Ibrahim El Hamdouni
Burak Naci Basaran

Laborbericht 4: Simulation eines Burger-Imbisses

Einleitung:

In diesem Praktikum wurde eine Simulation eines Burger-Imbisses erstellt, in der Mitarbeiter (Threads) Bestellungen von Kunden (ebenfalls Threads) verarbeiten. Die Herausforderung besteht darin, die Synchronisation der Threads zu gewährleisten und dabei Busy-Waiting zu vermeiden.

Implementierung:

Die Synchronisation von Mitarbeitern und Kunden erfolgt über Semaphoren (`sem_t queue_sem`) und Mutexe (`std::mutex`). Durch `std::unique_lock` und `std::lock_guard` wird sichergestellt, dass nur ein Thread gleichzeitig auf eine Ressource zugreift. Der Zugriff auf die Zutaten wird mittels Semaphoren geregelt, die blockieren, bis genügend Zutaten verfügbar sind (`sem_wait(&queue_sem)`), wodurch Busy-Waiting vermieden wird.

```
kali@kali-VirtualBox:~/praktburak$ g++ -std=c++20 -pthread main.cpp -o burger
kali@kali-VirtualBox:~/praktburak$ ./burger 10 100 50 20 100000
Welcome To My Imbiss

Employee 8 served customer 7 with 1 burgers.
Employee 1 served customer 5 with 1 burgers.
Employee 3 served customer 8 with 1 burgers.
Employee 5 served customer 6 with 3 burgers.
Employee 2 served customer 1 with 4 burgers.
Employee 7 served customer 4 with 4 burgers.
Employee 5 served customer 13 with 1 burgers.
Employee 8 served customer 10 with 4 burgers.
Employee 4 served customer 2 with 6 burgers.
Employee 9 served customer 9 with 6 burgers.
Employee 1 served customer 11 with 6 burgers.
Employee 6 served customer 3 with 8 burgers.
Employee 0 served customer 0 with 9 burgers.
Employee 3 served customer 12 with 8 burgers.
Employee 2 served customer 14 with 6 burgers.
Employee 8 served customer 17 with 5 burgers.
Employee 7 served customer 16 with 9 burgers.
Employee 4 served customer 18 with 8 burgers.
Employee 5 served customer 15 with 9 burgers.
Employee 9 served customer 19 with 5 burgers.
Total burgers served: 104
Elapsed time: 2.01475s
Throughput: 51.6194 burgers per second
kali@kali-VirtualBox:~/praktburak$
```

Code-Auszug:

```
////////////////////////////////////////////////////////////////////////////////
//  std::unique_lock<std::mutex> lock(container_ptr->mutex);  ||
//  while(container_ptr->quantity < order.num_burgers){       ||
//      lock.unlock();                                        ||
//      sem_wait(&(container_ptr->refill_sem));               ||
//      lock.lock();                                          ||
//  }                                                         ||
//  container_ptr->quantity -= order.num_burgers;             ||
////////////////////////////////////////////////////////////////////////////////||
```

Dieses Snippet zeigt, wie ein Thread auf die Verfügbarkeit von Zutaten wartet, ohne die CPU unnötig zu belasten.

Skript-Analyse:

Um die Effizienz der Implementierung zu testen, wurde ein Bash-Skript erstellt, das verschiedene Konfigurationen der Simulation automatisch ausführt.

Das Skript iteriert über verschiedene Kombinationen von Mitarbeiterzahlen(1, 2, 5, 10, 20) und Vorbereitungszeiten (1000, 10000, 100000) und führt das Hauptprogramm mit diesen Parametern aus. Dies ermöglicht eine umfassende Analyse der Systemleistung unter verschiedenen Betriebsbedingungen, ohne dass manuelle Eingriffe erforderlich sind.

Die Automatisierung trägt wesentlich zur Genauigkeit der Tests bei und erleichtert die Datensammlung für die grafische Auswertung.

```bash
#!/bin/bash

# Define the different sets of parameters
declare -a num_employees=("1" "2" "5" "10" "20")
declare -a prep_times=("1000" "10000" "100000") # in microseconds

# Fixed parameters based on your requirement
container_capacity=20
refill_rate=100
num_customers=50

# Loop over each combination of employees and prep times
for employees in "${num_employees[@]}"
do
    for prep_time in "${prep_times[@]}"
    do
        echo "Running simulation with $employees employees and $prep_time microseconds prep time"
        ./burger $employees $container_capacity $refill_rate $num_customers $prep_time
    done
done
~
~
~
```

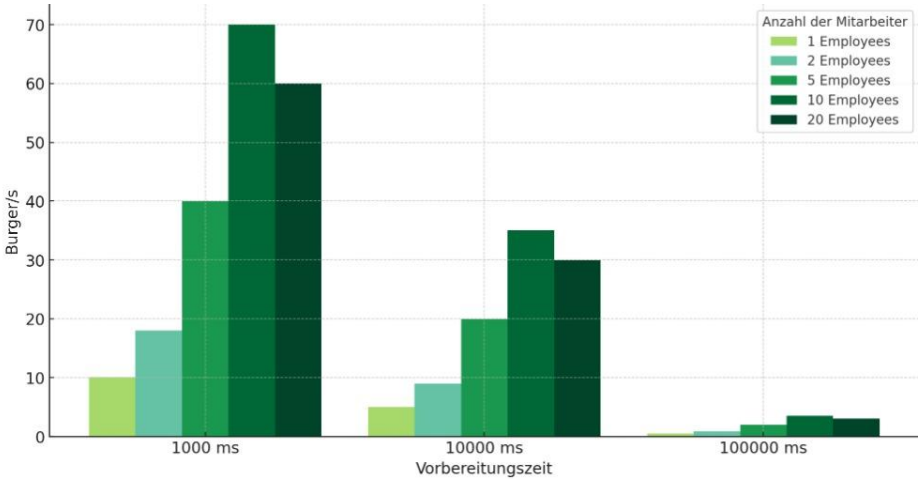
Fazit zum Skript:

Im Graphen sehen wir, dass die Anzahl der produzierten Burger pro Sekunde zunimmt, wenn mehr Mitarbeiter im Imbiss simuliert werden, aber nur bis zu einem gewissen Punkt.

Bei 20 Mitarbeitern verringert sich die Effizienz wieder. Dieses Ergebnis ist intuitiv vielleicht unerwartet, da man annimmt, mehr Mitarbeiter würden stets zu einer höheren Produktionsrate führen. Die Abnahme bei 20 Mitarbeitern lässt sich jedoch durch die Grenzen der Parallelverarbeitung eines Computers erklären. Es gibt einen Overhead bei der Verwaltung einer zu großen Anzahl von Threads, da der Kontextwechsel zwischen den

Threads Ressourcen verbraucht. Außerdem kann der Prozessor nicht unendlich viele Threads gleichzeitig ausführen, was zu einem Engpass führt.

Dieses Ergebnis demonstriert die Wichtigkeit der optimalen Anzahl von Threads zur Maximierung der Durchsatzrate und zeigt, dass mehr nicht immer besser ist, wenn es um parallele Verarbeitung geht.



Frohe Weihnachten.