

Onderzoeksverslag

– Producer/Consumer –



Student:	A.W. Janisse
Studentnummer:	2213829
Instelling:	Fontys hogescholen te Eindhoven
Opleiding:	ICT & Technology
Docent:	Dhr. Erik Dortmans
Opdracht:	5, Producer/Consumer
Datum:	22 maart 2015

Inleiding

In deze opdracht wordt onderzocht hoe een data producerend proces (of thread) en een data consumerend proces (of thread) via een eindig buffer kunnen communiceren.

Taken

In deze opdracht wordt de uitwerking van de volgende taken beschreven:

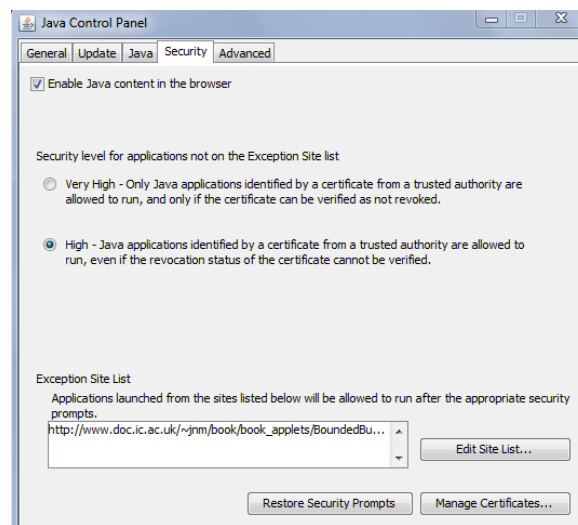
1. Beschrijven van bevindingen aan de hand van een bounded buffer applicatie (applet).
2. Implementeren van een Producer/Consumer applicatie.

Uitwerking taak 1

Het starten van de de applet http://www.doc.ic.ac.uk/~jnm/book/book_applets/BoundedBuffer.html gaat niet zonder problemen. Dit heeft te maken met de volgende oorzaken:

- In de Linux omgeving is er in Firefox geen Java plugin geïnstalleerd;
- Tegenwoordig kent Java alleen nog maar de security levels Ver High en High.

Ik heb er voor gekozen om in mijn Windows omgeving de Java omgeving aan te passen. Afbeelding 1 laat zien dat ik voor de bewuste site een exceptie heb toegevoegd. Nu draait de applet wel.



Afbeelding 1

Opdrachten:

- Start beide processen
- Laat een van beide processen pauzeren totdat de ander ook niet meer verder kan
- En daarna het andere proces

Beschrijf de bevindingen.

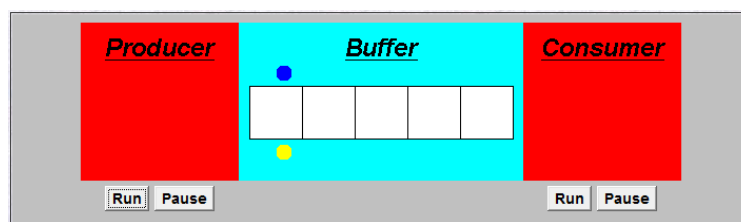
Algemeen:

De applet is een simulatie van een klassiek Producer/Consumer probleem. In het kort laat dit probleem zich als volgt beschrijven: Twee processen maken beide gebruik van een eindig stuk geheugen ruimte. Hierbij zal een proces, de Producer, data in het geheugen schrijven terwijl het andere proces, de Consumer, data uit het geheugen wil lezen. In deze voorstelling ligt het voor de hand dat de Consumer alleen maar data kan lezen als deze geschreven is door de Producer.

Het gedeelde geheugen is een circulaire buffer. Dit betekent dat als er ruimte is voor bijvoorbeeld 10 pakketjes data, er weer op geheugen locatie 0 wordt geschreven nadat locatie 9 beschreven is. Als de data niet geconsumeerd wordt zal in zekere zin de Producer zichzelf inhalen (*buffer overrun*) en ongelezen data weer gaan overschrijven.

Aan de andere kant is het zonder de juiste maatregelen voor een Consumer om zichzelf in te halen (*buffer underrun*). In deze situatie zal de Consumer reeds gelezen data opnieuw gaan lezen.

Zoals gezegd is de applet een simulatie van dit probleem. Afbeelding 2 laat zien hoe deze applet er uitziet als deze is opgestart.



