Übungen zu Systemprogrammierung 2

Ü 7 – Ringpuffer

Sommersemester 2019

Simon Ruderich, Dustin Nguyen, Christian Eichler, Jürgen Kleinöder

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg

Lehrstuhl für Verteilte Systeme
und Betriebssysteme

Agenda

7.1 Synchronisation des Ringpuffers

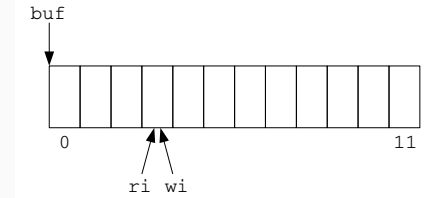
7.2 ABA-Problem bei der Verwendung von CAS

7.3 Vorteile nicht-blockierender Synchronisation

Agenda

Unter-/Überlaufsituationen

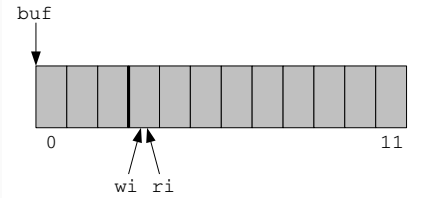
Leerer Ringpuffer:



Weiteres Lesen würde noch nicht gefüllten Slot liefern

$\rightarrow$ Unterlauf!

Voller Ringpuffer:



Weiteres Schreiben würde vollen Slot überschreiben

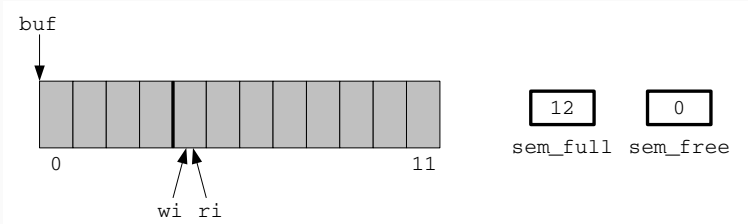
$\rightarrow$ Überlauf!

☞ Synchronisation mit Hilfe zweier Semaphore

Wettlauf der Leser

- Auslesen des Slots und Inkrementieren des Leseindex `ri` geschieht nicht atomar
  - Mehrere Threads könnten nebenläufig den selben Slot auslesen
- Synchronisation mittels *Compare and Swap* (CAS)

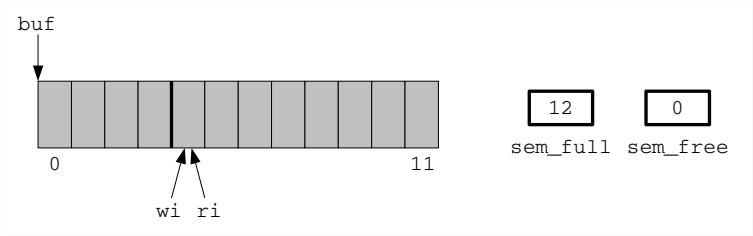
Wettlauf der Leser



- Erhöhen des Leseindex mittels CAS – vollständig korrekt?

```
int get(void) {
  int fd, pos, npos;
  P(sem_full);
  do { // Wiederhole...
    pos = ri;              // Lokale Kopie des Werts ziehen
    npos = (pos + 1) % 12; // Folgewert lokal berechnen
  } while(!cas(&ri, pos, npos)); // ... bis CAS erfolgreich
  fd = buf[pos];
  V(sem_free);
}
```

Wettlauf der Leser



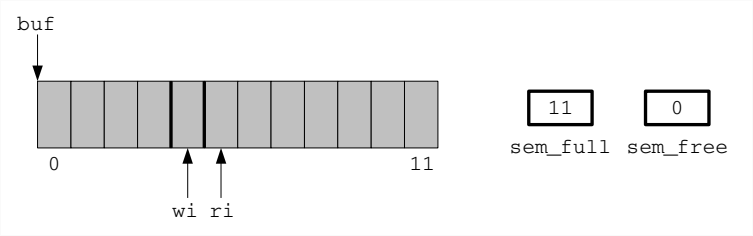
- Überlaufsituation: Schreiber blockiert, weil keine Slots frei

```
int get(void) {
  int fd, pos, npos;
  P(sem_full);
  do {
    pos = ri;
    npos = (pos + 1) % 12;
  } while(!cas(&ri, pos, npos));
  fd = buf[pos];
  V(sem_free);
  return fd;
}
```

```
void add(int val) {
  P(sem_free);
  buf[wi] = val;
  wi = (wi + 1) % 12;
  V(sem_full);
}
```

