**Ultima 2.0**

Giovanna Gorski

C435 - Operating Systems

Indiana University

Spring 2020

**Ultima 2.0**

**Phase I**

Scheduler and Semaphore

February 23, 2020

**Phase II**

Message Passing (IPC)

April 5, 2020

**Phase III**

Memory Management

April 13, 2020

**Phase IV**

File System

April 27, 2020

Author: Giovanna Gorski

(574) 302 - 1987

[ggorski@iu.edu](mailto:ggorski@iu.edu)

Instructor: Dr. Hossein Hakimzadeh

hhakimza@iusb.edu

Indiana University of South Bend

Spring 2020

# Resume

|  |  |
| --- | --- |
| **Giovanna Gorski**  8557 E John Emery Rd,  New Carlisle, IN 46552 | Home: (574) 797 - 0338  Cellphone: (574) 302 - 1987  ggorski@iu.edu |

SUMMARY

Self-starter, outgoing, goal-oriented, very detail-oriented, and professional individual seeking an opportunity to begin a career as a Data Scientist. I have conducted research in Computer Technologies to aid the delivery of health care to ‘at-risk’ communities. My strongest skill is the ability to learn new concepts and to easily adapt to new environments.

EDUCATION

* Indiana University, South Bend, IN

**Bachelor of Arts in Computer Science***,* May 2020 (Expected Graduation in May 2020)

Minor: Mathematics.

▪ Dean’s list nominee.

▪ Tutored first-year students in Computer Science and Informatics.

▪ Received the William J Knight Scholarship award of excellence in Computer Science and Technology.

* Ivy Tech Community College, South Bend, IN

**Associate Degree in Software Development***,* May 2017

▪ Dean’s list nominee.

▪ Magna Cum Laude graduate.

* Pontifícia Universidade Católica de Campinas, Brazil

**Bachelor’s in Computer Engineering***,* Feb 2013 (No degree obtained)

FOREIGN LANGUAGES

· Fluent in Portuguese (Brazil) and English.

WORK EXPERIENCE

* **Research fellow**

Indiana University, South Bend, IN

As a research fellow, I performed research on the applications of GIS in health care.

· Work presented at 2018 Indiana STEM Louis Stokes Alliance for Minority Participation (IN LSAMP) Annual Conference in Indianapolis – Indiana. Abstract: vCARE: Application of Geotagging to Improve Delivery of Public Health Services.

* **Tutor/Laboratory Assistant**, July 2017 - May 2019

Indiana University, South Bend, IN

Assisted freshmen students in Computer Science and Informatics with some of the major courses such as:

▪ Data Structures, Computer Programming I and II, Multiuser Operating Systems, and Discrete Structures.

▪ Assisted the lab lecture of the course Computer programming I.

SKILLS

· Programming languages

o C++, Python, PHP, HTML, CSS, JavaScript, and MySQL.

o Object-oriented programming.

o OpenLayers, and Bootstrap frameworks.

· Operational system

o macOS, Linux, and Windows.

CERTIFICATIONS

* **Social and Behavioral Responsible Conduct of Research**

CITI Program, A Division of BRANY

May 2017 - No Expiration

Credential ID: 27091071

Abstract

Ultima 2.0 is a project with the purpose to demonstrate some fundamental mechanisms that exist in an Operating System. Furthermore, Ultima 2.0 is divided into phases where at the end of each phase, a mechanism should be implemented. All together, to exemplify, the first phase of Ultima demonstrates the implementation of a Scheduler mechanism alongside a Binary Semaphore.

The Scheduler mechanism in Phase I should control which task is currently running by the CPU. Moreover, the Scheduler mechanism will use a round robin mechanism to ensure that each task has its Quantum time. On the other hand, the Binary Semaphore should serve as a critical region locking mechanism allowing only one task at a time to get the hold of the shared resource. As the project progresses, Phase II introduces the concept of inter process communication allowing each created task by the Scheduler to send messages to each other, and Phase III introduces the concept of Memory Management allowing each thread to write to memory as it is shared by each other.