Afbeelding 2

Aan de linker zijde zien we de Producer en aan de rechterzij de Consumer. Beide elementen kunnen worden gestart en gepauzeerd. Als ze in pauze staan zijn ze rood en als ze in run staan zijn ze groen. In het middelste gedeelte bevindt zich het buffer. Dit buffer is een visuele representatie van een circulair buffer met 5 geheugen plaatsen.

De blauwe stip geeft de locatie aan waar de Producer zijn data, letters a t/m z, wil gaan schrijven. De gele stip markeert de positie waar de Consumer wil gaan lezen. Gelezen data wordt uit het buffer verwijderd.

Bevindingen:

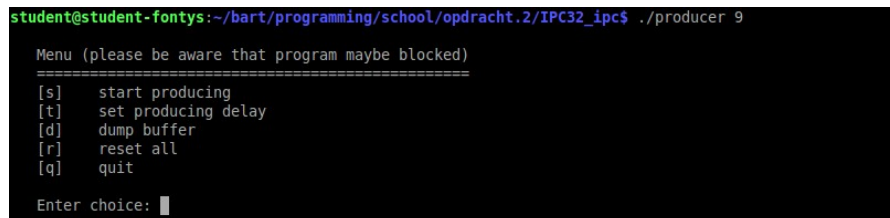
- Zoals op de website vermeld staat loopt de Consumer iets sneller dan de Producer. Ik heb dus als eerste de Producer gestart en toen deze eenmaal vier letters geschreven had, de Consumer gestart. De verwachting was dus dat er een moment moest zijn dat de Consumer op nieuwe data moest wachten. Dit bleek na enige minuten draaien ook zo te zijn. De Consumer had de Producer 'ingehaald' en moest iedere keer een kort moment wachten totdat de Producer nieuwe data had geschreven. Dit beeld geeft in ieder geval aan dat de applet op een juiste wijze is geïmplementeerd.
- Vervolgens heb ik de Consumer gepauzeerd. Nu kan de producer het buffer weer vullen tot dat deze helemaal vol is. Eenmaal vol stopt ook de Producer om niet ongelezen data te overschrijven.

- Vervolgens heb ik de Producer op pauze gezet en De Consumer weer op run. De verwachting hiervan is dat de Consumer het gehele buffer zal uitlezen. Als het buffer leeg is zou de Consumer moeten wachten op nieuwe data. Dit blijkt precies zo te werken.

Uitwerking taak 2

Voor de uitwerking van taak 2 heb ik twee applicaties¹ gemaakt. Een is de Producer en de andere is de Consumer. De producer kan gestart worden met een opstart parameter. Deze parameter geeft aan hoe groot het buffer moet zijn. De waarde hiervan mag variëren van 1 t/m 10000.

Als de Producer eenmaal is gestart dan wordt er een menu getoond. Afbeelding 3 laat dit zien.



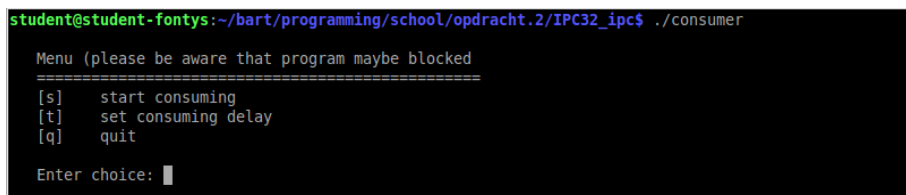
```
student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc$ ./producer 9
Menu (please be aware that program maybe blocked)
=====
[s]   start producing
[t]   set producing delay
[d]   dump buffer
[r]   reset all
[q]   quit
Enter choice: █
```

Afbeelding 3

De menu opties hierbij hebben de volgende betekenis:

- [s] Start producing. De producer gaat het buffer beschrijven met getallen.
- [t] Set producing delay. Hiermee wordt de snelheid van produceren bepaald.
- [d] Dump buffer. Toont de actuele inhoud van het buffer op het scherm.
- [r] Reset all. Reset het buffer door deze volledig met de waarde 0 te beschrijven. Tevens worden interne variabelen op 0 gezet zodat bij een hernieuwde start produceren weer alles vanaf 0 begint.
- [q] Quit. Sluit het programma af en 'ruimt' de semaforen en shared memory op.

