1 Aufgabe 7

- In `~i471000/src/mandel` auf der `gsm` befindet sich der serielle Code für die Berechnung der Mandelbrotmenge.
- Parallelisieren Sie den Source mit Hilfe von Pthreads.
  - ▲ Unterteilen Sie hierzu das Bild in einzelne Streifen und lassen sie die Streifen von eigenen Threads berechnen.
  - ▲ Vergessen sie nicht, in Ihr Program den Aufruf von `thr_setconcurrency(8)` einzubauen, um auf der `cssun` mit allen 8 Prozessoren rechnen zu können!
  - ▲ Warum ist die einfache Aufteilung in 8 Streifen nicht optimal? Die Rechenzeit verkürzt sich, wenn mehr Streifen und damit (Userlevel-)Threads eingesetzt werden. Was ist die optimale Anzahl von Threads?

© Michael Schröder – a7.doc: 12/2/97

2 Aufgabe 8

- In dieser Aufgabe soll ein HTTP Server parallelisiert werden. Der serielle Source findet sich in `~i471000/src/tinyhttp`
- Zum Testen kann der WWW Baum des RRZE verwendet werden. Ein Aufruf wäre:
  `./tinyhttp 2345 /proj/websource`
  Danach könnte über die URL
  `http://cssun.rrze:2345/docs`
  auf den Server zugegriffen werden können.
- Da der Server zu langsam arbeitet, sollen die einzelnen Requests parallel von Threads abgearbeitet werden.

© Michael Schröder – a7.doc: 12/2/97

3 cont.

- Zur Implementierung soll das “Pool of Threads” Modell verwendet werden. Hierbei erzeugt der Server eine feste Anzahl von Clientthreads, die sich alle an einer Requestqueue blockieren. Der Serverthread nimmt neue Aufträge entgegen und stellt sie in die Queue, wo sie von den Clientthreads abgearbeitet werden. Dieses Vorgehen hat folgende Vorteile:
  - ▲ Lastbegrenzung: Die Anzahl der Threads ist fest vorgegeben.
  - ▲ Geschwindigkeit: Ein (langsamer) Aufruf von `pthread_create` entfällt.

© Michael Schröder – a7.doc: 12/2/97

4 cont.

- Implementieren Sie zuerst die Requestqueue. Sie benötigen dazu einen Mutex und eine Condition Variable, an der die Clients sich schlafen legen können, falls die Queue leer ist.
- Ein Request soll folgendermaßen aussehen:

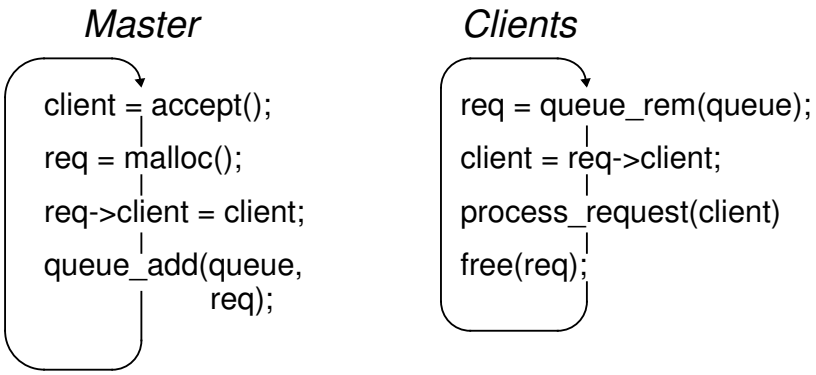
```
struct request {
    struct request *next; /* Verkettung */
    int client;           /* Descriptor */
};
```

- Über `queue_add(queue, request)` soll ein Request in die Queue eingehängt werden können und mit `queue_rem(queue)` soll der am frühesten eingehängte Request entfernt werden.

© Michael Schröder – a7.doc: 12/2/97

5 cont.

- Diagramm:



© Michael Schröder – a7.doc: 12/2/97