Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	WRIGHT	Student ID:	19779085
Other name(s):	JONATHAN		
Unit name:	OPERATING SYSTEMS	Unit ID:	COMP2006
Lecturer / unit coordinator:	SIE TENG SOH	Tutor:	SIE TENG SOH
Date of submission:	19/04/2020	Which assignment?	(Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature:  Date of signature: 19/04/2020

(By submitting this form, you indicate that you agree with all the above text.)

Table Of Contents

README

WHAT IS IT?: A Lift Simulator written in C
GOAL: I chose to try and make this assignment without memory leaks and global variables I think I succeeded?

Dependencies:
make (Required to easily compile, but you dont 'need' it if you know all the flags i've created.)
gcc (Required to compile the code, you can also use clang but I didn't test for clang.)
c (c89 Unix)
unix (built for unix environment ((use of pthreads + system v)))

OPTIONAL FLAGS:
DEBUG (Will output additional things to terminal to help with debugging.)
SANITIZETHREAD (Will add fsanitize=thread to the compiler flags)
NOTSLEEP (Will disable sleeping.)
OUTSIMASSTDOUT (Will output all things that are meant to be sent to out_sim to the standard output instead)
PROCESS (Will select to use the process implementation) <- DO NOT DEFINE BOTH OF THESE ONLY ONE
PTHREAD (Will select to use the pthread implementation, by default this is selected) <--|

How To Run:
1) cd into the directory containing code
2) (make PROCESS=1) OR (make PTHREAD=1) OR make (will select PTHREAD by default.)
3) Make sure you have a sim_input file. (siminputgenerator.java is included to help if you dont know how,
however the file you place must be called sim_input)
3) ./lift_sim_A m t (Where m is buffer size ((INTEGER)) and t is lift time ((INTEGER)))
NOTE: you can estimate the time it will take by doing the formula (t (liftTime) * siminputSize), this will
give you a upperbound estimate (It will not take longer than this.
4) Once done you can read the out_sim file to see the results.

Files:
- lifts.c
  Purpose: Provides Implementation of how the lifts will work.
- lifts.h
  Purpose: Headerfile for lifts.c
- list.c
  Purpose: Double Ended Doubly Linked list made for UCP 2019 Assignment Semester 2, Modified again by me.
- list.h
  Purpose: List.c headerfile.
- Makefile
  Purpose: Used by make for easy compilimation
- processLift.h
  Purpose: Provides the arrayQueue ADT and processLift struct
- program.c
  Purpose: Start and End point for program.
- program.h
  Purpose: Program headerfile
- queue.c
  Purpose: Contains list based queue ADT
- queue.h
  Purpose: queue.c headerfile
- request.c
  Purpose: contains request struct and the pthread/process implementation for polling requests from file into buffer.
- request.h
  Purpose: headerfile for request.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <time.h>
#include "queue.h"
#include "lifts.h"
#include "request.h"

/*****************************************************************************
*  PURPOSE OF C FILE: Provides Implementation of how the lifts will work.
*  DATE: 17/04/2020 - 9:20PM
*  AUTHOR: Jonathan Wright
*****************************************************************************/
/*****************************************************************************
*  PURPOSE: This method is the PThread function for the actual lifts going up/down,
*  it takes in a struct which points at memory such as mutexLock which allows it
*  to avoid race conditions.
*  IMPORTS: liftStruct* lift (fed in as a void* due to being a pthread.)
*  EXPORTS: NULL, (the lift's actual returns (request # and movement is done
*  in a array contained in the struct.))
*  DATE: 17/04/2020 - 9:20PM
*  AUTHOR: Jonathan Wright
*****************************************************************************/
void* lift(void* args) {
    void* currentRequest;
    request* tempRequest;
    int localFinishRead, localCount, sleepNeeded, distance;
    #ifndef NOTSLEEP
    int localSleepTimer;
    #endif
    liftStruct* thisLift = (liftStruct*)args;
    pthread_mutex_lock(thisLift->mutexLock);
    localFinishRead = *(thisLift->finishedRead);
    localCount = thisLift->buffer->list->size;
    #ifndef NOTSLEEP
    localSleepTimer = thisLift->liftTimer;
    #endif
    pthread_mutex_unlock(thisLift->mutexLock);
    sleepNeeded = FALSE;
    while (localFinishRead != TRUE || localCount != 0) {
        pthread_mutex_lock(thisLift->mutexLock);
        if (thisLift->buffer->list->size < 1 && *(thisLift->finishedRead) != TRUE) {
            #ifdef DEBUG
            printf("THREAD %ld (AKA: LIFT %d): Signaling condition empty in lifts.c (localFinishedRead = %d)\n", pthread_self(),
thisLift->liftNumber, localFinishRead);
            #endif
            pthread_cond_signal(thisLift->empty);
        } else {
            currentRequest = dequeue(thisLift->buffer);
            if (currentRequest != NULL) {
                if (thisLift->previousRequest == NULL) {
                    distance = abs(1 - ((request*)currentRequest)->requestFloor) + abs(((request*)currentRequest)->destinationFloor -
((request*)currentRequest)->requestFloor);
                    *((thisLift->liftReturnVals) + 0) += distance;
                    *((thisLift->liftReturnVals) + 1) += 1;
                    fprintf(thisLift->out_sim_file,
                        "Lift-%d Operation\nPrevious position: Floor 1 (I haven't had any requests yet.)\nRequest: Floor %d to Floor
%d\nDetail operations:\n    Go from Floor %d to Floor %d\n    Go from Floor %d to Floor %d\n    #movement for this request:
%d\n    #request: %d\n    Total #movement: %d\nCurrent position: Floor %d\n\n",
                        thisLift->liftNumber, ((request*)currentRequest)->requestFloor, ((request*)currentRequest)->destinationFloor, 1,
((request*)currentRequest)->requestFloor, ((request*)currentRequest)->requestFloor, ((request*)currentRequest)-
>destinationFloor,
                        distance, *((thisLift->liftReturnVals) + 1), *((thisLift->liftReturnVals) + 0), ((request*)currentRequest)-
>requestFloor
                    );
                    fflush(thisLift->out_sim_file);
```

```
                thisLift->previousRequest = ((request*)currentRequest);
                #ifndef NOTSLEEP
                sleepNeeded = TRUE;
                #endif
            } else {
                distance = abs(thisLift->previousRequest->destinationFloor - ((request*)currentRequest)->requestFloor) +
abs(((request*)currentRequest)->destinationFloor - ((request*)currentRequest)->requestFloor);
                *((thisLift->liftReturnVals) + 0) += distance;
                *((thisLift->liftReturnVals) + 1) += 1;
                fprintf(thisLift->out_sim_file,
                    "Lift-%d Operation\nPrevious position: Floor %d\nRequest: Floor %d to Floor %d\nDetail operations:\n    Go
from Floor %d to Floor %d\n    Go from Floor %d to Floor %d\n    #movement for this request: %d\n    #request: %d\n    Total
#movement: %d\nCurrent position: Floor %d\n\n",
                    thisLift->liftNumber, thisLift->previousRequest->destinationFloor, ((request*)currentRequest)->requestFloor,
((request*)currentRequest)->destinationFloor, thisLift->previousRequest->destinationFloor, ((request*)currentRequest)-
>requestFloor,
                    ((request*)currentRequest)->requestFloor, ((request*)currentRequest)->destinationFloor, distance, *((thisLift-
>liftReturnVals) + 1), *((thisLift->liftReturnVals) + 0), ((request*)currentRequest)->requestFloor
                );
                fflush(thisLift->out_sim_file);
                tempRequest = thisLift->previousRequest;
                thisLift->previousRequest = ((request*)currentRequest);
                free(tempRequest);
                #ifndef NOTSLEEP
                sleepNeeded = TRUE;
                #endif
            }
        } else {
            #ifdef DEBUG
            printf("\n!!!! RECIEVED A NULL ON THREAD %ld (AKA: LIFT %d)!!!!\n", pthread_self(), thisLift->liftNumber);
            #endif
            /* This occurs once at the end and doesnt happen again as it hasnt updated localFinishRead. */
        }
    }
    localFinishRead = *(thisLift->finishedRead);
    localCount = thisLift->buffer->list->size;
    pthread_mutex_unlock(thisLift->mutexLock);
    if (sleepNeeded == TRUE) {
        #ifndef NOTSLEEP
        sleepNeeded = FALSE;
        sleep(localSleepTimer);
        #endif
    }
  }
  free(thisLift->previousRequest);
  return NULL;
}

