Inhaltsverzeichnis

[1 Aufgabenstellung 3](#__RefHeading___Toc525_1561136669)

[2 Realisierung 3](#__RefHeading___Toc527_1561136669)

[2.1 Allgemein 3](#__RefHeading___Toc529_1561136669)

[3 C-Programm 3](#__RefHeading___Toc533_1561136669)

[3.1 Compilieren 3](#__RefHeading___Toc535_1561136669)

[3.2 Ausführen 3](#__RefHeading___Toc537_1561136669)

[4 Ergebnis 4](#__RefHeading___Toc539_1561136669)

[5 Code 4](#__RefHeading___Toc1826_1561136669)

[5.1 Beschreibung 4](#__RefHeading___Toc1828_1561136669)

[5.2 Code 4](#__RefHeading___Toc1830_1561136669)

# Aufgabenstellung

In einer Auslieferungshalle werden ständig Pakete von einem Lieferanten angeliefert und von

zwei Auslieferern abgeholt. Der Lieferant und die Auslieferer müssen dafür ein Tor

durchfahren. Das Tor kann immer nur von einer Person durchfahren werden. Der

Lieferant bringt mit jeder Lieferung 3 Pakete zum Wareneingang. An der Ausgabe

holt ein Auslieferer jeweils 2 Pakete ab, der andere Auslieferer 1 Paket

Es existiert genau ein Prozess Lieferant, ein Prozess Auslieferer\_A und ein Prozess Auslieferer\_B

Synchronisieren Sie die Prozesse, indem Sie die nötigen Semaphoren erzeugen,

diese mit Startwerten versehen und Semaphor-Operationen einfügen.

Folgende Bedingungen müssen erfüllt sein:

a) Es darf immer nur ein Prozess das Tor durchfahren.

b) Es darf immer nur einer der beiden Auslieferer die Warenausgabe betreten.

c) Es soll möglich sein, dass der Lieferant und ein Auslieferer gleichzeitig Waren

entladen bzw. aufladen.

d) Die Lagerhalle kann maximal 10 Pakete aufnehmen.

e) Zu Beginn sind keine Pakete in der Lagerhalle vorrätig und das Tor, der

Wareneingang und die Warenausgabe sind frei.

# Realisierung

## Allgemein

Der Code wurde auf Manjaro mit KDE Plasma getestet.

# C-Programm

## Compilieren

Der Code kann mit dem gcc-Compiler compiliert werden. Dazu gibt man folgenden Befehl ein,

wenn das .c-File mutex.c heißt:

$ gcc -pthread -o mutex mutex.c

jedoch muss je nach Linux-Art noch gcc nachinstalliert werden:

$ sudo apt install build-essential

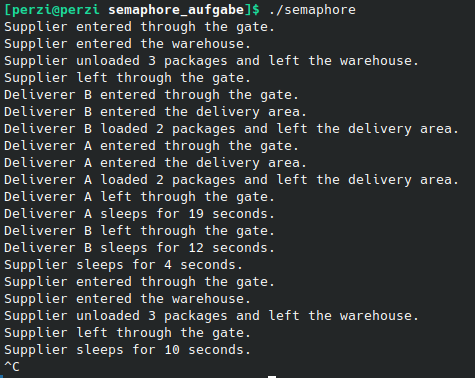
## Ausführen

Anschließend kann man dieses mit

$ ./mutex

ausführen.

# Ergebnis



# Code

## Beschreibung

Der "Supplier" (Lieferant): Dieser Thread repräsentiert den Lieferanten, der das Lagerhaus betritt und Waren abliefert. Der Lieferant verwendet die Semaphore "gate" und "packages", um sicherzustellen, dass er das Lagerhaus betreten und genügend Platz zum Abstellen der Waren hat. Zunächst wartet der Lieferant darauf, dass die Semaphore "gate" freigegeben wird, damit er das Lagerhaus betreten kann. Sobald er eingetreten ist, gibt er die Semaphore "gate" wieder frei. Anschließend lädt der Lieferant drei Pakete ab, indem er die Semaphore "packages" dreimal freigibt. Nachdem die Waren abgeliefert wurden, wartet der Lieferant wieder darauf, dass die Semaphore "gate" freigegeben wird, damit er das Lagerhaus verlassen kann. Sobald er das Lagerhaus verlassen hat, gibt er die Semaphore "gate" wieder frei und schläft dann für eine zufällige Zeit, bevor er erneut Waren abliefert.

