Zugriffskontrolle in Mehrkern-Systemen

Eine Ausarbeitung im Rahmen des KvBK-Seminars

Eduard Potwigin
Friedrich-Alexander-Universität Erlangen-Nürnberg
eduardpotwigin@web.de

ABSTRACT

Diese Ausarbeitung behandelt eine empirische Evaluation zweier Synchronisationsprotokolle in Mehrkern-Systemen. Dabei wird auf die Grundlagen der Protokolle sowie die Implementierung und die dabei auftretenden Herausforderungen als auch die Ergebnisse von zwei Fallstudien zu der Implementierung eingegangen.

1. EINLEITUNG

In Echtzeitsystemen mit harten oder festen Terminen ist die Einhaltung von Fristen sowohl periodischer als auch aperiodischer oder sporadischer Aufgaben ein wichtiges Kriterium für ein korrektes Systemverhalten. Dabei ist nicht nur ein guter Ablaufplan wichtig, sondern auch Synchronisationsprotokolle, die die Koordinationsschwierigkeiten lösen, die beim Betreten von kritischen Abschnitten und dem Zugriff auf gemeinsame Betriebsmittel(BM) zwischen verschiedenen Arbeitsaufträgen auftreten. Eines dieser Probleme lautet Prioritätsinversion. Das tritt ein, wenn ein Arbeitsauftrag trotz höherer Priorität auf einen anderen Arbeitsauftrag warten muss. Im schlimmsten Fall können Konflikte um BM ohne Synchronisationsprotokolle zu Verklemmungen führen. Daher sind diese Protokolle unverzichtbar. Die existierenden Synchronisationsprotokolle haben Stärken und Schwächen. Deshalb und weil im Vergleich zu Einkern-Systemen in Multikern-Systemen noch wenig Erfahrungen zu Synchronisationsprotokollen erzielt wurden, werden hier auf [5] und [2] aufbauend die Protokolle MPCP (*multiprocessor priority ceiling protocol*) und MSRP (*multiprocessor stack resource policy*) mit den Ablaufplänen EDF (*earliest deadline first*) und RM (*rate-monotonic*) miteinander verglichen. Im folgenden Abschnitt wird zuerst in Synchronisationsprotokolle bei Einkern-Systemen eingeführt. Anschließend folgt eine Erläuterung der angewendeten Protokolle. Daraufhin wird auf die Implementierung in Ada unter Zuhilfenahme von C/POSIX, da Ada laut [6] nicht die gewünschte Prozessorkernansteuerung bietet, eingegangen. In Kapitel 5 werden zwei Fallstudien präsentiert und in Kapitel 6 sind die von [6] dazu entsprechenden erhaltenen Ergebnisse zu finden. Zuletzt wird die Interpretation der Ergebnisse verdeutlicht.

ACM ISBN .
DOI:

2. ZUGRIFFSKONTROLLE IN EINKERN-SYSTEMEN

Ein Synchronisationsprotokoll kümmert sich um die Minimierung von unkontrollierter Prioritätsumkehr. Allgemein tritt Prioritätsumkehr ein, wenn ein Arbeitsauftrag mit hoher Priorität auf einen Arbeitsauftrag mit niedrigerer Priorität warten muss. Der Grund dafür sind unteilbare BM, weil sie gleichzeitig nur von einem Arbeitsauftrag belegt werden können. Sind zwei Arbeitsaufträge im Streit und der niedriger priore Auftrag besitzt das BM, dann wird der höher priorisierte Auftrag solange blockiert bis der kritische Abschnitt vom niedriger prioren Auftrag verlassen wurde. Man spricht von Prioritätsvererbung, wenn dies durch ein temporäres Überschreiben der Priorität des niedriger priorisierten Arbeitsauftrages mit der Priorität des höher priorisierten Auftrages für die Dauer des kritischen Abschnitts gelöst wird. Realisiert wird die Blockade durch wechselseitigen Ausschluss in Form von Semaphoren. Je nach Protokoll kann ein Arbeitsauftrag auf verschiedene Arten blockiert werden:

