A_7

Assignment VII

Connor Taffe. T no. 3742

March 10th, 2015

1 Report of Lab 4-3

1.1 Original Lab 4-3

Q. 4 I then completed all programs in file `prodcons++.cc` as follows:

Q. 4.1 In function `ProdCons`, I added the code to construct the three semaphores as follows:

```
mutex = new Semaphore("mutex", 1);
nempty = new Semaphore("nempty", BUFF_SIZE);
nfull = new Semaphore("nfull", 0);
```

I then added the ring instantiation code in the same function as follows:

```
ring = new Ring(BUFF_SIZE);
```

I then added the producer instantiation and forking code as follows:

```
Thread *t = new Thread(prod_names[i]);
t->Fork(Producer, i);
```

And similarly, the consumer forking code:

```
Thread *t = new Thread(cons_names[i]);
t->Fork(Consumer, i);
```

Q. 4.2 In function `Producer`, I added the following code before `ring->Put(message)`.

```
nempty->P();
mutex->P();
```

Similarly, the following syncronization code after `ring->Put(message)`.

```
mutex->V();
nfull->V();
currentThread->Yield();
```

Q. 4.3 In function `Consumer`, I added the following code before `ring->Get(message)`.

```
nfull->P();
mutex->P();
```

Similarly, the code after `ring->Get(message)`.

```
mutex->V();
nempty->V();
currentThread->Yield();
```

Q. 5 I then compiled a new nachos via command `make` and tested if my program was working or not as follows:

```
$ make
... (many lines omitted)
ln -sf arch/unknown-i386-linux/bin/nachos nachos
```

After running `make`, I ran nachos with `-rs 27` and `-rs 28`.

```
$ ./nachos -rs 27
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 890, idle 0, system 890, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
$ ls
arch     Makefile        Makefile.local~  prodcons++.cc  ring.h  tmp_1
main.cc  Makefile.local  nachos           ring.cc        tmp_0
$ cat tmp_0
producer id --> 0; Message number --> 0;
producer id --> 0; Message number --> 1;
producer id --> 1; Message number --> 1;
producer id --> 1; Message number --> 3;
producer id --> 0; Message number --> 3;
$ cat tmp_1
producer id --> 1; Message number --> 0;
producer id --> 0; Message number --> 2;
```

```
producer id --> 1; Message number --> 2;
$ ./nachos -rs 28
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 978, idle 8, system 970, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
$ cat tmp_0
producer id --> 0; Message number --> 0;
producer id --> 1; Message number --> 1;
producer id --> 1; Message number --> 2;
producer id --> 0; Message number --> 2;
producer id --> 0; Message number --> 3;
$ cat tmp_1
producer id --> 1; Message number --> 0;
producer id --> 0; Message number --> 1;
producer id --> 1; Message number --> 3;
```

As you can see from both runs at rs = 27 and rs = 28, there are 2 consumers (`N_CONS` is 2), and each has a non-repeating, sequential access for each producer. Moreover `N_MESSG` is 4, and 0 through 3 were given from producer, so all messages were recieved.

1.2 Configuration tests on Lab 4-3

I then proceeded to test my Lab 4-3 using the three test configurations given in Assignment 7.

Q. 1 Configuration one sets *buffer size* to 2, *number of producers* to 4, *number of messages per producer* to 3, and *number of consumers* to 2.

Q. 1.1 I implemented this by changing the following constants as stated here:

```
#define BUFF_SIZE   2
#define N_PROD      4
#define N_MESSG     3
#define N_CONS      2
```

I then ran make to recompile the reconfigured `nachos`.

```
$ make
... (many lines omitted)
ln -sf arch/unknown-i386-linux/bin/nachos nachos
```

I then completed parts **a** and **b** as follows:

(a) Here I ran the Nachos without randomization with command: `./nachos`.

```
$ ./nachos
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 1330, idle 0, system 1330, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
$ ls tmp_*
tmp_0  tmp_1
$ cat tmp_0
producer id --> 0; Message number --> 0;
producer id --> 0; Message number --> 1;
producer id --> 3; Message number --> 0;
producer id --> 3; Message number --> 1;
producer id --> 3; Message number --> 2;
$ cat tmp_1
producer id --> 1; Message number --> 0;
producer id --> 1; Message number --> 1;
producer id --> 0; Message number --> 2;
producer id --> 1; Message number --> 2;
producer id --> 2; Message number --> 0;
producer id --> 2; Message number --> 1;
producer id --> 2; Message number --> 2;
$ rm tmp_*
```

