

# U11 11. Übung

---

## U11-1 Überblick über die 11. Übung

---

- [ Besprechung 7. Aufgabe (job\_sh) (vorgezogen) ]
- Evaluation
- Klausur

# U11-2 Lösung zur Aufgabe 7 (job\_sh)

---

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

# 1 Hintergrundprozesse

## ■ mini\_sh:

```
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));
    }
}
```

## 2 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

### 3 Hintergrundprozesse

```

void execute_fg(char *commandLine, char *command, char **argv) {
    block_signal(SIGCHLD);
    switch(fg_pid=fork()) {
        case -1 : perror("fork failed"); return;
        case 0  : execvp(command, argv); /* ... */  exit(-1);
        default :
            while (fg_pid!=0) sigsuspend(&empty_sigmask);
            unblock_signal(SIGCHLD);
            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;
}

```

## 4 Listenoperationen

---

- Liste der aktiven Kindprozesse um bei SIGINT ein SIGQUIT 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)
- Weiteres Problem: die Listenoperationen sind wegen des internen Laufzeigers nicht nebenläufig aufrufbar
  - Problem mit dem jobs-Befehl  
(↳ SIGINT und SIGCHLD im jobs-Befehl blockieren)
  - Problem mit SIGINT-handler in Teilaufgabe e  
(↳ SIGINT während Listenoperationen blockieren)

## 5 Listenoperationen

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