Hochschule RheinMain
Fachbereich DCSM - Informatik
Prof. Dr. Reinhold Kröger
Prof. Dr. Martin Gergeleit

Betriebssysteme und Rechnerarchitektur
WS 2015/16
LV 3142

Übungsblatt 6
Bearbeitungszeit 2 Wochen
Abgabetermin: 25.01.2016, 4:00 Uhr

In dieser Übung werden gemeinsam genutzte Speichersegmente (Shared Memory Segment oder Shared Memory Object, vgl. Kap. 8.8) betrachtet. Als Programmierschnittstelle soll das POSIX-API benutzt werden:

`int shm_open(const char *name, int oflag, mode_t mode);`	Öffnen oder Erzeugen und Öffnen eines POSIX Shared Memory Segments mit einer initialen Größe von 0 Bytes.
`int ftruncate(const char *path, off_t length);`	Setzen der Größe des Shared Memory Segments.
`int shm_unlink(const char *name);`	Entfernen eines Shared Memory Segments.
`void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);`	Einblenden einer Datei oder eines Shared Memory Objektes in den Adressraum des aktuellen Prozesses.
`int munmap(void *addr, size_t length);`	„Ausblenden“ eines Teils des Adressraum.

Die POSIX Shared Memory Segmente sind im Dateisystem unter `/dev/shm` sichtbar. Weiterführende Informationen erhalten Sie als Übersicht in der Manual-Page `shm_overview`.

Aufgabe 6.1 (Vorbereitendes Beispiel, Abgabe):

In dieser Aufgabe wird ein Shared Memory Segment als einelementiger Kommunikationspuffer zur Realisierung einer Kommunikationsbeziehung zwischen mehreren Schreiber-Prozessen und einem Leser-Prozess verwendet. Die Schreiber-Prozesse legen Nachrichten ab, der Leser-Prozess liest auf Anforderung durch den Benutzer den Kommunikationspuffer und gibt die aktuell enthaltene Nachricht auf dem Bildschirm (über Standardausgabe) aus. Das Shared Memory Segment ist damit ein gemeinsam benutztes Betriebsmittel aller Prozesse.

(a) Schreiben Sie ein Programm `create.c`, das ein Shared Memory Segment mit einem über die Kommandozeile übergebenen Namen und einer festen Größe erzeugt, die für die im Folgenden betrachtete Kommunikationsaufgabe ausreichend ist. Die Struktur des Shared Memory Segments soll in `shared.h` beschrieben werden.

(b) Schreiben Sie ein Programm `destroy.c`, welches ein mit dem Programm aus (a) erzeugtes Shared Memory Segment löscht. Auch hier soll der Name der Shared Memory Segments über die Kommandozeile angegeben werden.

(c) Schreiben Sie ein Programm `schreiber.c`, dem beim Aufruf über `argv[]` der Name des Shared Memory Segments, ein Zeichen `x` (z.B. 'A'), eine Länge `length` (≤40) und

ein Wartezeit-Parameter `sleep_time` (in Millisekunden, z.B. 1) übergeben werden. Das Programm mache sich das in (a) erzeugte Segment zugänglich. Es schreibe dann zyklisch `ITERATIONS=10` mal eine Nachricht in das Shared Memory Segment mit folgender Struktur: `(int pid, int i, int length, xx...x)`. (Achtung: Es sind die Datentypen **binär** abzulegen und nicht in Zeichenketten zu wandeln!). Dabei sei `pid` die pid des ausführenden Prozesses, `i` die Nummer des aktuellen Iterationsschritts, `length` (s.o.) die mit dem Aufruf übergebene Anzahl der folgenden Zeichen xx...x, wobei x für das mit dem Aufruf übergebene Zeichen steht (hierdurch können Sie die Ausgaben von verschiedenen Prozessen einfach unterscheiden). Achtung: die Zeichenfolge xx..x bildet keinen Null-terminierten String, deshalb muss die Anzahl der abgelegten Zeichen über `length` explizit mitangegeben werden. Nach der Ausgabe eines Zeichens sowie (zusätzlich) nach Abschluss jeder Iteration lege sich der Prozess für `sleep_time` Millisekunden schlafen, bevor er das nächste Zeichen schreibt bzw. mit dem nächsten Iterationsschritt beginnt (zeitliche Dehnung, damit Sie besser beobachten können).

