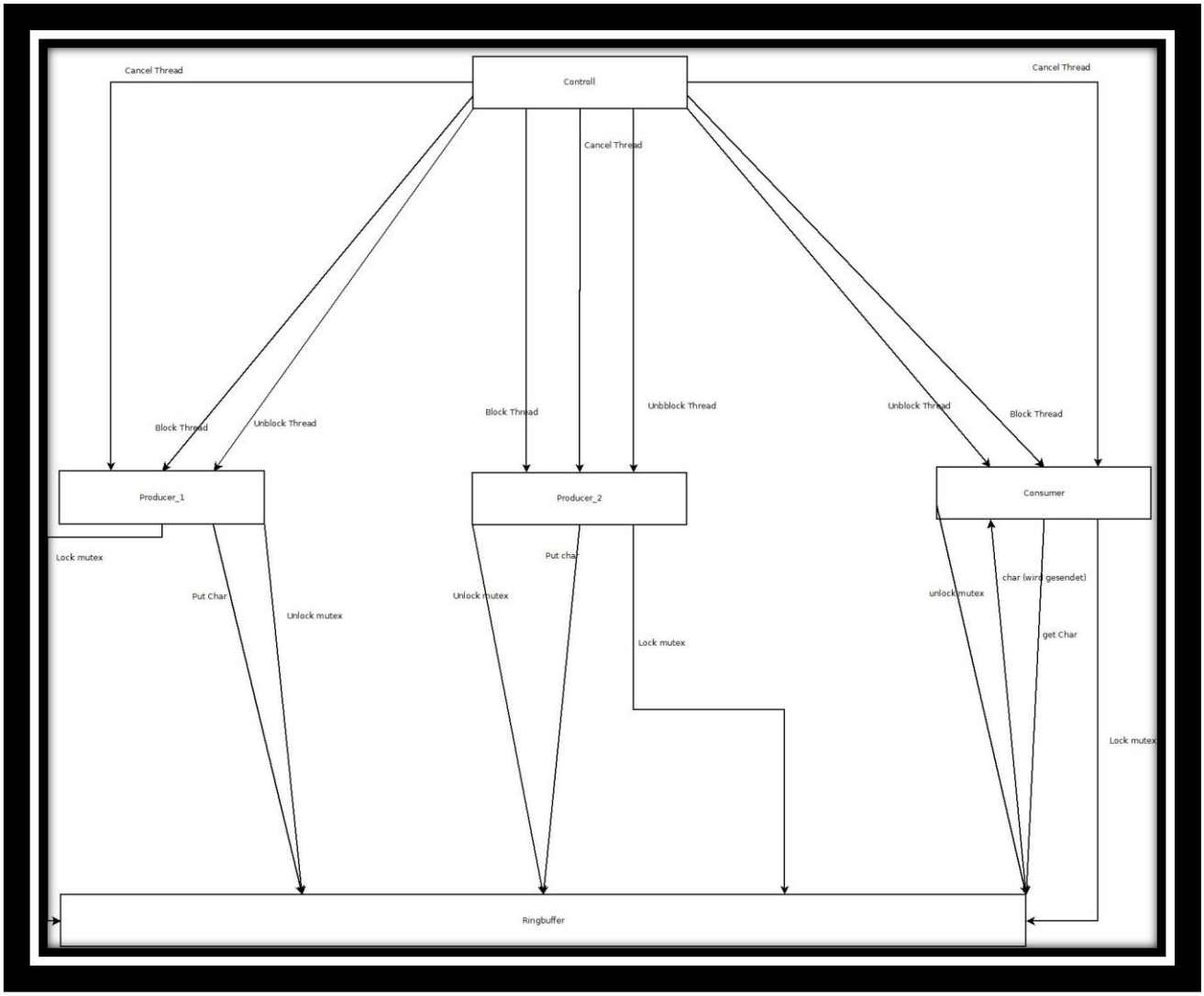
Kommunikationsdiagramm



Durch die Beschreibung des Diagramms möchten wir nochmal das FIFO-Prinzip verdeutlichen, da es aus dem Diagramm nicht deutlich wird. Wenn man vom ersten Producer ausgeht, so kann man es wie folgt beschreiben: Der Mutex wird gelockt, ein Zeichen wird in den Ringbuffer beschrieben, dananach wird der Mutex wieder freigeschaltet für den nächsten Producer. Das gleiche Prinzip verwendet Producer_2. Anschließend kann der Consumer den Mutext locken und ein Zeichen aus dem Ringbuffer auslesen. Nach diesem Verfahren, ist der Ringbuffer wieder frei für nächste Zeichen, die reingeschrieben werden. Mit Controll können wir bestimmen, zur welcher Zeit ein Thread blockiert werden soll und wann nicht. Des Weiteren ist man in der Lage alle Threads zu canceln und das Programm zu beenden.

Tabelle

	Rb_mutex Ctrl_mutex	
Condition_Var and Entry_Condition	CVs	ECs
	not_empty not_full read_condvar proc1_signal proc2_signal	count != 0 count <= MAX consumer_stop == 1 p1_stop == 1 p2_stop == 1
Threads	producer_1 producer_2 consumer	
Test auf die Funktionalität	-Wenn der producer_1 schreibt muss producer_2 in blocked gehen. -Das Gleiche Verhalten passiert mit dem producer_2. -Wenn der Ringbuffer voll ist, sollen beide producer1 und 2 blockieren. -Wenn beide producer blockieren, kann der consumer schreiben. -Wenn der Ringbuffer voll ist muss consumer ein Zeichen rauspicken und entweder producer1 o 2 aus dem blocked-Zustand wieder rausnehmen. -Wenn der Ringbuffer leer ist ist der consumer auf blocked zu schalten. -Wenn der Ringbuffer nicht leer ist, muss entweder producer 1 oder 2 den consumer aus dem blocked Zustand rausholen, damit der comsumer ein Zeichen aus dem Ringbuffer rausnehmen kann.	