Afbeelding 4 laat het menu van de Consumer zien.



```
student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc$ ./consumer
Menu (please be aware that program maybe blocked)
=====
[s]   start consuming
[t]   set consuming delay
[q]   quit
Enter choice: █
```

Afbeelding 4

De menu opties van de Consumer hebben de volgende betekenis:

- [s] Start consuming. De consumer begint uit het buffer te lezen.
- [t] Set consuming delay. Hiermee wordt de snelheid van consumeren bepaald.
- [q] Quit. Sluit het programma af.

¹ Bijlage: prodcon.tar.gz

Als eerste wordt de Producer gestart. Deze zal beginnen met het beschrijven van het buffer totdat deze geheel is gevuld. Zodra dit het geval is zal er gewacht worden totdat de Consumer het buffer begint te lezen. Afbeelding 5 laat zien dat het buffer geheel gevuld wordt met de getallen 1 t/m 9.

```
Menu (please be aware that program maybe blocked)
=====
[s]  start producing
[t]  set producing delay
[d]  dump buffer
[r]  reset all
[q]  quit

Enter choice: s

Press a p to pause producing
1 0 0 0 0 0 0 0 0
1 2 0 0 0 0 0 0 0
1 2 3 0 0 0 0 0 0
1 2 3 4 0 0 0 0 0
1 2 3 4 5 0 0 0 0
1 2 3 4 5 6 0 0 0
1 2 3 4 5 6 7 0 0
1 2 3 4 5 6 7 8 0
1 2 3 4 5 6 7 8 9
```

Afbeelding 5

Nu wordt de Consumer gestart waarbij een tijd is ingesteld van 1000ms. In afbeelding 6 is duidelijk te zien dat de gebruikte index van 0 t/m 8 loopt en dan weer bij 0 begint. Dit komt door het feit dat er een buffergrootte van 9 is gekozen bij het starten van de Producer. Verder is te zien dat de data netjes bij 1 begint te tellen en gewoon doorloopt.

```
Press a p to pause consuming
Index = 0 data = 1
Index = 1 data = 2
Index = 2 data = 3
Index = 3 data = 4
Index = 4 data = 5
Index = 5 data = 6
Index = 6 data = 7
Index = 7 data = 8
Index = 8 data = 9
Index = 0 data = 10
Index = 1 data = 11
Index = 2 data = 12
Index = 3 data = 13
Index = 4 data = 14
Index = 5 data = 15
Index = 6 data = 16
Index = 7 data = 17
Index = 8 data = 18
Index = 0 data = 19
```

Afbeelding 6

Leuk om te zien is dat de Producer afgeremd wordt door de Consumer. We hebben voor de Consumer immers een delay van 1000 ms gekozen. Hierdoor is duidelijk dat de Producer zichzelf niet inhaalt waarmee een *buffer overrun* wordt voorkomen.

Ook het pauzeren van de Producer geeft een leuk beeld. Door meerde dumps bij de Producer te maken kunnen we zien dat door de Consumer gelezen plaatsen met een 0 worden geschreven. Eenmaal alles 0 stopt ook de Consumer met lezen om zodoende een *buffer underrun* te voorkomen.

Bij deze taak is er ook de vraag gesteld of semafoor s nodig is. Om deze vraag te beantwoorden kij is eerst naar zijn functie. De functie van deze semafoor is het voorkomen dat zowel Producer als Consumer gelijktijdig is het buffer (*shared memory*) gaan lezen en schrijven. Dit zou dus een *race condition* opleveren. Het antwoord zal dus duidelijk zijn, semafoor s is noodzakelijk.

Vervolgens zijn er de volgende extra's gevraagd:

1. Uitbreiding van de programma's zodat meerdere producer processen en consumer processen kunnen worden opgestart. (in en uit worden dan wel shared variabelen).
2. Het genereren van random extra belasting bij producer en consumer (bijvoorbeeld met busy-waitloops). Zorg er dan voor dat:
 - soms heeft producer het erg druk
 - soms heeft consumer het erg druk
 - soms loopt producer een beetje sneller dan consumer
 - soms loopt consumer een beetje sneller dan producer.

beschrijf hoe je dit gerealiseerd is, en hoe je het getest is.