(b) Here I ran the Nachos with random seed 99: `./nachos -rs 99`.

```
$ ./nachos -rs 99
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 1494, idle 34, system 1460, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
```

```
Cleaning up...
$ ls tmp_*
tmp_0  tmp_1
$ cat tmp_0
producer id --> 0; Message number --> 0;
producer id --> 1; Message number --> 1;
producer id --> 0; Message number --> 2;
producer id --> 3; Message number --> 0;
producer id --> 3; Message number --> 1;
producer id --> 2; Message number --> 1;
producer id --> 2; Message number --> 2;
$ cat tmp_1
producer id --> 1; Message number --> 0;
producer id --> 0; Message number --> 1;
producer id --> 1; Message number --> 2;
producer id --> 2; Message number --> 0;
producer id --> 3; Message number --> 2;
$ rm tmp_*
```

Q. 1.2 I then examined the output files (`cat`'d above for clarity) of the consumers to see if the bounded buffer problem is solved and implemented correctly.

We can see that on both runs, no indice was repeated for the same producer, and that the consumers recieved indices in numerical order on a per producer basis.

Q. 1.3 The `cat`'d results are provided above.

Q. 2 Configuration one sets *buffer size* to 2, *number of producers* to 4, *number of messages per producer* to 3, and *number of consumers* to 2.

Q. 2.1 I implemented this by changing the following constants as stated here:

```
#define BUFF_SIZE   2
#define N_PROD      4
#define N_MESSG     3
#define N_CONS      2
```

I then ran make to recompile the reconfigured `nachos`.

```
$ make
... (many lines omitted)
ln -sf arch/unknown-i386-linux/bin/nachos nachos
```

I then completed parts **a** and **b** as follows:

(a) Here I ran the Nachos without randomization with command: `./nachos`.

```
$ ./nachos
No threads ready or runnable, and no pending interrupts.
```

```
Assuming the program completed.
Machine halting!

Ticks: total 1330, idle 0, system 1330, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
$ ls tmp_*
tmp_0  tmp_1
$ cat tmp_0
producer id --> 0; Message number --> 0;
producer id --> 0; Message number --> 1;
producer id --> 3; Message number --> 0;
producer id --> 3; Message number --> 1;
producer id --> 3; Message number --> 2;
$ cat tmp_1
producer id --> 1; Message number --> 0;
producer id --> 1; Message number --> 1;
producer id --> 0; Message number --> 2;
producer id --> 1; Message number --> 2;
producer id --> 2; Message number --> 0;
producer id --> 2; Message number --> 1;
producer id --> 2; Message number --> 2;
$ rm tmp_*
```

(b) Here I ran the Nachos with random seed 99: `./nachos -rs 99`.

```
$ ./nachos -rs 99
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 1494, idle 34, system 1460, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
$ ls tmp_*
tmp_0  tmp_1
$ cat tmp_0
producer id --> 0; Message number --> 0;
producer id --> 1; Message number --> 1;
```

```
producer id --> 0; Message number --> 2;
producer id --> 3; Message number --> 0;
producer id --> 3; Message number --> 1;
producer id --> 2; Message number --> 1;
producer id --> 2; Message number --> 2;
$ cat tmp_1
producer id --> 1; Message number --> 0;
producer id --> 0; Message number --> 1;
producer id --> 1; Message number --> 2;
producer id --> 2; Message number --> 0;
producer id --> 3; Message number --> 2;
$ rm tmp_*
```

Q. 2.2 I then examined the output files (`cat`'d above for clarity) of the consumers to see if the bounded buffer problem is solved and implemented correctly.

We can see that on both runs, no indice was repeated for the same producer, and that the consumers recieved indices in numerical order on a per producer basis.

Q. 2.3 The `cat`'d results are provided above.

Q. 3 Configuration one sets *buffer size* to 5, *number of producers* to 3, *number of messages per producer* to 4, and *number of consumers* to 3.