Keywords: Ultima, Operating Systems, Inter process communication, memory management

# Table of Contents

[**Resume**](#_b3q8zpeva4k2) **3**

[**Table of Contents**](#_1a484fiaimh6) **6**

[**Phase I**](#_md2efeps7v0) **9**

[Scheduler and Semaphore](#_n9wbbbu22e1c) 9

[Phase Abstract](#_1fdij81ji088) 10

[Phase Description](#_1n43ovx3t3pj) 11

[Design Documentation](#_10ca2198tk16) 15

[Source code](#_h0rzg970qzl) 16

[Main.cpp](#_y46b6la43484) 17

[LinkedList.h](#_tit7a3yzvmk3) 20

[Node.h](#_k4ttmrpw6spp) 29

[Queue.h](#_hu9tl2vq188z) 30

[Scheduler.h](#_4r8bc9kh2y5z) 34

[Semaphore.h](#_wl7vdg2nt6d7) 44

[ThreadTable.h](#_z8fvlwsnro67) 49

[Output](#_mk1igf24nthi) 59

[Testing Strategy](#_npc1st3qltv4) 61

[Discussion](#_31ubcyv7m2bb) 62

[**Phase II**](#_68eyxi7es72p) **64**

[Message Passing (IPC)](#_n9x368lu6hdj) 64

[Phase Abstract](#_3rdtz4rvlr5w) 65

[Phase Description](#_bd060ydj5pet) 66

[Design Documentation](#_t6mf7ickiap5) 72

[Source code](#_huujeeryxqxf) 73

[Main.cpp](#_nox8mhsi4wfn) 74

[IPC.h](#_d71jalez2eie) 78

[Queue.h](#_hbauzx714gf) 84

[LinkedList.h](#_qyk5693oplaz) 89

[MCB.h](#_drbizkvjkp75) 99

[Node.h](#_a53oj1itymkb) 101

[Scheduler.h](#_f0y194miglcg) 103

[Semaphore.h](#_oz4eok3h9pk6) 113

[ThreadTable.h](#_ui37dya9xe9o) 118

[Output](#_kvcr4d5o7x8t) 128

[Demonstration of message sent](#_k3700o15cefy) 129

[Demonstration of IPC being Dump](#_tqa9ha29qsjy) 130

[Testing Strategy](#_bi44e3n9gbvx) 131

[Discussion](#_1vlfajj1j81w) 131

[**Phase III**](#_t78ox2ijfzhq) **132**

[Memory Management](#_m9mrha7lwa49) 133

[Phase Abstract](#_8kb0t8ujf35g) 134

[Phase Description](#_qhgghk4gsqz3) 134

[Design Documentation](#_qtd7qb24evj5) 139

[Source code](#_esh2dx1i993m) 140

[Main.cpp](#_c98y4euo103g) 141

[IPC.h](#_6mceldvc8gpc) 146

[LinkedList.h](#_3pjaahxnrqss) 155

[MCB.h](#_l3e5wjbxaa3y) 166

[MemoryMgmt.h](#_mrrnhqi27h1j) 167

[Node.h](#_3l8bpgdgy6y4) 183

[Queue.h](#_9dr6nylpw50) 185

[Scheduler.h](#_5ydhwptsq2o0) 190

[Semaphore.h](#_p8y9g5euqlbd) 203

[ThreadTable.h](#_zbq3shr21xq4) 208

[Window.h](#_hp8ygus2mw4k) 218

[Output](#_fufpbtojfrft) 225

[Demonstration of Curses Library](#_mvyomg12ywi7) 226

[Demonstration of Memory Management](#_h89ad7bvo1ne) 227

[Testing Strategy](#_aopy4g7dnr5u) 227

[Discussion](#_mmf9o4wmtgeg) 229

[Brainstorm Ideas](#_90623r1r9aw2) 230

[Project Schedule](#_l1zjlfclhq49) 230

[**Phase IV**](#_cpzfz9kcuu3i) **230**

[File system](#_d107oey1fc00) 230

[Phase Abstract](#_cs09xfupykc1) 238

[Phase Description](#_e3rkbm9yi5qq) 238

[Design Documentation](#_5a506f7ldzw) 246

[Source code](#_8y570i2nb0fe) 246

[Main.cpp](#_ncrux1rv3ah8) 246

[IPC.h](#_8vp4uccpoh1n) 257

[LinkedList.h](#_qz7ji86g8efv) 267

[MCB.h](#_f7n0leefjgvt) 276

[MemoryMgmt.h](#_om0oe6z7g1hz) 279

[Node.h](#_uayjc36lkstq) 296

[Queue.h](#_y3ydmo3pmxbe) 298

[Scheduler.h](#_5jiuvifxchf5) 303

[Semaphore.h](#_li081o69bui6) 315

[ThreadTable.h](#_wfni8nizsl9b) 321

[Window.h](#_cl0777q8pk8o) 332

[Ufs.h](#_sxsyp2wb1qmx) 339

[Inode.h](#_1zk4l9y8xmi6) 363

[driver.h](#_iw907i6infxa) 364

[Output](#_7hpga07fbufv) 374

[Demonstration of new Curses Library](#_fhe7pl85o4lm) 376

[Demonstration of File system](#_7qbzbtn7iet4) 377

[Demonstration of datafile](#_cscw4akdwbkw) 379

[Testing Strategy](#_wvimprkf98e2) 379

[Some performed tests PREVIOUS MERGE:](#_1qjvkvahpm9k) 380

[Testing Ultima](#_r2vgnofx2xup) 380

[Discussion](#_y7ty7hrupsc7) 380

[Project Schedule](#_jcdyyoi0ze8o) 392

# Phase I

## Scheduler and Semaphore

February 23, 2020

Previous group partner: Casey Radeline

## Phase Abstract

This phase is the start of the ULTIMA 2.0 project for the course C435 – Operating Systems at Indiana University of South Bend. The purpose of this phase is to develop a Scheduler mechanism which will be responsible to control running tasks during the execution of the program alongside the development of a Binary Semaphore which prevents certain critical parts of the code to be executed by more than one task at the time, thus, preventing problems such as race conditions. Furthermore, phase I required the implementation of a Thread Table in the form of a linked list to store information over create tasks as well as some Standard Libraries (Queue and Linked List).

