VSCODE_PRINT_SCRIPT_TAGS

Selected files

9 printable files

Assignments\Assignment5\makefile
Assignments\Assignment5\Part_A.c
Assignments\Assignment5\Part_A.h
Assignments\Assignment5\Part_C.c
Assignments\Assignment5\Part_C.h
Assignments\Assignment5\queue.c
Assignments\Assignment5\queue.h
Assignments\Assignment5\st_pipeline.c
Assignments\Assignment5\st_pipeline.h

Assignments\Assignment5\makefile

```

CFLAGS = -Wall -Werror
all: st_pipeline


Part_A.o: Part_A.c Part_A.h
    gcc -c $(CFLAGS) Part_A.c -o Part_A.o -lm

queue.o: queue.c queue.h
    gcc -c $(CFLAGS) queue.c -o queue.o

Part_C.o: Part_C.c Part_C.h queue.o
    gcc -c $(CFLAGS) Part_C.c -o Part_C.o

st_pipeline.o: st_pipeline.c Part_C.o queue.o
    gcc -c $(CFLAGS) st_pipeline.c -o st_pipeline.o

st_pipeline: st_pipeline.o Part_A.o
    gcc st_pipeline.o Part_A.o Part_C.o queue.o -o st_pipeline -lm


clean:
    rm -f *.o st_pipeline
.PHONY: all clean
```

Assignments\Assignment5\Part_A.c

```c
// from gpt, request: write me a function in C that gets Unsigned int and checks if it's a prime number.

#include "Part_A.h"

int isPrime(unsigned int num) {
    if (num < 2) {
        return 0;
    }

    unsigned int limit = (unsigned int) sqrt(num);

    for (unsigned int i = 2; i <= limit; i++) {
        if (num % i == 0) {
            return 0;
        }
    }

    return 1;
}

```

Assignments\Assignment5\Part_A.h

```c
#ifndef ASSIGNMENT_5_PART_A_H
#define ASSIGNMENT_5_PART_A_H
#include <math.h>
int isPrime(unsigned int num);
#endif //ASSIGNMENT_5_PART_A_H

```

Assignments\Assignment5\Part_C.c

```c
#include "Part_C.h"

Queue * getQueue(struct AO * this){
    return this->queue;
}
void stop(struct AO this){
    pthread_cancel(this.thread); // ask the thread to stop
    pthread_join(this.thread, NULL); // wait until the thread will stop
}

void cleanupHandler(void * vao) {
    // Cleanup code here
    AO * ao = (AO*)vao;
    destroyQueue(ao->queue);
    free(ao->queue);
    free(ao);
}

void threadFunction(void * p2) {
    pparam p = (pparam) p2;

    pthread_cleanup_push(cleanupHandler, p->this) ;


            while (1) {
                // check for cancel
                pthread_testcancel();

                void *v = dequeue(p->this->queue);
                int *pnum = (int *) v;
                p->this->func(*pnum, p);
            }
    pthread_cleanup_pop(1);
}

AO *CreateActiveObject(void (*func)(int,pparam),pparam p) {
    AO *ao = malloc(sizeof(AO));
    Queue * q = malloc(sizeof(Queue));
    initializeQueue(q);
    ao->queue = q;
    ao->func = func;
    ao->getQueue = getQueue;
    ao->stop=stop;
    p->this = ao;
    pthread_create(&(ao->thread), NULL, (void *(*)(void *)) threadFunction, p);

    return ao;
}
```

Assignments\Assignment5\Part_C.h

```c
#ifndef ASSIGNMENT_5_PART_C_H
#define ASSIGNMENT_5_PART_C_H

#include "queue.h"

struct AO;
typedef struct param{
    struct AO* this;
    struct AO * next;
    int N;
    int seed;
    int * flag;

} param,*pparam;

typedef struct AO{
    pthread_t thread;
    Queue * queue;
    void (*func)(int,pparam);
    Queue* (*getQueue)(struct AO * this);
    void (*stop)(struct AO this);

}AO;


void threadFunction(void *);
AO *CreateActiveObject(void (*func)(int, pparam),pparam p);
void cleanupHandler(void * vao);


#endif //ASSIGNMENT_5_PART_C_H

```

Assignments\Assignment5\queue.c

```c
// from gpt. request 1: write for me a threads safe queue in C. the queue should hold void*.
//           request 2: now i want u to improve  this queue: the new queue will use cond to
//                     let the threads that try to dequeue not wait on busy loop( if the queue
//                     is empty or some other thread working on the queue, the thread should
//                     sleep until it can dequeue)
//          request 3: give me the header of this file.
#include "queue.h"


void initializeQueue(Queue* queue) {
    queue->isEmpty = isEmpty;
    queue->enqueue = enqueue;
    queue->dequeue = dequeue;
    queue->destroyQueue = destroyQueue;
    queue->front = NULL;
    queue->rear = NULL;
    pthread_mutex_init(&queue->mutex, NULL);
    pthread_cond_init(&queue->cond, NULL);
}

bool isEmpty(Queue* queue) {
    return queue->front == NULL;
}

void enqueue(Queue* queue, void* data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;

    pthread_mutex_lock(&queue->mutex);

    if (isEmpty(queue)) {
        queue->front = newNode;
        queue->rear = newNode;
    } else {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }

    pthread_cond_signal(&queue->cond);
    pthread_mutex_unlock(&queue->mutex);
}

void* dequeue(Queue* queue) {
    pthread_mutex_lock(&queue->mutex);

    while (isEmpty(queue)) {
        pthread_cond_wait(&queue->cond, &queue->mutex);
    }
    int i = 0;
    Node * temp = queue->front;
    while (temp!= NULL){
        temp = temp->next;
        i++;
    }

    Node* frontNode = queue->front;
    void* data = frontNode->data;

    queue->front = queue->front->next;
    free(frontNode);

    if (queue->front == NULL) {
        queue->rear = NULL;
    }

    pthread_mutex_unlock(&queue->mutex);

    return data;
}

void destroyQueue(Queue* queue) {
    while (!isEmpty(queue)) {
        dequeue(queue);
    }

    pthread_cond_destroy(&queue->cond);
    pthread_mutex_destroy(&queue->mutex);
}
void printQueue(Queue * queue){
    int i = 0;
    Node * temp = queue->front;
    while (temp!= NULL){
        if(temp->data == NULL){
            printf("data of %d: NULL\n",i);
        }
        else {
            printf("data of %d:\n", i);
        }
        temp = temp->next;
        i++;
    }
    printf("size: %d\n",i);

}