Q. 3.1 I implemented this by changing the following constants as stated here:

```
#define BUFF_SIZE   5
#define N_PROD      3
#define N_MESSG     4
#define N_CONS      3
```

I then ran make to recompile the reconfigured `nachos`.

```
$ make
... (many lines omitted)
ln -sf arch/unknown-i386-linux/bin/nachos nachos
```

I then completed parts **a** and **b** as follows:

(a) Here I ran the Nachos without randomization with command: `./nachos`.

```
$ ./nachos
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 1330, idle 0, system 1330, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
```

```
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
$ ls tmp_*
tmp_0  tmp_1  tmp_2
$ cat tmp_0
producer id --> 0; Message number --> 0;
producer id --> 0; Message number --> 1;
producer id --> 0; Message number --> 2;
producer id --> 0; Message number --> 3;
$ cat tmp_1
producer id --> 1; Message number --> 0;
producer id --> 1; Message number --> 1;
producer id --> 1; Message number --> 2;
producer id --> 1; Message number --> 3;
$ cat tmp_2
producer id --> 2; Message number --> 0;
producer id --> 2; Message number --> 1;
producer id --> 2; Message number --> 2;
producer id --> 2; Message number --> 3;
$ rm tmp_*
```

(b) Here I ran the Nachos with random seed 99: `./nachos -rs 99`.

```
$ ./nachos -rs 99
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 1494, idle 34, system 1460, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
$ ls tmp_*
tmp_0  tmp_1  tmp_2
$ cat tmp_0
producer id --> 0; Message number --> 0;
producer id --> 1; Message number --> 1;
producer id --> 2; Message number --> 1;
producer id --> 0; Message number --> 3;
producer id --> 2; Message number --> 3;
$ cat tmp_1
producer id --> 1; Message number --> 0;
```

```
producer id --> 0; Message number --> 1;
producer id --> 0; Message number --> 2;
producer id --> 2; Message number --> 2;
$ cat tmp_2
producer id --> 2; Message number --> 0;
producer id --> 1; Message number --> 2;
producer id --> 1; Message number --> 3;
$ rm tmp_*
```

Q. 3.2 I then examined the output files (`cat`'d above for clarity) of the consumers to see if the bounded buffer problem is solved and implemented correctly.

We can see that on both runs, no indice was repeated for the same producer, and that the consumers recieved indices in numerical order on a per producer basis.

Q. 3.3 The `cat`'d results are provided above.

Q. 4 Configuration one sets *buffer size* to 4, *number of producers* to 5, *number of messages per producer* to 20, and *number of consumers* to 4.

Q. 4.1 I implemented this by changing the following constants as stated here:

```
#define BUFF_SIZE   4
#define N_PROD      5
#define N_MESSG     20
#define N_CONS      4
```

I then ran make to recompile the reconfigured `nachos`.

```
$ make
... (many lines omitted)
ln -sf arch/unknown-i386-linux/bin/nachos nachos
```

I then completed parts **a** and **b** as follows:

