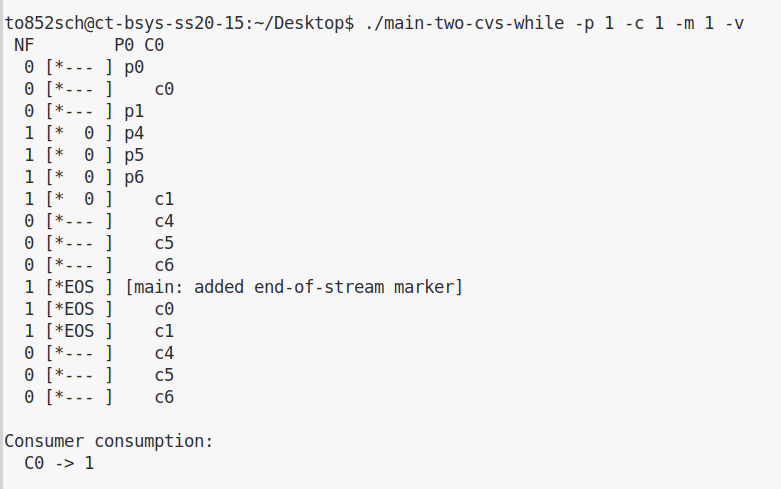
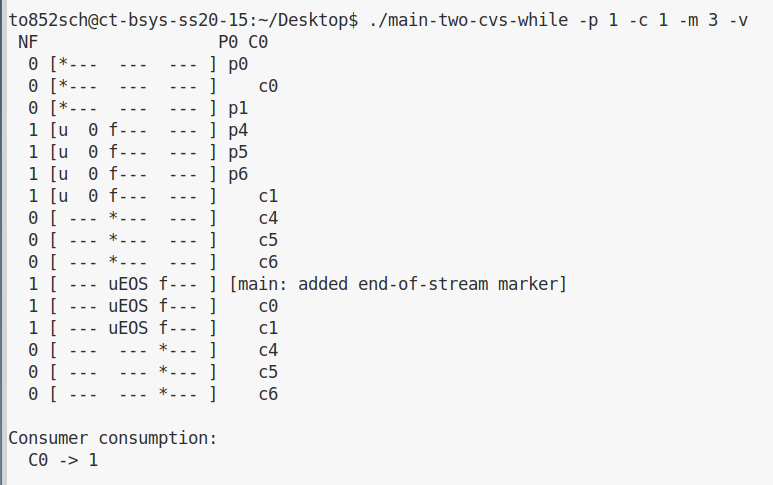
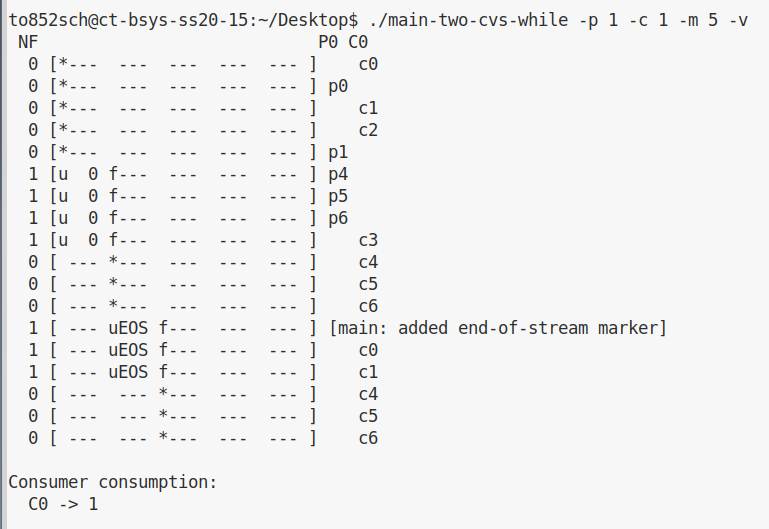
**Gruppe 15 | Tobias Schoch, Luis Nothvogel**

**Simulation wurde auf dem HTWG Container ausgeführt**

**30.2**

****

****

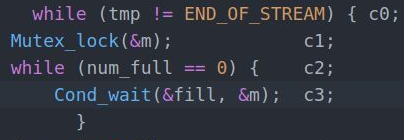
****

**30.4**

**Vermutung:** Da der Producer insgesamt mit „-l 10“ 10 Produkte herstellt, ist die Laufzeit des Programmes mindestens 10 Sekunden lang, da die Produkte eine Sekunde lang in der Instruktion C3 vom Consumer „verarbeitet“ werden.