Uitwerking

De uitwerking van de extra's is betrekkelijk eenvoudig en bestaat in de basis uit de volgende zaken:

1. Aanpassing van menu in producer zodat hier o.a gekozen kan worden om shared memory en semaforen aan te maken of te openen.
2. Het onderbrengen van de *in* variabele in het shared memory voor de Producers
3. Het onderbrengen van de actuele data (getal om te schrijven) in shared memory bij de producers.
4. Aanpassen van de `append()` functie bij de Producers zodat deze met het shared memory overweg kan.
5. Het verwijderen van het menu bij de Consumer i.v.m. redirect
6. Het onderbrengen van de *out* variabele in het shared memory voor de Consumers
7. Bij Producer en Consumer zorgen dat de al aanwezige delay een random karakter kan krijgen.

Om variatie in het produceren en het consumeren te krijgen is er gebruik gemaakt van de functie *usleep()* in combinatie met de functie *random()*. Bij *usleep()* kan een 'slaaptijd' in μ s opgegeven worden. De functie *random()* zorgde hierbij voor random tijdsintervallen tussen 0 en 1 seconden.

Om het geheel te testen is de output van de Consumers geredirect naar een console. Hierdoor was eenvoudig te observeren of de getallen netjes opliepen.

De source code voor beide programma's² is meegeleverd.

² baprodcon.tar.gz

Onderstaand zullen diverse scenario's worden uitgevoerd. Bij afbeelding 7 is de uitgangssituatie te zien waarbij er aan de linker zijde twee producers zijn, in het midden drie Consumers en rechts de output verkregen door het `tty` commando.

```

student@student-fontys: ~/bart/...ing/school/opdracht.2/IPC32_ipc
student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc$ ./baproducer 9
bash: ./BAPRODUCER: No such file or directory
student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc$ ./baproducer
Usage: ./baproducer <bufferize 1..10000>
student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc$ ./baproducer 9

Menu (please be aware that program maybe blocked)
=====
[c] create semaphores and SHM's
[o] open semaphores end SHM's
[s] start producing
[q] quit
[Q] quit and cleanup

Enter choice: [c]

student@student-fontys:~/bart/...ing/school/opdracht.2/IPC32_ipc
student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc$ ./baproducer 9

Menu (please be aware that program maybe blocked)
=====
[c] create semaphores and SHM's
[o] open semaphores end SHM's
[s] start producing
[q] quit
[Q] quit and cleanup

Enter choice: [c]

student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc
student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc$ ./baconsumer > /dev/pts/17

student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc
student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc$ ./baconsumer > /dev/pts/17

student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc
student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc$ ./baconsumer > /dev/pts/17

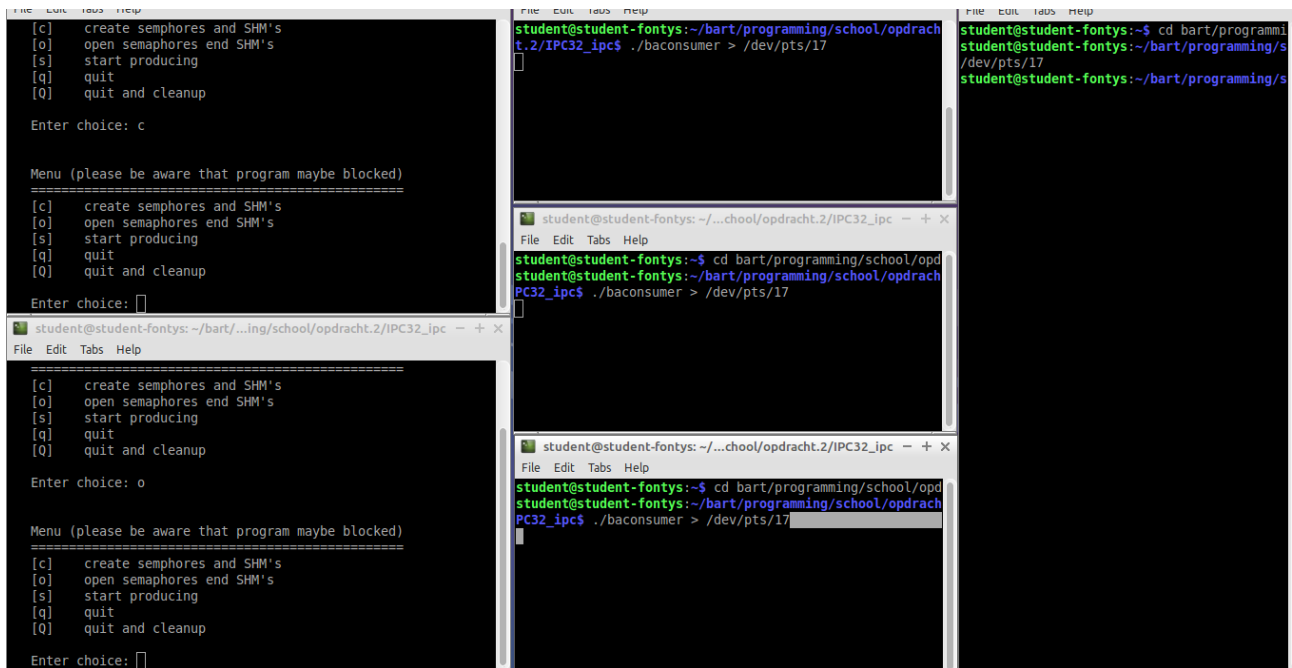
```