(a) Here I ran the Nachos without randomization with command: `./nachos`.

```
$ ./nachos
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 10190, idle 0, system 10190, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
```

```
$ ls tmp_*
tmp_0  tmp_1  tmp_2  tmp_3
$ cat tmp_0
producer id --> 0; Message number --> 0;
producer id --> 0; Message number --> 1;
producer id --> 0; Message number --> 2;
producer id --> 0; Message number --> 3;
producer id --> 0; Message number --> 4;
producer id --> 0; Message number --> 5;
producer id --> 0; Message number --> 6;
producer id --> 0; Message number --> 7;
producer id --> 0; Message number --> 8;
producer id --> 0; Message number --> 9;
producer id --> 0; Message number --> 10;
producer id --> 0; Message number --> 11;
producer id --> 0; Message number --> 12;
producer id --> 0; Message number --> 13;
producer id --> 0; Message number --> 14;
producer id --> 0; Message number --> 15;
producer id --> 0; Message number --> 16;
producer id --> 0; Message number --> 17;
producer id --> 0; Message number --> 18;
producer id --> 0; Message number --> 19;
producer id --> 4; Message number --> 0;
producer id --> 4; Message number --> 1;
producer id --> 4; Message number --> 2;
producer id --> 4; Message number --> 3;
producer id --> 4; Message number --> 4;
producer id --> 4; Message number --> 5;
producer id --> 4; Message number --> 6;
producer id --> 4; Message number --> 7;
producer id --> 4; Message number --> 8;
producer id --> 4; Message number --> 9;
producer id --> 4; Message number --> 10;
producer id --> 4; Message number --> 11;
producer id --> 4; Message number --> 12;
producer id --> 4; Message number --> 13;
producer id --> 4; Message number --> 14;
producer id --> 4; Message number --> 15;
producer id --> 4; Message number --> 16;
producer id --> 4; Message number --> 17;
producer id --> 4; Message number --> 18;
producer id --> 4; Message number --> 19;
$ cat tmp_1
producer id --> 1; Message number --> 0;
producer id --> 1; Message number --> 1;
producer id --> 1; Message number --> 2;
producer id --> 1; Message number --> 3;
```

```
producer id --> 1; Message number --> 4;
producer id --> 1; Message number --> 5;
producer id --> 1; Message number --> 6;
producer id --> 1; Message number --> 7;
producer id --> 1; Message number --> 8;
producer id --> 1; Message number --> 9;
producer id --> 1; Message number --> 10;
producer id --> 1; Message number --> 11;
producer id --> 1; Message number --> 12;
producer id --> 1; Message number --> 13;
producer id --> 1; Message number --> 14;
producer id --> 1; Message number --> 15;
producer id --> 1; Message number --> 16;
producer id --> 1; Message number --> 17;
producer id --> 1; Message number --> 18;
producer id --> 1; Message number --> 19;
$ cat tmp_2
producer id --> 2; Message number --> 0;
producer id --> 2; Message number --> 1;
producer id --> 2; Message number --> 2;
producer id --> 2; Message number --> 3;
producer id --> 2; Message number --> 4;
producer id --> 2; Message number --> 5;
producer id --> 2; Message number --> 6;
producer id --> 2; Message number --> 7;
producer id --> 2; Message number --> 8;
producer id --> 2; Message number --> 9;
producer id --> 2; Message number --> 10;
producer id --> 2; Message number --> 11;
producer id --> 2; Message number --> 12;
producer id --> 2; Message number --> 13;
producer id --> 2; Message number --> 14;
producer id --> 2; Message number --> 15;
producer id --> 2; Message number --> 16;
producer id --> 2; Message number --> 17;
producer id --> 2; Message number --> 18;
producer id --> 2; Message number --> 19;
$ cat tmp_3
producer id --> 3; Message number --> 0;
producer id --> 3; Message number --> 1;
producer id --> 3; Message number --> 2;
producer id --> 3; Message number --> 3;
producer id --> 3; Message number --> 4;
producer id --> 3; Message number --> 5;
producer id --> 3; Message number --> 6;
producer id --> 3; Message number --> 7;
producer id --> 3; Message number --> 8;
producer id --> 3; Message number --> 9;
```

```
producer id --> 3; Message number --> 10;
producer id --> 3; Message number --> 11;
producer id --> 3; Message number --> 12;
producer id --> 3; Message number --> 13;
producer id --> 3; Message number --> 14;
producer id --> 3; Message number --> 15;
producer id --> 3; Message number --> 16;
producer id --> 3; Message number --> 17;
producer id --> 3; Message number --> 18;
producer id --> 3; Message number --> 19;
$ rm tmp_*
```

