Dr. M. Grabert
Johannes Mayer
Abteilung Angewandte Informationsverarbeitung

2. Mai 2001
Blatt 1

Systemnahe Software (SS 2001)

Abgabetermin: 7. Mai 2001

Beispiellösung

1 Winzige Pseudo-Shell (8 Punkte)

Sie haben die Aufgabe, eine winzige Shell zu entwickeln. Diese gibt in einer Schleife immer wieder den Prompt aus und wartet dann auf die Eingabe einer Kommandozeile von der Standardeingabe solange, bis das Kommando „exit“ eingegeben wird. Wurde ein Kommando eingegeben, so führt die Shell ein `fork()` aus. Im Kindprozess wird dann das eingegebene Kommando via `system(...)` ausgeführt. Der Elternprozess, d. h. die Shell selbst, wartet bis der Kindprozess terminiert ist und gibt dann den nächsten Prompt aus. Das Warten soll via Synchronisation der beiden Prozesse erfolgen. In Teil 2 des AI3-Skriptes finden Sie im Kapitel 25.17 geeignete Mechanismen.

Lösung:

```
/*
 * Pseudo Microshell via System
 * Vater nimmt Kommando entgegen und Sohn fuehrt dieses via system() aus
 * Prozesssynchronisation via link
 */
#include <stdio.h>

#define TOKEN    "lfile"            /* Ressource */

int main(int argc, char **argv)
{
    char buf[200];
    int pid;

    printf("tinish > ");
    while (gets(buf) && strcmp(buf, "exit")) {
        if (link(argv[0], TOKEN)==-1)    /* Ressource sperren    */
```

```
            puts("can'get token ..."),
            exit(1);

        switch (pid = fork()) {
            case -1:    /* error */
                break;
            case 0:       /* Sohn */
                /* Warten auf Freigabe durch Vater    */
                while (link(argv[0], TOKEN)==-1)
                    ;

                system(buf);
                unlink(TOKEN);
                exit(0);
            default:
                unlink(TOKEN);    /* Sohn freigeben    */
                while (!link(argv[0], TOKEN))
                    unlink(TOKEN);

                /* Warten auf Ende vom Sohn    */
                while (link(argv[0], TOKEN)==-1)
                    ;
                unlink(TOKEN);
        }
        printf("tinish > ");
    }
}
```

2 ps -l (2 Punkte)

```
thales$ ps -l
F S   UID   PID  PPID  C PRI NI     ADDR     SZ    WCHAN TTY      TIME CMD
8 S   509 16507  1370  0  51 20        ?    312        ? pts/63   0:06 bash
```

Sie sollten Ihrem Tutor die folgenden Ausgaben des `ps -l` Befehls erklären können: S, UID, PID, PPID, PRI, NI, TTY, TIME, CMD

Lösung:

S Status, in dem sich der Prozess gerade befindet (z. B. S = schläft, T = gestoppt, ...).

UID effektive User-ID[1], unter der der Prozess läuft.

PID ID des Prozesses.

[1] ID = (eindeutige) Identifikationsnummer

PPID ID des Vaterprozesses.

PRI Priorität des Prozesses.

NI `nice`-Nummer des Prozesses (eine größere `nice`-Nummer führt zu geringerer Priorität des Prozesses $\leadsto$ rechenintensive Prozesse).

TTY Kontroll-Terminal an dem der Prozess „hängt“ bzw. `?`, falls der Prozess an keinem Kontroll-Terminal hängt.

TIME bisher angesammelte Ausführungszeit des Prozesses.

CMD Name des ausgeführten Kommandos.