*Keywords*: Scheduler, Binary Semaphore, Thread Table, race conditions

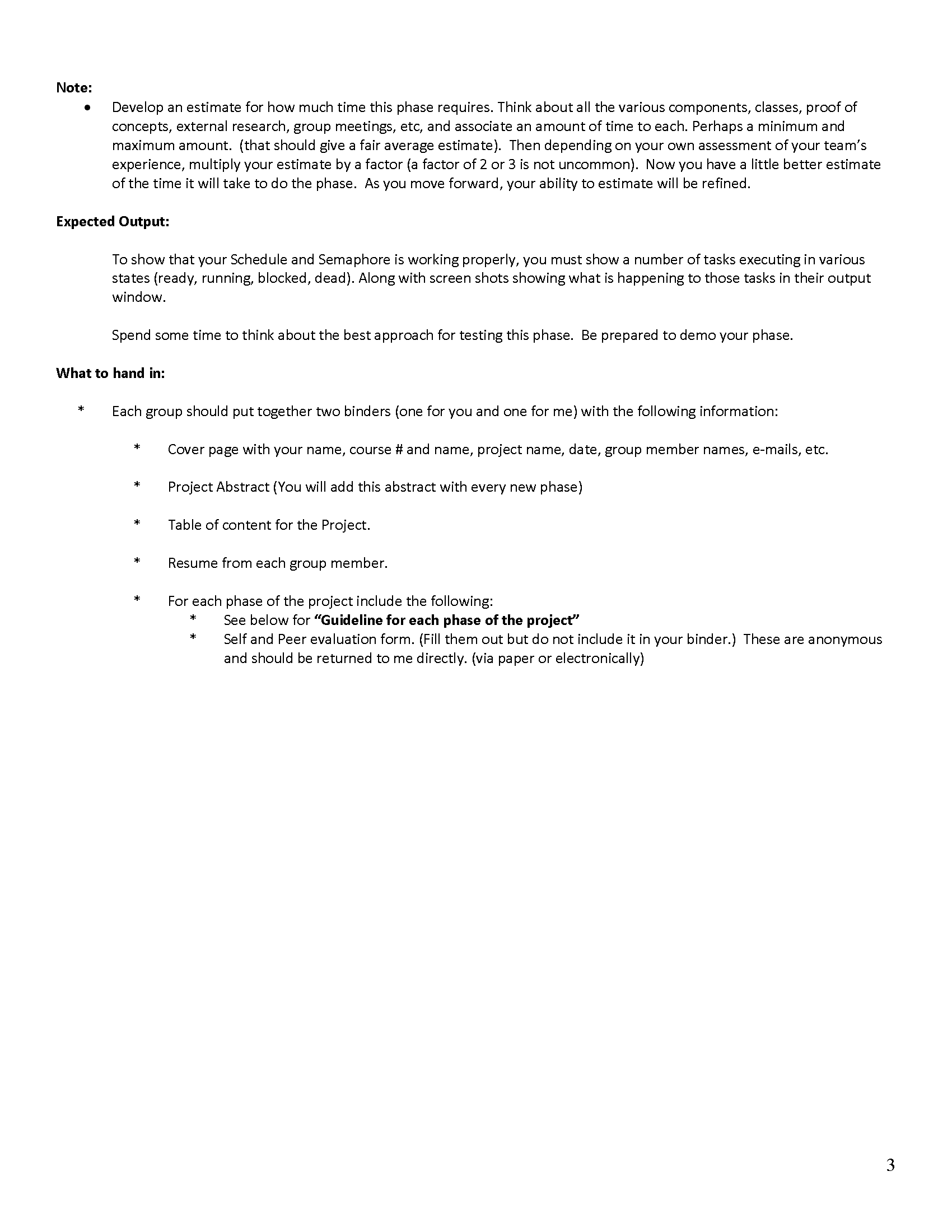