1. *Direkte Blockierung:* Ein höher priorisierter Arbeitsauftrag fordert ein gesperrtes BM an, das von einem niedriger priorisiertem Arbeitsauftrag belegt ist. Tritt auf bei NPCS *(non-preemptive critical sections)*. Dabei wird ein Arbeitsauftrag unverdrängbar, wenn er ein BM verwendet.
2. *Blockierung durch Vererbung:* Ein niedriger priorisierter Arbeitsauftrag erbt aus der direkten Blockierung die Priorität des höher priorisierten Arbeitsauftrages. Dabei wird ein mittel priorisierter Arbeitsauftrag blockiert, der das BM aber nicht anfordert. Tritt auf bei Prioritätsvererbung und PCP *(priority ceiling protocol)* siehe [1].
3. *Blockierung durch Prioritätsobergrenze:* Erweiterung des Vererbungsprinzips. Ein Arbeitsauftrag wird von der Prioritätsobergrenze des Systems blockiert, da sich ein Arbeitsauftrag noch im kritischen Abschnitt befindet, dessen Obergrenze eine höhere Priorität besitzt als der blockierte Arbeitsauftrag. Tritt bei PCP auf und entspricht dem Prinzip der Obergrenze.

Bei einer Menge $\Omega = \{t1, t2, ..., tn\}$ von Arbeitsaufträgen mit einer Priorität p_i lautet die Prioritätsobergrenze für ein

BM r^k: *ceil(r^k) = max_i ($p_i|t_i$ benutzt r^k)*. Wenn sich kein Arbeitsauftrag in einem kritischen Abschnitt befindet, dann ist die Prioritätsobergrenze kleiner als die kleinste vorhandene Priorität.

3. ZUGRIFFSKONTROLLE IN MEHRKERN-SYSTEMEN

Zusätzlich zu den im vorangegangenen Kapitel angegebenen Arten der Blockierung kommt in Mehrkern-Systemen die *entfernte Blockierung* hinzu. Diese tritt ein, wenn ein Arbeitsauftrag auf einen anderen Arbeitsauftrag, der auf einen anderen Prozessor ausgeführt wird, warten muss. Dabei spielt die Priorität keine Rolle.

3.1 MPCP

MPCP angewandt auf einem Einkern-System gleicht PCP. Daher ist im Grunde MPCP eine Erweiterung von PCP. Für den Fall, dass zwei Arbeitsaufträge, die auf zwei verschiedenen Prozessoren ausgeführt werden, das gleiche BM belegen möchten, gibt es die globale Prioritätsobergrenze. Um unkontrollierte Prioritätsumkehr zwischen verschiedenen Prozessorkernen zu vermeiden, muss die globale Prioritätsobergrenze über den Prioritäten aller Aufträge liegen. PCP nutzt lokale Semaphoren um BM zu schützen. Zusätzlich werden im Multikernbetrieb (MPCP) globale Semaphoren verwendet. Wenn nun ein Auftrag eine blockierte globale Semaphore sperren möchte, dann wird er in eine prioritätsorientierte Warteschlange mit seiner aktuellen Priorität eingereiht. Während er wartet, verändert sich seine Priorität nicht. Dementsprechend darf derjenige Arbeitsauftrag aus der Warteschlange mit der höchsten Priorität die Semaphore belegen, sobald eine globale Semaphore freigegeben wird. Ist ein Auftrag blockiert und belegt dabei aber keine Semaphore(z.B. entfernte Blockierung), darf ein niedriger priorer Auftrag eingelastet werden. Nach [7] unterscheidet man PCP in in zwei Formen: *OCPP (Original Ceiling Priority Protocol)* und *ICPP (Immediate Ceiling Priority Protocol)*. Der Unterschied liegt im Zeitpunkt, wann die Prioritäten angepasst werden, sobald ein Auftrag ein BM belegt und ihn ein höher priorisierter Auftrag anschließend verdrängen möchte. Bei OCPP erbt der Arbeitsauftrag, nachdem er ein BM belegt hat, erst die Priorität eines höherprioren Auftrages, wenn dieser versucht ihn zu verdrängen. Danach (falls nötig) wird seine Priorität an die Prioritätsobergrenze des BMs nochmals angepasst. Bei ICPP hingegen wird die Priorität direkt nach Belegen des BMs an die Obergrenze angepasst. Wenn im Folgenden von MPCP die Rede ist, dann ist die OCPP Variante zu verstehen.