Afbeelding 7

Scenario 1

- Producer 1 maakt semaforen en SHM aan.
- Producer 2 opent semaforen en SHM
- Consumer 1, 2 en 3 opstarten.

Afbeelding 8 toont het resultaat.



```
[c] create semaphores and SHM's
[o] open semaphores end SHM's
[s] start producing
[q] quit
[Q] quit and cleanup

Enter choice: c

Menu (please be aware that program maybe blocked)
=====
[c] create semaphores and SHM's
[o] open semaphores end SHM's
[s] start producing
[q] quit
[Q] quit and cleanup

Enter choice: [ ]

student@student-fontys: ~/bart/...ing/school/opdracht.2/IPC32_ipc -- + X
File Edit Tabs Help

=====
[c] create semaphores and SHM's
[o] open semaphores end SHM's
[s] start producing
[q] quit
[Q] quit and cleanup

Enter choice: o

Menu (please be aware that program maybe blocked)
=====
[c] create semaphores and SHM's
[o] open semaphores end SHM's
[s] start producing
[q] quit
[Q] quit and cleanup

Enter choice: [ ]

student@student-fontys: ~/bart/programming/school/opdracht.2/IPC32_ipc -- + X
File Edit Tabs Help

student@student-fontys: ~$ cd bart/programming/school/opdracht.2/IPC32_ipc
student@student-fontys: ~$ ./baconsumer > /dev/pts/17

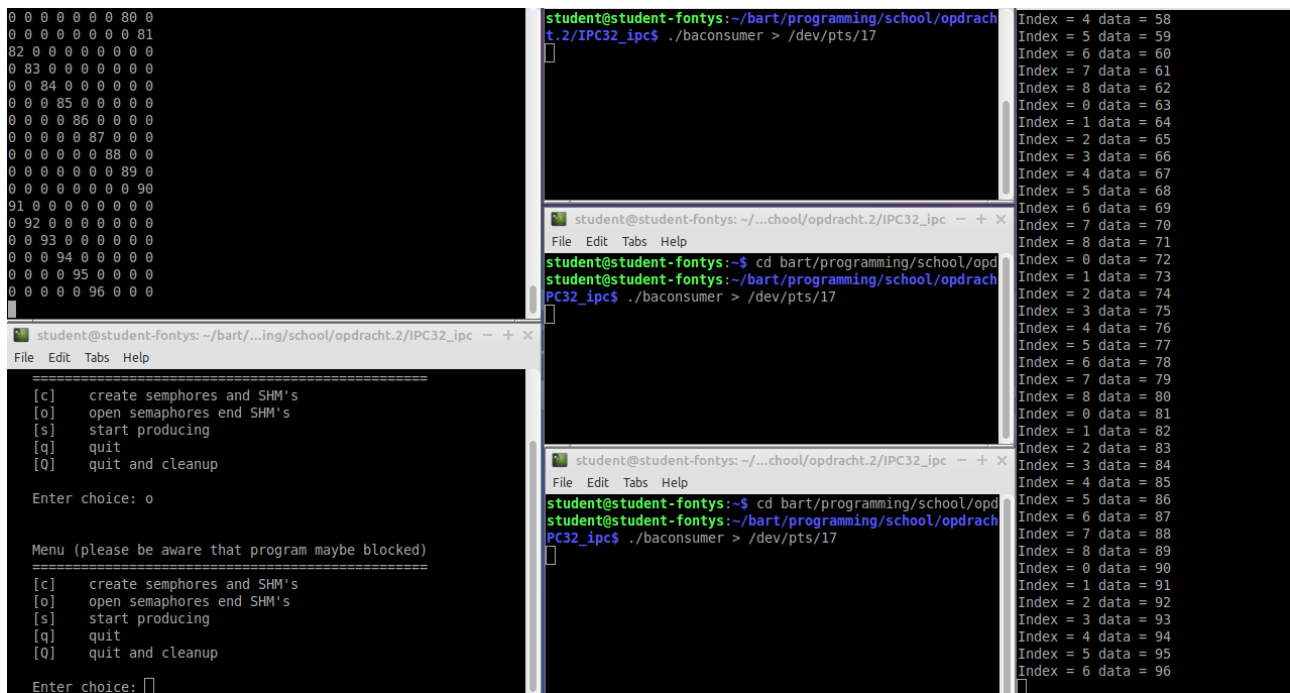
student@student-fontys: ~$ cd bart/programming/school/opdracht.2/IPC32_ipc
student@student-fontys: ~$ ./baconsumer > /dev/pts/17
```

