42 Überblick über die 11. Übung

- Besprechung 8. Aufgabe (Shared-Memory und Semaphore)
- Musterlösung zur 5. Aufgabe (tsh)
- Wiederholung: sockets

43 Musterlösung zur Aufgabe 5 (tsh)

- Hintergrundprozesse (Teilaufgabe b und c)
- Listenoperationen (Teilaufgabe f)
- Zeiterfassung (Teilaufgabe g)

43.1 Hintergrundprozesse

- yash:

```
void execute(char *commandLine, char *command, char **argv) {
  int   statloc;
  pid_t pid, ret;
  switch(pid=fork()) {
    case -1 : perror("fork failed");return;
    case 0  :
      execvp(command, argv);
      perror(command);
      exit(EXIT_FAILURE);
    default :
      while(((ret = wait(&statloc)) != pid)
          && (errno == EINTR));
       if(ret != pid)
         perror("wait failed");
       else if(WIFEXITED(statloc))
         printStatus (commandLine, WEXITSTATUS(statloc));
  }
}
```


43.1 Hintergrundprozesse

- Anforderungen:
  - Shell soll nicht auf Hintergrundprozess warten
  - bei einem Vordergrundprozess muss die Shell auf den richtigen Prozess warten
- mögliche Lösungen:
  - waitpid im Vaterprozess
    - waitpid kann von SIGCHLD unterbrochen werden
    - kein wait im SIGCHLD-Handler möglich
  - waitpid im SIGCHLD-Handler

43.1 Hintergrundprozesse

```
void execute_fg(char *commandLine,char *command,char **argv){
    switch(fg_pid=fork()) {
    case -1 : perror("fork failed"); return;
    case 0  : execvp(command, argv); /* ... */  exit(-1);
    default :
     block_all_signals(&sigmask);
     while (fg_pid!=0) sigsuspend&(sigmask);
     restore_signals(&sigmask);
     printStatus (commandLine, WEXITSTATUS(fg_status));
}  }
```

```
void sigchild_handler(int signo) {
  int status, errnobak = errno;
  while ((pid=waitpid(-1,&status,WNOHANG))>0) {
    if (!WIFEXITED(status)) continue;
    if (pid==fg_pid) {
      fg_pid=0;
      fg_status=status;
  } }
  errno = errnobak;
}
```


43.1 Listenoperationen

```
void execute_bg(char *commandLine,char *command,char **argv){
  pid_t pid;
  sigset_t sigmask;
  block_all_signals(&sigmask);
  switch(pid=fork()) {
    case -1 : perror("fork failed");return;
    case 0  :
      restore_signals(&sigmask);
      block_signal(SIGINT);
      execvp(command, argv);
      perror(command);
      exit(EXIT_FAILURE);
    default :
      if (list_insert(pid, command)) perror ("list_insert");
   }
   restore_signals(&sigmask);
 }
```


43.1 Listenoperationen

- Liste der aktiven Kindprozesse um bei SIGINT ein SIGKILL zuzustellen
- Einfügen in Liste kann durch SIGCHLD unterbrochen werden
  - Problem, wenn im SIGCHLD Handler ebenfalls Listenoperationen untergebracht sind
  - Alternativ wird das Listenelement im SIGCHLD-Handler nur markiert und im “Hauptprogramm” ausgetragen
- Einfügen muss vor Austragen/Markieren geschehen (“atomar” mit fork)

43.1 Zeiterfassung

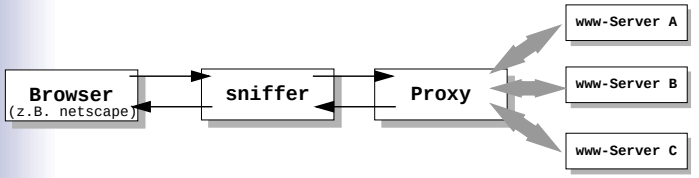
- times liefert die verbrauchten Zeiteinheiten des aktuellen Prozesses
- und die verbrauchte Zeit seiner Kinder
- ! die Zeitinformationen eines Kindes werden erst durch ein wait zum Vater übertragen

Zeiterfassung

```
void sigchild_handler(int signo) {
  int pid,status;
  struct tms time_buf1, time_buf2;
  struct tms *t1,*t2;
  clock_t ut,st;
  /* ... */
  t1 = &time_buf1;
  t2 = &time_buf2;
  if (times(t1)==(clock_t)-1) perror("times");
  while ((pid=waitpid(-1,&status,WNOHANG))>0) {
    if (times(t2)==(clock_t)-1) perror("times");
    ut = t2->tms_cutime - t1->tms_cutime;
    st = t2->tms_cstime - t1->tms_cstime;
    time_buf1 = time_buf2;
    /* ... */
  }
}
```


43.1 Sniffer

- Server in Richtung Web-Browser
- Client in Richtung Proxy



43 Wiederholung: TCP-Sockets