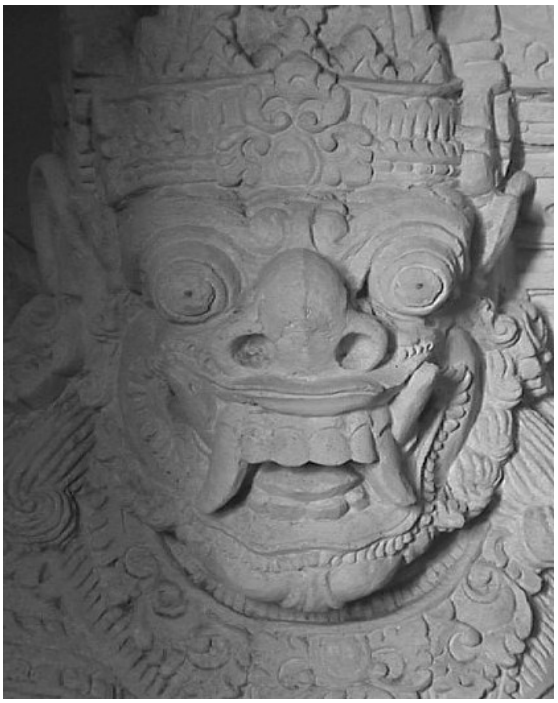
SNP: Dämon Prozesse



1 Übersicht

Dieser Teil des Praktikums behandelt spezielle Prozesse: die Dämon Prozesse («daemon processes»). Es ist gedacht als Zusatz zum Basis Praktikum über Prozesse und Threads.

Auch dieser Teil ist ein «Analyse»- und «Experimentier»-Praktikum.

1.1 Nachweis

Dieses Praktikum ist eine leicht abgewandelte Variante des ProcThreads Praktikum des Moduls BSY, angepasst an die Verhältnisse des SNP Moduls. Die Beispiele und Beschreibungen wurden, wo möglich, eins-zu-ein übernommen.

Als Autoren des BSY Praktikums sind genannt: M. Thaler, J. Zeman.

2 Lernziele

In diesem Praktikum werden Sie sich mit Dämon Prozessen beschäftigen.

- Sie können die Problemstellung der Dämon Prozesse erklären
- Sie können einen Dämon Prozess kreieren
- Sie können aus dem Dämon Prozess mit der Umgebung kommunizieren

3 Aufgabe: Dämon Prozesse

Ziele

- Problemstellungen um Daemons kennenlernen:
  - wie wird ein Prozess zum Daemon?
  - wie erreicht man, dass nur ein Daemon vom gleichen Typ aktiv ist?
  - wie teilt sich ein Daemon seiner Umwelt mit?
  - wo "lebt" ein Daemon?

Einleitung

Für diese Aufgabe haben wir einen Daemon implementiert: **`MrTimeDaemon`** gibt auf Anfrage die Systemzeit Ihres Rechners bekannt. Abfragen können Sie diese Zeit mit dem Programm **`WhatsTheTimeMr localhost`**. Die Kommunikation zwischen den beiden Prozessen haben wir mit TCP/IP Sockets implementiert. Weitere Infos zum Daemon finden Sie nach den Aufgaben.

Im Abschnitt 4 finden Sie Zusatzinformationen über diese Implementation eines Dämon Prozesses plus weiterführende Informationen.

Aufgaben

a) Für die folgende Aufgabe benötigen Sie mindestens zwei Fenster (Kommandozeilen-Konsolen). Übersetzen Sie die Programme mit **`make`** und starten Sie das Programm **`PlapperMaul`** in einem der Fenster. Das Programm schreibt (ca.) alle 0.5 Sekunden ***`Hallo, ich bins.... Pidi`*** plus seine Prozess-ID auf den Bildschirm. Mit dem Shell Befehl **`ps`** können Sie Ihre aktiven Prozesse auflisten, auch **`PlapperMaul`**. Überlegen Sie sich zuerst, was mit **`PlapperMaul`** geschieht, wenn Sie das Fenster schliessen: läuft **`PlapperMaul`** weiter? Was geschieht mit **`PlapperMaul`** wenn Sie sich ausloggen und wieder einloggen? Testen Sie Ihre Überlegungen, in dem Sie die entsprechenden Aktionen durchführen. Stimmen Ihre Überlegungen?