Dies wird definiert mit -C 0,0,0,1,0,0: … Dabei ist die erste Stelle C0, die zweite C1 usw.

Wenn man also nun bei cond\_wait() angekommen ist, dann schläft der Consumer für 1 Sekunde.

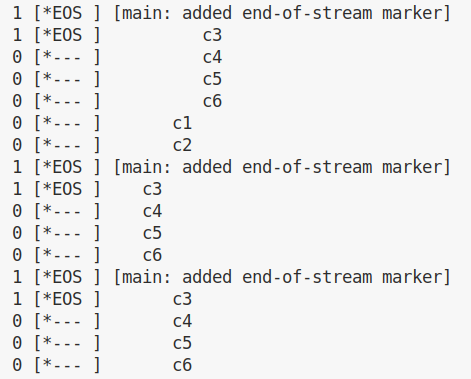


**Test:** Wenn wir das Programm ausführen mit dem Tag -t erhalten wir die Laufzeit von 13.01 Sekunden:

Warum also erhalten wir bei 10 Ausführungen eine Zeit von mehr als 13 Sekunden?

Dies kommt dadurch, dass zum Schluss noch die 3 Customer verarbeitet werden mit EOS (End-of-String), nachdem die einzelnen Customer fertig sind.

Wenn wir von der Grundzeit der Wiederholungen ausgehen ohne die abschließenden c3’s (blau umrandet), haben wir eine Zeit von 10.01 Sekunden.

****

**30.5**

**Vermutung:** Das Programm sollte mit dem größeren Buffer keinen spürbaren zeitlichen Unterschied machen, da die Customer immer noch jeweils 1 Sekunde warten müssen und lediglich der Producer durch den größeren Buffer mehr Produkte generieren kann bevor er sich schlafen legt.

****Die Zeit sollte zudem über 10 Sekunden bleiben, da wir immer noch 10 Ausführungen haben.

**Test:**

Allerdings haben wir nun eine geringere Laufzeit von 12.01 statt 13.01 Sekunden

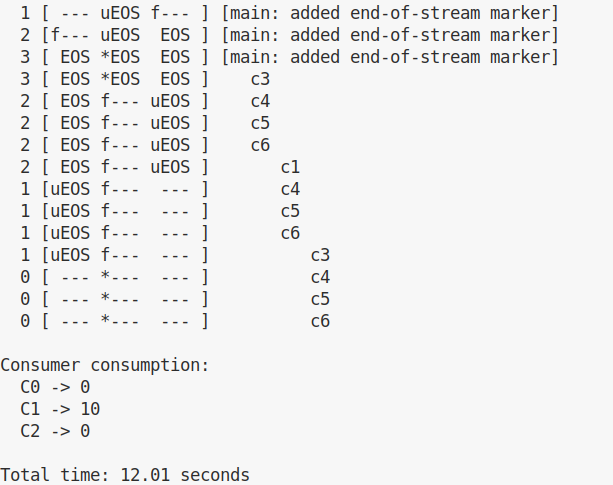
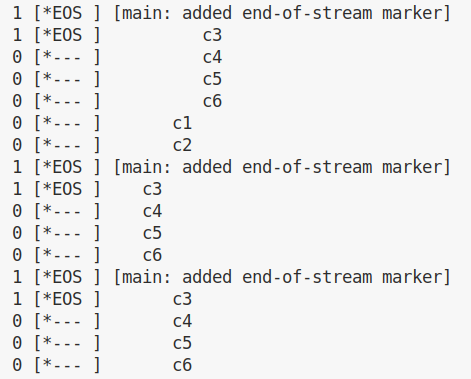
Wenn wir hier ebenfalls die abschließenden 2 c3’s ausblenden, haben wir ebenfalls eine Zeit von 10.01 Sekunden logischerweise.