/*****************************************************************************
*  PURPOSE: This is the process implementation of the actual lifts going up and
*  down, this uses semaphores fed in through a struct.
*  IMPORTS: processLift** thisLift (fed in as void* args because I originally
*  thought that processes ran functions the same way pthreads did.)
*  EXPORTS: NULL (Exports again happen through a array contained in the struct.)
*  DATE: 17/04/2020 - 9:20PM
*  AUTHOR: Jonathan Wright
*****************************************************************************/
void liftProcess(void* args) {
  struct timespec tims;
  #ifdef DEBUG
  int tester;
  #endif
  #ifndef NOTSLEEP
  int localSleepTimer, localCount;
  #endif
  int distance, sleepNeeded;
```

```
    processLift** thisLift;
    request currentRequest;
    localCount = 0;
    thisLift = (processLift**)args;
    #ifndef NOTSLEEP
    localSleepTimer = (*thisLift)->liftTimer;
    #endif
    while (**((*thisLift)->finishedRead) == FALSE || localCount != 0) {
        #ifdef DEBUG
        sem_getvalue( *((*thisLift)->semaphoreFull), &tester);
        printf("FULL SEM VALUE BEFORE: %d\n", tester);
        #endif
        clock_gettime(CLOCK_REALTIME, &tims);
        tims.tv_sec += 1; /* after 4 seconds, check the file hasnt been finished read. */
        if (sem_timedwait(*((*thisLift)->semaphoreFull), &tims) == 0) {
            #ifdef DEBUG
            sem_getvalue( *((*thisLift)->semaphoreFull), &tester);
            printf("FULL SEM VALUE AFTER: %d\n", tester);
            #endif
            localCount =(*(*thisLift)->buffer)->size;
            if (localCount > 0) {
                currentRequest = arrayDequeue((*thisLift)->buffer);
                if (((*thisLift)->previousRequest).destinationFloor == -1) { /* Equivalent of Not Null from the thread implementation,
however, I used a stack variable so i just set it to -1. */
                    distance = abs(1 - currentRequest.requestFloor) + abs(currentRequest.destinationFloor -
currentRequest.requestFloor);
                    *((*((*thisLift)->liftReturnVals)) + 0) += distance;
                    *((*((*thisLift)->liftReturnVals)) + 1) += 1;
                    sem_wait(*((*thisLift)->requestFileSem));
                    fprintf(**((*thisLift)->out_sim_file),
                        "Lift-%d Operation\nPrevious position: Floor 1 (I haven't had any requests yet.)\nRequest: Floor %d to Floor
%d\nDetail operations:\n    Go from Floor %d to Floor %d\n    Go from Floor %d to Floor %d\n    #movement for this request:
%d\n    #request: %d\n    Total #movement: %d\nCurrent position: Floor %d\n\n",
                        (*thisLift)->liftNumber, currentRequest.requestFloor, currentRequest.destinationFloor, 1,
currentRequest.requestFloor, currentRequest.requestFloor, currentRequest.destinationFloor,
                        distance, *((*((*thisLift)->liftReturnVals)) + 1), *((*((*thisLift)->liftReturnVals)) + 0), currentRequest.requestFloor
                    );
                    fflush(**((*thisLift)->out_sim_file));
                    (*thisLift)->previousRequest = currentRequest;
                    #ifndef NOTSLEEP
                    sleepNeeded = TRUE;
                    #endif
                    sem_post(*((*thisLift)->liftZeroFileSem));
                } else {
                    distance = abs(((*thisLift)->previousRequest).destinationFloor - currentRequest.requestFloor) +
abs(currentRequest.destinationFloor - currentRequest.requestFloor);
                    *((*((*thisLift)->liftReturnVals)) + 0) += distance;
                    *((*((*thisLift)->liftReturnVals)) + 1) += 1;
                    #ifdef DEBUG
                    sem_getvalue( *((*thisLift)->requestFileSem), &tester);
                    printf("REQUEST FILE SEM VALUE BEFORE END: %d\n", tester);
                    #endif
                    sem_wait(*((*thisLift)->requestFileSem));
                    fprintf(**((*thisLift)->out_sim_file),
                        "Lift-%d Operation\nPrevious position: Floor %d\nRequest: Floor %d to Floor %d\nDetail operations:\n    Go
from Floor %d to Floor %d\n    Go from Floor %d to Floor %d\n    #movement for this request: %d\n    #request: %d\n    Total
#movement: %d\nCurrent position: Floor %d\n\n",
                        (*thisLift)->liftNumber, ((*thisLift)->previousRequest).destinationFloor, currentRequest.requestFloor,
currentRequest.destinationFloor, ((*thisLift)->previousRequest).destinationFloor, currentRequest.requestFloor,
                        currentRequest.requestFloor, currentRequest.destinationFloor, distance, *((*((*thisLift)->liftReturnVals)) + 1),
*((*((*thisLift)->liftReturnVals)) + 0), currentRequest.requestFloor
                    );
                    fflush(**((*thisLift)->out_sim_file));
                    (*thisLift)->previousRequest = currentRequest;
                    #ifndef NOTSLEEP
                    sleepNeeded = TRUE;
```

```
            #endif
            sem_post(*((*thisLift)->liftZeroFileSem));
        }
    }


    #ifdef DEBUG
    sem_getvalue( *((*thisLift)->semaphoreFull), &tester);
    printf("EMPTY SEM VALUE BEFORE: %d\n", tester);
    #endif
    sem_post(*((*thisLift)->semaphoreEmpty));
    #ifdef DEBUG
    sem_getvalue( *((*thisLift)->semaphoreFull), &tester);
    printf("EMPTY SEM VALUE AFTER: %d\n", tester);
    #endif
    if (sleepNeeded == TRUE) {
        #ifndef NOTSLEEP
        sleepNeeded = FALSE;
        sleep(localSleepTimer);
        #endif
    }
  } else {
    sem_post(*((*thisLift)->semaphoreFull));
    /* Check the initial condition again! */
  }
 }
}

/*****************************************************************************
*  PURPOSE: This initializes the struct used by lifts in the pthreads implementation
*  IMPORTS: queue* inBuffer, int inTimer, int whatLift, pthread_mutex_t* inLock
*  pthread_cond_t* inFullCond, pthread_cond_t* inEmptyCond,
*  int* inFinishedRead, int inMaxBufferSize, FILE* inFile
*  EXPORTS: liftStruct* newLiftStruct
*  DATE: 17/04/2020 - 9:20PM
*  AUTHOR: Jonathan Wright
*****************************************************************************/
liftStruct* initLiftStruct(queue* inBuffer, int inTimer, int whatLift,
pthread_mutex_t* inLock, pthread_cond_t* inFullCond, pthread_cond_t* inEmptyCond,
int* inFinishedRead, int inMaxBufferSize, FILE* inFile) {
  liftStruct* newLiftStruct = (liftStruct*)malloc(sizeof(liftStruct));
  newLiftStruct->buffer = inBuffer;
  newLiftStruct->previousRequest = NULL;
  newLiftStruct->finishedRead = inFinishedRead;
  newLiftStruct->liftTimer = inTimer;
  newLiftStruct->liftNumber = whatLift;
  newLiftStruct->mutexLock = inLock;
  newLiftStruct->full = inFullCond;
  newLiftStruct->empty = inEmptyCond;
  newLiftStruct->maxBufferSize = inMaxBufferSize;
  newLiftStruct->out_sim_file = inFile;
  newLiftStruct->liftReturnVals = (int*)malloc(sizeof(int) * 2);
  *((newLiftStruct->liftReturnVals) + 0) = 0;
  *((newLiftStruct->liftReturnVals) + 1) = 0;
  return newLiftStruct;
}