```

Assignments\Assignment5\queue.h

```c
#ifndef ASSIGNMENT_5_QUEUE_H
#define ASSIGNMENT_5_QUEUE_H

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

typedef struct Node {
    void* data;
    struct Node* next;
} Node;

typedef struct Queue {
    Node* front;
    Node* rear;
    pthread_mutex_t mutex;
    pthread_cond_t cond;
    bool (*isEmpty)(struct Queue* queue);
    void (*enqueue)(struct Queue* queue, void* data);
    void* (*dequeue)(struct Queue* queue);
    void (*destroyQueue)(struct Queue* queue);
} Queue;

void initializeQueue(Queue* queue);
bool isEmpty(Queue* queue);
void enqueue(Queue* queue, void* data);
void* dequeue(Queue* queue);
void destroyQueue(Queue* queue);
void printQueue(Queue * queue);

#endif //ASSIGNMENT_5_QUEUE_H

```

Assignments\Assignment5\st_pipeline.c

```
#include "st_pipeline.h"

int generateRandomNumber() {

    int randomNum = 0;
    int min = 100000;  // Minimum value for a 6-digit number
    int max = 999999;  // Maximum value for a 6-digit number

    // Generate random number within the desired range
    randomNum = (rand() % (max - min + 1)) + min;

    return randomNum;
}

void func4(int num, pparam p) {
    printf("%d\n", num);
    num += 2;
    printf("%d\n", num);
    *(p->flag) = *(p->flag) - 1;
}

void func3(int num, pparam p) {
    printf("%d\n", num);
    if (num < 0) // not necessary
        num = -num;
    unsigned int u_num = (unsigned int) num;
    if (isPrime(u_num)) {
        printf("true\n");
    } else {
        printf("false\n");
    }
    num = num - 13;

    p->next->queue->enqueue(p->next->queue, &num);


}

```

```c
void func2(int num, pparam p) {
    printf("%d\n", num);
    if (num < 0) // not necessary
        num = -num;
    unsigned int u_num = (unsigned int) num;
    if (isPrime(u_num)) {
        printf("true\n");
    } else {
        printf("false\n");
    }
    num = num + 11;

    p->next->queue->enqueue(p->next->queue, &num);

}

void func1(int num, pparam p) {
    srand(p->seed);  // Set the seed for random number generation
    Queue *next_q = p->next->getQueue(p->next);
    for (int i = 0; i < p->N; ++i) {
        int rand = generateRandomNumber();
        next_q->enqueue(next_q, (void *) &rand);
        usleep(1000);
    }
}

int main(int argc, char *argv[]) {
    int seed = 0;
    if (argc != 2 && argc != 3) {
        return 1;
    } else if (argc == 2) {
        seed = time(NULL);
    } else {
        seed = atoi(argv[2]);
    }
    int N = atoi(argv[1]);


    int left = N;
    // create th active objects
    pparam p4 = malloc(sizeof(param));
    p4->this = NULL;
    p4->flag = &left;
    p4->next = NULL;
    p4->seed = seed;
    p4->N = N;
    AO *ao4 = CreateActiveObject(func4, p4);

//    param p3 = {NULL, ao4, N, seed, &left};
    pparam p3 = malloc(sizeof(param));
    p3->this = NULL;
    p3->flag = &left;
    p3->next = ao4;
    p3->seed = seed;
    p3->N = N;
    AO *ao3 = CreateActiveObject(func3, p3);

    pparam p2 = malloc(sizeof(param));
    p2->this = NULL;
    p2->flag = &left;
    p2->next = ao3;
    p2->seed = seed;
    p2->N = N;
    AO *ao2 = CreateActiveObject(func2, p2);

    pparam p1 = malloc(sizeof(param));
    p1->this = NULL;
    p1->flag = &left;
    p1->next = ao2;
    p1->seed = seed;
    p1->N = N;
    AO *ao1 = CreateActiveObject(func1, p1);
    usleep(1000000);
    ao1->getQueue(ao1)->enqueue(ao1->getQueue(ao1), &left); //enqueue some data so the first AO will start.

    while (left > 0) {
        usleep(1000);
    }
    // kill threads
    ao1->stop(*ao1);
    ao2->stop(*ao2);
    ao3->stop(*ao3);
    ao4->stop(*ao4);
    free(p1);
    free(p2);
    free(p3);
    free(p4);

    return 0;
}
```

Assignments\Assignment5\st_pipeline.h

```c
#ifndef ASSIGNMENT_5_ST_PIPELINE_H
#define ASSIGNMENT_5_ST_PIPELINE_H

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include "Part_C.h"
#include "Part_A.h"
#include <unistd.h>

#define TRUE 1

int generateRandomNumber();
void func4(int num,pparam p);
void func3(int num,pparam p);
void func2(int num,pparam p);
void func1(int num,pparam p);

#endif //ASSIGNMENT_5_ST_PIPELINE_H
```