Pointer-Semantik

Pointer des Buffers sind p_in und p_out.
p_in zeigt auf das nächste Feld im Array, welches beschrieben werden kann. p_out zeigt auf das nächste Arrayfeld welches ausgelesen werden soll.
Auf dem Ringbuffer können wir zwei Operationen durchführen, Einlesen und Schreiben. Beim Einlesen wird wird p_in auf die nächste Adresse des Arrays(Speicherzelle) gesetzt. Beim Schreiben wird p_out auf die nachfolgende Adresse der Speicherzelle verweisen. Wichtig für den Ablauf des Programms ist, dass der p_in den p_out nicht überholt, damit die Elemente nicht gelesen wurden nicht überschrieben werden(somit falsche Ausgabe). Da wir einen Ringbuffer simulieren, ist es wichtig, dass die Pointer p_in und p_out nicht über die Grenze der Speicherzellen hinauswachsen, um dies zu vermeiden setzen wir bei einer Überschreitung, den jeweiligen Pointer wieder auf die Startadresse des Ringbuffers.

Beobachtungen

Man kann feststellen, dass bei der Ausführung des Programms, ganz normal das Alphabet aufgelistet und in den Ringbuffer geschrieben wird. Wenn man den Consumer stoppt, so erfolgt keine Ausgabe der Buchstaben, da dieser dafür zuständig ist. Wenn man einen Producer rausnimmt als Beispiel Producer_1, so werden nur die Großbuchstaben in den Ringbuffer geschrieben und von dem Consumer ausgegeben. Sobald man beide Producer stoppt, der Ringbuffer aber nicht leer ist, so kann man sehen, wie der Consumer die einzelnen Buchstaben auf dem Bildschirm ausgibt, bis der Ringbuffer leer ist. Sobald man am Ende des Alphabets angekommen ist(Z), startet der Input wieder am Anfang also beim A, da wir in einem Zyklus arbeiten. Das Programm arbeitet solange bis man alle Threads beendet.