/*****************************************************************************
*  PURPOSE: Will free the memory allocated for a liftStruct.
*  IMPORTS: liftStruct* toFree
*  EXPORTS: None
*  DATE: 17/04/2020 - 9:20PM
*  AUTHOR: Jonathan Wright
*****************************************************************************/
void freeLiftStruct(liftStruct* toFree) {
  free(toFree->liftReturnVals);
  free(toFree);
```

```
}

/*****************************************************************************
*   PURPOSE: Will create a lift for the process implementation.
*   IMPORTS: arrayQueue** inBuffer, int** inFinishedRead, int inTimer,
*   int inNumber, int myCapacity, FILE*** inFile, sem_t** inFullSem, sem_t** inEmptySem,
*   sem_t** inFileSem, sem_t** inRequestFileSem, int** inReturnAddress
*   EXPORTS: processLift* myLift
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
processLift* createProcessLift(arrayQueue** inBuffer, int** inFinishedRead, int inTimer,
int inNumber, int myCapacity, FILE*** inFile, sem_t** inFullSem, sem_t** inEmptySem,
sem_t** inFileSem, sem_t** inRequestFileSem, int** inReturnAddress) {
    request nullValue;
    processLift* myLift = (processLift*)malloc(sizeof(processLift));
    nullValue.requestFloor = -1; nullValue.destinationFloor = -1;
    myLift->buffer = inBuffer;
    myLift->previousRequest = nullValue;
    myLift->finishedRead = inFinishedRead;
    myLift->liftReturnVals = inReturnAddress;
    myLift->liftTimer = inTimer;
    myLift->liftNumber = inNumber;
    myLift->maxBufferSize = myCapacity;
    myLift->out_sim_file = inFile;
    myLift->semaphoreFull = inFullSem;
    myLift->semaphoreEmpty = inEmptySem;
    myLift->liftZeroFileSem = inFileSem;
    myLift->requestFileSem = inRequestFileSem;
    return myLift;
}
```

```c
 #ifndef LIFTS_H
#define LIFTS_H
#define TRUE 1
#define FALSE 0
#define CALCDISTANCE(a,b,c,d) ((a - b) + (c - d))
#include <pthread.h>
#include <stdio.h>
#include <semaphore.h>
#include "queue.h"
#include "request.h"
#include "processLift.h"
/*****************************************************************************
*   PURPOSE: This is the struct for each lift in the PThread implementation
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
typedef struct liftStruct { /* Lift One */
    queue* buffer;
    request* previousRequest;
    int* finishedRead;
    int* liftReturnVals; /* [0] = #movement, [1] = #requests */
    int liftTimer;
    int liftNumber;
    int maxBufferSize;
    pthread_mutex_t* mutexLock;
    pthread_cond_t* full;
    pthread_cond_t* empty;
    FILE* out_sim_file;
} liftStruct;
liftStruct* initLiftStruct(queue* inBuffer, int inTimer, int whatLift,
pthread_mutex_t* inLock, pthread_cond_t* inFullCond, pthread_cond_t* inEmptyCond,
int* inFinishedRead, int inMaxBufferSize, FILE* inFile);
void freeLiftStruct(liftStruct* toFree);
void* lift(void* args);
void liftProcess(void* args);
processLift* createProcessLift(arrayQueue** inBuffer, int** inFinishedRead, int inTimer,
int inNumber, int myCapacity, FILE*** inFile, sem_t** inFullSem, sem_t** inEmptySem,
sem_t** inFileSem, sem_t** inRequestFileSem, int** inReturnAddress);
#endif
```

```c
/*
    ----- NOTICE -----
    SUBMITTED FOR UCP 2019 ASSIGNMENT SEMESTER 2
    AUTHOR: JONATHAN WRIGHT
    DATE: 8/10/2019
*/
#include "list.h"
#include <stdlib.h>
#include <stdio.h>
/*
    Purpose: This file is about manipulating linked lists, it was originally written for prac 7, it is a
    doubly linked double ended generic linked list with a size.
    Author: Jonathan Wright
    Date: 8/10/2019
*/

/*
    Purpose: To create and export a empty linked list.
    Date: 8/10/2019
    Imports: None
    Exports: list (Empty)
*/
linkedList* createLinkedList()
{
    linkedList* list;
    list = malloc(sizeof(linkedList));
    list->size = 0;
    list->head = NULL;
    list->tail = NULL;
    return list;
}
/*
    Purpose: To insert generic data at the start of the linked list.
    Date: 8/10/2019
    Imports: list and data
    Exports: none
*/
void insertStart(linkedList* list, void* inData)
{
    listNode* node;
    node = (listNode*)malloc(sizeof(listNode));
    node->data = inData;
    list->size += 1;
    if (list->head == NULL)
    { /* NO HEAD */
        list->head = node;
        list->tail = node;
        node->next = NULL;
        node->prev = NULL;
    }
    else
    { /* HEAD */
        list->head->prev = node;
        node->next = list->head;
        node->prev = NULL;
        list->head = node;
    }
}
/*
    Purpose: Remove the data at head of linked list and return it.
    Date: 8/10/2019
    Imports: list
    Exports: data (removed from start)
*/
void* removeStart(linkedList* list)
{
    listNode* removed = NULL;
```

```
    void* outData = NULL;
    if (list->head != NULL)
    {
        removed = list->head;
        list->head = list->head->next;
        list->head->prev = NULL;
        list->size -= 1;
        outData = removed->data;
        free(removed);
    }
    else
    {

    }
    return outData; /* NOT FREED */
}
/*
    Purpose: Insert data at tail of linked list.
    Date: 8/10/2019
    Imports: list, data
    Exports: None
*/
void insertLast(linkedList* list, void* inData)
{
    listNode* node;
    node = (listNode*)malloc(sizeof(listNode));
    node->data = inData;
    list->size += 1;
    node->next = NULL;
    if (list->tail == NULL)
    { /* NO TAIL */
        list->head = node;
        list->tail = node;
        node->next = NULL;
        node->prev = NULL;
    }
    else
    { /* TAIL */
        list->tail->next = node;
        node->prev = list->tail;
        list->tail = node;
        node->next = NULL;
    }
}
/*
    Purpose: Remove data at tail of linked list.
    Date: 8/10/2019
    Imports: list
    Exports: data (removed data)
*/
void* removeLast(linkedList* list)
{
    void* outData = NULL;
    listNode* removed = NULL;
    if (list->tail == NULL)
    {
        /* NO TAIL */
    }
    else
    {
        removed = list->tail;
        if (list->tail->prev == NULL)
        {
            list->head = NULL;
            list->tail = NULL;
            list->size -= 1;
```

```
        }
        else
        {
            list->tail = list->tail->prev;
            list->tail->next = NULL;
            list->size -= 1;
        }
        outData = removed->data;
        free(removed);
    }
    return outData;
}
/*
    Purpose: To print contents of linked list.
    Date: 8/10/2019
    Imports: list
    Exports: None
*/
void printLinkedList(linkedList* list)
{
    listNode* curr;
    curr = list->head;
    while (curr != NULL)
    {
        printf("%s,",(char*)curr->data);
        curr = curr->next;
    }
    printf("\n");
}
/*
    Purpose: To free linked list.
    Date: 8/10/2019
    Imports: list
    Exports: None
*/
void freeLinkedList(linkedList* list, void (*freeMethod)(void* data))
{
    listNode* curr; listNode* prev;
    curr = list->head;
    while (curr != NULL)
    {
        prev = curr;
        curr = curr->next;
        (*freeMethod)(prev->data /* Return A Game */);
        free(prev);
    }
    free(list);
    list = NULL;
}
```