(b) Here I ran the Nachos with random seed 99: `./nachos -rs 99`.

```
$ ./nachos -rs 99
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 11334, idle 54, system 11280, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
$ ls tmp_*
tmp_0  tmp_1  tmp_2  tmp_3
$ cat tmp_0
producer id --> 3; Message number --> 0;
producer id --> 4; Message number --> 0;
producer id --> 4; Message number --> 1;
producer id --> 0; Message number --> 5;
producer id --> 1; Message number --> 3;
producer id --> 3; Message number --> 5;
producer id --> 0; Message number --> 8;
producer id --> 2; Message number --> 7;
producer id --> 3; Message number --> 6;
producer id --> 0; Message number --> 11;
producer id --> 4; Message number --> 7;
producer id --> 4; Message number --> 8;
producer id --> 2; Message number --> 12;
producer id --> 3; Message number --> 11;
producer id --> 0; Message number --> 15;
producer id --> 4; Message number --> 13;
producer id --> 1; Message number --> 12;
producer id --> 3; Message number --> 16;
```

```
producer id --> 1; Message number --> 13;
producer id --> 1; Message number --> 14;
producer id --> 3; Message number --> 17;
producer id --> 1; Message number --> 15;
producer id --> 1; Message number --> 16;
$ cat tmp_1
producer id --> 0; Message number --> 0;
producer id --> 2; Message number --> 0;
producer id --> 0; Message number --> 1;
producer id --> 0; Message number --> 2;
producer id --> 3; Message number --> 1;
producer id --> 2; Message number --> 5;
producer id --> 1; Message number --> 2;
producer id --> 0; Message number --> 6;
producer id --> 4; Message number --> 3;
producer id --> 1; Message number --> 5;
producer id --> 2; Message number --> 10;
producer id --> 2; Message number --> 11;
producer id --> 3; Message number --> 7;
producer id --> 3; Message number --> 8;
producer id --> 3; Message number --> 9;
producer id --> 1; Message number --> 9;
producer id --> 1; Message number --> 11;
producer id --> 3; Message number --> 13;
producer id --> 3; Message number --> 15;
producer id --> 4; Message number --> 16;
producer id --> 4; Message number --> 17;
producer id --> 2; Message number --> 18;
producer id --> 1; Message number --> 19;
$ cat tmp_2
producer id --> 1; Message number --> 0;
producer id --> 2; Message number --> 1;
producer id --> 2; Message number --> 3;
producer id --> 0; Message number --> 3;
producer id --> 1; Message number --> 1;
producer id --> 3; Message number --> 3;
producer id --> 3; Message number --> 4;
producer id --> 2; Message number --> 6;
producer id --> 4; Message number --> 5;
producer id --> 2; Message number --> 8;
producer id --> 2; Message number --> 9;
producer id --> 1; Message number --> 6;
producer id --> 4; Message number --> 6;
producer id --> 1; Message number --> 8;
producer id --> 0; Message number --> 14;
producer id --> 1; Message number --> 10;
producer id --> 3; Message number --> 12;
producer id --> 4; Message number --> 12;
```

```
producer id --> 0; Message number --> 16;
producer id --> 0; Message number --> 17;
producer id --> 4; Message number --> 15;
producer id --> 0; Message number --> 18;
producer id --> 4; Message number --> 19;
producer id --> 3; Message number --> 19;
$ cat tmp_3
producer id --> 2; Message number --> 2;
producer id --> 2; Message number --> 4;
producer id --> 0; Message number --> 4;
producer id --> 3; Message number --> 2;
producer id --> 4; Message number --> 2;
producer id --> 0; Message number --> 7;
producer id --> 1; Message number --> 4;
producer id --> 4; Message number --> 4;
producer id --> 0; Message number --> 9;
producer id --> 0; Message number --> 10;
producer id --> 1; Message number --> 7;
producer id --> 0; Message number --> 12;
producer id --> 0; Message number --> 13;
producer id --> 3; Message number --> 10;
producer id --> 4; Message number --> 9;
producer id --> 4; Message number --> 10;
producer id --> 4; Message number --> 11;
producer id --> 2; Message number --> 13;
producer id --> 3; Message number --> 14;
producer id --> 2; Message number --> 14;
producer id --> 4; Message number --> 14;
producer id --> 2; Message number --> 15;
producer id --> 2; Message number --> 16;
producer id --> 4; Message number --> 18;
producer id --> 2; Message number --> 17;
producer id --> 3; Message number --> 18;
producer id --> 2; Message number --> 19;
producer id --> 0; Message number --> 19;
producer id --> 1; Message number --> 17;
producer id --> 1; Message number --> 18;
$ rm tmp_*
```

Q. 4.2 I then examined the output files (`cat`'d above for clarity) of the consumers to see if the bounded buffer problem is solved and implemented correctly.

We can see that on both runs, no indice was repeated for the same producer, and that the consumers recieved indices in numerical order on a per producer basis.