Afbeelding 8

Scenario 2

- Producer 1 start met produceren.
- Consumers 1, 2 en 3 starten automatisch met consumeren.

Afbeelding 9 toont hiervan het resultaat waarbij duidelijk te zien is aan de output dat de getallen netjes oplopen en de index van 0 t/m 8 loopt.



```
0 0 0 0 0 0 0 80 0
0 0 0 0 0 0 0 81
82 0 0 0 0 0 0 0
0 83 0 0 0 0 0 0
0 0 84 0 0 0 0 0
0 0 0 85 0 0 0 0
0 0 0 0 86 0 0 0
0 0 0 0 0 87 0 0
0 0 0 0 0 0 88 0
0 0 0 0 0 0 0 89
0 0 0 0 0 0 0 90
91 0 0 0 0 0 0 0
0 92 0 0 0 0 0 0
0 0 93 0 0 0 0 0
0 0 0 94 0 0 0 0
0 0 0 0 95 0 0 0
0 0 0 0 0 96 0 0

student@student-fontys: ~/bart/programming/school/opdracht.2/IPC32_ipc
t.2/IPC32_ipc$ ./baconsumer > /dev/pts/17

student@student-fontys: ~/...chool/opdracht.2/IPC32_ipc
File Edit Tabs Help

student@student-fontys:~$ cd bart/programming/school/opdracht.2/IPC32_ipc
student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc$ ./baconsumer > /dev/pts/17

student@student-fontys:~/...chool/opdracht.2/IPC32_ipc
File Edit Tabs Help

student@student-fontys:~$ cd bart/programming/school/opdracht.2/IPC32_ipc
student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc$ ./baconsumer > /dev/pts/17

Index = 4 data = 58
Index = 5 data = 59
Index = 6 data = 60
Index = 7 data = 61
Index = 8 data = 62
Index = 0 data = 63
Index = 1 data = 64
Index = 2 data = 65
Index = 3 data = 66
Index = 4 data = 67
Index = 5 data = 68
Index = 6 data = 69
Index = 7 data = 70
Index = 8 data = 71
Index = 0 data = 72
Index = 1 data = 73
Index = 2 data = 74
Index = 3 data = 75
Index = 4 data = 76
Index = 5 data = 77
Index = 6 data = 78
Index = 7 data = 79
Index = 8 data = 80
Index = 0 data = 81
Index = 1 data = 82
Index = 2 data = 83
Index = 3 data = 84
Index = 4 data = 85
Index = 5 data = 86
Index = 6 data = 87
Index = 7 data = 88
Index = 8 data = 89
Index = 0 data = 90
Index = 1 data = 91
Index = 2 data = 92
Index = 3 data = 93
Index = 4 data = 94
Index = 5 data = 95
Index = 6 data = 96

[c] create semaphores and SHM's
[o] open semaphores end SHM's
[s] start producing
[q] quit
[Q] quit and cleanup

Enter choice: o

Menu (please be aware that program maybe blocked)

[c] create semaphores and SHM's
[o] open semaphores end SHM's
[s] start producing
[q] quit
[Q] quit and cleanup

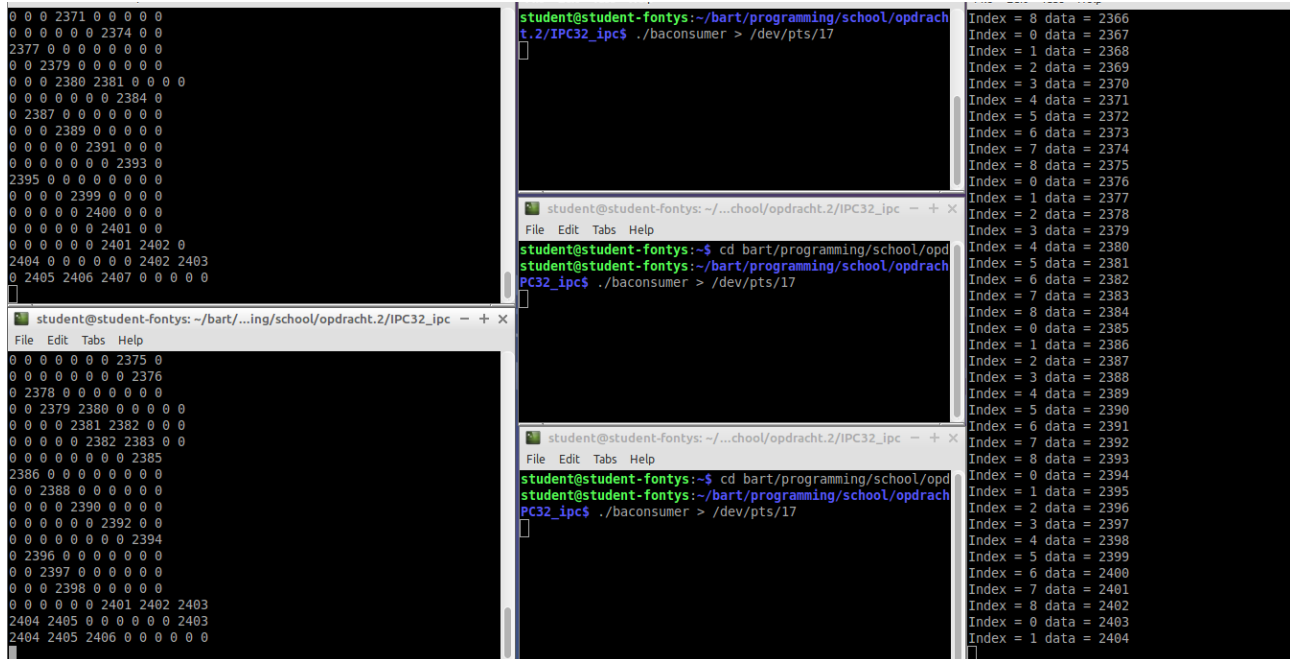
Enter choice: 
```

