Opdracht 8

Valgrind:

je kunt met valgrind bepalen of je een memory leak hebt door valgrind --leak-check=full ./<naam_van_programma> te gebruiken. Nadat het programma is gestopt krijg je te zien hoeveel bytes verloren zijn gegaan.

Wanneer de laatste deleteQueue in main.c uit wordt gecomment krijg je te zien dat er 32 bytes verloren zijn. Dit komt doordat de deleteQueue deze bytes freed.

```
==10520==
==10520== HEAP SUMMARY:
==10520==     in use at exit: 32 bytes in 1 blocks
==10520==   total heap usage: 6 allocs, 5 frees, 1,184 bytes allocated
==10520==
==10520== 32 bytes in 1 blocks are definitely lost in loss record 1 of 1
==10520==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64
-linux.so)
==10520==    by 0x400744: createQueue (Queue.c:18)
==10520==    by 0x400CC0: main (main.c:57)
==10520==
==10520== LEAK SUMMARY:
==10520==    definitely lost: 32 bytes in 1 blocks
==10520==    indirectly lost: 0 bytes in 0 blocks
==10520==      possibly lost: 0 bytes in 0 blocks
==10520==    still reachable: 0 bytes in 0 blocks
==10520==         suppressed: 0 bytes in 0 blocks
==10520==
==10520== For counts of detected and suppressed errors, rerun with: -v
==10520== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Als deleteQueue wel aangeroepen wordt krijg je te zien dat alle blocks gefreed zijn.

```
==10620==
==10620== HEAP SUMMARY:
==10620==     in use at exit: 0 bytes in 0 blocks
==10620==   total heap usage: 6 allocs, 6 frees, 1,184 bytes allocated
==10620==
==10620== All heap blocks were freed -- no leaks are possible
==10620==
==10620== For counts of detected and suppressed errors, rerun with: -v
==10620== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

SharedQueue

Bij deze opdracht was het de bedoeling om met meerdere threads dezelfde sharedQueue te vullen en deze na elke 15 seconde uit te lezen en in een bestand te zetten met een andere thread. Met ctrl+C moeten alle threads hun cycle afmaken en daarna stoppen.

Wanneer een producer thread aan wordt gemaakt zal het aantal runningProducers(globale variabele) worden opgehoogd. Daarna wordt een oneindige loop aangemaakt. De loop begint met een sleep die net zo lang duurt als is aangegeven in de input data.
de mutex wordt gelockt nadat de sleep is afgelopen. Hierdoor kan een variabele niet halverwege het overschrijven worden aangepast door een andere thread waardoor de data niet meer klopt.
nadat de queue is aangepast wordt de mutex weer unlocked zodat andere threads verder kunnen werken met de queue. Daarna wordt gecheckt of ctrl+C is ingedrukt, en als dit zo is wordt runningProducers verlaagd en stopt de thread.

```
void *producer(void *input)
{
  runningProducers++;
  data_t *arguments = input;
  while(1)
    {
      sleep(arguments->intVal);
      pthread_mutex_lock(&lock);

      if(backQueue(&queue) == NULL)
        {
          createQueue(&queue, *arguments);
        }
      else
        {
          pushQueue(&queue, *arguments);
        }
      pthread_mutex_unlock(&lock);
      if(CTRLCPressed == 1)
        {
          runningProducers--;
          pthread_exit(NULL);
        }
    }
}
```

De consumer thread slaapt eerst voor 15 seconde waarna het de queue lockt zodat er geen data naar de queue kan worden geschreven zolang de consumer bezig is met het schrijven naar het log bestand. De queue wordt laten zien in de terminal en geschreven naar log.txt. de queue wordt weggegooid en daarna wordt de mutex unlocked.
als het aantal runningProducers nul is stopt ook de consumer thread.

showQueueToFile is bijna dezelfde functie als showQueue, alleen is het verschil dat deze alles wat normaal in de terminal geschreven wordt nu in een tekst bestand geschreven wordt behalve de lastnode.

```
void *consumer(void *input)
{
  while(1)
    {
      sleep(15);
      pthread_mutex_lock(&lock);
      showQueue(&queue);
      showQueueToFile(&queue, "log.txt");
      deleteQueue(&queue);
      pthread_mutex_unlock(&lock);
      if(runningProducers<=0)
        {
          deleteQueue(&queue);
          pthread_exit(NULL);
        }
    }
}
```