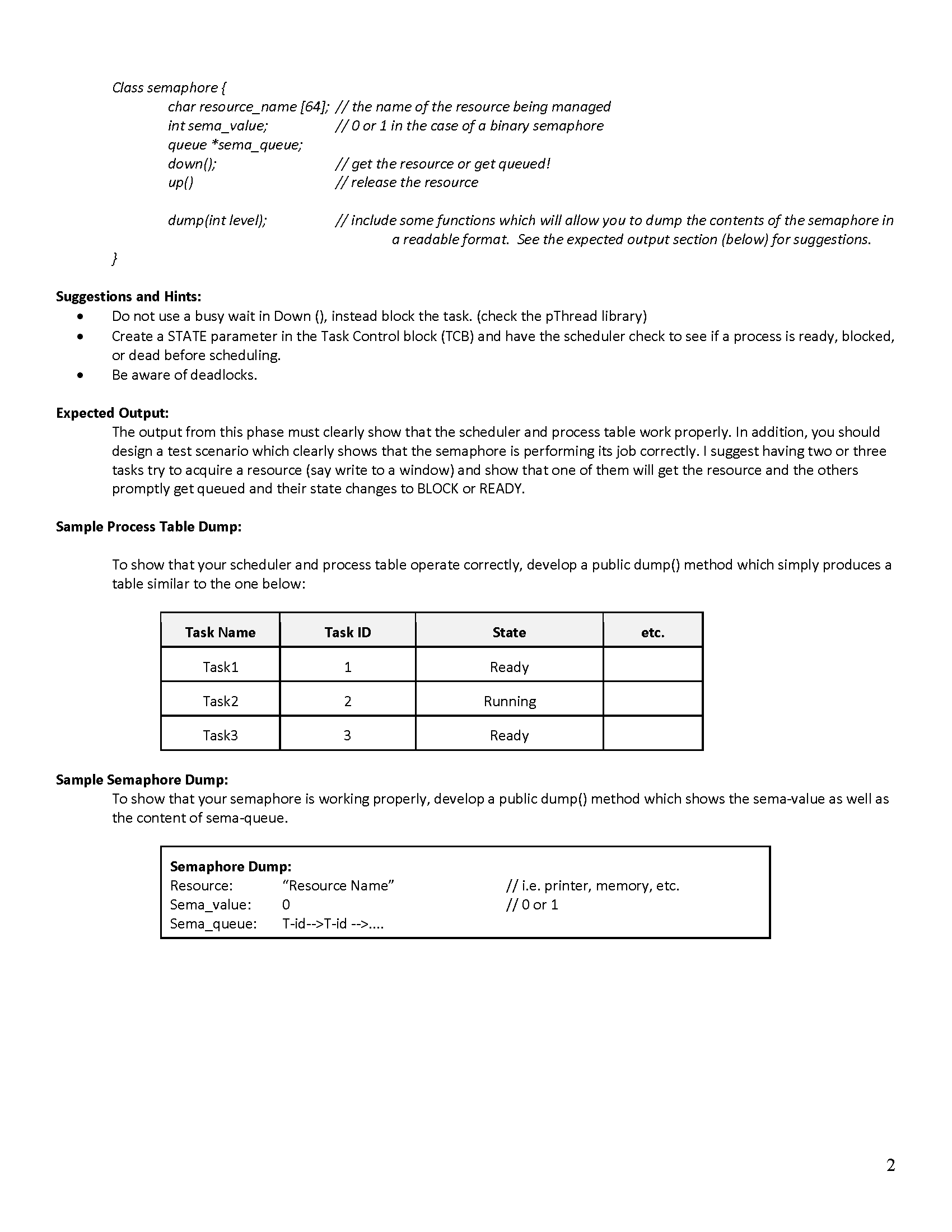
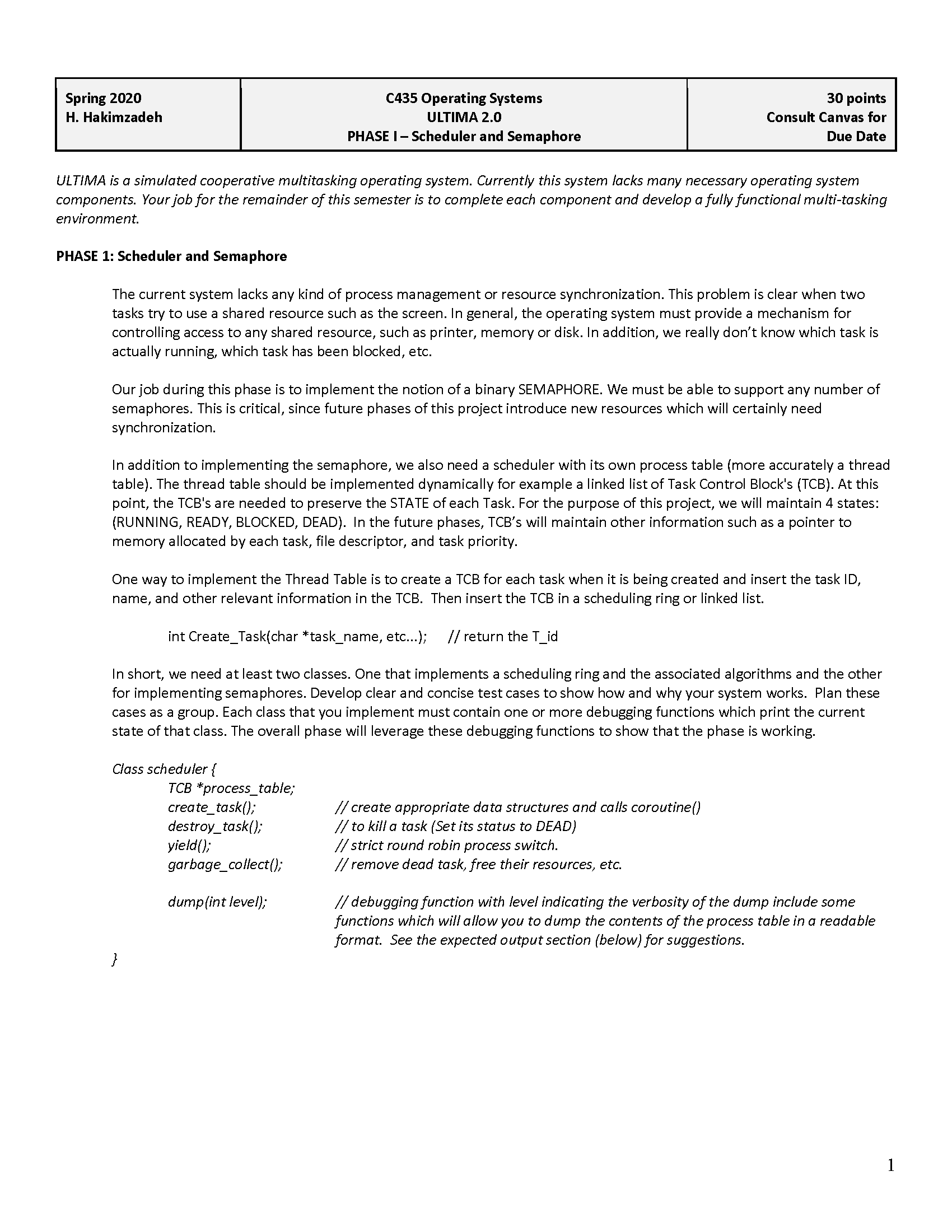
## 