Afbeelding 9

Scenario 3

- Producer 2 start ook met produceren

Afbeelding 10 laat hierbij zien dat de output netjes volgens verwachting doorloopt.



```
0 0 0 2371 0 0 0 0 0
0 0 0 0 0 0 2374 0 0
2377 0 0 0 0 0 0 0 0
0 0 2379 0 0 0 0 0 0
0 0 0 2380 2381 0 0 0 0
0 0 0 0 0 0 0 2384 0
0 2387 0 0 0 0 0 0 0 0
0 0 0 2389 0 0 0 0 0 0
0 0 0 0 0 2391 0 0 0
0 0 0 0 0 0 0 2393 0
2395 0 0 0 0 0 0 0 0 0
0 0 0 0 2399 0 0 0 0
0 0 0 0 0 2400 0 0 0
0 0 0 0 0 0 2401 0 0
0 0 0 0 0 0 2401 2402 0
2404 0 0 0 0 0 0 2402 2403
0 2405 2406 2407 0 0 0 0 0

student@student-fontys: ~/bart/programming/school/opdracht.2/IPC32_ipc
t.2/IPC32_ipc$ ./baconconsumer > /dev/pts/17

Index = 8 data = 2366
Index = 0 data = 2367
Index = 1 data = 2368
Index = 2 data = 2369
Index = 3 data = 2370
Index = 4 data = 2371
Index = 5 data = 2372
Index = 6 data = 2373
Index = 7 data = 2374
Index = 8 data = 2375
Index = 0 data = 2376
Index = 1 data = 2377
Index = 2 data = 2378
Index = 3 data = 2379
Index = 4 data = 2380
Index = 5 data = 2381
Index = 6 data = 2382
Index = 7 data = 2383
Index = 8 data = 2384
Index = 0 data = 2385
Index = 1 data = 2386
Index = 2 data = 2387
Index = 3 data = 2388
Index = 4 data = 2389
Index = 5 data = 2390
Index = 6 data = 2391
Index = 7 data = 2392
Index = 8 data = 2393
Index = 0 data = 2394
Index = 1 data = 2395
Index = 2 data = 2396
Index = 3 data = 2397
Index = 4 data = 2398
Index = 5 data = 2399
Index = 6 data = 2400
Index = 7 data = 2401
Index = 8 data = 2402
Index = 0 data = 2403
Index = 1 data = 2404
```

Afbeelding 10

Scenario 4

- Stoppen van Consumer 1 en 2.