3.2 MSRP

SRP aus [1] funktioniert wie ICPP, bis auf Verdrängeungswerte anstatt Prioritäten zu verwenden. Die Verdrängeungswerte basieren auf den relativen Fristen den Arbeitsaufträge. Will ein neuer Arbeitsauftrag einen eingelasteten Arbeitsauftrag verdrängen, so geschieht dies nur, wenn seine absolute Frist kürzer und sein Verdrängeungswert größer als die Obergrenze der eventuell belegten Semaphore ist. Weiterhin ist zu beachten, dass sobald ein Arbeitsauftrag eine Semaphore belegt, dieser für diese Zeitspanne nicht verdrängbar ist. Da bei MSRP keine Prioritäten existieren, werden die auf BM wartenden Arbeitsaufträge in eine FCFS *(first come first served)* Warteschlange eingereiht. In MSRP gerät nach [2, 3] ein Auftrag in einen aktiven Wartezustand *(spinlock)*, wenn er auf ein belegtes globales BM wartet, das von einem Arbeitsauftrag eines anderen Kerns blockiert wird. Obwohl das aktive Warten Laufzeit in Anspruch nimmt, liegt so eine Lösung nahe, da zwischen den Kernen keine Synchronisation bezüglich der laufenden Arbeitsaufträgen auf Hardwareebene besteht.

4. IMPLEMENTIERUNG

Für die Implementierung hat sich der Autor aus [6] für Ada-2005 entschieden, da die Sprache für Echtzeitanwendungen geeignet ist und bereits implementierte Konstrukte bietet, die die Entwicklung so einer Anwendung vereinfachen. Eines dieser vorteilhaften Konstrukte ist das Schlüsselwort *protected*. Wird ein Objekt als *protected* deklariert, so wird der Zugriff darauf durch Semaphoren geschützt. Wenn ein Arbeitsauftrag nun ein BM anfrägt, wird ein Server-Prozess in Gang gesetzt, der alle Semaphorenbelegungen überprüft und entscheidet ob das BM gewährt wird. In der Implementierung heißt dieser Server "Resource Scheduler" und wird ebenfalls mit dem *protected* Schlüsselwort versehen.

```
Type Scheduler is protected interface;
Procedure Request is abstract;
Procedure Release is abstract;
Protected type Resource_Scheduler is new Scheduler
...
```

Dieses Objekt bietet für anfordernde Arbeitsaufträge zwei Funktionen an. Das Anfordern (*Request*) und Freigeben (*Release*) von Semaphoren.

Request(Sj): P(Sj) oder Requeue(Sj) oder raise Ceiling_Error
Release(Sj): V(Sj) oder raise unauthorized Release_Error

Entsprechend den zwei Protokollen MPCP und MSRP in den vorangegangenen Kapiteln wird bei den beiden Funktionen verfahren. Ein "Ceiling_Error" entsteht, wenn die Grundregeln der Protokolle verletzt werden und ein "unauthorized Release_Error" entsteht, wenn versucht wird eine Semaphore freizugeben, die nicht belegt ist. Eine Schwäche von Ada-2005 ist die fehlende Unterstützung Prozesse auf ausgewählten Kernen ausführen zu lassen. Mit Ada-2005 ist Mehrfädigkeit zwar möglich, jedoch kann man die Kerne nicht direkt ansteuern. Deshalb nutzt man das "pragma"-Schlüsselwort um C-Quellcode zu importieren. Mit dem Einbinden der *sched.h*-Datei werden Funktionen, wie "CPU-SET", angeboten um einen Prozess auf einem speziellen Kern laufen zu lassen. Dabei wird eine Bitmaske manipuliert, die aus *n = Anzahl der Prozessorkerne* besteht. Somit kann man jeden Kern separat ansteuern. Um einen Kern direkt anzusprechen sieht der Quellcode wie folgt aus:

```
package SMP is - not all features shown
   function available_cpus return integer;
   pragma Import(C, available_cpus);
   - sets the affinity for a processor
   function set_affinity(cpu_n: natural; pid: natural) return integer;
   pragma Import(C, set_affinity);
   function get_affinity return integer; - returns
   the task's affinity
```

