INF FAKULTÄT FÜR INFORMATIK

Institut für Intelligente Kooperierende Systeme (IKS)
Prof. Dr. David Hausheer

Technische Informatik 2 - SS 2018 - Praktische Übung 3

Abgabe: 13.05.2018

1 Speicherverwaltung

Copyright Ariane Keller, Markus Happe
Communication Systems Group
ETH Zurich

Inhaltsverzeichnis

1 Einleitung

Ziel dieser Übung ist es, Messungen in Bezug auf CPU-Caching durchzuführen sowie den IPC-Mechanismus des Shared Memorys kennenzulernen. Für das Caching-Experiment wird ein allokierter Speicherbereich auf zwei verschiedene Weisen durchlaufen, welche zu unterschiedlichen Cache-Misses führen. Beim zweiten Teil der Übung sollen Sie mit verschiedenen Prozessen auf einen gemeinsam geteilten Speicherbereich zugreifen. Auch hier sollten Sie sich zunächst mit den zugehörigen Manpages vertraut machen wie z.B. `shmget(2)` und `shmat(2)`.

2 Aufgaben

2.1 Caching

Schreiben Sie zwei Funktionen die einen gegebenen Speicherbereich (Matrix) modifizieren. Dabei soll die Modifikation das Inkrementieren jedes Bytes des Speicherbereichs sein. Das Durchlaufen des Speicherbereichs soll dabei auf zwei verschiedene Art und Weisen geschenen:

1. Funktion `f` durchläuft den Speicherbereich spaltenweise
2. Funktion `g` durchläuft den Speicherbereich zeilenweise

Dies bedeutet also, dass die Eingabe und Ausgabe beider Funktionen identisch sind, nur die Strategie die Modifikationen auszuführen eine andere ist.

Für die Implementierung wird das Template `caching.c` bereitgestellt. Das Programm zum Messen der Caching-Statistiken (`perf`) entnehmen wir dem Linux Kernel. Es liegt bereits vorkompiliert im gleichen Verzeichnis wie das Template.

Das heisst also, dass Sie das Tool `perf` auf Ihr Programm folgendermassen anwenden können: `./perf stat ./a.out 0` für Funktion `f` bzw. `./perf stat ./a.out 1` für Funktion `g`.

Was lässt sich feststellen und warum?

2.2 Shared Memory

Linux verfügt über verschiedene Mechanismen, die der Kommunikation zwischen Prozessen dienen. Einer dieser Mechanismen beruht auf der Verwendung eines gemeinsamen Speicherbereiches (Shared Memory): Verschiedene Prozesse allozieren einen gemeinsamen Speicherplatz und gebrauchen diesen, um Daten auszutauschen. `shmget(2)` ist ein solcher System Call, dessen Aufruf einem Prozess ein gemeinsames Speicher-Segment alloziert.

Bevor Sie mit Programmieren beginnen, sind die nachfolgend aufgelisteten Fragen zu beantworten. Zur Hilfestellung sind beispielsweise die Manpages zu heranzuziehen.

1. Was passiert beim ersten Aufruf von `shmget`?
2. Was passiert, wenn `shmget` mit identischen Parametern (innerhalb des gleichen oder aus einem anderen Prozess heraus) erneut aufgerufen wird?
3. Wofür wird der erste Parameter `key` der Funktion `shmget` benötigt?
4. Wie können Sie mit der Funktion `ftok` in zwei Prozessen einen übereinstimmenden `key`-Wert erzeugen?
5. Mit welcher Funktion können Sie einen Pointer auf den von `shmget` erzeugten Shared Memory erhalten?
6. Wie teilen Sie `shmat` mit, auf welchen Shared Memory der Pointer verweisen soll?
7. Mit welcher Funktion kann ein Prozess dem System mitteilen, dass er den Shared Memory nicht mehr benötigt?

Gegeben sei ein Prozess 'Producer', der ein gemeinsames Segment im Speicher alloziert. Dieses Segment besteht aus x Bytes, wobei x die Anzahl Bytes ist, die für die Instanzierung einer Integer Datenstruktur vom System verwendet wird. Nachdem der Producer sicherstellt, dass das Segment erfolgreich alloziert wurde, schreibt dieser 27 Integer nacheinander in das gemeinsame Segment. Nach dem Schreiben jedes Integers wartet der Producer, bis der 'Consumer' den Integer gelesen hat. Nachdem alle Integer geschrieben sind, wird das gemeinsame Segment dealloziert und der Prozess beendet. Den Code des Producers finden Sie in der Vorlage `producer.c`.

Implementieren Sie nun einen 'Consumer' Prozess, der folgende Anforderungen erfüllt:

- Das gemeinsame Segment, das vom Producer initialisiert wird, wird vom 'Consumer' referenziert und dazu benutzt, die 27 vom Producer geschriebenen Integer zu lesen.
- Die 27 Integer sollen in der gleichen Reihenfolge gelesen werden, in der sie vom Producer geschrieben wurden. Vermeiden Sie Synchronisationsprobleme, indem Sie Spinlocks bei Producer und Consumer gebrauchen. Definieren Sie dafür geeignete Integer-Konstanten, die als Signalisierungsdaten über das gemeinsame Segment ausgetauscht werden. Nehmen Sie dabei an, dass der Producer Prozess immer vor dem Consumer Prozess gestartet wird.

Benutzen Sie als Vorlage für die Implementierung des Consumer Prozesses die Vorlage `consumer.c`. Ihr Consumer ist korrekt implementiert, wenn ein berühmtes Einstein-Zitat angezeigt wird durch `tester.c`.

Zum Testen starten Sie die Prozesse in verschiedenen Shells oder als Hintergrundprozess. Da `producer.c` und `consumer.c` while-Schleifen verwenden, kann es passieren, dass eine while-Schleife nicht terminiert, falls Ihr Programm fehlerhaft ist. Sie könnnen dann die entsprechende Prozess-ID mit `ps -ef | grep consumer` oder `ps -ef | grep producer` herausfinden und den Prozess mit dem `kill -9 <PID>` beenden.

Die Beantwortung der folgenden Fragen ist freiwillig:

1. Mit welchem Befehl wird shared memory gelöscht?
2. Wird der Shared Memory direkt beim Aufruf von (Antwort von obiger Frage) gelöscht?
3. Lassen Sie die Shared Memory ID `shmid` des Producers ausgeben. Starten Sie nun Producer und Consumer wie zuvor und rufen Sie mehrmals die Funktion `ipcs` auf, während Producer und Consumer laufen (und einmal, nachdem beide Programme beendet worden sind). Was können Sie beobachten?