W

Wettlauf der Leser



- R1 sichert sich Leseindex 4, wird nach erfolgreichem CAS verdrängt

```
int get(void) {
  int fd, pos, npos;
  P(sem_full);
  do {
    pos = ri;
    npos = (pos + 1) % 12;
  } while(!cas(&ri, pos, npos));
  fd = buf[pos];
  V(sem_free);
  return fd;
}
```

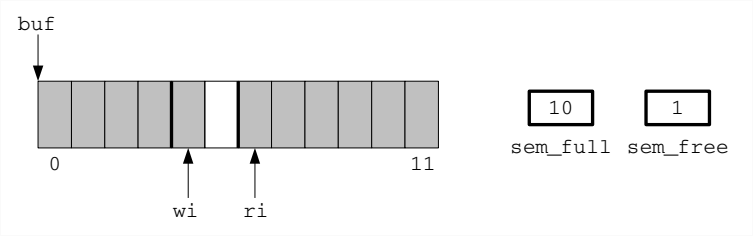
R1

pos: 4

```
void add(int val) {
  P(sem_free);
  buf[wi] = val;
  wi = (wi + 1) % 12;
  V(sem_full);
}
```

W

Wettlauf der Leser



- R2 durchläuft `get()` komplett, entnimmt Datum in Slot 5

```
int get(void) {
  int fd, pos, npos;
  P(sem_full);
  do {
    pos = ri;
    npos = (pos + 1) % 12;
  } while(!cas(&ri, pos, npos));
  fd = buf[pos];
  V(sem_free);
  return fd;
}
```

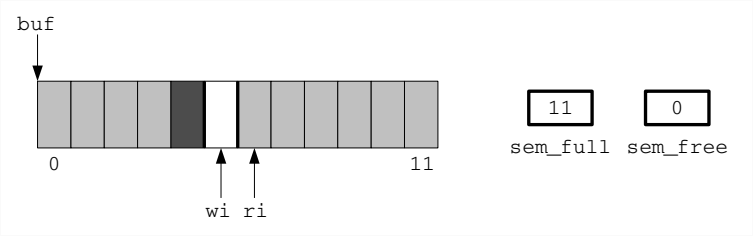
R1 pos: 4

R2 pos: 5

```
void add(int val) {
  P(sem_free);
  buf[wi] = val;
  wi = (wi + 1) % 12;
  V(sem_full);
}
```

W

Wettlauf der Leser



- W wird deblockiert, komplettiert `add()` und überschreibt Slot 4

```
int get(void) {
  int fd, pos, npos;
  P(sem_full);
  do {
    pos = ri;
    npos = (pos + 1) % 12;
  } while(!cas(&ri, pos, npos));
  fd = buf[pos];
  V(sem_free);
  return fd;
}
```

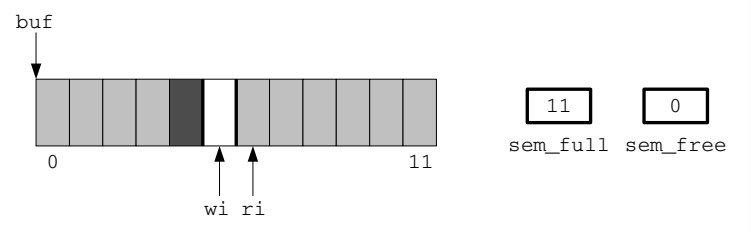
R1 pos: 4

R2 pos: 5

```
void add(int val) {
  P(sem_free);
  buf[wi] = val;
  wi = (wi + 1) % 12;
  V(sem_full);
}
```