Table 1: GAP-Satz und Worst-Case-Antwortzeiten. Auszug aus [6]

Task	Period T	Deadline D	WCET C	WCRT PCP	WCRT SRP
1	200000	5000	3000	4180	4180
2	25000	25000	2100	6480	6480
3	25000	25000	4200	13180	14930
4	40000	40000	1000	14280	16030
5	50000	50000	3000	17580	19330
6	50000	50000	5000	23344	24694
7	59000	59000	8000	38848	40198
8	80000	80000	9000	49648	50998
9	80000	100000	2000	40780	43630
10	99995	115000	5000	98954	122604
11	200000	200000	1000	138794	140144
12	200000	200000	1000	138860	158510
13	200000	200000	1000	139926	159576
14	200000	200000	3000	144540	144540
15	199995	200000	3100	146488	146488
16	1000000	1000000	1000	147554	147554
17	1000000	1000000	1000	148620	148620
			Total:	1312056	1388506

Table 2: GAP-Satzt und Normalfall-Antwortzeiten. Auszug aus [6]

Task	Period T	Deadline D	WCET C	ACRT PCP	ACRT SRP
1	200000	5000	3000	3236	3236
2	25000	25000	2100	2704	2704
3	25000	25000	4200	7804	7834
4	40000	40000	1000	4207	5214
5	50000	50000	3000	11959	12018
6	50000	50000	5000	17764	18199
7	59000	59000	8000	18798	19681
8	80000	80000	9000	29628	33424
9	80000	100000	2000	19763	40260
10	99995	115000	5000	84008	89773
11	200000	200000	1000	115190	119060
12	200000	200000	1000	117776	122786
13	200000	200000	1000	118842	123852
14	200000	200000	3000	107736	115676
15	199995	200000	3100	122894	122894
16	1000000	1000000	1000	147554	147554
17	1000000	1000000	1000	148620	148620
			Total:	1078483	1132785

```
    pragma Import(C, get_affinity);
  end SMP;

int set_affinity (int cpu_n, pid_t pid) {
   int num_procs = sysconf (_SC_NPROCESSORS_CONF);
   if( (cpu_n < 0) || (cpu_n > (num_procs-1) ) )
   return Affinity_Error;
   cpu_set_t mask;
   // set all the bits in the mask to zero
   CPU_ZERO(&mask);
   // set only the bit corresponding to cpu_n
   CPU_SET(cpu_n, &mask);
   // set the CPU affinity mask of the task denoted
   // by pid
   return sched_setaffinity (pid, sizeof(mask),
   &mask);
  }
```

5. FALLSTUDIEN

Es wurden zwei Fallstudien behandelt um die beiden Protokolle zu vergleichen. Fallstudie 1 ist ein theoretischer Test, der die Antwortzeiten im Einkern-Fall (PCP und SRP) misst und Fallstudie 2 ist ein in der Industrie angewandter praktischer Test, der die Leistung basierend auf nicht eingehaltene Fristen, Kontextwechsel und Prioritätsänderungen im Multikern-Fall (MPCP und MSRP) evaluiert.