Abbildung zeigt die Ausführung mit einer Buffer Größe von 3.

Warum nur 2 EOS?



Diese kommt dadurch, dass wir nur zwei c3 Instruktionen (orange umrandet) haben beim EOS statt 3 wie zuvor (blau umrandet)

Abbildung zeigt Programmausführung mit einer Buffer Größe von 1.

**30.6**

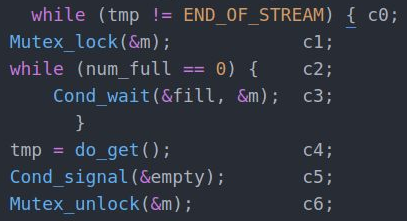
**Vermutung:** Bisher hatten wir die Reihenfolge, dass ein Lock gesetzt wurde und im Lock hat der Customer daraufhin die Produkte des Producers bearbeitet 1 Sekunde lang (sleep), woraufhin erst im Anschluss der Lock wieder befreit wurde.

Nun ist es so, dass der Consumer erst den Lock setzt, ihn im Anschluss freigibt und danach die Produkte bearbeitet 1 Sekunde lang bei c6 (rot umrandet).

Dadurch ist der Lock offen und ein anderer Consumer kann nun den Lock benutzen, während der andere schläft.

Da wir 3 Consumer haben, vermute ich, dass die 10 Wiederholungen auch 3x (~3,34 Sekunden | 3s-4s) so schnell sind, da die Consumer nicht mehr 1s auf die Freigabe des Locks warten müssen, sondern direkt nacheinander ohne große Wartezeiten den Lock reservieren können.

Nachdem die Consumer aufgeweckt werden, geht das Spiel von vorne los.

**Test:**

Wir erhalten eine Laufzeit von 5 Sekunden. In der 5-sekündigen Laufzeit ist auch die verwendete Zeit mit 1s für EOS mitinbegriffen. Daraus ergibt sich eine Laufzeit ohne EOS von 4 Sekunden.

Kann das sein?

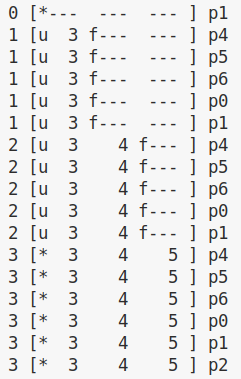
**30.7**

**Vermutung:** Da weiterhin lediglich die Consumer die Produkte abarbeiten mit einer Geschwindigkeit von 1s und die Producer keine „Produktionsverzögerung“ haben, sollte sich durch den größere Buffer keine Zeitänderung ergeben.

Es könnte eine kleine Zeitersparnis geben, dadurch dass weniger Context Switches stattfinden, denn durch den größeren Buffer kann der Producer nun mehr Produkte generieren bevor er sich schlafen legt.

**Test:**

Wir erhalten dieselbe Laufzeit von 5 Sekunden, da wie bereits erwähnt der Producer zwar den Buffer vollmacht, aber dennoch stets 1s gewartet werden muss bis die Produkte abgearbeitet wurden durch den Consumer.

****

Hier sehen wir, wie der größere Buffer gefüllt wird durch den Producer, aber dennoch dieselbe Laufzeit erzielt wird durch die Wartezeiten.

**30.8**

**Nein**, das Problem mit sleep strings zu verursachen wäre mit diesem Beispiel nicht möglich, denn dafür bräuchte man entweder mehr als 1 Producer bzw. mehr als 1 Consumer und wir haben lediglich ein Consumer und ein Producer.