W

Wettlauf der Leser



- Ursache: FIFO-Entnahmeeigenschaft des Puffers nicht sichergestellt

```
int get(void) {
  int fd, pos, npos;
  P(sem_full);
  do {
    pos = ri;
    npos = (pos + 1) % 12;
  } while(!cas(&ri, pos, npos));
  fd = buf[pos];
  V(sem_free);
  return fd;
}
```

R1 R2

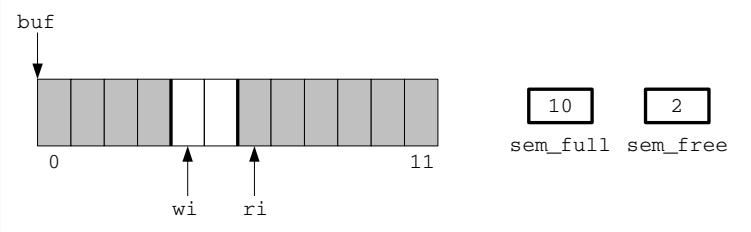
pos: 4

pos: 5

```
void add(int val) {
  P(sem_free);
  buf[wi] = val;
  wi = (wi + 1) % 12;
  V(sem_full);
}
```

W

Wettlauf der Leser



- Lösung: Entnahme des Datums **innerhalb** der CAS-Schleife

```
int get(void) {
  int fd, pos, npos;
  P(sem_full);
  do {
    pos = ri;
    npos = (pos + 1) % 12;
    fd = buf[pos]; // Datum bereits vorsorglich entnehmen
  } while(!cas(&ri, pos, npos));
  V(sem_free);
  return fd;
}
```

Brauchen wir das `volatile`-Schlüsselwort?

Schreibindex

- Szenario: nur ein Produzenten-Thread
  - Kein nebenläufiger Zugriff auf den Schreibindex
  - `volatile` nicht erforderlich

Leseindex

- Szenario: mehrere Konsumenten-Threads möglich
  - Nebenläufiger Zugriff auf den Leseindex möglich
  - C11 Atomics: *[Strong] atomic operations [like* `atomic_compare_exchange_strong`*] not only order memory [such that] everything that happened-before a store in one thread becomes a visible side effect in [another] thread, but also establish a single total modification order of all [strong] atomic operations.*
  - `volatile` also nicht falsch, aber nicht zwangsläufig erforderlich

Agenda

ABA-Problem bei der Verwendung von CAS

T1

```
bbGet();
```

T2

```
bbGet();
bbPut(7);
bbGet();
```


ABA-Problem bei der Verwendung von CAS

T1

```
bbGet();
```

```
bbGet() {
  ...
  int retVal = 0;
  P(full);
  do {
    ...
    retVal = 5;
  } while(!cas(&r, 0, 1));
  ...
  V(empty);
}
```

T2

```
bbGet();
bbPut(7);
bbGet();
```

ABA-Problem bei der Verwendung von CAS

T1

```
bbGet();
```

```
bbGet() {
  ...
  int retVal = 0;
  P(full);
  do {
    ...
    retVal = 5;
  } while(!cas(&r, 0, 1));
  ...
  V(empty);
}
```

T2

```
bbGet();
bbPut(7);
bbGet();
```


ABA-Problem bei der Verwendung von CAS

T1

```
bbGet();
```

```
bbGet() {
  ...
  int retVal = 0;
  P(full);
  do {
    ...
    retVal = 5;
  } while(!cas(&r, 0, 1));
  ...
  V(empty);
}
```

T2

```
/* 5 */ bbGet();
bbPut(7);
bbGet();
```

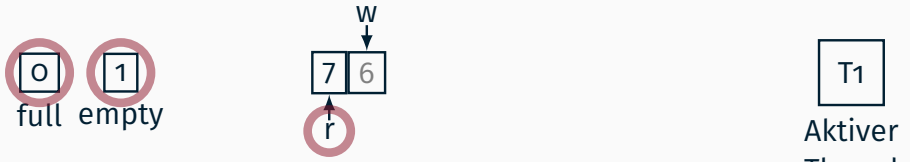
ABA-Problem bei der Verwendung von CAS

T1 T2

```
bbGet();      bbGet() {
                ...
                int retVal = 0;
                P(full);
                do {
                  ...
                  retVal = 5;
                } while(!cas(&r, 0, 1));
                ...
                V(empty);
              }
```

```
/* 5 */ bbGet();
        bbPut(7);
        bbGet();
```