(d) Schreiben Sie ein Programm `leser.c`, das sich zu Beginn das Shared Memory Segment zugänglich macht (Name über Kommandozeile) und anschließend bei Eingabe eines Zeichens von Standardeingabe mittels `getchar()` die Datenstruktur im Shared Memory Segment liest und als lesbare Zeile z.B. in der Form `pid, i: xx...x` ausgibt.

(e) Legen Sie ein Shell-Skript run.sh an, welches mehrere Schreiber-Prozesse im Hintergrund erzeugt.

(f) Dokumentieren Sie Ihre Beobachtungen in die Datei `beobachtungen.txt`. Welchen Effekt können Sie erkennen, wenn Sie mit dem Programm `leser` wiederholt den Puffer auslesen? Was passiert, wenn Sie das Shared Memory Segment löschen, während Leser und Schreiber noch darauf zugreifen?

Aufgabe 6.2 (Synchronisierung, Abgabe):

In dieser Aufgabe sollen, wie schon im vorherigen Übungsblatt, die Ausgaben der Schreiber-Prozesse synchronisiert werden. Legen Sie dazu zusätzlich eine POSIX-Semaphore mit im Shared-Memory Segment an, die die Schreiber untereinander synchronisiert.

(a) Erweitern Sie das Programm `create.c` um die Initialisierung des Semaphors. Beachten Sie, dass auf das Semaphor-Objekt von mehreren Prozessen gleichzeitig zugegriffen werden soll (`pshared = 1`).

(b) Erweitern Sie `schreiber.c` entsprechend um die geforderte Synchronisierung.

Aufgabe 6.3 (Speisende Philosophen, Abgabe):

In dieser Aufgabe wird wiederholt das Problem der speisenden Philosophen betrachtet. Gehen Sie wieder von der korrekten Lösung des Philosophenproblems aus, wie sie in der Vorlesung (Folien 5-60ff) für N Philosophen vorgestellt wurde. Die Philosophen werden jetzt auf UNIX-Prozesse abgebildet (anstelle von Threads in Aufgabe 4.2). Programmieren Sie die Lösung, indem Sie den gemeinsam benutzten Zustandsvektor `state` aller Philosophen sowie die benötigten Semaphoren in einem Shared Memory Segment anlegen. Gehen Sie so vor, dass der initiale Prozess alle gemeinsamen IPC-Objekte (Anonymes Shared Memory Segment, siehe `MAP_ANONYMOUS` bei `mmap()`, und Semaphoren aus dem bekannten POSIX API) erzeugt und für jeden Philosophen einen Kindprozess erzeugt, der die Funktion `philosopher()` ausführt (eine Überlagerung mittels `exec()` etc. wird nicht gefordert). Machen Sie sich zunächst klar, wieviele Semaphoren benötigt werden, sowie über deren Bedeutung und Initialwerte. Die Kindprozesse sollen wieder RUNDEN=10 mal speisen,

bevor sie terminieren. Der Elternprozess wartet; bis alle Kindprozesse fertig sind und gibt danach die benutzten Betriebsmittel wieder frei.

Organisieren Sie die Ausgabe des Programms so, dass die relevanten Zustände der Philosophen sichtbar werden und alle Ausgaben eines Philosophen untereinander in einer Gruppe von Spalten erscheinen (vgl. 4.2). Testen Sie die Lösung für N=1, 2, 3, 5 und 8 Philosophen. Dokumentieren Sie die Ausgaben Ihrer Testläufe in einer Datei `philosophen.txt`.

Bewertung:

Aufgabe	**Kriterien**	**Punkte**
6.1 (a)	SHM erzeugen	1
	SHM Größe setzen	1
(b)	SHM löschen	0.5
(c)	SHM einblenden	1
	Funktionalität Schreiber-Prozesse	2
(d)	Funktionalität Leser-Prozess	1
(e)	run.sh-Skript	0.5
(f)	Beobachtungen / Dokumentation	1
6.2 (a)	Semaphor Initialisierung	1
(b)	Funktionalität Synchronisation	1
6.3	Bereitstellung der IPC-Objekte	2
	Philosophen als Kindprozesse	1
	Synchronisierung	2
	Korrekte Funktionalität	3
	Initialer Prozess wartet auf Kindprozesse	1
	Aufräumen / Freigabe IPC-Objekte	1
	Beobachtungen / Dokumentation	1
	Abzüge bei fehlender Return-Code-Behandlung	(-3)
	Abzüge bei Compiler-Warnungen	(-2)
	Abzüge Lesbarkeit / Kommentare	(-2)
	Gesamt	**20 / 21**