**Praktikum 2 – Betriebssysteme**

**Marvin Kiehl**

**Niklas Debbrecht**

#include <pthread.h>

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include "msocket.h"

#include <string.h>

#include <time.h>

#define CONSUMER 3

#define QUEUESIZE 5

**typedef** **struct** queue queue;

**typedef** **struct** thread thread;

**typedef** **struct** arguments arguments;

**typedef** **enum** {false,true} boolean;

**void**\* producer (**void**\* args);

**void**\* consumer (**void**\* args);

queue\* queueInit (**void**);

arguments\* argsInit();

**void** queueDelete (queue\* q);

**void** queueAdd (queue\* q, **char**\* line);

**char**\* queueDel (queue\* q, **char**\* line);

**void** createFileName(**char**\* saveData, **int** id, **int** fileNum);

**struct** queue

{

**char**\* websites[QUEUESIZE];

**int** head;

**int** tail;

boolean end;

boolean empty;

boolean full;

pthread\_mutex\_t\* mut;

pthread\_cond\_t\* notFull;

pthread\_cond\_t\* notEmpty;

};

**struct** thread

{

pthread\_t reader;

**int** id;

};

**struct** arguments

{

queue\* qu;

thread\* th;

**char**\* file;

};

**int** main(**int** argc, **char** \*\*argv)

{

**int** i;

queue\* fifo;

pthread\_attr\_t attr;

thread pro;

arguments\* args;

args = argsInit();

thread consumers[CONSUMER];

//Eindeutige ID zuweisen

**for**(i = 0; i<CONSUMER; i++)

{

consumers[i].id = i+1;

}

/\*\*QUEUE INIT\*\*/

fifo = queueInit();

**if** (fifo == NULL)

{

fprintf (stderr, "main: Queue Init failed.\n");

exit (1);

}

args->file = argv[1];

args->qu = fifo;

/\*\*CREATE THREADS\*\*/

//Producer

clock\_t start = clock();

**if**(pthread\_create (&(pro.reader), NULL, producer, args)!=0)

{

fprintf (stderr, "Can´t create Producerthread.\n");

exit (1);

}

//Consumer

**for**(i = 0; i<CONSUMER; i++)

{

args->th = &consumers[i];

**if**(pthread\_create (&(args->th->reader), NULL, consumer, args)!=0)

{

fprintf (stderr, "Can´t create Consumerthread.\n");

exit (1);

}

}

/\*\*JOIN THREADS\*\*/

**for**(i = 0; i<CONSUMER; i++)

{

args->th = &consumers[i];

pthread\_join (args->th->reader, NULL);

}

pthread\_join (pro.reader, NULL);

/\*\*ENDE\*\*/

clock\_t end = clock();

queueDelete (fifo);

free(args);

**double** time = ((end-start)\*1000)/CLOCKS\_PER\_SEC;

printf("\n%.2fms\n\n", time);

**return** 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*DEFINITION\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

queue\* queueInit (**void**)

{

queue\* q;

//Speicher fuer Queue, und auf erfolg pruefen

q = (queue\*)malloc (**sizeof** (queue));

**if** (q == NULL)

{

**return** (NULL);

}

//Werte vorbelegen

q->empty = true;

q->full = false;

q->end = false;

q->tail = 0;

q->head = 0;

//Mutex erstellen

q->mut = (pthread\_mutex\_t \*) malloc (**sizeof** (pthread\_mutex\_t));

pthread\_mutex\_init (q->mut, NULL);

//Bedingungsvariablen

q->notFull = (pthread\_cond\_t \*) malloc (**sizeof** (pthread\_cond\_t));

pthread\_cond\_init (q->notFull, NULL);

q->notEmpty = (pthread\_cond\_t \*) malloc (**sizeof** (pthread\_cond\_t));

pthread\_cond\_init (q->notEmpty, NULL);

**return** (q);

}

arguments\* argsInit()

{

arguments\* q;

q = (arguments\*)malloc (**sizeof** (arguments));

**if** (q == NULL)

{

**return** (NULL);

}

//q->file;

**return**(q);

}

**void**\* producer (**void**\* q)

{

arguments\* args;

args = (arguments\*) q;

queue\* fifo = args->qu;