```c
#ifndef LIST
#define LIST
/* --- NOTICE ---
    ORIGINALLY SUBMITTED FOR UCP ASSIGNMENT 2019
    AUTHOR: JONATHAN WRIGHT
    DATE: 8/10/2019
*/
/*
    Purpose: Nodes for a linked list (Contains generic data, doubly linked (prev and next))
    Date: 8/10/2019 -- Taken From Prac 7 UCP Credit to Jonathan Wright
    Author: Jonathan Wright
*/
typedef struct listNode {
    void* data;
    struct listNode* next;
    struct listNode* prev;
} listNode;
/*
    Purpose: Linked List Structure, double ended (head and tail) and contains a size.
    Date: 8/10/2019 -- Taken from Prac 7 UCP Credit To Jonathan Wright
    Author: Jonathan Wright
*/
typedef struct linkedList {
    int size;
    listNode* head;
    listNode* tail;
} linkedList;
linkedList* createLinkedList();
void insertStart(linkedList* list, void* data);
void* removeStart(linkedList* list);
void insertLast(linkedList* list, void* data);
void* removeLast(linkedList* list);
void printLinkedList(linkedList* list);
void freeLinkedList(linkedList* list, void (*freeMethod)(void* data));
#endif
```

```
#ifndef PROCESSLIFT_H
#define PROCESSLIFT_H
#include <semaphore.h>
/*****************************************************************************
*   PURPOSE: Queue ADT with a array data structure
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
typedef struct arrayQueue {
    int front;
    int back;
    int capacity;
    int size;
    request** myBuffer;
} arrayQueue;
/*****************************************************************************
*   PURPOSE: Lift Struct for each Lift in the Process Implementation.
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
typedef struct processLift {
    arrayQueue** buffer;
    request previousRequest;
    int** finishedRead;
    int** liftReturnVals; /* [0] = #movement, [1] = #requests */
    int liftTimer;
    int liftNumber;
    int maxBufferSize;
    FILE*** out_sim_file;
    sem_t** semaphoreFull;
    sem_t** semaphoreEmpty;
    sem_t** liftZeroFileSem;
    sem_t** requestFileSem;
} processLift;
#endif
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <limits.h>
#ifdef PTHREAD
#include <pthread.h>
#endif
#ifdef PROCESS
#include <pthread.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#endif
#include "lifts.h"
#include "queue.h"
#include "program.h"
#include "request.h"
#include "list.h"

/*****************************************************************************
*   PURPOSE: Starting point for program.
*   IMPORTS: int argc, char** argv
*   EXPORTS: int errorStatus
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
int main(int argc, char** argv) {
    /*
        TODO: FILTER OUT NEGATIVES
            FILTER OUT ZEROS.
    */
    char* endptr;
    int bufferSize, liftTime;
    if (argc == 3) {
        bufferSize = strtol(argv[1], &endptr, 10);
        if ((errno == ERANGE && (bufferSize == LONG_MAX || bufferSize == LONG_MIN)) || (errno != 0 && bufferSize == 0)) {
            fprintf(stderr, "Invalid Number Entered for m: %s\n", argv[1]);
            exit(EXIT_FAILURE);
        }
        if (endptr == argv[1]) {
            fprintf(stderr, "No Digits were found for m.\n");
            exit(EXIT_FAILURE);
        }
        if (*endptr != '\0')
            printf("Only integers can be entered, decimal point has been cut off for buffer.\n");
        liftTime = strtol(argv[2], &endptr, 10);
        if (endptr == argv[2]) {
            fprintf(stderr, "No Digits were found for t.\n");
            exit(EXIT_FAILURE);
        }
        if (*endptr != '\0')
            printf("Only integers can be entered, decimal point has been cut off for time.\n");
        if ((errno == ERANGE && (liftTime == LONG_MAX || liftTime == LONG_MIN)) || (errno != 0 && liftTime == 0)) {
            fprintf(stderr, "Invalid Number Entered for m: %s\n", argv[1]);
            exit(EXIT_FAILURE);
        }
        #ifdef DEBUG
        printf("Reached with values m:%d and t:%d\n",bufferSize,liftTime);
        #endif
        if (bufferSize <= 0) {
            fprintf(stderr, "A Non-Zero Integer Number has to be input for bufferSize.\n");
            exit(EXIT_FAILURE);
        }
        if (liftTime <= 0) {
```

```
            fprintf(stderr, "A Non-Zero Integer Number has to be input for liftTime.\n");
            exit(EXIT_FAILURE);
        }
        beginSimulation(bufferSize, liftTime);
    } else {
        fprintf(stderr, "Invalid format entered!\nExpected: lift_sim_A m t\n\nm = buffer size\nt = time required by each lift\n");
        exit(EXIT_FAILURE);
    }
    return(0);
}

/*****************************************************************************
*   PURPOSE: PThread based implementation of the simulation
*   IMPORTS: int bufferSize, int liftTime
*   EXPORTS: none
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
#ifdef PTHREAD
void beginSimulation(int bufferSize, int liftTime) {
    int totalMovements = 0, totalRequests = 0;
    FILE* out_sim;
    liftStruct* liftZero, *liftOne, *liftTwo, *liftThree;
    pthread_mutexattr_t canShare;
    pthread_mutex_t liftLock;
    pthread_cond_t full, empty;
    pthread_t LiftR, Lift_1, Lift_2, Lift_3;
    int finishedRead = FALSE;
    queue* buffer = createQueue(bufferSize);
    out_sim = fopen("out_sim", "w");
    if (out_sim == NULL) {
        fprintf(stderr, "Unable to write to a file called \"out_sim\".\n");
        exit(EXIT_FAILURE);
    }
    pthread_mutexattr_init(&canShare);
    pthread_mutexattr_setpshared(&canShare, PTHREAD_PROCESS_SHARED);
    pthread_mutex_init(&liftLock, &canShare);
    pthread_cond_init(&full, NULL);
    pthread_cond_init(&empty, NULL);
    liftZero = initLiftStruct(buffer, liftTime, 0, &liftLock, &full,
    &empty, &finishedRead, bufferSize, out_sim);
    liftOne = initLiftStruct(buffer, liftTime, 1, &liftLock, &full,
    &empty, &finishedRead, bufferSize, out_sim);
    liftTwo = initLiftStruct(buffer, liftTime, 2, &liftLock, &full,
    &empty, &finishedRead, bufferSize, out_sim);
    liftThree = initLiftStruct(buffer, liftTime, 3, &liftLock, &full,
    &empty, &finishedRead, bufferSize, out_sim);
    pthread_create(&Lift_1, NULL, &lift, liftOne);
    pthread_create(&Lift_2, NULL, &lift, liftTwo);
    pthread_create(&Lift_3, NULL, &lift, liftThree);
    pthread_create(&LiftR, NULL, &requestt, liftZero);
    pthread_join(LiftR, NULL);
    pthread_join(Lift_1, NULL);
    pthread_join(Lift_2, NULL);
    pthread_join(Lift_3, NULL);
    totalMovements += *((liftOne->liftReturnVals) + 0);
    totalMovements += *((liftTwo->liftReturnVals) + 0);
    totalMovements += *((liftThree->liftReturnVals) + 0);
    totalRequests += *((liftOne->liftReturnVals) + 1);
    totalRequests += *((liftTwo->liftReturnVals) + 1);
    totalRequests += *((liftThree->liftReturnVals) + 1);
    fprintf(out_sim, "Total number of movements: %d\nTotal number of requests: %d\n", totalMovements, totalRequests);
    freeLiftStruct(liftZero);
    freeLiftStruct(liftOne);
    freeLiftStruct(liftTwo);
    freeLiftStruct(liftThree);
    freeQueue(buffer, free);
```