b) Starten Sie nun das Programm bzw. den Daemon **`MrTimeDaemon`**. Stellen Sie die gleichen Überlegungen an wie mit **`PlapperMaul`** und testen Sie wiederum, ob Ihre Überlegungen stimmen. Ob **`MrTimeDaemon`** noch läuft können Sie feststellen, indem Sie die Zeit abfragen oder den Befehl **`ps ajx | grep MrTimeDaemon`** eingeben: was fällt Ihnen am Output auf? Was schliessen Sie aus Ihren Beobachtungen?

c) Starten Sie **`MrTimeDaemon`** erneut, was geschieht?

d) Stoppen Sie nun **`MrTimeDaemon`** mit **`killall MrTimeDaemon`**.

e) Starten Sie **`MrTimeDaemon`** und fragen Sie mit **`WhatsTheTimeMr localhost`** oder mit **`WhatsTheTimeMr 127.0.0.1`** die aktuelle Zeit auf Ihrem Rechner ab.

Optional:

Fragen Sie die Zeit bei einem Ihrer Kollegen ab. Dazu muss beim Server (dort wo **`MrTimeDaemon`** läuft) ev. die Firewall angepasst werden. Folgende Befehle müssen dazu mit **root-Privilegien** ausgeführt werden:

```
iptables-save > myTables.txt # sichert die aktuelle Firewall
iptables -I INPUT 1 -p tcp --dport 65534 -j ACCEPT
iptables -I OUTPUT 2 -p tcp --sport 65534 -j ACCEPT
```

Nun sollten Sie über die IP-Nummer oder über den Rechner-Namen auf den **`TimeServer`** mit **`WhatsTheTimeMr`** zugreifen können.

Die Firewall können Sie mit folgendem Befehl wiederherstellen:

```
iptables-restore myTables.txt
```

f) Studieren Sie **`MrTimeDaemon.c`**, **`Daemonizer.c`** und **`TimeDaemon.c`** und analysieren Sie, wie die Daemonisierung abläuft. Entfernen Sie die Kommentare im Macro **`OutPutPIDs`** am Anfang des Moduls **`Daemonizer.c`**. Übersetzen Sie die Programme mit **`make`** und starten Sie **`MrTimeDaemon`** erneut. Analysieren Sie die Ausgabe, was fällt Ihnen auf? Notieren Sie alle für die vollständige Daemonisierung notwendigen Schritte.

g) Setzen Sie beim Aufruf von **`Daemonizer()`** in **`MrTimeDaemon.c`** anstelle von **`lockFilePath`** den Null-Zeiger **`NULL`** ein. Damit wird keine lock-Datei erzeugt. Übersetzen Sie die Programme und starten Sie erneut **`MrTimedaemon`**. Was geschieht bzw. wie können Sie feststellen, was geschehen ist?

Hinweis: lesen Sie das log-File: **`/tmp/timeDaemon.log`**.

Wenn Sie noch Zeit und Lust haben: messen Sie die Zeit, zwischen Start der Zeitanfrage und Eintreffen der Antwort. Dazu müssen Sie die Datei **`WhatsTheTimeMr.c`** entsprechend anpassen.

4 Zusatzinformationen

4.1 Diese Implementation

Dieser Daemon besteht aus den 3 Komponenten.

