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		

Inhaltsangabe

Inhaltsangabe 2
1. Aufgabenstellung 3
2. Ausprogrammierter Code 4
3. Bild der Funktion 6

1. 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

```
(2) Unsynchronized cook code
        While True:
                Put ServingsInPot(M)
```

The synchronization constraints are:

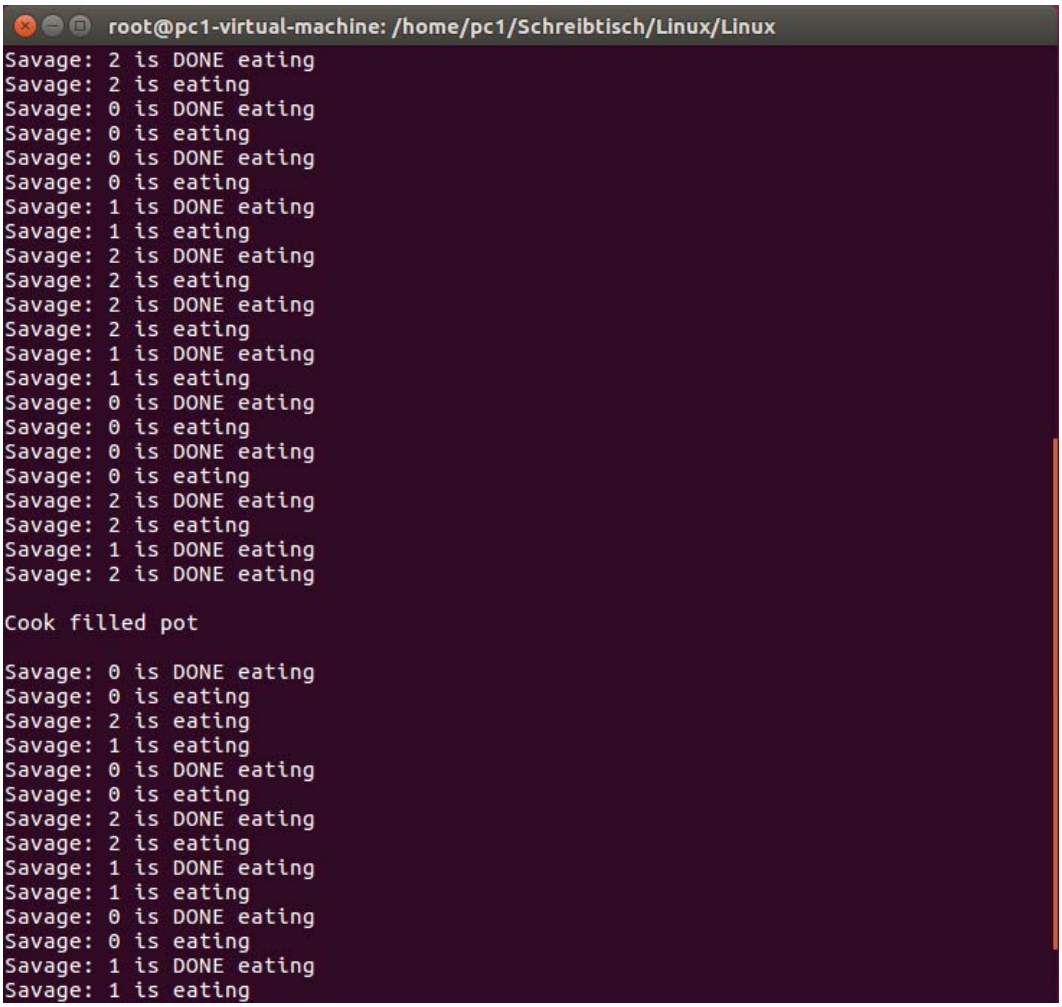
.) Savages cannot invoke getServingFromPot if the pot is empty
.) The cook can invoke putServingsInPot if the pot is empty

2. 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 sema-
phore
        {
            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);
    }
}
```

3. Bild der Funktion



4. Github

https://github.com/Bernhard97/Thediningsavagesproblem