```
    fclose(out_sim);
}
#endif

/*****************************************************************************
*   PURPOSE: PThread based implementation selected when PTHREAD and PROCESS not
*   defined.
*   IMPORTS: int BufferSize, int liftTime
*   EXPORTS: none
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
#if !defined(PTHREAD) && !defined(PROCESS)
void beginSimulation(int bufferSize, int liftTime) {
    int totalMovements = 0, totalRequests = 0;
    FILE* out_sim;
    liftStruct* liftZero, *liftOne, *liftTwo, *liftThree;
    pthread_mutexattr_t canShare;
    pthread_mutex_t liftLock;
    pthread_cond_t full, empty;
    pthread_t LiftR, Lift_1, Lift_2, Lift_3;
    int finishedRead = FALSE;
    queue* buffer = createQueue(bufferSize);
    printf("The Assumption is made you have selected PThread\nIf you wish to choose Process please compile with Process
defined.\n");
    out_sim = fopen("out_sim", "w");
    if (out_sim == NULL) {
        fprintf(stderr, "Unable to write to a file called \"out_sim\".\n");
        exit(EXIT_FAILURE);
    }
    pthread_mutexattr_init(&canShare);
    pthread_mutexattr_setpshared(&canShare, PTHREAD_PROCESS_SHARED);
    pthread_mutex_init(&liftLock, &canShare);
    pthread_cond_init(&full, NULL);
    pthread_cond_init(&empty, NULL);
    liftZero = initLiftStruct(buffer, liftTime, 0, &liftLock, &full,
    &empty, &finishedRead, bufferSize, out_sim);
    liftOne = initLiftStruct(buffer, liftTime, 1, &liftLock, &full,
    &empty, &finishedRead, bufferSize, out_sim);
    liftTwo = initLiftStruct(buffer, liftTime, 2, &liftLock, &full,
    &empty, &finishedRead, bufferSize, out_sim);
    liftThree = initLiftStruct(buffer, liftTime, 3, &liftLock, &full,
    &empty, &finishedRead, bufferSize, out_sim);
    pthread_create(&Lift_1, NULL, &lift, liftOne);
    pthread_create(&Lift_2, NULL, &lift, liftTwo);
    pthread_create(&Lift_3, NULL, &lift, liftThree);
    pthread_create(&LiftR, NULL, &requestt, liftZero);
    pthread_join(LiftR, NULL);
    pthread_join(Lift_1, NULL);
    pthread_join(Lift_2, NULL);
    pthread_join(Lift_3, NULL);
    totalMovements += *((liftOne->liftReturnVals) + 0);
    totalMovements += *((liftTwo->liftReturnVals) + 0);
    totalMovements += *((liftThree->liftReturnVals) + 0);
    totalRequests += *((liftOne->liftReturnVals) + 1);
    totalRequests += *((liftTwo->liftReturnVals) + 1);
    totalRequests += *((liftThree->liftReturnVals) + 1);
    fprintf(out_sim, "Total number of movements: %d\nTotal number of requests: %d\n", totalMovements, totalRequests);
    freeLiftStruct(liftZero);
    freeLiftStruct(liftOne);
    freeLiftStruct(liftTwo);
    freeLiftStruct(liftThree);
    freeQueue(buffer, free);
    fclose(out_sim);
}
#endif
```

```
/*****************************************************************************
*   PURPOSE: Process implementation for simulation
*   IMPORTS: int bufferSize, int liftTime
*   EXPORTS: none
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
#ifdef PROCESS
void beginSimulation(int bufferSize, int liftTime) {
    /*int totalMovements = 0, totalRequests = 0;*/
    processLift *liftOne, *liftTwo, *liftThree,  *liftZero;
    pid_t mainProcess, LiftR, Lift_1, Lift_2, Lift_3;
    arrayQueue* reqQueue;
    int totalMovements, totalRequests;
    int* liftOneReturns = (int*)mmap(NULL,sizeof(int) * 2,PROT_READ|PROT_WRITE,MAP_SHARED|MAP_ANON,-1,0);
    int* liftTwoReturns = (int*)mmap(NULL,sizeof(int) * 2,PROT_READ|PROT_WRITE,MAP_SHARED|MAP_ANON,-1,0);
    int* liftThreeReturns = (int*)mmap(NULL,sizeof(int) * 2,PROT_READ|PROT_WRITE,MAP_SHARED|MAP_ANON,-1,0);
    request* requestBuffer = (request*)mmap(NULL,sizeof(request) *
bufferSize,PROT_READ|PROT_WRITE,MAP_SHARED|MAP_ANON,-1,0);
    int* readDone = (int*)mmap(NULL,sizeof(int),PROT_READ|PROT_WRITE,MAP_SHARED|MAP_ANON,-1,0);
    FILE** out_sim_file = (FILE**)mmap(NULL,sizeof(FILE*),PROT_READ|PROT_WRITE,MAP_SHARED|MAP_ANON,-1,0);
    sem_t* fullSem = (sem_t*)mmap(NULL,sizeof(sem_t),PROT_READ|PROT_WRITE,MAP_SHARED|MAP_ANON,-1,0);
    sem_t* emptySem = (sem_t*)mmap(NULL,sizeof(sem_t),PROT_READ|PROT_WRITE,MAP_SHARED|MAP_ANON,-1,0);
    sem_t* liftZeroSem = (sem_t*)mmap(NULL,sizeof(sem_t),PROT_READ|PROT_WRITE,MAP_SHARED|MAP_ANON,-1,0);
    sem_t* requestLiftFileSem =
(sem_t*)mmap(NULL,sizeof(sem_t),PROT_READ|PROT_WRITE,MAP_SHARED|MAP_ANON,-1,0);
    sem_init(emptySem, 5, bufferSize);
    sem_init(fullSem, 5, 0);
    sem_init(liftZeroSem, 5, 1);
    sem_init(requestLiftFileSem, 5, 0);
    liftOne = createProcessLift(&reqQueue, &readDone, liftTime, 1, bufferSize, &out_sim_file, &fullSem, &emptySem,
&liftZeroSem, &requestLiftFileSem, &liftOneReturns);
    liftTwo = createProcessLift(&reqQueue, &readDone, liftTime, 2, bufferSize, &out_sim_file, &fullSem, &emptySem,
&liftZeroSem, &requestLiftFileSem, &liftTwoReturns);
    liftThree = createProcessLift(&reqQueue, &readDone, liftTime, 3, bufferSize, &out_sim_file, &fullSem, &emptySem,
&liftZeroSem, &requestLiftFileSem, &liftThreeReturns);
    liftZero = createProcessLift(&reqQueue, &readDone, liftTime, 0, bufferSize, &out_sim_file, &fullSem, &emptySem,
&liftZeroSem, &requestLiftFileSem, NULL);
    *out_sim_file = fopen("out_sim", "w");
    *readDone = 0;
    #ifdef OUTSIMASSTDOUT
    fclose(*out_sim_file);
    *out_sim_file = stdout;
    #endif
    if (*out_sim_file == NULL) {
        fprintf(stderr, "Unable to write to file called out_sim.\n");
        exit(EXIT_FAILURE);
    }
    reqQueue = createArrayQueue(&requestBuffer, bufferSize);
    mainProcess = getppid();
    if (mainProcess == getppid())
        LiftR = fork();
    if (mainProcess == getppid())
        Lift_1 = fork();
    if (mainProcess == getppid())
        Lift_2 = fork();
    if (mainProcess == getppid())
        Lift_3 = fork();
    if (LiftR == 0) {
        processRequest(&liftZero);
    } else if (Lift_1 == 0) {
        liftProcess(&liftOne);
    } else if (Lift_2 == 0) {
        liftProcess(&liftTwo);
    } else if (Lift_3 == 0) {
        liftProcess(&liftThree);
```

