**HÖHERE TECHNISCHE BUNDESLEHRANSTALT HOLLABRUNN**

Höhere Abteilung für Elektronik – Technische Informatik

|  |  |  |
| --- | --- | --- |
| Klasse/ Jahrgang:  5BHEL | Aufgabe:  5 | Lehrer:  Reisinger & Stoll |
| Aufgabentitel  The dining savages problem | Teilnehmer:  Altenburger, Oberhamberger | |
| Datum der Abgabe:  10.1.2017 | Schriftführer:  Altenburger | Unterschrift: |

|  |  |  |
| --- | --- | --- |
|  | | Beurteilung |
| Aufgabenstellung |  |  |
| Dokumentation |  |  |
| Messschaltungen |  |  |
| Messtabellen |  |  |
| Berechnungen |  |  |
| Programmlistings |  |  |
| Auswertung |  |  |
| Diagramme |  |  |
| Berechnungen |  |  |
| Simulationen |  |  |
| Schlußfolgerungen |  |  |
| Kommentare |  |  |
| Inventarliste |  |  |
| Messprotokoll |  |  |
| Form |  |  |
| Summe | |  |

# Inhaltsverzeichnis

Inhaltsverzeichnis 2

**1.** **Aufgabenstellung** 3

**2.** **Ausprogrammierter Code** 4

**3.** **Bild der Funktion** 6

**4.** **Github** 6

# **Aufgabenstellung**

Task 5: **The dining savages problem**

A tribe of savages eats communal dinner from a large pot that can hold M servings of stewed missionary. When a savage wants to eat, he helps himself from the pot, unless it is empty. I the pot is empty, the savage wakes up the cook and then waits until the cook has refilled the pot.

Any number of savage threads run the following code:

1. Unsynchronized savage code

While True:

getServingFromPot()

eat()

And one cook thread runs this code

1. Unsynchronized cook code

While True:

Put ServingsInPot(M)

The synchronization constraints are:

.) Savages cannot invoke getServingFromPot if the pot is empty

.) The cook can invoke putServingsInPot if the pot is empty

# **Ausprogrammierter Code**

//Altenburger, Oberhamberger

#include <stdio.h>

#include <unistd.h>

#include <pthread.h>

#include <semaphore.h>

#define NUM\_SAVAGES 3 //3 Savages

sem\_t emptyPot**;** //create the semaphore emptyPot

sem\_t fullPot**;** //&fullPot

void **\***savage **(**void**\*);**

void **\***cook **(**void**\*);**

static pthread\_mutex\_t servings\_mutex**;** //create the mutex servings\_mutex&

static pthread\_mutex\_t print\_mutex**;** //&print\_mutex

static int servings **=** 15**;** // create meal-counter

int getServingsFromPot**(**void**)** //get servings from Pot

**{**

int retVal**;**

**if** **(**servings **<=** 0**)**

**{**

sem\_post **(&**emptyPot**);** //In case servings run low -> unlock the semaphore

retVal **=** 0**;**

**}**

**else** //Else -> decrement servings

**{**

servings**--;**

retVal **=** 1**;**

**}**

pthread\_mutex\_unlock **(&**servings\_mutex**);**

//retVal = 0 if no servings are left

**return** retVal**;** //retVal = 1 if servings are left

**}**

void putServingsInPot **(**int num**)** //fill the pot

**{**

servings **+=** num**;**

sem\_post **(&**fullPot**);**

**}**

void **\***cook **(**void **\***id**)** //cooker, refill the pot

**{**

int cook\_id **=** **\*(**int **\*)**id**;**

int meals **=** 2**;**

int i**;**

**while** **(** meals **)**

**{**

sem\_wait **(&**emptyPot**);** //decrements (lock) the semaphore

putServingsInPot **(**15**);** //fill the pot

meals**--;**

pthread\_mutex\_lock **(&**print\_mutex**);** //printing on the screen must be locked by a mutex

printf **(**"\nCook filled pot\n\n"**);**

pthread\_mutex\_unlock **(&**print\_mutex**);** //unlock it afterwards

**for** **(**i**=**0**;** i**<**NUM\_SAVAGES**;** i**++)**

sem\_post **(&**fullPot**);**

**}**

**return** **NULL;**

**}**

void **\***savage **(**void **\***id**)** //savage

**{**

int savage\_id **=** **\*(**int **\*)**id**;**

int myServing**;**

int meals **=** 11**;**

**while** **(** meals **)**

**{**

//find out, if no servings are left

pthread\_mutex\_lock **(&**servings\_mutex**);**

myServing **=** getServingsFromPot**();**

**if** **(**servings **==** 0**)** //if yes -> decrements (lock) the semaphore

**{**

sem\_wait **(&**fullPot**);**

myServing **=** getServingsFromPot**();**

**}**

pthread\_mutex\_unlock **(&**servings\_mutex**);**

meals**--;**

pthread\_mutex\_lock **(&**print\_mutex**);** //printing on the screen must be locked by a mutex

printf **(**"Savage: %i is eating\n"**,** savage\_id**);**

pthread\_mutex\_unlock **(&**print\_mutex**);** //unlock it afterwards

sleep**(**2**);**

pthread\_mutex\_lock **(&**print\_mutex**);** //printing on the screen must be locked by a mutex

printf **(**"Savage: %i is DONE eating\n"**,** savage\_id**);**

pthread\_mutex\_unlock **(&**print\_mutex**);** //unlock it afterwards

**}**

**return** **NULL;**

**}**

int main**()**

**{**

int i**,** id**[**NUM\_SAVAGES**+**1**];**

pthread\_t tid**[**NUM\_SAVAGES**+**1**];**

pthread\_mutex\_init**(&**servings\_mutex**,** **NULL);** // Initialize the mutex locks

pthread\_mutex\_init**(&**print\_mutex**,** **NULL);**

sem\_init**(&**emptyPot**,** 0**,** 0**);** // Initialize the semaphores

sem\_init**(&**fullPot**,** 0**,** 0**);**

**for** **(**i**=**0**;** i**<**NUM\_SAVAGES**;** i**++)** //Create an amount of NUM\_SAVAGES of savages (pthread)

**{**

id**[**i**]** **=** i**;**

pthread\_create **(&**tid**[**i**],** **NULL,** savage**,** **(**void **\*)&**id**[**i**]);**

**}**

pthread\_create **(&**tid**[**i**],** **NULL,** cook**,** **(**void **\*)&**id**[**i**]);** //Create cooker

**for** **(**i**=**0**;** i**<**NUM\_SAVAGES**;** i**++)**

**{**

pthread\_join**(**tid**[**i**],** **NULL);**

**}**

**}**

# **Bild der Funktion**

# **Github**

<https://github.com/Bernhard97/Thediningsavagesproblem>