Das Problem wird verursacht, da wir nur eine Condition Variable haben. Nehmen wir an, wir haben 2 Consumer und einen Producer. Wenn nun die beiden Consumer sich schlafen legen, weil der Buffer leer ist, legt der Producer einen Wert in den Buffer und aktiviert einen anderen Thread (Consumer 1). Consumer 1 wiederrum verarbeitet den Wert, leert den Buffer und aktiviert den Consumer 2. Im Anschluss legt Consumer 1 sich schlafen. Danach prüft Consumer 2 den Buffer, welcher leer ist und legt sich auch schlafen.

Dies ist jedoch in der Aufgabenstellung mit einem Producer und einem Consumer nicht möglich.

Consumer und Producer = Threads?

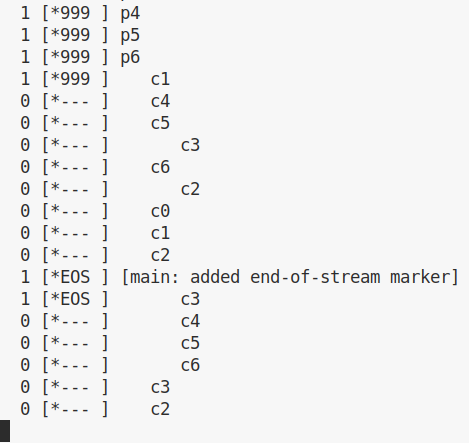
**30.9**

**Ja**, dies ist mit der Aufgabenstellung möglich da wir nun mind. 2 Threads derselben Klasse haben.

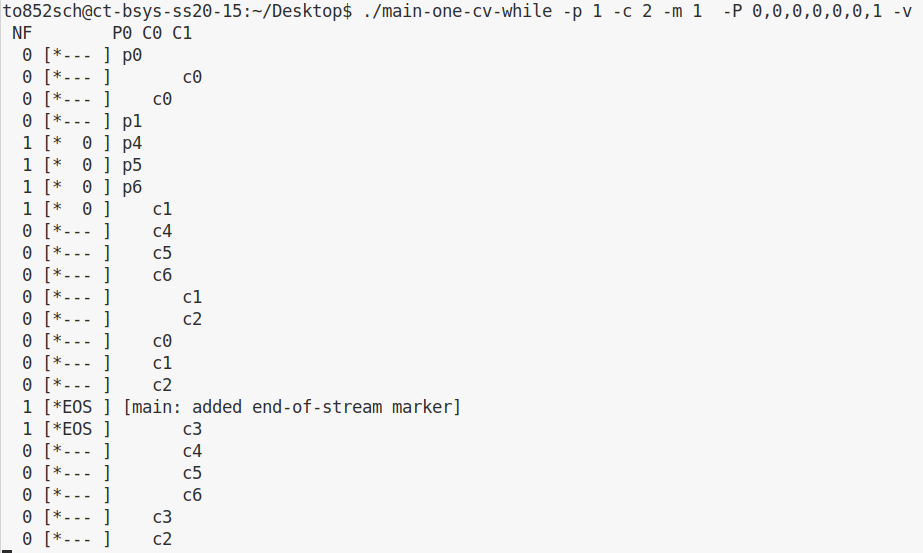


Zuerst haben wir versucht das Programm mit einem Producer, zwei Customer und einem Buffer der Größe eins laufen zu lassen. Das hat nicht funktioniert.

Als nächstes haben wir versucht die Anzahl der Wiederholungen zu erhöhen mit -l auf 1000. 

****Dies hat bei der zweiten Ausführung funktioniert. Dadurch erhalten wir allerdings keinen garantierten Fehler, denn der Fehler passiert nur durch Zufall und passiert nur alle paar Ausführungen (nicht deterministisch).

Consumer 1 weckt Consumer 2 auf (orange markiert).

****

Im Anschluss haben wir versucht, den Producer nach der Lock Befreiung eine Sekunde zu beschäftigen (p6), so dass die Consumer sich nur gegenseitig aufwecken können(c5), was auch passiert (rot markiert).

Im Anschluss haben sich dann die einzelnen Threads schlafen gelegt, da der Buffer nicht mehr aufgefüllt wurde durch den Producer.

**30.10**

**30.11**