## 

## 

## 

## Phase Description



## Design Documentation

## 

## 

## 

## 

## 

## 

## Source code

### Main.cpp

|  |
| --- |
| /\*  Filename: main.cpp  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To implement the main class for Ultima 2.0.Furthermore,  this main should call some implemented functions in Phase II  so that its functionality is demonstrated.  \*/  #include <iostream>  #include <sched.h>  #include <time.h>  #include <pthread.h>  #include "Semaphore.h"  #include "Scheduler.h"  using namespace std;  //StartMCB() starts the Master Control Block  //object by registering its variables to the  //correspondent class objects.  int main()  {  int t\_id;  Scheduler \*mysch = new Scheduler();  Semaphore \*mysema = new Semaphore(1, "resource1", mysch);    t\_id = mysch->create\_task();  t\_id = mysch->create\_task();  t\_id = mysch->create\_task();  t\_id = mysch->create\_task();  //Uncomment the below statements to  //test the Scheduler::destroyTask(int)  //and garbageCollector() functions.  //mysch->destroyTask(2);  //mysch->garbageCollector();  mysch->start();  //Binary Semaphore  //---------- DOWN  t\_id = mysch->getCurrTaskID();  cout << "Task " << t\_id << " is trying to obtain the semaphore (Resource1)" << endl;  mysema->down(t\_id); // hold on to the resource  mysch->wasteTime(3);  mysch->yield();  t\_id = mysch->getCurrTaskID();  cout << "Task " << t\_id << " is trying to obtain the semaphore (Resource1)" << endl;  mysema->down(t\_id); // hold on to the resource  mysch->wasteTime(3);  mysch->yield();  t\_id = mysch->getCurrTaskID();  cout << "Task " << t\_id << " is trying to obtain the semaphore (Resource1)" << endl;  mysema->down(t\_id); // hold on to the resource  mysch->wasteTime(3);  mysch->yield();  //---------- UP  cout << "START OF MY UP-----------------------" << endl<< endl;  t\_id = mysch->getCurrTaskID();  cout << "Task " << t\_id << " is trying to obtain the semaphore (Resource1)" << endl;  mysema->up(); // hold on to the resource  mysch->wasteTime(3);  mysch->yield();  t\_id = mysch->getCurrTaskID();  cout << "Task " << t\_id << " is trying to obtain the semaphore (Resource1)" << endl;  mysema->up(); // hold on to the resource  mysch->wasteTime(3);  mysch->yield();  t\_id = mysch->getCurrTaskID();  cout << "Task " << t\_id << " is trying to obtain the semaphore (Resource1)" << endl;  mysema->up(); // hold on to the resource  mysch->wasteTime(3);  mysch->yield();  return 0;  } |

