der Bundeswehr
Universität München

Universität der Bundeswehr
Fakultät für Elektrotechnik und Technische Informatik
ETTI – Institut 2
Prof. Dr. H. Görl

HT 21
Praktikumsblatt 1
18. Oktober 2021

Praktikum zu
Sicherheit moderner Betriebssysteme

Aufgabe 0 (Kurzwiederholung der Praktika)

Im Ilias-Verzeichnis zu diesem Praktikum finden Sie einen Ordner „Betriebssysteme-Praktikum-WT2021“ mit den fünf Aufgabenblättern des ersten Teils des Praktikums zu Betriebssysteme aus dem Wintertrimester. Bitte sehen Sie sich die Aufgaben nochmals an und überlegen Sie, wie Sie die Aufgaben gelöst haben und welche Probleme dabei aufgetreten waren.
Dabei liegt auch ein Aufgabenblatt aus dem Praktikum zu Systemsicherheit aus dem Frühjahrstrimester FT2021. Dessen Aufgabe 2 bis Aufgabe 4 sind wichtig für das Verständnis der Vorlesungsinhalte zur Speicherverwaltung, das bitte auch nochmals durchdenken.

Aufgabe 1 (Erzeugung eines Daemon)

Für die Lösung dieser Aufgaben sind folgende Kenntnisse nötig:
(Befehle, die mit einem > beginnen, sind Shellbefehle, die die mit `()` enden, sind C-Befehle)

Befehl	Bedeutung	Folien	Video
> top	Livebild der Prozessliste		
keine Optionen	Testen Sie mal `htop`		
> ps	Ausgabe der Prozessliste		Unix - Kapitel 3
-a -x -f	Alle Prozesse Auch Prozesse ohne Ausgabe Prozesshierarchie als Baum		
> kill	Signal an Prozess senden		
Option PID	SIGTERM an Prozess PID schicken		
> man X	Anleitung für Befehl X ansehen		
Option > man 2 X	Handbuch 2 für Befehl X ansehen		
r = fork(void)	Anleitung für Befehl X ansehen	Kapitel 4	
Rückgabewert $r > 0$ Rückgabewert $r == 0$ Rückgabewert $r < 0$	im Elternprozess im Kindprozess Fehler		
r = setsid(void)	Neue Session erzeugen		
Rückgabewert $r > 0$	erfolgreich		
r = chdir(const char*)	Arbeitsverzeichnis wechseln		
Rückgabewert $r == 0$	erfolgreich		
r = umask(0)	Standardrechte neuer Dateien		
Rückgabewert egal	immer erfolgreich		

Daemonprozesse sind Prozesse, die unabhängig von Benutzersitzungen Aufgaben ausführen und i.W. nur durch das Herunterfahren des Systems beendet werden. Sie besitzen gegenüber anderen Prozessen folgende Eigenschaften:

- Systemdaemonen haben die UID 0
- Elternprozess muss der Init-Prozess sein
- Daemon besitzen kein Kontrollterminal
- Terminalprozessgruppe ist -1

Damit der Daemon beobachtet werden kann, müssen Sie am Ende dieser Schritte Sie dafür sorgen, dass der Prozess nicht beendet wird, beispielsweise durch eine Endlosschleife.

(a) Überprüfen Sie bitte die Eigenschaften an Ihrem System mit dem Aufruf `ps` und den entsprechenden Optionen, z.B. `-axj` und erklären Sie die Ausgabe. Welche der angezeigten Prozesse sind Benutzerprozesse und welche stellen vermutlich einen Hintergrundprozess/Daemon dar und warum?

(b) Schreiben Sie einen eigenen Daemonprozess unter Linux in der Programmiersprache C. Der Ablauf sollte dabei wie folgt sein:

- Rufen Sie `fork()` auf und beenden Sie den Elternprozess mit `exit(0)`.
- Erzeugen Sie für den neu erzeugten und nun zu Init gehörigen Prozess eine neue Session mittels dem Systemruf `setsid()`. Der Prozess wird dadurch Sessionführer einer neuen Session, er wird Prozessgruppenführer der neuen Prozessgruppe und er hat kein zugeordnetes Terminal.
- Jeder Prozess hat ein Arbeitsverzeichnis, was zur Folge hat, dass dieser Daemon das Arbeitsverzeichnis blockiert, in dem er gestartet wurde. Ändern Sie dies durch den Aufruf von `chdir()` und wechseln Sie in das Wurzelverzeichnis.
- Der Daemon hat von seinem Vater eine Rechteliste bzw. sogenannte Maske geerbt, die beschreibt, wie die Rechte der von ihm erzeugten Dateien und Verzeichnisse aussehen. Ändern Sie dies mittels dem Aufruf `umask(0)`.
- Schließen Sie noch alle geöffneten Dateideskriptoren.

