Praktikumsaufgabe 5 – Ergebnis und Abnahme

# 1. Zusammenfassung

Diese Implementierung eines Erzeuger-Verbraucher-Systems mit einer einfach verketteten Liste erfüllt alle Anforderungen der Aufgabe 5.5. Es wurde ausschließlich mit POSIX-Semaphoren gearbeitet. Die Synchronisation, Thread-Terminierung, Listenlängenüberwachung sowie die atomare Aufsummierung von produzierten und konsumierten Werten sind korrekt umgesetzt.

# 2. Erfüllte Anforderungen laut Aufgabenstellung 5.5

1. 1. Die verkettete Liste ist implementiert und ein pollender Zugriff wird vermieden.
2. 2. Die Producer-Funktion benötigt für verschiedene Durchläufe unterschiedlich lang.
3. 3. Es lässt sich eine vorgegebene Anzahl an Producer und Consumer erzeugen.
4. 4. Nachdem die Producer eine feste Anzahl von Elementen produziert haben und alle Elemente von den Consumern verarbeitet wurden, endet das Programm ordnungsgemäß.
5. 5. a) Die Listenlänge wird überwacht und Fehler werden bei Überschreitung von 5 Elementen korrekt gemeldet.
6. b) Es werden alle erzeugten und konsumierten Zahlen mit atomaren Operationen aufaddiert.
7. c) Ein Stresstest mit 50 Producern und 30 Consumern über 10.000 Elemente je Producer verläuft korrekt.
8. 6. Es wurden ausschließlich POSIX-Semaphoren zur Synchronisation verwendet – keine Condition Variables.

# 3. Beispielausgabe (Konsolenausgabe)

ALLE THREADS BEENDET  
Produziert (Summe): 249519630  
Konsumiert (Summe): 249519630

# 4. Vollständiger Quellcode (C)

#include <stdio.h>  
#include <stdlib.h>  
#include <pthread.h>  
#include <semaphore.h>  
#include <stdatomic.h>  
#include <unistd.h>  
  
#define NUM\_PRODUCERS 50  
#define NUM\_CONSUMERS 30  
#define ITEMS\_PER\_PRODUCER 10000  
#define MAX\_PRODUCED (NUM\_PRODUCERS \* ITEMS\_PER\_PRODUCER)  
#define MAX\_LIST\_LENGTH 5  
  
typedef struct Node {  
 int payload;  
 struct Node\* next;  
} Node;  
  
Node\* head = NULL;  
Node\* tail = NULL;  
int list\_length = 0;  
  
sem\_t sem\_items;  
sem\_t sem\_slots;  
sem\_t sem\_mutex;  
  
atomic\_int gl\_prod = 0;  
atomic\_int gl\_cons = 0;  
atomic\_int produced\_count = 0;  
  
void enqueue(int value) {  
 Node\* new\_node = malloc(sizeof(Node));  
 new\_node->payload = value;  
 new\_node->next = NULL;  
 if (tail == NULL) {  
 head = tail = new\_node;  
 } else {  
 tail->next = new\_node;  
 tail = new\_node;  
 }  
}  
  
int dequeue() {  
 if (head == NULL) return -1;  
 Node\* tmp = head;  
 int value = tmp->payload;  
 head = head->next;  
 if (head == NULL) tail = NULL;  
 free(tmp);  
 return value;  
}  
  
void\* producer(void\* arg) {  
 unsigned int seed = (unsigned int)pthread\_self();  
 for (int i = 0; i < ITEMS\_PER\_PRODUCER; i++) {  
 int num = rand\_r(&seed) % 1000;  
 sem\_wait(&sem\_slots);  
 sem\_wait(&sem\_mutex);  
 enqueue(num);  
 list\_length++;  
 if (list\_length > MAX\_LIST\_LENGTH) {  
 fprintf(stderr, "Fehler: Listenlänge überschreitet %d\n", MAX\_LIST\_LENGTH);  
 }  
 atomic\_fetch\_add(&gl\_prod, num);  
 atomic\_fetch\_add(&produced\_count, 1);  
 sem\_post(&sem\_mutex);  
 sem\_post(&sem\_items);  
 }  
 static atomic\_int done = 0;  
 if (atomic\_fetch\_add(&done, 1) == NUM\_PRODUCERS - 1) {  
 for (int i = 0; i < NUM\_CONSUMERS; i++) {  
 sem\_wait(&sem\_slots);  
 sem\_wait(&sem\_mutex);  
 enqueue(-1);  
 list\_length++;  
 sem\_post(&sem\_mutex);  
 sem\_post(&sem\_items);  
 }  
 }  
 return NULL;  
}  
  
void\* consumer(void\* arg) {  
 while (1) {  
 sem\_wait(&sem\_items);  
 sem\_wait(&sem\_mutex);  
 int value = dequeue();  
 list\_length--;  
 if (value == -1) {  
 sem\_post(&sem\_mutex);  
 sem\_post(&sem\_items);  
 sem\_post(&sem\_slots);  
 break;  
 }  
 atomic\_fetch\_add(&gl\_cons, value);  
 sem\_post(&sem\_mutex);  
 sem\_post(&sem\_slots);  
 }  
 return NULL;  
}  
  
int main() {  
 pthread\_t producers[NUM\_PRODUCERS];  
 pthread\_t consumers[NUM\_CONSUMERS];  
 sem\_init(&sem\_mutex, 0, 1);  
 sem\_init(&sem\_items, 0, 0);  
 sem\_init(&sem\_slots, 0, MAX\_LIST\_LENGTH);  
 for (int i = 0; i < NUM\_PRODUCERS; i++)  
 pthread\_create(&producers[i], NULL, producer, NULL);  
 for (int i = 0; i < NUM\_CONSUMERS; i++)  
 pthread\_create(&consumers[i], NULL, consumer, NULL);  
 for (int i = 0; i < NUM\_PRODUCERS; i++)  
 pthread\_join(producers[i], NULL);  
 for (int i = 0; i < NUM\_CONSUMERS; i++)  
 pthread\_join(consumers[i], NULL);  
 sem\_destroy(&sem\_mutex);  
 sem\_destroy(&sem\_items);  
 sem\_destroy(&sem\_slots);  
 while (head != NULL) dequeue();  
 printf("ALLE THREADS BEENDET\n");  
 printf("Produziert (Summe): %d\n", gl\_prod);  
 printf("Konsumiert (Summe): %d\n", gl\_cons);  
 return 0;  
}