5.1 Fallstudie 1

Hier wird die *GAP(Generic Avionics Platform)* Fallstudie aus [4] repliziert. Es geht hierbei um 16 periodische Prozesse und einen sporadischen Prozess, welcher aber pro Periode ein bestimmtes Zeitfenster zugeteilt bekommt, in dem er eingelastet werden kann. Das Verfahren wurde leicht angepasst, damit kein Protokoll bevorzugt wird. Wie in den Tabellen "Table 1" und "Table 2" zu sehen ist, absolviert PCP für diese Version der GAP-Fallstudie in allen Punkten besser als oder gleich wie SRP. Als Scheduling-Algorithmus wurde RM (rate monotonic) verwendet. Man beachte den Unterschied zwischen der maximalen Laufzeit WCET *(worst case execution time)* und der maximalen Antwortzeit WCRT *(worst case response time)* und mittleren Antwortzeit ACRT *(average case response time)*.

5.2 Fallstudie 2

Die zweite Fallstudie befasst sich mit Echtzeit-Datenerfassung und -Verarbeitung von Sensorwerten und Informationen von verteilten Steuerungsrechnern in einem Stahlwerk. Genauere Informationen dazu sind in [6] zu finden. Einerseits wurden die Anzahl und Abfolge der Arbeitsaufträge von 10 bis 100 variiert. Andererseits wurde ebenso mit verschiedener Periodendauer getestet, die von 50 bis 1000 Millisekunden reichte. Dabei endet jede Frist eines Arbeitsauftrages mit seiner Periode. Ein Arbeitsauftrag fordert eine Semaphore von 0 bis 8 mal an und gibt sie nach 0,02 bis 0,2 Millisekunden im kritischen Abschnitt wieder frei.

6. ERGEBNISSE

Um eine bessere Perspektive zu gewinnen wurden die beiden Protokolle jeweils mit EDF und RM Ablaufplanern getestet und mit dem FCFS-Protokoll *(first come first served)* verglichen. Die gemessenen Eigenschaften umfassen:

1. verpasste Frist
2. Wartezeit für eine Betriebsmittelanforderung
3. Prioritätsänderung
4. Kontextwechsel

Die Ergebnisse sind in den Abbildungen "Figure 1" bis "Figure 4" zu sehen. Obwohl MSRP weniger Kontextwechsel (Abbildung: "Figure 4") benötigt als MPCP, schneidet MPCP aufgrund der stärkeren Orientierung an Prioritätsplanung leicht besser ab siehe Abbildungen "Figure 1" und "Figure 2".

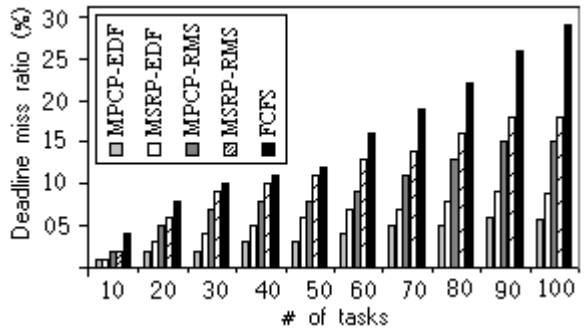

Figure 1: Prozentanteil verpasster Fristen – zufällig gewählte Perioden und starke Ressourcennutzung. Auszug aus [6]

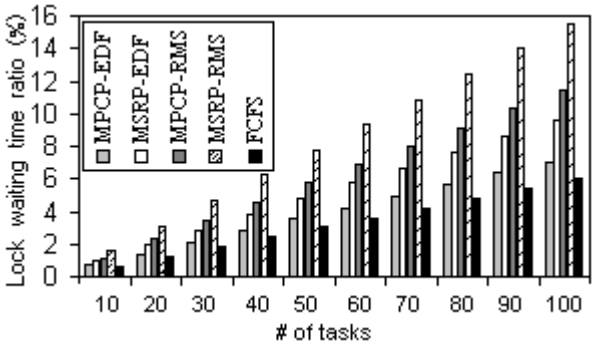

Figure 2: Wartezeitenanteil – zufällig gewählte Perioden und starke Ressourcennutzung. Auszug aus [6]