### 

### LinkedList.h

|  |
| --- |
| /\*  Filename: LinkedList.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To create a linked list data structure.  \*/  #ifndef LINK\_LIST\_H  #define LINK\_LIST\_H  #include <iostream>  using namespace std;  #include "Node.h"  template <class T>  class LinkedList  {  private:  Node<T> \*head;  Node<T> \*tail;  int listSize;  public:  LinkedList();  ~LinkedList();  void insertFront(T data, int num = 0);  void insertBack(T data, int num = 0);  void removeFront();  void removeBack();  void removeNode(T data);  void removeNode(int num);  Node<T> \*front();  Node<T> \*back();  int size();  bool empty();  Node<T> \*searchNode(T data);  Node<T> \*searchNode(int data);  bool isNode(int data);  void dumpList();  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor.  template <class T>  LinkedList<T>::LinkedList()  {  head = tail = NULL;  listSize = 0;  }  //----------------------------------------  //Destructor.  template <class T>  LinkedList<T>::~LinkedList()  {  Node<T> \*tmp = head;  while (tmp)  {  tmp = tmp->next;  delete (head);  head = tmp;  }  head = tail = NULL;  listSize = 0;  }  //----------------------------------------  //insertFront(T, int) - Insert node at the begin  // of the linked list.  template <class T>  void LinkedList<T>::insertFront(T data, int num)  {  Node<T> \*newNode = new Node<T>;  newNode->data = data;  newNode->num = num;  newNode->next = NULL;  if (empty())  {  head = tail = newNode;  }  else  {  newNode->next = head;  head = newNode;  }  listSize++;  }  //----------------------------------------  //insertBack(T,int) - Insert node at the end of the linked list.  template <class T>  void LinkedList<T>::insertBack(T data, int num)  {  Node<T> \*newNode = new Node<T>;  newNode->data = data;  newNode->num = num;  newNode->next = NULL;  if (empty())  head = tail = newNode;  else  {  tail->next = newNode;  tail = newNode;  }  listSize++;  }  //----------------------------------------  //removeFront() - Remove node at the beginning of the list.  template <class T>  void LinkedList<T>::removeFront()  {  if (!empty())  {  Node<T> \*tmp = head->next;  delete (head);  head = tmp;  }  listSize--;  }  //----------------------------------------  //removeBack() - Remove node at the end of the list.  template <class T>  void LinkedList<T>::removeBack()  {  if (head->next == NULL)  delete (head);  else  {  Node<T> \*tmp = head;  while (tmp->next->next != NULL)  tmp = tmp->next;  delete (tmp->next);  tmp->next = NULL;  tail = tmp;  }  listSize--;  }  //----------------------------------------  //removeNode() - Remove node by specification type T.  template <class T>  void LinkedList<T>::removeNode(T data)  {  if (head->data == data)  removeFront();  else if (tail->data == data)  removeBack();  else  {  Node<T> \*tmp = head;  Node<T> \*prev;  while (tmp->next && tmp->data != data)  {  prev = tmp;  tmp = tmp->next;  }  if (tmp->data == data)  {  prev->next = tmp->next;  delete tmp;  listSize--;  }  else  exit(-1);  }  }  //----------------------------------------  //removeNode(int) - Remove node by specification type int.  template <class T>  void LinkedList<T>::removeNode(int num)  {  if (num == 0)  removeFront();  else if (listSize == num)  removeBack();  else  {  Node<T> \*tmp = head;  Node<T> \*prev;  int count = 0;  while (tmp->next && count != num)  {  prev = tmp;  tmp = tmp->next;  count++;  }  if (count == num)  {  prev->next = tmp->next;  delete tmp;  listSize--;  }  else  exit(-1);  }  }  //----------------------------------------  //front() - Return head/first node of the linked list.  template <class T>  Node<T>\*LinkedList<T>::front()  {  return head;  }  //back() - Return the last node of the linked list.  template <class T>  Node<T> \*LinkedList<T>::back()  {  return tail;  }  //----------------------------------------  //size() - Return size of linked list.  template <class T>  int LinkedList<T>::size()  {  return listSize;  }  //----------------------------------------  //empty() - Return if the linked list is empty or not.  template <class T>  bool LinkedList<T>::empty()  {  if (listSize > 0)  return false;  else  return true;  }  //----------------------------------------  //searchNode(T) - Search node by specification type T.  template <class T>  Node<T> \*LinkedList<T>::searchNode(T data)  {  Node<T> \*tmp = new Node<T>;  tmp = head;  while (tmp)  {  if (data == tmp->data)  return tmp;  tmp = tmp->next;  }  exit(-1);  }  //----------------------------------------  //searchNode(int) - Search node by specification type int.  template <class T>  Node<T> \*LinkedList<T>::searchNode(int data)  {  Node<T> \*tmp = new Node<T>;  tmp = head;  while (tmp)  {  if (data == tmp->num)  return tmp;  tmp = tmp->next;  }  exit(-1);  }  //----------------------------------------  //isNode(int) - Return if there is such a node in the list.  template <class T>  bool LinkedList<T>::isNode(int data)  {  Node<T> \*tmp = new Node<T>;  tmp = head;  while (tmp)  {  if (data == tmp->num)  return true;  tmp = tmp->next;  }  return false;  }  //----------------------------------------  //dumpList() - Output the linked list if there is one,  // if not exit with error.  template <class T>  void LinkedList<T>::dumpList()  {  Node<T> \*tmp = head;  if (!empty())  {  while (tmp != NULL)  {  cout << tmp->data << " ";  tmp = tmp->next;  }  cout << endl;  }  else  cout << "---- EMPTY ----" << endl;  }  //----------------------------------------  #endif |

### 

### Node.h

|  |
| --- |
| /\*  Filename: Node.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To define the structure of a Node for a linked list, or queue.  \*/  #ifndef NODE\_H  #define NODE\_H  #include <iostream>  using namespace std;  template <class T>  struct Node  {  T data;  int num;  Node <T> \*next;  };  #endif |

