Übungen zur Vorlesung
Betriebssysteme II
Wintersemester 2005/2006

AG BS
Jan Peleska

Blatt 3

Prozesswechselsperre

Eine Prozesswechselsperre verhindert, dass auf einer gegebenen CPU eine andere rechenbereite Applikation die CPU erhält, bevor nicht die aktive Applikation die Sperre wieder aufhebt. Interrupts sind in dieser Zeit weiterhin zugelassen, so dass Interrupthandler ggf. auch auf dieser CPU aktiv werden können. Bei 1-CPU Rechnern kann dieses Verfahren wirkungsvoll zum Schutz kritischer Sektionen verwendet werden, wenn sichergestellt ist, dass Interrupthandler niemals dieselben Datenbereiche manipulieren wie die Anwendungen in ihren kritischen Abschnitten. Im Gegensatz zu Semaphoren hat die Prozesswechselsperre den Vorteil, dass sie unabhängig davon wirkt, ob die anderen Applikationen auch einen Sperrbefehl geben (z.B. DOWN auf die Semaphore) oder nicht. Ein weiterer Vorteil besteht darin, dass die sperrende Anwendung zügig wieder aus dem kritischen Abschnitt kommt, denn die CPU wird ihr höchstens kurzfristig durch einen Interrupthandler weggenommen.
Bei Mehr-CPU-Systemen ist es dagegen nicht sinnvoll, alle CPUs für Applikationen zu sperren, da dann in dieser Phase nur die Leistung eines 1-CPU Systems zur Verfügung stünde.

Aufgabe 1: Entwurf der Systemaufrufe (30%)

Beschreiben Sie Signatur und Wirkungsweise eines Satzes von Systemaufrufen, der auch in Mehr-CPU-Systemen die sinnvolle Verwendung einer Prozesswechselsperre zulässt, indem zumindest folgende Anforderungen realisiert werden:

1. Ein aus mehreren Prozessen bestehendes Anwendungssystem kann dafür sorgen, dass diese Prozesse nur auf einer dedizierten CPU laufen. Hierzu wird ein Systemaufruf definiert, der die Wahl der CPU ermöglicht. Der Scheduler wird einem solchen Prozess dann niemals eine andere CPU geben.

2. Eine Applikation, die einer dedizierten CPU zugeordnet ist, kann mit 2 weiteren Systemaufrufen die Prozesswelchselsperre auf dieser CPU aktivieren und wieder aufheben. Der Scheduler trägt ein Sperrflag in die um dieses Flag zu erweiterende Prozesstabelle ein und gibt so lange keinem anderen Prozess die CPU, bis die Sperre wieder aufgehoben wird.

3. Blockierende System Calls (read, write, sleep, semop etc.) kehren mit Fehler `EACCES` (permission denied) zurück. Die Prozesswechselsperre bleibt jedoch bestehen. Bitte eine vollständige Liste dieser Aufrufe erstellen.

4. In keinem Fall blockierende Systemaufrufe (gettimeofday, kill, exit) sind erlaubt. Auch hier wieder eine vollständige Liste dieser Aufrufe erstellen.

5. Bei manchen Systemaufrufen ist es sinnvoll (z. B. `exit()`), die Prozesswechselsperre automatisch aufzuheben. Auch hier wieder eine vollständige Liste dieser Aufrufe erstellen.

6. Wird das Programm auf Grund eines Signals (z. B. SIGSEGV) beendet, wird die Sperre automatisch wieder aufgehoben.

Aufgabe 2: Implementierung der Systemaufrufe (40%)

Implementieren Sie die beschriebenen Systemaufrufe wieder mit dem BS2-eigenen Trap.

Aufgabe 3: Test der Implementierung (30%)

Programmieren Sie eine neue Lösung des in BS1 mit Semaphoren vorgestellten Producer-/Consumer-Problems, wobei alle beteiligten Programme auf der selben CPU aktiv sind und ihren kritischen Abschnitt mit Hilfe einer Prozesswechselsperre schützen. Entwickeln Sie eine geeignete Kodierung der zu produzierenden/konsumierenden Items, so dass man die Protokolle aller Producer und Consumer verwenden kann, um die korrekte Behandlung der kritischen Abschnitte nachzuweisen. Im Test sollen auch automatisch Signale an einzelne Programme geschickt werden.

Abgabe: Bis Mittwoch, 11.01.2006, in der Übung.

Sowohl bei der schriftlichen Lösung als auch im Source-Code die Namen aller Gruppenmitglieder nicht vergessen!