ulm university universität
uulm

Institut für Numerische Mathematik

Dr. Andreas F. Borchert

16. Juli 2019
Blatt 12

Systemnahe Software II (SS 2019)

Abgabe bis zum 23. Juli 2019, 14:00 Uhr

Lernziele:

- Parallele Sitzungen auf Basis von *epoll*

Aufgabe 12: Chat-Dienst

Im Rahmen dieser Aufgabe ist ein Chat-Dienst zu entwickeln, der für einen Hostport beliebig viele Sitzungen unterstützt.

Die Nachrichten im Rahmen des Chat-Protokolls bestehen aus beliebig vielen Zeichen ohne Zeilentrenner und werden mit CR LF (oder nur LF) terminiert. Zu Beginn jeder Sitzung erfolgt eine Begrüßung, worauf der Klient seinen Namen eingibt. Danach wird jede vom Klienten eingehende Nachricht an alle anderen Klienten unverändert weitergegeben. Jeder Klient kann jederzeit seine Verbindung terminieren und wird dann nicht mehr weiter berücksichtigt.

Dieser Chat-Dienst ist mit nur einem einzigen Prozess bzw. nur einem Thread auf Basis der *epoll*-Schnittstelle unter Linux zu entwickeln.

Wenn Sie möchten, können Sie hierzu eine auf den Modulen *mpx_session* und *multiplexor* der Vorlesungsbibliothek beruhende Implementierung verwenden. Wenn Sie dies tun, müssen Sie nur *multiplexor.c* ins zugehörige Verzeichnis kopieren und so anpassen, dass es *epoll* statt *poll* verwendet. Alternativ steht es Ihnen natürlich frei, eine davon unabhängige Lösung zu entwickeln.

Einige Hinweise: Es ist einfacher, dies pegelgesteuert zu realisieren (d.h. ohne *EPOLLET*). Sie müssten sonst die Netzwerkverbindungen auf nicht-blockierend umkonfigurieren und im Falle eines Ereignisses dieses umfassend abarbeiten. Das passt nicht zur aktuellen Schnittstelle von *multiplexor*, da *read_from_link* durch die Anwendung aufgerufen wird und die nicht auf einen nicht-blockierenden Modus vorbereitet ist.

Wenn Sie pegelgesteuert arbeiten, müssen Sie jeweils daran denken, *EPOLLOUT* hinzuzunehmen, wenn sich die Ausgabe-Warteschlange füllt, und das Flag wieder wegzunehmen, sobald die Ausgabe-Warteschlange leer ist. Solche Veränderungen sind möglich mit der Ope-

ration *EPOLL_CTL_MOD* bei *epoll_ctl*. Ebenso müssen Sie *EPOLLIN* wegnehmen, wenn die Eingabe beendet ist, aber die Ausgabe-Warteschlange noch gefüllt ist.

Bevor Sie eine Netzwerkverbindung endgültig schließen, sollten Sie diese noch aus der Filterliste von *epoll* mit *EPOLL_CTL_DEL* entfernen.

Bei *epoll_ctl* geben Sie jeweils ein **struct** *epoll_event* an, das eine **union** *epoll_data* enthält. Beachten Sie, dass Sie vollkommen frei entscheiden können, was Sie bei *data* angeben. Das muss nicht der jeweilige Dateideskriptor sein, Sie können auch alternativ einen beliebigen Zeiger (*ptr*) auf eine eigene Datenstruktur angeben. Konkret könnten Sie hier z.B. Zeiger des Typs *connection*$*$ angeben. Bei *epoll_wait* erhalten Sie für die eingetretenen Ereignisse dieses *data*-Feld zurück. Wenn Sie da zuvor einen Zeiger hinterlassen haben, finden Sie dann sofort ohne größere Suche ihre jeweils zugehörige Datenstruktur.

Ihre Lösung können Sie wiederum mit Hilfe von *tar* verpacken und dann mit *submit* auf der Theon einreichen:

```
tar cvf chatd.tar *.[ch] [mM]akefile
submit ss2 12 team [notes] chatd.tar
```

Viel Erfolg!