Der "Deliverer A" (Auslieferer A): Dieser Thread repräsentiert den ersten Auslieferer, der Waren aus dem Lagerhaus abholt. Der Auslieferer A verwendet die Semaphore "gate", "delivery" und "packages", um sicherzustellen, dass er das Lagerhaus betreten, Waren abholen und genügend Platz zum Beladen seines Fahrzeugs haben kann. Zunächst wartet der Auslieferer A darauf, dass die Semaphore "gate" freigegeben wird, damit er das Lagerhaus betreten kann. Sobald er eingetreten ist, gibt er die Semaphore "gate" wieder frei und wartet dann darauf, dass die Semaphore "delivery" freigegeben wird, damit er in den Bereich gelangen kann, in dem er Waren abholen kann. Sobald er in diesem Bereich ist, gibt er die Semaphore "packages" zweimal frei, um zwei Pakete aufzunehmen, und gibt dann die Semaphore "delivery" frei, um den Bereich zu verlassen. Anschließend wartet der Auslieferer A wieder darauf, dass die Semaphore "gate" freigegeben wird, damit er das Lagerhaus verlassen kann. Sobald er das Lagerhaus verlassen hat, gibt er die Semaphore "gate" wieder frei und schläft dann für eine zufällige Zeit, bevor er erneut Waren ausliefert.

Der "Deliverer B" (Auslieferer B): Dieser Thread repräsentiert den zweiten Auslieferer, der ähnlich wie Auslieferer A funktioniert, aber nur ein Paket aufnehmen und ausliefern kann.

## Code

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#include <stdlib.h>

// gcc -pthread -o semaphore semaphore.c

//./semaphore

sem\_t gate; // Semaphore for the gate with start value 1

sem\_t delivery; // Semaphore for the delivery area with start value 1

sem\_t packages; // Semaphore for counting used space with start value 0

void \*supplier(void \*arg) {

while (1) {

sem\_wait(&gate);

printf("Supplier entered through the gate.\n");

sem\_post(&gate);

printf("Supplier entered the warehouse.\n");

sem\_post(&packages);

sem\_post(&packages);

sem\_post(&packages);

printf("Supplier unloaded 3 packages and left the warehouse.\n");

sem\_wait(&gate);

printf("Supplier left through the gate.\n");

sem\_post(&gate);

//sleep for 3 to 20 seconds

int i = rand() % 18 + 3;

printf("Supplier sleeps for %d seconds.\n", i);

sleep(i);

}

}

void \*deliverer\_a(void \*arg) {

while (1) {

sem\_wait(&gate);

printf("Deliverer A entered through the gate.\n");

sem\_post(&gate);

sem\_wait(&delivery);

printf("Deliverer A entered the delivery area.\n");

sem\_post(&packages);

sem\_post(&packages);

sem\_post(&delivery);

printf("Deliverer A loaded 2 packages and left the delivery area.\n");

sem\_wait(&gate);

printf("Deliverer A left through the gate.\n");

sem\_post(&gate);

//sleep for 3 to 20 seconds

int i = rand() % 18 + 3;

printf("Deliverer A sleeps for %d seconds.\n", i);

sleep(i);

}

}

void \*deliverer\_b(void \*arg) {

while (1) {

sem\_wait(&gate);

printf("Deliverer B entered through the gate.\n");

sem\_post(&gate);

sem\_wait(&delivery);

printf("Deliverer B entered the delivery area.\n");

sem\_post(&packages);

sem\_post(&delivery);

printf("Deliverer B loaded 2 packages and left the delivery area.\n");

sem\_wait(&gate);

printf("Deliverer B left through the gate.\n");

sem\_post(&gate);

//sleep for 3 to 20 seconds

int i = rand() % 18 + 3;

printf("Deliverer B sleeps for %d seconds.\n", i);

sleep(i);

}

}

int main() {

// Initialize the semaphores

sem\_init(&gate, 0, 1);

sem\_init(&delivery, 0, 1);

sem\_init(&packages, 0, 10);

// Create the threads

pthread\_t supplier\_thread, deliverer\_a\_thread, deliverer\_b\_thread;

pthread\_create(&supplier\_thread, NULL, supplier, NULL);

pthread\_create(&deliverer\_a\_thread, NULL, deliverer\_a, NULL);

pthread\_create(&deliverer\_b\_thread, NULL, deliverer\_b, NULL);

// Wait for the threads to finish

pthread\_join(supplier\_thread, NULL);

pthread\_join(deliverer\_a\_thread, NULL);

pthread\_join(deliverer\_b\_thread, NULL);

// Destroy the semaphores

sem\_destroy(&gate);

sem\_destroy(&delivery);

sem\_destroy(&packages);

return 0;

}