Hauptprogramm: MrTimeDaemon.c

Hier werden die Pfade für die lock-Datei, die log-Datei und der ”Aufenthaltsort” des Daemons gesetzt. Die lock-Datei wird benötigt um sicherzustellen, dass der Daemon nur einmal gestartet werden kann. In die lock-Datei schreibt der Daemon z.B. seine PID und sperrt sie dann für Schreiben. Wird der Daemon ein zweites Mal gestartet und will seine PID in diese Datei schreiben, erhält er eine Fehlermeldung und terminiert (es soll ja nur ein Daemon arbeiten). Terminiert der Daemon, wird die Datei automatisch freigegeben. Weil Daemonen sämtliche Kontakte mit ihrer Umwelt im Normalfall abbrechen und auch kein Kontrollterminal besitzen, ist es sinnvoll, zumindest die Ausgabe des Daemons in eine log-Datei umzuleiten. Dazu stehen einige Systemfunktionen für Logging zur Verfügung. Der Einfachheit halber haben wir hier eine normale Datei im Verzeichnis **`/tmp`** gewählt.

> **Anmerkung:** die Wahl des Verzeichnisses **`/tmp`** für die lock- und log-Datei ist für den normalen Betrieb problematisch, weil der Inhalt dieses Verzeichnisses jederzeit gelöscht werden kann, bzw. darf. Wir haben dieses Verzeichnis gewählt, weil wir die beiden Dateien nur für die kurze Zeit des Praktikums benötigen.

Der Daemon erbt sein Arbeitsverzeichnis vom Elternprozesse, er sollte deshalb in ein festes Verzeichnis des Systems wechseln, um zu verhindern, dass er sich in einem montierten (gemounteten) Verzeichnis aufhält, das dann beim Herunterfahren nicht demontiert werden könnte (wir haben hier wiederum **`/tmp`** gewählt).

Daemonizer: Daemonizer.c

Der Daemonizer macht aus dem aktuellen Prozess einen Daemon. Z.B. sollte er Signale (eine Art Softwareinterrupts) ignorieren: wenn Sie die CTRL-C Taste während dem Ausführen eines Vordergrundprozess drücken, erhält dieser vom Betriebssystem das Signal SIGINT und bricht seine Ausführung ab. Weiter sollte er die Dateierzeugungsmaske auf 0 setzen (Dateizugriffsrechte), damit kann er beim Öffnen von Dateien beliebige Zugriffsrechte verlangen (die Datei-

erzeugungsmaske erbt er vom Elternprozess). Am Schluss startet der Daemonizer das eigentliche Daemonprogramm: TimeDaemon.e.

Daemonprogramm: TimeDaemon.c

Das Daemonprogramm wartet in einer unendlichen Schleife auf Anfragen zur Zeit und schickt die Antwort an den Absender zurück. Die Datenkommunikation ist, wie schon erwähnt, mit Sockets implementiert, auf die wir aber im Rahmen dieses Praktikums nicht weiter eingehen wollen (wir stellen lediglich Hilfsfunktionen zur Verfügung).

4.2 Zusatzinformation zu Dämon Prozessen

Dämonen oder englisch Daemons sind eine spezielle Art von Prozessen, die vollständig unabhängig arbeiten, d.h. ohne direkte Interaktion mit dem Anwender. Dämonen sind Hintergrundprozesse und terminieren i.A. nur, wenn das System heruntergefahren wird oder abstürzt. Dämonen erledigen meist Aufgaben, die periodisch ausgeführt werden müssen, z.B. Überwachung von Systemkomponenten, abfragen, ob neue Mails angekommen sind, etc.

Ein typisches Beispiel unter Unix ist der Printer Daemon **`lpd`**, der periodisch nachschaut, ob ein Anwender eine Datei zum Ausdrucken hinterlegt hat. Wenn ja, schickt er die Datei auf den Drucker.

Hier wird eine weitere Eigenschaft von Daemons ersichtlich: meist kann nur ein Dämon pro Aufgabe aktiv sein: stellen Sie sich vor, was passiert, wenn zwei Druckerdämonen gleichzeitig arbeiten. Andererseits muss aber auch dafür gesorgt werden, dass ein Dämon wieder gestartet wird, falls er stirbt.

5 Bewertung

Die gegebenenfalls gestellten Theorieaufgaben und der funktionierende Programmcode müssen der Praktikumsbetreuung gezeigt werden. Die Lösungen müssen mündlich erklärt werden.

Aufgabe	**Kriterium**	**Punkte**
	Sie können die gestellten Fragen erklären.	
1	*Dämon Prozesse*	4