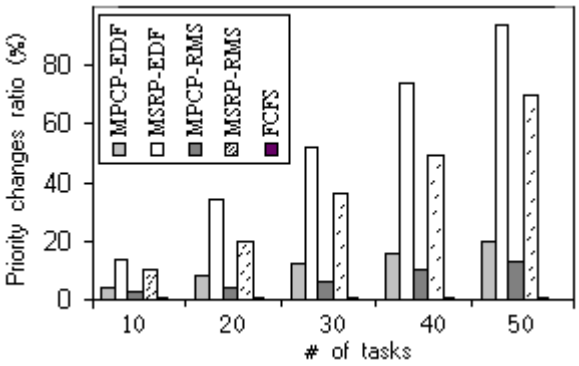

Figure 3: Prozentanteil der Prioriätswechselverhältnisse von MPCP, MSRP, FCFS. Auszug aus [6]

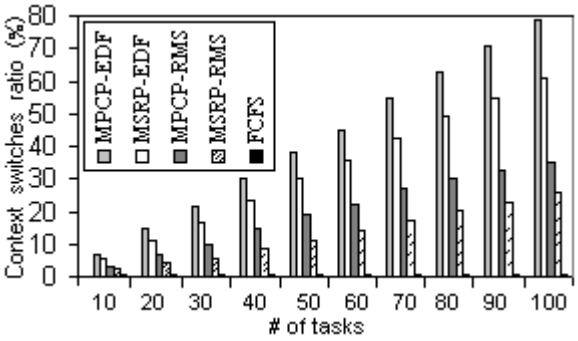

Figure 4: Prozentanteil der Kontextwechsel von MPCP, MSRP, FCFS. Auszug aus [6]

Denn erst wenn es wirklich nötig ist, dann werden Prioritäten unter MPCP geändert. Da MSRP direkt alle anfordernden Arbeitsaufträge blockiert, sobald ein BM belegt ist, führt es zu verlängerten Ausführungszeiten für die anfordernden Arbeitsaufträge. Unter FCFS sind die Kontextwechsel und Prioritätsänderungen am geringsten, da FCFS alle Arbeitsaufträge gleich behandelt. Da RM ein statischer und EDF ein dynamischer und optimaler Ablaufplaner sind, schneidet EDF besser ab. Für eine detaillierte Analyse der Ergebnisse ist auf [6] zu verweisen.

7. SCHLUSS

Zusammenfassend geben sich MPCP und MSRP nicht viel. Vor allem wenn man bedenkt, dass MSRP laut [6] eine bessere Implementierung durch einen angepassten Kernel erfahren könnte. Das könnte den Overhead reduzieren und die Leistung somit erhöhen. Beide Protokolle Haben ihre Stärken und Schwächen. Einerseits verursacht bei MPCP die Entscheidung, ob eine Prioritätsänderung geschehen soll, einen größeren Overhead, aber andererseits führt dies zu weniger Kontextwechsel als bei MSRP.

8. REFERENCES

[1] T. P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, 1991.

[2] P. Gai, M. Di Natale, G. Lipari, A. Ferrari, C. Gabellini, and P. Marceca. A comparison of mpcp and msrp when sharing resources in the janus multiple-processor on a chip platform. In *Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE*, pages 189–198. IEEE, 2003.

[3] P. Gai, G. Lipari, and M. D. Natale. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In *Real-Time Systems Symposium, 2001.(RTSS 2001). Proceedings. 22nd IEEE*, pages 73–83. IEEE, 2001.

[4] C. D. Locke, D. R. Vogel, T. Mesler, et al. Building a predictable avionics platform in ada: a case study. In *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, pages 181–189. IEEE, 1991.

[5] R. Rajkumar. *Synchronization in real-time systems: a priority inheritance approach*, volume 151. Springer Science & Business Media, 2012.

[6] J. Ras and A. M. Cheng. An evaluation of the dynamic and static multiprocessor priority ceiling protocol and the multiprocessor stack resource policy in an smp system. In *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE*, pages 13–22. IEEE, 2009.

[7] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *Computers, IEEE Transactions on*, 39(9):1175–1185, 1990.