```
    } else {
        waitpid(LiftR, NULL, 0);
        waitpid(Lift_1, NULL, 0);
        waitpid(Lift_2, NULL, 0);
        waitpid(Lift_3, NULL, 0);
        totalMovements = 0;
        totalRequests = 0;
        totalMovements += *(liftOneReturns + 0);
        totalMovements += *(liftTwoReturns + 0);
        totalMovements += *(liftThreeReturns + 0);
        totalRequests += *(liftOneReturns + 1);
        totalRequests += *(liftTwoReturns + 1);
        totalRequests += *(liftThreeReturns + 1);
        fprintf(*out_sim_file, "Total number of movements: %d\nTotal number of requests: %d\n", totalMovements, totalRequests);
    }
    cleanupArrayQueue(&reqQueue);
    fclose(*out_sim_file);
    free(liftOne);
    free(liftTwo);
    free(liftThree);
    free(liftZero);
    munmap(requestBuffer, sizeof(request) * bufferSize);
    munmap(readDone, sizeof(int));
    munmap(out_sim_file, sizeof(FILE*));
    munmap(fullSem, sizeof(sem_t));
    munmap(emptySem, sizeof(sem_t));
    munmap(liftZeroSem, sizeof(sem_t));
    munmap(requestLiftFileSem, sizeof(sem_t));
    munmap(liftOneReturns, sizeof(int) * 2);
    munmap(liftTwoReturns, sizeof(int) * 2);
    munmap(liftThreeReturns, sizeof(int) * 2);
    #ifdef DEBUG
    printf("pid of main: %d\n", mainProcess);
    #endif
}
#endif
```

```
#ifndef PROGRAM_H
#define PROGRAM_H
void beginSimulation(int bufferSize, int liftTime);
#endif
```

```c
#include <stdlib.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <stdio.h>
#include "queue.h"
#include "list.h"
#include "request.h"
/* LIST BASED */
/*****************************************************************************
*   PURPOSE: To create a list based queue.
*   IMPORTS: int inBuffer (Not actually needed, its a list but I included it)
*   EXPORTS: queue* newQueue
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
queue* createQueue(int inBuffer) {
    queue* newQueue = (queue*)malloc(sizeof(queue));
    newQueue->list = createLinkedList();
    newQueue->bufferSize = inBuffer;
    return newQueue;
}
/*****************************************************************************
*   PURPOSE: frees the queue and the list.
*   IMPORTS: queue* myQueue and the Free method
*   EXPORTS: none
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
void freeQueue(queue* myQueue, void (*freeMethod)(void* inData)) {
    freeLinkedList(myQueue->list, freeMethod);
    free(myQueue);
}
/*****************************************************************************
*   PURPOSE: enqueues objects into the queue
*   IMPORTS: void* inObject, queue* myQueue
*   EXPORTS: none
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
void enqueue(void* inObject, queue* myQueue) {
    insertStart(myQueue->list, inObject);
}
/*****************************************************************************
*   PURPOSE: dequeue objects from the queue
*   IMPORTS: queue* myQueue
*   EXPORTS: void* outObject
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
void* dequeue(queue* myQueue) {
    return removeLast(myQueue->list);
}
/*****************************************************************************
*   PURPOSE: returns the object at the top of the queue but does not free it from
*   list.
*   IMPORTS: queue* myQueue
*   EXPORTS: void* outObject
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
void* peek(queue* myQueue) {
    return myQueue->list->tail;
}

/* ARRAY BASED SHUFFLING QUEUE */
/*****************************************************************************
*   PURPOSE: Creates an array based queue.
```

```
*   IMPORTS: request** inBuffer, int inBufferSize
*   EXPORTS: arrayQueue* outQueue
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
arrayQueue* createArrayQueue(request** inBuffer, int inBufferSize) {
    arrayQueue* myQueue =
(arrayQueue*)mmap(NULL,sizeof(arrayQueue),PROT_READ|PROT_WRITE,MAP_SHARED|MAP_ANON,-1,0);
    myQueue->myBuffer = inBuffer;
    myQueue->front = 0;
    myQueue->back = myQueue->front;
    myQueue->capacity = inBufferSize;
    myQueue->size = 0;
    return myQueue;
}
/*****************************************************************************
*   PURPOSE: Dequeues an object from the array based queue.
*   IMPORTS: arrayQueue** inBuffer
*   EXPORTS: request outRequest
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
request arrayDequeue(arrayQueue** inBuffer) {
    request outRequest, temp;
    outRequest.destinationFloor = -1;
    outRequest.requestFloor = -1;
    temp.destinationFloor = -1;
    temp.requestFloor = -1;
    if ((*inBuffer)->size != 0) {
        outRequest = ((*((*inBuffer)->myBuffer))[0]);
        shuffle((*inBuffer)->myBuffer, (*inBuffer)->back - 1);
        if ((*inBuffer)->back < (*inBuffer)->capacity) {
            ((*((*inBuffer)->myBuffer))[(*inBuffer)->back]) = temp;
        }
        (*inBuffer)->back -= 1;
        (*inBuffer)->size -= 1;
    }
    else {
        fprintf(stderr, "Unable to dequeue. Array is empty.\n");
    }
    return outRequest;
}
/*****************************************************************************
*   PURPOSE: Enqueue an object into an array based queue.
*   IMPORTS: request inData, arrayQueue** inBuffer
*   EXPORTS: none
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
void arrayEnqueue(request inData, arrayQueue** inBuffer) {
    if ((*inBuffer)->capacity == (*inBuffer)->size) {
        fprintf(stderr, "Unable to enqueue, Array is full.\n");
    } else {
        (*((*inBuffer)->myBuffer))[(*inBuffer)->back] = inData;
        (*inBuffer)->back += 1;
        (*inBuffer)->size += 1;
    }
}
/*****************************************************************************
*   PURPOSE: Shuffles the array in the queue.
*   IMPORTS: request** inArray, int whereBackIs
*   EXPORTS: none
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
void shuffle(request** inArray, int whereBackIs) {
```

```
    int k;
    for (k = 0; k < whereBackIs; k++) {
        (*inArray)[k] = (*inArray)[k + 1];
    }
}
/*******************************************************************************
*   PURPOSE: Removes the memory mapping of the arrayQueue from shared memory.
*   IMPORTS: arrayQueue** myAddr
*   EXPORTS: none
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*******************************************************************************/
void cleanupArrayQueue(arrayQueue** myAddr) {
    munmap(*myAddr, sizeof(arrayQueue));
}
```

```c
#ifndef QUEUE_H
#define QUEUE_H
#include "request.h"
#include "list.h"
#include "processLift.h"
/*******************************************************************************
*   PURPOSE: list based queue struct
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*******************************************************************************/
typedef struct queue {
   linkedList* list;
   int bufferSize;
} queue;
queue* createQueue(int bufferSize);
void freeQueue(queue* myQueue, void (*freeMethod)(void* inData));
void enqueue(void* inObject, queue* myQueue);
void* dequeue(queue* myQueue);
void* peek(queue* myQueue);