Q. 4.3 The `cat`'d results are provided above.

2 Mental Trace of Augmented Lab 4-3

Q. 1 List all the context switches occurred in the order of time during this test run of the Nachos. For each context switch, indicate (1) the current th read, (2) the next thread, (3) the thread function causing the context switch (Yield() or Sleep()), (4) the content of the ready-queue after the context switch, and (5) the value and the content of the queque of each of semaphores mutex, nempty and nfull after the context switch, by filling the table as follows:

1.
   - Current: M
   - Next: P_0
   - Function: Sleep
   - Ready Queue: head $\longrightarrow P_1 \longrightarrow P_2 \longrightarrow C_0 \longrightarrow C_1 \longrightarrow \varnothing$
   - Queue of semaphore `mutex`: head $\longrightarrow \varnothing$
   - Queue of semaphore `nempty`: head $\longrightarrow \varnothing$
   - Queue of semaphore `nfull`: head $\longrightarrow \varnothing$
2.
   - Current: P_0
   - Next: P_1
   - Function: Yield
   - Ready Queue: head $\longrightarrow P_2 \longrightarrow C_0 \longrightarrow C_1 \longrightarrow P_0 \longrightarrow \varnothing$
   - Queue of semaphore `mutex`: head $\longrightarrow \varnothing$
   - Queue of semaphore `nempty`: head $\longrightarrow \varnothing$
   - Queue of semaphore `nfull`: head $\longrightarrow \varnothing$
3.
   - Current: P_1
   - Next: P_2
   - Function: Sleep
   - Ready Queue: head $\longrightarrow C_0 \longrightarrow C_1 \longrightarrow P_0 \longrightarrow \varnothing$
   - Queue of semaphore `mutex`: head $\longrightarrow P_1 \longrightarrow \varnothing$
   - Queue of semaphore `nempty`: head $\longrightarrow \varnothing$
   - Queue of semaphore `nfull`: head $\longrightarrow \varnothing$
4.
   - Current: P_2
   - Next: C_0
   - Function: Sleep
   - Ready Queue: head $\longrightarrow C_1 \longrightarrow P_0 \longrightarrow \varnothing$
   - Queue of semaphore `mutex`: head $\longrightarrow P_1 \longrightarrow P_2 \longrightarrow \varnothing$
   - Queue of semaphore `nempty`: head $\longrightarrow \varnothing$
   - Queue of semaphore `nfull`: head $\longrightarrow \varnothing$
5.
   - Current: C_0
   - Next: C_1

- Function: Sleep
- Ready Queue: head $\longrightarrow P_0 \longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow P_1 \longrightarrow P_2 \longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow C_0 \longrightarrow \varnothing$

6.
- Current: C_1
- Next: P_0
- Function: Sleep
- Ready Queue: head $\longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow P_1 \longrightarrow P_2 \longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow C_0 \longrightarrow C_1 \longrightarrow \varnothing$

7.
- Current: P_0
- Next: P_1
- Function: Sleep
- Ready Queue: head $\longrightarrow C_0 \longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow P_2 \longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow P_0 \longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow C_1 \longrightarrow \varnothing$

8.
- Current: P_1
- Next: C_0
- Function: Yield
- Ready Queue: head $\longrightarrow P_1 \longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow P_2 \longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow P_0 \longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow C_1 \longrightarrow \varnothing$

9.
- Current: C_0
- Next: P_1
- Function: Sleep
- Ready Queue: head $\longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow P_2 \longrightarrow C_0 \longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow P_0 \longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow C_1 \longrightarrow \varnothing$

10.
- Current: P_1
- Next: P_2
- Function: Sleep

- Ready Queue: head $\longrightarrow C_1 \longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow C_0 \longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow P_0 \longrightarrow P_1 \longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow \varnothing$

11.
- Current: P_2
- Next: C_1
- Function: Yield
- Ready Queue: head $\longrightarrow P_2 \longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow C_0 \longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow P_0 \longrightarrow P_1 \longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow \varnothing$

12.
- Current: C_1
- Next: P_2
- Function: Sleep
- Ready Queue: head $\longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow C_0 \longrightarrow C_1 \longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow P_0 \longrightarrow P_1 \longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow \varnothing$