(c) Erzeugen Sie vom Daemon aus Syslog-Einträge für jeden der Erzeugungsschritte aus der vorigen Teilaufgabe durch die Nutzung folgender Funktion:

```
1    #include <syslog.h>
2    syslog (LOG_INFO, "Daemon BSPRAKT erzeugt");
```

Listing 1: Erzeugung eines Syslog-Eintrags

Ausgaben nach Syslog können Sie z.B. in der Datei `/var/log/syslog` ansehen.

(d) Wieviel Rechenzeit verbraucht der Daemon in Ihrem System? Nutzen Sie dazu das Kommando `top` oder moderner `htop`. Wie können Sie diese (sinnlose) Rechenzeitverschwendung erklären und wie können Sie diese reduzieren?

Aufgabe 2 (Linux – Prozesserzeugung und Prozessumgebungen)

Für die Lösung dieser Aufgaben sind folgende Kenntnisse nötig:
(Befehle, die mit einem > beginnen, sind Shellbefehle, die die mit `()` enden, sind C-Befehle)

Befehl	Bedeutung	Folien	Video
r = fork(void)	Anleitung für Befehl X ansehen	Kapitel 4	
Rückgabewert $r > 0$ Rückgabewert $r == 0$ Rückgabewert $r < 0$	im Elternprozess im Kindprozess Fehler		
r = sleep(u_int x)	x Sekunden warten		
Rückgabewert $r == 0$	Zeit abgelaufen		
r = strncpy(dest,src,n)	String-Copy		
Rückgabewert $r = *dest$	erfolgreich		

(a) Schreiben Sie ein C-Programm nach folgendem Muster und geben Sie die Inhalte des Arrays `argv[]` an der angegebene Stelle aus. Welche Inhalte liegen in diesem Array? Was steht in `argv[0]`? Wie kann dieser Inhalt von `argv[0]` durch den Aufruf des Programms verändert werden? Wie sieht der Prozessbaum nach dem Start aus, erklären Sie bitte den Aufbau des Baums?

```c
#include <stdio.h>
int main(int argc, char **argv){
  int pid;
  pid = fork();
  pid = fork();
  // Hier Ausgabe von argv[]
  while(1){
    sleep(1);
  }
}
```

(b) Was steht als Inhalt im ersten Element des Arrays `argv[]`. Verändern Sie den Inhalt des ersten Elementes des Arrays `argv[]`. Was passiert dann, wenn Sie die anschließend die Prozessliste ansehen? Wie könnte der Inhalt in einen Wert verändert werden, der länger ist, als der ursprüngliche?

(c) Welche Bedeutung hat der Rückgabewert der Funktion `main`. Informieren Sie sich über die Funktion `exit()`. Wer wertet diesen Wert aus und wann wird dieser Wert genau ausgewertet?

(d) Wie greift man in einem C-Programm auf die Umgebungsvariablen zu? Sehen Sie sich dazu die Anleitung zu den Funktionen `char ** environ` und `char * getenv (const char *name)` an.

(e) Wie können wir Prozesse erzeugen, die einen anderen Programmcode ausführen, als den eigenen? Informieren Sie sich dazu über die Funktionsfamilie `exec()`. Hinweis: `l=Liste, v=Vektor, e=Environment, p=Path`. Schreiben Sie ein Programm, welches einen neuen Prozess erzeugt und dieser neu erzeugte Prozess mit dem Prozess `ls -l` überladen wird.

Aufgabe 3 (Unix-crypt())

Nutzen Sie unten stehendes Programm zur Analyse der Passwort-crypt()-Funktion unter Unix. Testen Sie die Ausgabe für verschiedene Salt-Werte bei gleichem Passwort und vergleichen Sie die Ausgaben auch mit den Einträgen der beiden Nutzer „prakt“ und „root“ in Ihrem System in der `/etc/shadow`-Datei (oder der angelegten User bei einer selbst installierter Unix-Variante).

```c
#include <unistd.h>
#include <stdio.h>
int main(int argc, char* argv[])
{
  char password[] = "passwd";
  printf("%s\n", password);
  char *passwordcipher = crypt(password, "11");
  printf("%s\n", passwordcipher);
  return 0;
}
```

Aufgabe 4 (Ein erstes Kernel-Modul)

Um eigenen Programmcode in den Linux-Kernel zu laden, bietet es sich an, ein sogenanntes Kernel-Modul zu schreiben. Es besteht aus einem C-Quelltext, der einige Standard-Makros und Standard-Funktionen besitzen muss und mit einem speziellen Compileraufruf übersetzt wird. Die vom Linker erzeugte Objektdatei kann direkt in den laufenden Kern eingeladen werden.
Zur Umsetzung werden einige Pakete aus der installierten Distribution (Ubuntu) benötigt, u.a. „kernel-header“, und die Compiler-Infrastruktur für den „gcc“.
Achtung: Fehlerhafter Kernelcode bringt das Betriebssystem und alle Anwendungen u.U. zum Absturz und führt dazu, dass Sie die Maschine (hier die virtuelle Maschine) neu starten müssen!