arrayQueue* createArrayQueue(request** inBuffer, int inBufferSize);
void arrayEnqueue(request inRequest, arrayQueue** inBuffer);
request arrayDequeue(arrayQueue** inBuffer);
void shuffle(request** arrayToShuffle, int whereBackIs);
void cleanupArrayQueue(arrayQueue** myAddr);
#endif
```

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <sys/mman.h>
#include "request.h"
#include "lifts.h"
/*****************************************************************************
*   PURPOSE: This C File contains definitions to do with requests aswell as the
*   implmentation of the request elevators.
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/

/*****************************************************************************
*   PURPOSE: creates a request on the heap.
*   IMPORTS: int inRequest, int inDestination
*   EXPORTS: request* outRequest
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
request* createRequest(int inRequest, int inDestination) {
    request* newRequest = malloc(sizeof(request));
    newRequest->requestFloor = inRequest;
    newRequest->destinationFloor = inDestination;
    return newRequest;
}
/*****************************************************************************
*   PURPOSE: creates a request on the stack.
*   IMPORTS: int inRequest, int inDestination
*   EXPORTS: request outRequest
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
request createStackRequest(int inRequest, int inDestination) {
    request newRequest;
    newRequest.requestFloor = inRequest;
    newRequest.destinationFloor = inDestination;
    return newRequest;
}
/*****************************************************************************
*   PURPOSE: creates a request on the shared memory.
*   IMPORTS: int inRequest, int inDestination
*   EXPORTS: request* outRequest
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
request* createSharedRequest(int inRequest, int inDestination) {
    request* newRequest =
(request*)mmap(NULL,sizeof(request),PROT_READ|PROT_WRITE,MAP_SHARED|MAP_ANON,-1,0);
    newRequest->requestFloor = inRequest;
    newRequest->destinationFloor = inDestination;
    return newRequest;
}
/*****************************************************************************
*   PURPOSE: Lift 1-3 PTHREAD implementation.
*   IMPORTS: void* args
*   EXPORTS: NULL
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
void* requestt(void* args) {
    int linecount;
    FILE* file;
    liftStruct* fakeLift;
    int destination, from, fscanfReturn;
    file = fopen("sim_input", "r");
    linecount = 1;
```

```
    fakeLift = (liftStruct*)args; /* This is not a actual lift */
    fscanfReturn = fscanf(file, "%d %d\n", &from, &destination);
    while (fscanfReturn != EOF) {
        pthread_mutex_lock(fakeLift->mutexLock);
        #ifdef DEBUG
        printf("LIFT ZERO WAITING ON EMPTY\n");
        #endif
        pthread_cond_wait(fakeLift->empty, fakeLift->mutexLock);
        while (fscanfReturn != EOF && (fakeLift->buffer->list->size != fakeLift->maxBufferSize)) {
            if (fscanfReturn != 2) {
                printf("A Line contains invalid amount of values, program will now stop.\n");
                *(fakeLift->finishedRead) = TRUE;
                fclose(file);
                pthread_mutex_unlock(fakeLift->mutexLock);
                return NULL;
            } else {
                if (from < 1 || from > 20) {
                    fprintf(stderr, "Line %d contained a number greater than 20 or less than 1 for the request floor, ignoring line.\n",
linecount);
                    linecount += 1;
                    fscanfReturn = fscanf(file, "%d %d\n", &from, &destination);
                } else if (destination < 1 || destination > 20) {
                    fprintf(stderr, "Line %d contained a number greater than 20 or less than 1 for the destination floor, ignoring line.\n",
linecount);
                    linecount += 1;
                    fscanfReturn = fscanf(file, "%d %d\n", &from, &destination);
                } else {
                    enqueue(createRequest(from, destination), fakeLift->buffer);
                    fprintf(fakeLift->out_sim_file,"----------------------------------------\nNew Lift Request From Floor %d to Floor
%d\nRequest No: %d\n----------------------------------------\n\n",
                    from, destination, linecount);
                    fflush(fakeLift->out_sim_file);
                    fscanfReturn = fscanf(file, "%d %d\n", &from, &destination);
                    linecount += 1;
                }
            }
        }
        pthread_mutex_unlock(fakeLift->mutexLock);
    }
    pthread_mutex_lock(fakeLift->mutexLock);
    *(fakeLift->finishedRead) = TRUE;
    pthread_mutex_unlock(fakeLift->mutexLock);
    fclose(file);
    return NULL;
}

/*****************************************************************************
*   PURPOSE: Lift 1-3 Process Implementation.
*   IMPORTS: void* args
*   EXPORTS: none
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*****************************************************************************/
void processRequest(void* args) {
    #ifdef DEBUG
    int tester;
    #endif
    processLift** liftZero;
    FILE* file;
    int destination, from, fscanfReturn, linecount;
    liftZero = (processLift**)args;
    file = fopen("sim_input", "r");
    linecount = 1;
    fscanfReturn = fscanf(file, "%d %d\n", &from, &destination);
    while (fscanfReturn != EOF)
    {
```

```
        #ifdef DEBUG
        sem_getvalue( *((*liftZero)->semaphoreEmpty), &tester);
        printf("EMPTY SEM VALUE BEFORE: %d\n", tester);
        #endif
        sem_wait(*((*liftZero)->semaphoreEmpty));
        #ifdef DEBUG
        sem_getvalue( *((*liftZero)->semaphoreEmpty), &tester);
        printf("SEM VALUE AFTER: %d\n", tester);
        #endif
        if (fscanfReturn != 2) {
            printf("A Line contains too many values or an invalid character, program will now stop.\n");
            **((*liftZero)->finishedRead) = TRUE;
            fclose(file);
            exit(EXIT_FAILURE);
        } else {
            if (from < 1 || from > 20) {
                fprintf(stderr, "Line %d contained a number greater than 20 or less than 1 for the request floor, ignoring line.\n",
linecount);
                linecount += 1;
                fscanfReturn = fscanf(file, "%d %d\n", &from, &destination);
                sem_post(*((*liftZero)->semaphoreEmpty));
            } else if (destination < 1 || destination > 20) {
                fprintf(stderr, "Line %d contained a number greater than 20 or less than 1 for the destination floor, ignoring line.\n",
linecount);
                linecount += 1;
                fscanfReturn = fscanf(file, "%d %d\n", &from, &destination);
                sem_post(*((*liftZero)->semaphoreEmpty));
            } else {
                arrayEnqueue(createStackRequest(from, destination), (*liftZero)->buffer);
                sem_wait(*((*liftZero)->liftZeroFileSem));
                fprintf((**((*liftZero)->out_sim_file)),"-----------------------------------------\nNew Lift Request From Floor %d to Floor
%d\nRequest No: %d\n-----------------------------------------\n\n",
                from, destination, linecount);
                fflush(**((*liftZero)->out_sim_file));
                fscanfReturn = fscanf(file, "%d %d\n", &from, &destination);
                linecount += 1;
                #ifdef DEBUG
                sem_getvalue( *((*liftZero)->semaphoreFull), &tester);
                printf("FULL SEM VALUE BEFORE: %d\n", tester);
                #endif
                sem_post(*((*liftZero)->semaphoreFull));
                #ifdef DEBUG
                sem_getvalue( *((*liftZero)->semaphoreFull), &tester);
                printf("FULL SEM VALUE AFTER: %d\n", tester);
                #endif /* VERIFICATION NEEDED */
                sem_post(*((*liftZero)->requestFileSem));
            }
        }
    }
    **((*liftZero)->finishedRead) = TRUE;
    fclose(file);
}
```

```c
#ifndef REQUEST_H
#define REQUEST_H
/*******************************************************************************
*   PURPOSE: Struct for holding details about requests.
*   DATE: 17/04/2020 - 9:20PM
*   AUTHOR: Jonathan Wright
*******************************************************************************/
typedef struct request {
    int requestFloor;
    int destinationFloor;
} request;
request createStackRequest(int inRequest, int inDestination);
request* createRequest(int inRequest, int inDestination);
request* createSharedRequest(int inRequest, int inDestination);
void* requestt(void* args);
void processRequest(void* args);
#endif
```

MUTUAL EXCLUSION – PTHREADS

All four threads share the same mutex I titled mutexLock in the code, this is passed into each thread's function call through the liftStruct I made (I could have used a global mutex, but I made it a goal of mine to avoid using global in this assignment). All the data within the liftStruct that is passed to each thread is what I consider to be 'shared resources' that are required, at the start of each thread function I made sure to lock the mutex, copy static values across into stack variables so I can avoid any race conditions on static values that need to be accessed. Then each loop in the thread functions I make sure at the top I lock the mutex, access the critical section, then unlock the mutex, this assures mutual exclusion, and running the program with the flag '-fsanitize=thread' shows that there is no race conditions or synchronization issues, I also ran it through helgrind to check. FILE* is handled by the mutex lock and unlock loop so it will not be accessed by multiple instances at once.
As well as this there are no memory leaks as checked by Valgrind.