13.
- Current: P_2
- Next: C_0
- Function: Sleep
- Ready Queue: head $\longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow C_1 \longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow P_0 \longrightarrow P_1 \longrightarrow P_2 \longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow \varnothing$

14.
- Current: C_0
- Next: C_1
- Function: Yield
- Ready Queue: head $\longrightarrow P_0 \longrightarrow C_0 \longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow P_1 \longrightarrow P_2 \longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow \varnothing$

15.
- Current: C_1
- Next: P_0
- Function: Yield
- Ready Queue: head $\longrightarrow C_0 \longrightarrow C_1 \longrightarrow \varnothing$

- Queue of semaphore `mutex`: head $\longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow P_1 \longrightarrow P_2 \longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow \varnothing$

16.
- Current: P_0
- Next: C_0
- Function: Sleep
- Ready Queue: head $\longrightarrow C_1 \longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow P_0 \longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow P_1 \longrightarrow P_2 \longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow \varnothing$

17.
- Current: C_0
- Next: C_1
- Function: Sleep
- Ready Queue: head $\longrightarrow P_0 \longrightarrow P_1 \longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow P_2 \longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow C_0 \longrightarrow \varnothing$

18.
- Current: C_1
- Next: P_0
- Function: Sleep
- Ready Queue: head $\longrightarrow P_1 \longrightarrow P_2 \longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow C_0 \longrightarrow C_1 \longrightarrow \varnothing$

19.
- Current: P_0
- Next: P_1
- Function: Yield
- Ready Queue: head $\longrightarrow P_2 \longrightarrow P_0 \longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow C_0 \longrightarrow C_1 \longrightarrow \varnothing$

20.
- Current: P_1
- Next: P_2
- Function: Yield
- Ready Queue: head $\longrightarrow P_0 \longrightarrow P_1 \longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow \varnothing$

- Queue of semaphore `nempty`: head $\longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow C_0 \longrightarrow C_1 \longrightarrow \varnothing$

21.
- Current: P_2
- Next: P_0
- Function: Sleep
- Ready Queue: head $\longrightarrow P_1 \longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow P_2 \longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow C_0 \longrightarrow C_1 \longrightarrow \varnothing$

22.
- Current: P_0
- Next: P_1
- Function: Sleep
- Ready Queue: head $\longrightarrow P_2 \longrightarrow C_0 \longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow C_1 \longrightarrow \varnothing$

23.
- Current: P_1
- Next: P_2
- Function: Sleep
- Ready Queue: head $\longrightarrow C_0 \longrightarrow C_1 \longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow \varnothing$

24.
- Current: P_2
- Next: C_0
- Function: Yield
- Ready Queue: head $\longrightarrow C_1 \longrightarrow P_2 \longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow \varnothing$

25.
- Current: C_0
- Next: C_1
- Function: Yield
- Ready Queue: head $\longrightarrow P_2 \longrightarrow C_0 \longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow \varnothing$

- Queue of semaphore `nfull`: head $\longrightarrow \varnothing$

26.
- Current: C_1
- Next: P_2
- Function: Sleep
- Ready Queue: head $\longrightarrow C_0 \longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow C_1 \longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow \varnothing$

27.
- Current: P_2
- Next: C_0
- Function: Sleep
- Ready Queue: head $\longrightarrow C_1 \longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow \varnothing$

28.
- Current: C_0
- Next: C_1
- Function: Yield
- Ready Queue: head $\longrightarrow C_0 \longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow \varnothing$

29.
- Current: C_1
- Next: C_0
- Function: Sleep
- Ready Queue: head $\longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow C_1 \longrightarrow \varnothing$

30.
- Current: C_0
- Next: $\varnothing$
- Function: Sleep
- Ready Queue: head $\longrightarrow \varnothing$
- Queue of semaphore `mutex`: head $\longrightarrow \varnothing$
- Queue of semaphore `nempty`: head $\longrightarrow \varnothing$
- Queue of semaphore `nfull`: head $\longrightarrow C_1 \longrightarrow C_0 \longrightarrow \varnothing$