ABA-Problem bei der Verwendung von CAS

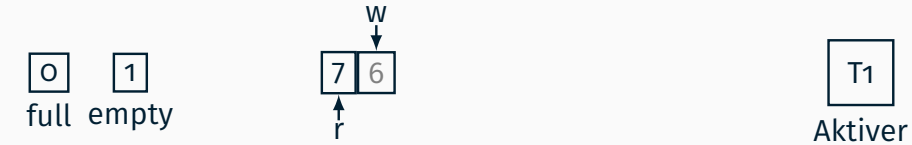


T1 T2

```
bbGet();      bbGet() {
                ...
                int retVal = 0;
                P(full);
                do {
                  ...
                  retVal = 5;
                } while(!cas(&r, 0, 1));
                ...
                V(empty);
              }
```

```
/* 5 */ bbGet();
        bbPut(7);
/* 6 */ bbGet();
```

ABA-Problem bei der Verwendung von CAS



T1

```
bbGet();      bbGet() {
                ...
                int retVal = 0;
                P(full);
                do {
                  ...
                  retVal = 5;
                } while(!cas(&r, 0, 1));
                ...
                V(empty);
              }
```

T2

```
/* 5 */ bbGet();
        bbPut(7);
/* 6 */ bbGet();
```


ABA-Problem bei der Verwendung von CAS

- bbGet() liefert 5 statt 7 zurück
  - CAS schlägt nicht fehl, weil r nach dem Wiedereinlasten des Threads den selben Wert hat wie vor dessen Verdrängung
  - Zwischenzeitliche Wertänderung von r wird nicht erkannt
- Grundsätzliches Problem von inhaltsbasierten Elementaroperationen wie CAS
- Erhöhte Auftrittswahrscheinlichkeit, je kleiner der Puffer und je höher die Systemlast
- Gegenmaßnahmen siehe Vorlesung C | X-4 S. 24ff.

ABA-Problem in den Griff bekommen

- Einführen eines Generationszählers, der bei jeder erfolgreichen Operation inkrementiert wird
- ABA-Situation: Leseindex hat nach Umlaufen des Ringpuffers wieder den alten Wert – aber Generationszähler hat anderen Wert
  $\rightarrow$ CAS schlägt fehl
- **Möglichkeit 1:** separate Zählvariable
  - Erfordert *Double-Word-CAS*
- **Möglichkeit 2:** eingebetteter Generationszähler
  - Nutzung der oberen Bits des Leseindex
- Keine hundertprozentige Sicherheit möglich:
  - Generationszähler hat begrenzten Wertebereich und kann überlaufen
  - Je nach Größe des Zählers und konkretem Szenario (hoffentlich) ausreichend unwahrscheinlich

Agenda

Vorteile nicht-blockierender Synchronisation

- Vorteile gegenüber sperrenden oder blockierenden Verfahren (Auswahl):
  - Rein auf Anwendungsebene, keine teuren Systemaufrufe
  - Geringere Mehrkosten als bei Locking, wenn die CAS-Operation auf Anhieb funktioniert
  - Konkurrierende Fäden werden vom Scheduler nach dessen Kriterien eingeplant
  - Durch Locks wird eine Abhängigkeit vom Halter des Locks geschaffen:
    - Halter des Locks wird möglicherweise im kritischen Abschnitt verdrängt
    - Der „Zweite“, „Dritte“ usw. werden durch den „Ersten“ verzögert
- In unserem konkreten Anwendungsbeispiel kommen diese Vorteile nicht wirklich zum Tragen
  - Übungsbeispiel zum Begreifen des Konzepts