De sighandler zet CTRLCpressed op 1 wanneer sig_num (CTRL+C) binnen komt. Dit is een globale variabele zodat de producer threads er bij kunnen.

```
void sighandler(int sig_num)
{
  CTRLCPressed = 1;
}
```

In de main wordt de mutex eerst aangemaakt zodat de threads deze kunnen gebruiken. Daarna wordt de sighandler gekoppeld met CTRL+C.
Drie producer threads, elk met andere waardes in hun argument, en een consumer thread worden aangemaakt.
nadat alle threads zijn aangemaakt zal de main wachten totdat alle threads zijn gestopt en wordt de mutex weggegooid.

```
int main()
{
  pthread_mutex_init(&lock, NULL);
  signal(SIGINT, sighandler);

  pthread_t tid;
  pthread_create(&tid, NULL, &producer, (void *)&thread1);
  pthread_create(&tid, NULL, &producer, (void *)&thread2);
  pthread_create(&tid, NULL, &producer, (void *)&thread3);
  pthread_create(&tid, NULL, &consumer, NULL);
  pthread_join(tid, NULL);
  pthread_mutex_destroy(&lock);
  return 0;
}
```

In de output is te zien dat de queue na 15 seconde bestaat uit 14 nodes. Dit komt doordat thread 1 elke 2 seconde iets toevoegd, thread 2 elke 3 seconde en thread 3 elke 4 seconde. Thread 1 heeft in deze 15 seconde de tijd gehad om 7.5x iets toe te voegen. Je kunt iets niet half toevoegen en voegt dus 7x iets toe. Dit zorgt ervoor dat in de volgende 15 seconde thread 1 al de helft van de tijd gewacht heeft en dus 8x iets kan sturen. Thread 2 stuurt elke 3 seconde iets waardoor deze altijd 5x iets kan sturen. Voor thread 3 geldt hetzelfde als thread 1 want hij heeft tijd om 3.75x iets toe te voegen. Thread 3 hoeft dus maar een kwart van de tijd te wachten en kan in de volgende 15 seconde 4.5x iets toevoegen.

```
student@UbuntuSE:~/exercises/ex8$ ./sharedQueue
Queue contains 14 nodes:
Last node: 0x7fc3b80009e0
pNode = 0x7fc3b80008c0  Data = '2' 'Thread 1'  nextN = 0x7fc3b00008c0
pNode = 0x7fc3b00008c0  Data = '3' 'Thread 2'  nextN = 0x7fc3b40008c0
pNode = 0x7fc3b40008c0  Data = '4' 'Thread 3'  nextN = 0x7fc3b80008f0
pNode = 0x7fc3b80008f0  Data = '2' 'Thread 1'  nextN = 0x7fc3b00008f0
pNode = 0x7fc3b00008f0  Data = '3' 'Thread 2'  nextN = 0x7fc3b8000920
pNode = 0x7fc3b8000920  Data = '2' 'Thread 1'  nextN = 0x7fc3b40008f0
pNode = 0x7fc3b40008f0  Data = '4' 'Thread 3'  nextN = 0x7fc3b8000950
pNode = 0x7fc3b8000950  Data = '2' 'Thread 1'  nextN = 0x7fc3b0000920
pNode = 0x7fc3b0000920  Data = '3' 'Thread 2'  nextN = 0x7fc3b8000980
pNode = 0x7fc3b8000980  Data = '2' 'Thread 1'  nextN = 0x7fc3b0000950
pNode = 0x7fc3b0000950  Data = '3' 'Thread 2'  nextN = 0x7fc3b4000920
pNode = 0x7fc3b4000920  Data = '4' 'Thread 3'  nextN = 0x7fc3b80009b0
pNode = 0x7fc3b80009b0  Data = '2' 'Thread 1'  nextN = 0x7fc3b80009e0
pNode = 0x7fc3b80009e0  Data = '2' 'Thread 1'  nextN = 0x7fc3b80008c0
Queue contains 16 nodes:
Last node: 0x7fc3b80009e0
pNode = 0x7fc3b0000950  Data = '3' 'Thread 2'  nextN = 0x7fc3b4000920
pNode = 0x7fc3b4000920  Data = '4' 'Thread 3'  nextN = 0x7fc3b80009b0
pNode = 0x7fc3b80009b0  Data = '2' 'Thread 1'  nextN = 0x7fc3b0000920
pNode = 0x7fc3b0000920  Data = '3' 'Thread 2'  nextN = 0x7fc3b8000980
pNode = 0x7fc3b8000980  Data = '2' 'Thread 1'  nextN = 0x7fc3b40008f0
pNode = 0x7fc3b40008f0  Data = '4' 'Thread 3'  nextN = 0x7fc3b8000950
pNode = 0x7fc3b8000950  Data = '2' 'Thread 1'  nextN = 0x7fc3b00008f0
pNode = 0x7fc3b00008f0  Data = '3' 'Thread 2'  nextN = 0x7fc3b8000920
pNode = 0x7fc3b8000920  Data = '2' 'Thread 1'  nextN = 0x7fc3b00008c0
pNode = 0x7fc3b00008c0  Data = '3' 'Thread 2'  nextN = 0x7fc3b40008c0
pNode = 0x7fc3b40008c0  Data = '4' 'Thread 3'  nextN = 0x7fc3b80008f0
pNode = 0x7fc3b80008f0  Data = '2' 'Thread 1'  nextN = 0x7fc3b80008c0
pNode = 0x7fc3b80008c0  Data = '2' 'Thread 1'  nextN = 0x7fc3b0000980
pNode = 0x7fc3b0000980  Data = '3' 'Thread 2'  nextN = 0x7fc3b4000950
pNode = 0x7fc3b4000950  Data = '4' 'Thread 3'  nextN = 0x7fc3b80009e0
pNode = 0x7fc3b80009e0  Data = '2' 'Thread 1'  nextN = 0x7fc3b0000950
```