**char** line[2048];

**int** fileNum = 1;

pthread\_mutex\_lock(args->qu->mut);

**char**\* linePtr = args->file;

FILE\* file;

**if**(fopen(linePtr,"r"))

{

file = fopen(linePtr,"r");

}

**else**

{

printf("Die Datei ist nicht vorhanden!\n(oder sie ist nicht als Argument '<file>.<format>' angeben)\n");

exit(-1);

}

pthread\_mutex\_unlock(fifo->mut);

**while**(fgets(line, **sizeof**(line), file))

{

pthread\_mutex\_lock (fifo->mut);

**while** (fifo->full)

{

pthread\_cond\_wait (fifo->notFull, fifo->mut);

}

sprintf(line,"%s|%d",line, fileNum++);

queueAdd (fifo, line);

printf("\n\nPRO: %s\n\n", fifo->websites[fifo->head]);

pthread\_mutex\_unlock (fifo->mut);

pthread\_cond\_signal (fifo->notEmpty);

}

fifo->end=true;

printf("Ich bin fertig PRODUCER\n");

fclose(file);

pthread\_cond\_broadcast (fifo->notEmpty);

**return** (NULL);

}

**void**\* consumer (**void**\* q)

{

arguments\* args;

args = (arguments\*) q;

queue\* fifo = args->qu;

**char** lineArr[1024];

**char**\* line = lineArr;

**char**\* addressPointer;

**char**\* pagePointer;

**char** filename[1024];

**char**\* filenamePointer = filename;

**char**\* saveptr; //für strtok\_r

**int** id = args->th->id;

**int** fileNum;

**while**(true)

{

**if**(fifo->empty && fifo->end)

{

**break**;

}

pthread\_mutex\_lock (fifo->mut);

**while**(fifo->empty && !fifo->end)

{

pthread\_cond\_wait(fifo->notEmpty, fifo->mut);

}

**if**(!fifo->empty)

{

line = queueDel(fifo, line);

line = strtok\_r(line, "|", &saveptr);

fileNum = atoi(strtok\_r(NULL, "\0", &saveptr));

addressPointer = strtok\_r(line, " ", &saveptr);

pagePointer = strtok\_r(NULL, "\n", &saveptr);

createFileName(filenamePointer, id, fileNum);

printf("CONSUMER:%d\nFIFO:%s %s\nSEITE:%s\nUNTERSEITE:%s\nNUMMER:%d\n\n\n",id ,addressPointer, pagePointer, addressPointer, pagePointer, fileNum);

}

pthread\_mutex\_unlock (fifo->mut);

pthread\_cond\_signal (fifo->notFull);

askServer(addressPointer, pagePointer, filenamePointer);

}

printf("Ich bin fertig CONSUMER\n");

**return** (NULL);

}

/\*\*setzt namen zsm\*\*/

**void** createFileName(**char**\* filename, **int** id, **int** fileNum)

{

**char** buffer[1024];

sprintf(buffer,"%d",id);

strcpy(filename,"index\_");

strcat(filename, buffer);

strcat(filename, "\_");

sprintf(buffer,"%d", fileNum);

strcat(filename, buffer);

strcat(filename, ".html");

}

/\*\*loescht queue\*\*/

**void** queueDelete (queue\* q)

{

pthread\_mutex\_destroy (q->mut);

free (q->mut);

pthread\_cond\_destroy (q->notFull);

free (q->notFull);

pthread\_cond\_destroy (q->notEmpty);

free (q->notEmpty);

free (q);

}

/\*\*fuegt char\* in queue\*\*/

**void** queueAdd (queue\* q, **char**\* line)

{

**char**\* newString = (**char**\*)malloc (1024);

strcpy(newString, line);

q->head++;

**if** (q->head >= QUEUESIZE)

{

q->head = 0;

}

q->websites[q->head] = newString;

**if** (q->head == q->tail)

{

q->full = true;

}

q->empty = false;

}

/\*\*veraendert tail (loescht)\*\*/

**char**\* queueDel(queue\* q, **char**\* line)

{

q->tail++;

**if**(q->tail >= QUEUESIZE)

{

q->tail = 0;

}

line = q->websites[q->tail];

**if** (q->tail == q->head)

{

q->empty = true;

}

q->full = false;

**return** line;

}