(a) Erzeugen Sie eine C-Quelltextdatei „praktmodul.c“ mit folgendem Inhalt:

```
1   #include <linux/kernel.h>
2   #include <linux/module.h>
3   #include <linux/timer.h>
4   #include <linux/types.h>
5
6   #define PRINTK(x...) printk(KERN_DEBUG "Praktmodul: " x)
7
8   static int __init prakt_module_init(void)
9   {
10    PRINTK("Praktikumsmodul init\n");
11    return 0;
12  }
13
14  static void __exit prakt_module_exit(void)
15  {
16    PRINTK("Praktikumsmodul exit\n");
17  }
18
19  module_init(prakt_module_init);
20  module_exit(prakt_module_exit);
21
22  MODULE_DESCRIPTION("Praktikum Kernelmodul");
23  MODULE_LICENSE("GPL");
```

Listing 2: Ein einfaches Kernelmodul

(b) Erzeugen Sie für die Übersetzung des Moduls die Datei „Makefile“:

```
1   obj-m += praktmodul.o
2   KVERSION = $(shell uname -r)
3   all:
4   <TAB>    make -C /lib/modules/$(KVERSION)/build M=$(PWD) modules
5   clean:
6   <TAB>    make -C /lib/modules/$(KVERSION)/build M=$(PWD) clean
```

Listing 3: Ein einfaches Kernelmodul laden

Achtung: Hier werden ausnahmsweise die Leerzeichen dargestellt, da das Makefile sonst fehlerhaft sein kann. Beachten Sie bitte auch, dass Sie in den Zeilen 4 und 6 wirklich die Tabulator-Taste nutzen, um die Einrückungen zu erzeugen. Anderenfalls wird das Makefile nicht funktionieren.

(c) Übersetzen Sie das Modul mit dem Shell-Kommando „`make`“. und laden Sie die erzeugte Objektdatei (das erzeugte Kernmodul) mit folgendem Kommando in den Kern (*root*-Rechte nötig!):

```
1   > insmod praktmodul.ko
```

Überprüfen Sie das erfolgreiche Laden des Moduls dadurch, dass Sie die Liste aller geladenen Module ansehen (Kommando „`lsmod`“) und in dieser Liste ihr neues Modul suchen (Name: „praktmodul“)

(d) Alle Log-Ausgaben, die unter Linux systemweit erzeugt werden, werden an den sog. „syslog-Dienst“ übergeben. Eine Log-Meldung besteht grundsätzlich aus einer Meldung (String) und einer sog. „Facility“, die den Ursprung der Meldung zeigt. Der Schweregrad der Meldung wird u.a. durch die Schlüsselwörter `INFO, DEBUG, WARNING, ERROR` beschrieben. Anhand der Facility , der Quelle und des Schweregrads, von der die Meldung stammt, kann der „syslog-Dienst“ die Meldung unterschiedlich verarbeiten und abspeichern oder an andere Rechner weiterleiten. Unter Linux behandelt dieser Dienst auftretende Meldungen so, wie es in seiner Konfigurationsdatei `/etc/rsyslog.conf` angegeben wird.

Wir möchten für unser Praktikum alle Meldungen („*“) aller Facilities („*“) in eine Datei schreiben und fügen dazu folgende Zeile in die Konfigurationsdatei `/etc/rsyslog.conf` am Ende ein:

```
1    [... Datei /etc/rsyslog.conf ...]
2    *.*       /var/log/allmessages
```

und starten Sie den Dienst („rsyslog“) neu:

```
1    > systemctl restart rsyslog
```

Die Meldungen des Betriebsystemkerns werden durch das C-Makro `PRINTK(...)` generiert. Diesen Mechanismus nutzen wir auch in dem Modul, daher können wir die Ausgaben des Moduls im Systemlog wie folgt ansehen:

```
1    > tail -100f /var/log/allmessages
```

(e) Sehen Sie sich Informationen zum Kernelmodul mit dem Kommando „`modinfo`“ an.

(f) **Zusatzaufgabe** Bauen Sie eine einfache Endlosschleife in die Funktion `prakt_module_init` ein und testen Sie, was beim Laden des Moduls passiert.