Figuur 1: normale output

Wanneer CTRL+C ergens in de 15 seconde in wordt gedrukt is te zien dat er maar 6 nodes zijn. Je kunt hier dus zien dat de producer threads gestopt zijn. Het programma stopt nadat deze output is laten zien.

```
Queue contains 15 nodes:
Last node: 0x7fc3b80008c0
pNode = 0x7fc3b0000950  Data = '3' 'Thread 2'  nextN = 0x7fc3b80009e0
pNode = 0x7fc3b80009e0  Data = '2' 'Thread 1'  nextN = 0x7fc3b0000920
pNode = 0x7fc3b0000920  Data = '3' 'Thread 2'  nextN = 0x7fc3b4000920
pNode = 0x7fc3b4000920  Data = '4' 'Thread 3'  nextN = 0x7fc3b80009b0
pNode = 0x7fc3b80009b0  Data = '2' 'Thread 1'  nextN = 0x7fc3b8000980
pNode = 0x7fc3b8000980  Data = '2' 'Thread 1'  nextN = 0x7fc3b00008f0
pNode = 0x7fc3b00008f0  Data = '3' 'Thread 2'  nextN = 0x7fc3b40008f0
pNode = 0x7fc3b40008f0  Data = '4' 'Thread 3'  nextN = 0x7fc3b8000950
pNode = 0x7fc3b8000950  Data = '2' 'Thread 1'  nextN = 0x7fc3b00008c0
pNode = 0x7fc3b00008c0  Data = '3' 'Thread 2'  nextN = 0x7fc3b8000920
pNode = 0x7fc3b8000920  Data = '2' 'Thread 1'  nextN = 0x7fc3b40008c0
pNode = 0x7fc3b40008c0  Data = '4' 'Thread 3'  nextN = 0x7fc3b80008f0
pNode = 0x7fc3b80008f0  Data = '2' 'Thread 1'  nextN = 0x7fc3b0000980
pNode = 0x7fc3b0000980  Data = '3' 'Thread 2'  nextN = 0x7fc3b80008c0
pNode = 0x7fc3b80008c0  Data = '2' 'Thread 1'  nextN = 0x7fc3b0000950
^C
Queue contains 6 nodes:
Last node: 0x7fc3b40008f0
pNode = 0x7fc3b0000980  Data = '3' 'Thread 2'  nextN = 0x7fc3b40008c0
pNode = 0x7fc3b40008c0  Data = '4' 'Thread 3'  nextN = 0x7fc3b80008f0
pNode = 0x7fc3b80008f0  Data = '2' 'Thread 1'  nextN = 0x7fc3b8000920
pNode = 0x7fc3b8000920  Data = '2' 'Thread 1'  nextN = 0x7fc3b00008c0
pNode = 0x7fc3b00008c0  Data = '3' 'Thread 2'  nextN = 0x7fc3b40008f0
pNode = 0x7fc3b40008f0  Data = '4' 'Thread 3'  nextN = 0x7fc3b0000980
```