MUTUAL EXCLUSION – PROCESSES

Similar to the pthreads implementation all four processes share a struct, this struct contains two semaphores 'fullSemaphore and emptySemaphore', to achieve mutual exclusion the lift processes wait on the fullSemaphore which indicates how many objects is in the buffer and the enqueuing process (liftZero) waits on the emptySemaphore which indicates how many spots are left in the buffer, as objects are entered into the buffer fullSemaphore is incremented and emptySemaphore is decremented, the opposite is also true. Waiting on the semaphore assures mutual exclusion as they won't touch the buffer unless they have waited on the semaphore and it passes (waiting is atomic). To access the FILE* in this implementation is very different to the pthreads implementation, and to be honest, I'm not happy with how I implemented it, I used a semaphore called fileSem which when someone is accessing it, then everyone else is waiting on it when they finish accessing the FILE* the semaphore is signaled so the next person can access it, this means that it goes:
liftZero -> actual lifts -> liftZero -> actual lift -> …
This doesn't fit what I like about it as it defeats the purpose of greater than 2 processes, but unfortunately, I just couldn't come up with a better solution.
Running this program with `-fsanitize=thread` shows no race conditions but I believe this is because it is designed for threads, I am almost certain I have made an error somewhere.
Running this program through helgrind shows that semaphores have a bug in libg which I searched up and is apparently a common issue, so I am not sure how to test multi-process programs.

Cases Where My Program Will Fail/Known Issues

Issues

- Helgrind will report a bug for the Process version in libpthread for sem_wait succeeding before a prior sem_post, I believe this is because I initialize it with a non-zero value. This does appear to be the only issue helgrind has with my process implementation.
- If a line is skipped request no is not incremented (makes sense), however, the line count is so it looks slightly off in the sim_out but it is correct.
- If a sim_input file contains a completely incorrect format for the line, it will terminate the program completely (gracefully not just memory leak and crash), this is because I am using fscanf rather than fgets, I don't see a need to account for this as skipping the line could cause issues.
- Running the program with DEBUG=1 defined in the compiling stage will output a redundant amount of information, I have decided to keep this as it makes debugging far easier to be able to follow the program line by line.

Testing

- For sim_input files, I wrote a java program called siminputgenerator.java, you can use this for testing if you would like but I am not 'officially' submitting this java file to be marked as it is a java file, not c, I simply did this because writing the same program in c would have taken me longer.
- For testing mutual exclusion and race conditions for pthread implementation I used the fsanitize flag for gcc and helgrind, both are completely fine with the pthread version.
- For testing mutual exclusion and race conditions for the process implementation I was a bit lost, as the fsanitize flag for gcc seems to only work for pthreads, as such I was only able to use it in helgrind which also seems to be bugged as it has issues with initializing a semaphore with a nonzero value.

Sample Input and Associated Outputs

SAMPLE INPUT 1 – 99 LINES

7 19	5 4	13 9	6 9	8 8
11 18	6 17	11 4	8 17	8 18
16 2	9 8	9 17	19 5	13 5
5 17	20 17	13 13	18 8	13 8
18 1	11 17	11 18	11 12	1 11
6 16	15 8	9 2	19 5	18 13
8 20	19 18	8 9	13 9	5 20
4 3	15 11	13 14	19 11	7 10
1 9	18 14	20 3	20 5	15 6
15 2	3 16	11 9	20 14	5 9
3 1	7 13	3 16	14 20	20 19
4 10	16 11	11 19	13 9	
4 19	4 13	6 3	20 20	
14 17	9 1	13 3	17 6	
7 10	2 2	3 18	3 3	
11 20	10 9	15 18	20 1	
19 3	16 14	10 14	6 5	
4 13	8 10	14 7	4 20	
17 16	4 3	10 3	15 4	
3 13	3 2	5 19	6 8	
3 17	9 14	16 18	17 13	
6 20	17 16	8 7	6 8	

PTHREADS (File 1: m=1, t=1; File 2: m=5, t=5; File3: m=20, t=20)


pthread m=1 t=1.txt


pthread m=5 t=5.txt


pthread m=20 t=20.txt

(Processes output all looks the same, but it is in fact following the buffer rule + time rule, its due to the FILE* being locked behind a separate semaphore, you can verify this by using DEBUG=1)
PROCESSES (File 1: m=1, t=1; File2: m=5, t=5; File3: m=20, t=20)


process m=1 t=1.txt


process m=5 t=5.txt


process m=20 t=20.txt

Other Tests Performed and passed:

Replacing a Request Floor with a letter (Program Ends Gracefully Informing User)
Replacing a Destination Floor with a letter (Program Ends Gracefully Informing User)
Entering a invalid number e.g. Floor 0 or Floor >20, (Line skip)
Entering m or t wrong in program start (Informs User it is wrong.)

Sorry, if you are not using Acrobat the txt files wont open when you click them, you can find these included in the zip upload, I'm not sure how to format those to look nice for a pdf (its over 1400+ lines). If you use Acrobat you can click the text document symbol and it will open, if you look in the zip file I've included each file in-case you don't have Acrobat.

Sample Input 2 – Invalid Floors Spread Throughout

8 45	10 4	3 3	16 17	2 19
500 9	4 4	1 4	15 5	12 12
10 9000	5 15	7 6	8 9	5 14
800 9	13 6	16 11	20 8	20 8
9 14	5 9	17 4	9 1	17 15
13 19	3 4	17 6	10 16	1 11
1 7	19 11	9 4	6 10	9 7
6 20	17 18	1 8	18 18	10 12
16 20	4 19	5 4	14 19	
7 2	11 3	11 10	13 14	
11 7	19 7	3 15	19 17	
			18 14	

Due to already testing at various buffer sizes and times for other data sets I will be testing the rest of the samples with NOTSLEEP=1 defined, this means the lifts will not be sleeping so I can perform testing faster.

PTHREAD (m=1 t=1, m=5 t=5, m=20 t=20)
* Lines 1, 2, 3 & 4 are all skipped as they have floors that do not exist (Floor 45, Floor 500, Floor 9000, Floor 800).


pthread m=1 t=1.txt


pthread m=5 t=5.txt


pthread m=20 t=20.txt

PROCESS (m=1 t=1, m=5 t=5, m=20 t=20)
* Lines 1, 2, 3 & 4 are all skipped as they have floors that do not exist (Floor 45, Floor 500, Floor 9000, Floor 800).


process m=1 t=1.txt


process m=5 t=5.txt


process m=20 t=20.txt

As with the last sample outputs, if the links aren't working please check the zip files, I have organized them into folders such as 'Sample_input_2' which contains each of these files

Sample Input 3 – Invalid Characters

8 45 500 a hello world 9000 HI 9 9 I m no 19 1 this 6 20 16 20 7 2 11 7	10 4 4 4 5 15 13 6 5 9 3 4 19 11 17 18 4 19 11 3 19 7	3 3 1 4 7 6 16 11 17 4 17 6 9 4 1 8 5 4 11 10 3 15	16 17 15 5 8 9 20 8 9 1 10 16 6 10 18 18 14 19 13 14 19 17 18 14	2 19 12 12 5 14 20 8 17 15 1 11 9 7 10 12

For both PTHREAD and PROCESS implementations of this lift I have chosen to end gracefully if the user does something like this, as I didn't really see the need in skipping the line if they have entered something like '5 helloworld', this is because I believe the user should be informed of the error so they can fix their test file (unlike a invalid floor which may mean it was designed for a different elevator/lift system, so I can just ignore that floor).

There are no test outputs because the user is informed through stderr that it is going to end due to a invalid character in a line.