Afbeelding 11 toont het resultaat. Ook nu loopt weer alles volgens verwachting.

[illegible]

Afbeelding 11

Scenario 5

- Stoppen van de laatste Consumer.

Afbeelding 12 laat hiervan het resultaat zien. Het is een beetje lastig op een plaatje maar Producer 1 en 2 zijn beide ook gestopt omdat het buffer vol is.

```
student@student-fontys: ~/bart/programming/school/opdracht.2/IPC32_ipc - + X
File Edit Tabs Help
9910 9911 9903 9904 9905 9906 9907 9908 9909
9910 9911 9912 9913 9905 9906 9907 9908 9909
9910 9911 9912 9913 9914 9915 9916 9908 9909
9910 9911 9912 9913 9914 9915 9916 9917 9918
9919 9920 9921 9922 9914 9915 9916 9917 9918
9919 9920 9921 9922 9923 9924 9916 9917 9918
9919 9920 9921 9922 9923 9924 9925 9926 9918
9928 9920 9921 9922 9923 9924 9925 9926 9927
9928 9929 9930 9922 9923 9924 9925 9926 9927
9928 9929 9930 9931 9932 9924 9925 9926 9927
9928 9929 9930 9931 9932 9933 9934 9926 9927
9928 9929 9930 9931 9932 9933 9934 9935 9936
9937 9929 9930 9931 9932 9933 9934 9935 9936
9937 9938 9939 9931 9932 9933 9934 9935 9936
9937 9938 9939 9940 9941 9942 9934 9935 9936
9937 9938 9939 9940 9941 9942 9943 9944 9936
9946 9947 9939 9940 9941 9942 9943 9944 9945

student@student-fontys: ~/bart/programming/school/opdracht.2/IPC32_ipc - + X
File Edit Tabs Help
student@student-fontys:~$ cd bart/programming/school/opdracht.2/IPC32_ipc
student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc$ ./baconsumer > /dev/pts/17
^C
student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc$

student@student-fontys: ~/bart/programming/school/opdracht.2/IPC32_ipc - + X
File Edit Tabs Help
student@student-fontys:~$ cd bart/programming/school/opdracht.2/IPC32_ipc
student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc$ ./baconsumer > /dev/pts/17
^C
student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc$

student@student-fontys: ~/bart/programming/school/opdracht.2/IPC32_ipc - + X
File Edit Tabs Help
student@student-fontys:~$ cd bart/programming/school/opdracht.2/IPC32_ipc
student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc$ ./baconsumer > /dev/pts/17
^C
student@student-fontys:~/bart/programming/school/opdracht.2/IPC32_ipc$

Index = 0 data = 9900
Index = 1 data = 9901
Index = 2 data = 9902
Index = 3 data = 9903
Index = 4 data = 9904
Index = 5 data = 9905
Index = 6 data = 9906
Index = 7 data = 9907
Index = 8 data = 9908
Index = 0 data = 9909
Index = 1 data = 9910
Index = 2 data = 9911
Index = 3 data = 9912
Index = 4 data = 9913
Index = 5 data = 9914
Index = 6 data = 9915
Index = 7 data = 9916
Index = 8 data = 9917
Index = 0 data = 9918
Index = 1 data = 9919
Index = 2 data = 9920
Index = 3 data = 9921
Index = 4 data = 9922
Index = 5 data = 9923
Index = 6 data = 9924
Index = 7 data = 9925
Index = 8 data = 9926
Index = 0 data = 9927
Index = 1 data = 9928
Index = 2 data = 9929
Index = 3 data = 9930
Index = 4 data = 9931
Index = 5 data = 9932
Index = 6 data = 9933
Index = 7 data = 9934
Index = 8 data = 9935
Index = 0 data = 9936
Index = 1 data = 9937
Index = 2 data = 9938
```

Afbeelding 12

Zo zijn er nog veel scenario's te bedenken maar uitproberen is toch het beste!