

## Aufgabe 5: mach (12.0 Punkte)

Implementieren Sie ein Programm `mach` (multithreaded absurd compilation helper), das ähnlich dem UNIX-Kommando `make` arbeitet. Es führt beliebige Befehle parallel aus und wird wie folgt aufgerufen:

```
user@host:~$ ./mach <anzahl threads> <mach-datei>
```

### a) mach-Datei (1 Punkt)

Eine `mach`-Datei besteht aus durch Leerzeilen getrennte Gruppen von Befehlen (ein Befehl pro Zeile). Alle Befehle einer Gruppe werden parallel (mit maximal `<anzahl threads>`) ausgeführt. Danach wird gewartet, bis alle Befehle beendet wurden, bevor die nächste Gruppe ausgeführt wird. Beispiel für eine `mach`-Datei:

```
gcc -c file1.c
gcc -c file2.c

gcc -o program file1.o file2.o
```

Diese kann folgende Ausgabe erzeugen (die Reihenfolge der Ausgabe innerhalb von Blöcken ist nicht deterministisch):

```
Running 'gcc -c file1.c' ...
Running 'gcc -c file2.c' ...
Completed 'gcc -c file2.c': "Multiple definitions for func".
Completed 'gcc -c file1.c': "".
Running 'gcc -o program file1.o file2.o' ...
Completed 'gcc -o program file1.o file2.o': "".
```

Erstellen Sie eine `mach`-Datei für die `mach` selbst, die wie üblich zuerst die Objektdateien kompiliert und diese dann linkt.

### b) mach-Programm (11 Punkte)

Das Programm soll wie folgt strukturiert werden:

- Der Hauptthread (`main()`) liest die `mach`-Datei ein und startet die benötigten Threads. Er steuert auch die Synchronisation zum Warten auf Gruppen von Befehlen.
- Die Ausgabe erfolgt in einem eigenen Thread. Dieser nimmt die Ausgaben von den Arbeiterthreads entgegen und gibt diese jeweils sofort aus. Die Kommunikation nutzt eine Warteschlange, die als *einfach verkettete Liste* implementiert ist. Dabei entspricht jedes Element in der Warteschlange einer Ausgabe. Orientieren Sie sich bei der verketteten Liste für die Warteschlange an der `lilo` und erweitern Sie das `struct` um benötigte Felder (wie beispielsweise den Programmnamen, Ausgabe, etc.).
- Pro Befehl wird ein eigener detachter Arbeiterthread gestartet (`pthread_detach(3)`), der den Befehl ausführt. Vor dem Start des Befehls wird dem Ausgabethread der Befehl mitgeteilt und nach dem Ende des Befehls wird der Befehl und die Ausgabe dem Ausgabethread mitgeteilt. Nutzen Sie zum Ausführen der Befehle die `run_cmd()`-Funktion in der vorgegebenen `run.o`-Datei (Dokumentation in `run.h`). Dabei sollen Fehler beim Ausführen (inklusive nicht erfolgreicher Exit-Codes) *vollständig* ignoriert werden.

Alle Threads dürfen *ausschließlich* passiv warten. Achten Sie ebenfalls auf eine passende Synchronisation der geteilten Datenstrukturen; nutzen Sie dazu `P()` und `V()` aus der vorgegebenen `sem.o`. Langsame Funktionen (z. B. `printf(3)`) dürfen nicht in kritischen Abschnitten ausgeführt werden. Beachten Sie, dass sich alle Programme beendet haben, bevor die nächste Gruppe ausgeführt wird. Stellen Sie ebenfalls sicher, dass am Ende des Programms immer sicher die letzte Ausgabe des Ausgabethreads erfolgt ist. Der Fokus der Bepunktung dieser Aufgabe liegt auf korrekter Synchronisation. Alle Ausgaben dürfen *nur* im Ausgabethread stattfinden (Ausnahme sind Fehlermeldungen, die `mach` beenden). Sie können davon ausgehen, dass jede Zeile maximal 4096 Zeichen lang ist (inklusive Newline).

### Hinweise zur Abgabe:

- Erforderliche Dateien: `mach.c`, `machfile`
- Hilfreiche *Manual-Pages*: `atoi(3)`, `fopen(3)`, `fgets(3)`, `strdup(3)`
- Im Verzeichnis `/proj/i4sp1/pub/aufgabe5` finden Sie eine `mach`-Implementierung und einige Beispiele, mit der Sie Ihre Lösung vergleichen können. Ebenfalls liegt dort `nethack-3.6.0-4-mach.tar.xz` zum Testen. Mit `tar xf` entpacken, in das Verzeichnis wechseln und dort die `mach`-Datei ausführen (hohe Parallelität lohnt sich). Dann mit `./nethack-console` das Spiel starten.

Bearbeitung: Einzel

Bearbeitungszeit: 11 Werktage (ohne Wochenenden und Feiertage)

Abgabezeit: 17:30 Uhr