Figuur 2: output na ctrl + C

Het log.txt bestand ziet er precies hetzelfde uit als deze outputs van de terminal alleen dan dus zonder de lastnode want anders leken de terminal en het tekstbestand precies hetzelfde.

```
 51 Queue contains 15 nodes:
 52 pNode = 0x7fc3b0000950  Data = '3' 'Thread 2'  nextN = 0x7fc3b80009e0
 53 pNode = 0x7fc3b80009e0  Data = '2' 'Thread 1'  nextN = 0x7fc3b0000920
 54 pNode = 0x7fc3b0000920  Data = '3' 'Thread 2'  nextN = 0x7fc3b4000920
 55 pNode = 0x7fc3b4000920  Data = '4' 'Thread 3'  nextN = 0x7fc3b80009b0
 56 pNode = 0x7fc3b80009b0  Data = '2' 'Thread 1'  nextN = 0x7fc3b8000980
 57 pNode = 0x7fc3b8000980  Data = '2' 'Thread 1'  nextN = 0x7fc3b00008f0
 58 pNode = 0x7fc3b00008f0  Data = '3' 'Thread 2'  nextN = 0x7fc3b40008f0
 59 pNode = 0x7fc3b40008f0  Data = '4' 'Thread 3'  nextN = 0x7fc3b8000950
 60 pNode = 0x7fc3b8000950  Data = '2' 'Thread 1'  nextN = 0x7fc3b00008c0
 61 pNode = 0x7fc3b00008c0  Data = '3' 'Thread 2'  nextN = 0x7fc3b8000920
 62 pNode = 0x7fc3b8000920  Data = '2' 'Thread 1'  nextN = 0x7fc3b40008c0
 63 pNode = 0x7fc3b40008c0  Data = '4' 'Thread 3'  nextN = 0x7fc3b80008f0
 64 pNode = 0x7fc3b80008f0  Data = '2' 'Thread 1'  nextN = 0x7fc3b0000980
 65 pNode = 0x7fc3b0000980  Data = '3' 'Thread 2'  nextN = 0x7fc3b80008c0
 66 pNode = 0x7fc3b80008c0  Data = '2' 'Thread 1'  nextN = 0x7fc3b0000950
 67 Queue contains 6 nodes:
 68 pNode = 0x7fc3b0000980  Data = '3' 'Thread 2'  nextN = 0x7fc3b40008c0
 69 pNode = 0x7fc3b40008c0  Data = '4' 'Thread 3'  nextN = 0x7fc3b80008f0
 70 pNode = 0x7fc3b80008f0  Data = '2' 'Thread 1'  nextN = 0x7fc3b8000920
 71 pNode = 0x7fc3b8000920  Data = '2' 'Thread 1'  nextN = 0x7fc3b00008c0
 72 pNode = 0x7fc3b00008c0  Data = '3' 'Thread 2'  nextN = 0x7fc3b40008f0
 73 pNode = 0x7fc3b40008f0  Data = '4' 'Thread 3'  nextN = 0x7fc3b0000980
```