Universität Heidelberg
Sommersemester 2019

Lehrstuhl für Parallele und Verteilte Systeme (PVS)
Diego Costa, Artur Andrzejak

3. Übung zur Vorlesung „Betriebssysteme und Netzwerke“ (IBN)

Abgabedatum: 14.05.2019, 11:00 Uhr

Aufgabe 1 (5 Punkte)

Schreiben Sie ein Programm (Shell-Skript oder C), bei dem zwei Prozesse erzeugt werden. Der Prozess A schickt einen String an einen anderen Prozess B. Dann kehrt Prozess B die Reihenfolge der Zeichen in dem empfangenen String um, und anschließend schickt er das Ergebnis an den Prozess A zurück. So z.B. wird aus dem String „Hello World“ der String „dlroW olleH“. Verwenden Sie für die Kommunikation der Prozesse anonyme Pipes (wahlweise unter Linux oder Windows).

Aufgabe 2 (2 Punkte)

Fügen Sie in das Programm mit Bezeichnung Algorithmus 1 für die Platzhaltern `???` geeignet folgende Befehle ein: `shmat, shmctl, shmdt, shmget`. Erklären Sie jeden der Befehle kurz. Kompilieren Sie das Programm und führen es aus. Welches Verhalten können Sie beobachten (ist dies wie erwartet)? Versuchen Sie, falls möglich, Ihre Beobachtungen zu erklären.

Aufgabe 3 (4 Punkte)

Lesen Sie den Abschnitt 4.4 des Buches *The Linux Kernel*[1] durch (siehe *Processes / 4.4 Files*).

1. Wie groß ist die maximale Anzahl der geöffneten Dateien eines Prozesses laut diesem Text? Was ist diese Anzahl auf heutigen Linux-Systemen, bzw. kann sie verändert werden? Begründen Sie die zweite Antwort mit einer Quellenangabe.
2. Weiterhin betrachten Sie den Quelltext der `task_struct` (PCB in Linux) des neuesten Linux-Kernels[2]. In welchen Zeilen finden Sie eine Datenstruktur mit den Informationen zu allen geöffneten Dateien eines Prozesses?
3. Finden Sie den Quelltext von `struct file`, der Datenstruktur mit der Information zu einer *einzigen* geöffnetem Datei (Hinweis: Betrachten Sie das Diagramm in dem o.g. Abschnitt 4.4, und suchen Sie via Google nach relevanten Variablennahmen). Geben Sie den Link zum Quelltext und die Zeilennummer an. Schätzen Sie dann den minimalen Speicherbedarf dieser

[1] http://www.tldp.org/LDP/tlk/tlk-toc.html
[2] http://elixir.free-electrons.com/linux/latest/source/include/linux/sched.h

Datenstruktur. Zur Vereinfachung nehmen Sie an, dass jeder Eintrag (Variable, Pointer, interne struct usw.) genau 4 Bytes belegt. Nehmen Sie zur weiteren Vereinfachung an, dass alle `#ifdef` erfüllt sind (falls vorhanden).

4. Können Sie in der Datenstruktur `struct file` den Wert des Dateideskriptors zu der entsprechenden geöffneten Datei identifizieren? Falls ja, geben Sie die Zeile und den Variablennamen an. Bei „nein" erläutern Sie, wo man den Wert des zugehörigen Dateideskriptors findet. Überlegen Sie sich einen Grund, warum man das so implementiert hat, und geben Sie diesen an.

Aufgabe 4 (1 Punkt)

Wenn ein Threadwechsel auf einer CPU stattfindet, entsteht ein gewisser Overhead an Zeit, in dem keiner der beteiligten Threads seine Rechnung fortführen kann. Beschreiben Sie, was beim Prozesswechsel passiert und nennen Sie die tatsächlichen Gründe, warum dieser Overhead entsteht. Schauen Sie sich dazu die in der Vorlesung genannten Videos an.

Aufgabe 5 (4 Punkte)

Bei dem Programm aus Vorlesung 6 („POSIX Pthreads – Beispiel") sind die Aktionen „Erzeugung eines Threads" und „Nachricht drucken" zufällig miteinander verschachtelt.

1. Schreiben Sie das Programm so um, dass die Ausführung die folgende strikte Reihenfolge einhält: „Erzeugung von Thread 1"; „Nachricht von Thread 1 drucken"; „Beenden von Thread 1"; „Erzeugung von Thread 2"; ..., usw. Testen Sie Ihr Programm anschließend.
2. Was ist (bei Ihrer Implementierung und Ihrem System) der maximale Wert $N_{\max}$ der Konstante *NUM_THREADS*, bei dem die Ausführung des Programms etwa 10 Sekunden dauert (eine Näherung reicht)?

Algorithmus 1 Programm (mit Lücken), das shared memory verwenden soll

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <stdlib.h>
int main(){

    int i, shmID, *shared_mem, count=0, total=0,rnd;
    shmID = ???(IPC_PRIVATE, sizeof(int), IPC_CREAT | 0644);
    shared_mem = (int*)???(shmID,0,0);
    *shared_mem = 0;
    if (fork())
        for (i=0; i<500; i++){
            *shared_mem+=1;
            printf("\n Elternprozess: %i", *shared_mem);
            sleep(2);
        }
    else
        for (i=0; i<500;i++){
            *shared_mem+=1;
            printf("\n Kindprozess: %i", *shared_mem);
            rnd=rand();
            sleep(rnd%3);
        }
    ???(shared_mem);
    ???(shmID, IPC_RMID, 0);
    return 0;

}
```