

Aufgabenblatt 9

Musterlösung

Vorlesung Verteilte Systeme

Wintersemester 2025/26

Aufgabe 1: Ring-Algorithmus

Wenn ein Prozess eine ELECTION-Nachricht empfängt, prüft er, wer sie gesendet hat. Hat er sie selbst gesendet (d.h. seine Nummer befindet sich am Listenanfang), wandelt er die Nachricht in eine COORDINATOR-Nachricht um, wie in der Vorlesung beschrieben. Hat er keine ELECTION-Nachricht gesendet, trägt er seine Prozessnummer ein und gibt die Nachricht im Ring weiter.

Hat er jedoch zuvor seine eigene ELECTION-Nachricht gesendet und entdeckt soeben einen Konkurrenten, vergleicht er die Prozessnummer des Ursprungsprozesses mit seiner eigenen. Hat der andere Prozess eine niedrigere Nummer, verwirft er die Nachricht, statt sie weiterzugeben.

Hat der andere Prozess eine höhere Nummer, wird die Nachricht wie üblich weitergegeben. Auf diese Weise bleibt nur die ELECTION-Nachricht mit der höheren Nummer übrig, wenn mehrere ELECTION-Nachrichten gesendet werden. Die restlichen werden entlang des Weges verworfen.

Aufgabe 2: Zentralisierter Algorithmus zum wechselseitigen Ausschluß

- a) Anforderungen könnten abhängig von ihrer Bedeutung Prioritätsstufen zugeordnet werden. Der Koordinator könnte dann die Anforderungen mit der höchsten Priorität zuerst bedienen.
- b) Angenommen, der Algorithmus gibt vor, dass jede Anforderung unmittelbar beantwortet wird, entweder durch eine Erlaubnis oder durch eine Verweigerung. Befinden sich keine Prozesse in kritischen Bereichen und in den Warteschlangen, ist ein Absturz nicht fatal.

Der nächste Prozess, der eine Berechtigung anfordert, erhält keine Antwort und kann dann die Wahl eines neuen Koordinators initiieren. Das System kann noch robuster gemacht werden, indem der Koordinator jede eingehende Anforderung auf der Festplatte speichert, bevor er eine Antwort zurücksendet. Auf diese Weise kann im Falle eines Absturzes der neue Koordinator die Liste der aktiven kritischen Bereiche und die Warteschlange rekonstruieren, indem er die Datei von der Festplatte liest.

Aufgabe 3: Algorithmus von Ricart und Agrawala

Angenommen, ein Prozess verweigert die Berechtigung und stürzt dann ab. Der anfordernde Prozess glaubt, er sei noch am Leben, aber die Berechtigung trifft niemals ein. Ein Ausweg ist, den Anforderer zu veranlassen, dass er nicht nur blockiert, sondern dass er eine bestimmte Zeiddauer lang *schläft*, wonach er alle Prozesse abfragt, die die Berechtigung verweigert haben, um zu prüfen, ob sie noch laufen.

Aufgabe 4: Deadlocks - Ricart und Agrawala

Das ist von den Grundregeln abhängig. Wenn die Prozesse streng sequentiell in kritische Bereiche eintreten, d.h. ein Prozess in einem kritischen Bereich darf nicht versuchen, in einen weiteren kritischen Bereich einzutreten, gibt es keine Möglichkeit, dass er blockiert, während er eine Ressource (d.h. einen kritischen Bereich) hält, die ein anderer Prozess

braucht. Das System ist Deadlock-frei. Kann andererseits Prozess 0 in den kritischen Bereich A eintreten und dann versuchen, in den kritischen Bereich B einzutreten, kann ein Deadlock auftreten, wenn ein anderer Prozess versucht, die kritischen Bereiche in umgekehrter Reihenfolge zu betreten. Der Algorithmus von Ricart und Agrawala selbst trägt nicht zu dem Deadlock bei, weil jeder kritische Bereich unabhängig von den anderen behandelt wird.