semctl(semid, 0, IPC_RMID, arg) == -1) {
        perror("semctl");
        exit(1);
    }

    return 0;
}
```

und erklären Sie ihre Wirkungsweise Zeile für Zeile. Führen Sie bitte vor und nach Aufruf jedes Programms das Kommando `ipcs` aus und erläutern Sie.

Aufgabe 7. *shared memory*

Bringen Sie das folgende Programm zum Ablauf

```
/*
** shmdemo.c -- read and write to a shared memory segment
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SHM_SIZE 1024  /* make it a 1K shared memory segment */

int main(int argc, char *argv[])
{
    key_t key;
    int shmid;
    char *data;
    int mode;

    if (argc > 2) {
        fprintf(stderr, "usage: shmdemo [data_to_write]\n");
        exit(1);
    }

    /* make the key: */
    if ((key = ftok("shmdemo.c", 'R')) == -1) {
        perror("ftok");
        exit(1);
    }

    /* connect to (and possibly create) the segment: */
    if ((shmid = shmget(key, SHM_SIZE, 0644 | IPC_CREAT)) == -1) {
        perror("shmget");
        exit(1);
    }

    /* attach to the segment to get a pointer to it: */
    data = shmat(shmid, (void *)0, 0);
    if (data == (char *)(-1)) {
        perror("shmat");
        exit(1);
    }

    /* read or modify the segment, based on the command line: */
    if (argc == 2) {
        printf("writing to segment: \"%s\"\n", argv[1]);
        strncpy(data, argv[1], SHM_SIZE);
    } else
        printf("segment contains: \"%s\"\n", data);

    /* detach from the segment: */
    if (shmdt(data) == -1) {
        perror("shmdt");
        exit(1);
    }

    return 0;
}
```

und erklären Sie seine Wirkungsweise Zeile für Zeile. Benutzen Sie wiederum `ipcs`. Schreiben Sie ein Programm, das das Segment mit dem Schlüssel `"shmdemo.c"` wieder aus dem shared memory entfernt und testen Sie.

Aufgabe 8. *memory mapped files*

Bringen Sie das folgende Programm zum Ablauf

```
/*
** mmapdemo.c -- demonstrates memory mapped files lamely.
*/

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <errno.h>

int main(int argc, char *argv[])
{
    int fd, offset;
    char *data;
    struct stat sbuf;

    if (argc != 2) {
        fprintf(stderr, "usage: mmapdemo offset\n");
        exit(1);
    }

    if ((fd = open("mmapdemo.c", O_RDONLY)) == -1) {
        perror("open");
        exit(1);
    }

    if (stat("mmapdemo.c", &sbuf) == -1) {
        perror("stat");
        exit(1);
    }

    offset = atoi(argv[1]);
    if (offset < 0 || offset > sbuf.st_size-1) {
        fprintf(stderr, "mmapdemo: offset must be in the range 0-%d\n",
                sbuf.st_size-1);
        exit(1);
    }

    if ((data = mmap((caddr_t)0, sbuf.st_size,
                PROT_READ, MAP_SHARED, fd, 0)) == (caddr_t)(-1)) {
        perror("mmap");
        exit(1);
    }
    printf("byte at offset %d is '%c'\n", offset, data[offset]);

    return 0;
}
```

und erklären Sie seine Wirkungsweise Zeile für Zeile.