### 

### Queue.h

|  |
| --- |
| /\*  Filename: Queue.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To create a queue data structure.  \*/  #ifndef QUEUE\_LIST\_H  #define QUEUE\_LIST\_H  #include "LinkedList.h"  #include <iostream>  using namespace std;  #include "Node.h"  template <class T>  class Queue  {  private:  int queueSize;  Node <T> \*head;  public:  Queue();  ~Queue();  void enqueue(T data, int num = 0);  void dequeue();  bool empty();  Node<T>\* top();  int size();  void dumpQueue();  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor  template <class T>  Queue<T>::Queue()  {  head = NULL;  queueSize = 0;  }  //----------------------------------------  //Destructor  template <class T>  Queue<T>::~Queue()  {  Node<T> \*tmp = head;  while(tmp)  {  tmp = tmp->next;  delete(head);  head = tmp;  }  head = NULL;  queueSize = 0;  }  //----------------------------------------  //empty() - Returns if Queue is empty or not.  template <class T>  bool Queue<T>::empty()  {  if(size() > 0) return false;  else return true;  }  //----------------------------------------  //enqueue(T,int) - Insert node at the begin of  // the queue list.  template <class T>  void Queue<T>::enqueue(T data, int num)  {  Node<T> \*newNode = new Node<T>;  newNode->data = data;  newNode->num = num;  newNode->next = NULL;  if(empty())  {  head = newNode;  }  else  {  newNode->next = head;  head = newNode;  }  queueSize++;  }  //----------------------------------------  //dequeue() - Remove node at the beginning of the list.  template <class T>  void Queue<T>::dequeue()  {  if(!empty())  {  Node<T> \*tmp = head->next;  delete(head);  head = tmp;  }  queueSize--;  }  //----------------------------------------  //top() - Return the last node of the linked list.  template <class T>  Node<T>\* Queue<T>::top()  {  if(head)  return head;  else  {  cout << "Not found!" << endl;  }    }  //----------------------------------------  //size() - Return size of linked list.  template <class T>  int Queue<T>::size()  {  return queueSize;  }  //----------------------------------------  //dumpQueue() - Output all node in Queue.  template <class T>  void Queue<T>::dumpQueue()  {  Node<T> \*tmp = head;  if(!empty())  {  while(tmp!= NULL)  {  cout << tmp->data << " | " ;  tmp = tmp->next;  }  cout << endl;  }else cout << "---- EMPTY ----" << endl;  }  //----------------------------------------  #endif |

### Scheduler.h

|  |
| --- |
| /\*  Filename: IPC.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To implement a Round Robin scheduler mechanism.  \*/  #ifndef SCHEDULER\_H  #define SCHEDULER\_H  #ifndef THREAD\_STATES  #define THREAD\_STATES  enum STATE  {  READY,  RUNNING,  BLOCKED,  DEAD  };  #endif  const int NO\_OF\_THREADS = 3;  #include <pthread.h>  #include <iostream>  #include <unistd.h>  #include <assert.h>  #include "ThreadTable.h"  using namespace std;  class Scheduler  {  private:  int currentThread;  long currentQuantum;  int nextAvailableThreadID;  ThreadTable threadTable;  public:  Scheduler();  ~Scheduler();  int create\_task();  void \*worker(void \*arguments);  void start();  void yield();  void setQuantum(long quantum);  long getQuantum();  void setState(int task\_id, STATE the\_state);  int getCurrTaskID();  TCB\* getTCB(int task\_id);  void dump();  void wasteTime(int x);  void destroyTask(int task\_id);  void garbageCollector();  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor  Scheduler::Scheduler()  {  currentThread = -1;//No active process.  nextAvailableThreadID = 0;  currentQuantum = 300; //default quantum 300ms.  }  //----------------------------------------  //Destructor  Scheduler::~Scheduler()  {  currentThread = -1;//No active process.  nextAvailableThreadID = 0;  currentQuantum = 300; //default quantum 300ms.  }  //----------------------------------------  //CreateTask() -- Create an operating system process.  int Scheduler::create\_task()  {  if (nextAvailableThreadID < NO\_OF\_THREADS)  {  cout << "Creating task # " << nextAvailableThreadID << endl;  threadTable.insertTCB(nextAvailableThreadID, clock(), READY);  //Source: https://thispointer.com/c-how-to-pass-class-member-function-to-pthread\_create/  typedef void \*(\*THREADFUNCPTR)(void \*);  int err = pthread\_create(threadTable.getTCBProcessID(nextAvailableThreadID), NULL, (THREADFUNCPTR)&Scheduler::worker, this);  assert(!err);  nextAvailableThreadID++;  return (nextAvailableThreadID - 1);  }  else  {  cout << "Create task FAILED: Number of available threads exceeded the limit.." << endl;  return (-1);// return error in case cannot create a task.  }  }  //----------------------------------------  //\*worker() - Context swapping function.  void \*Scheduler::worker(void \*arguments)  {  int mythreadNum = 0;  //Find the TCB related to the thread running in the OS background.  while (\*threadTable.getTCBProcessID(mythreadNum) != pthread\_self())  mythreadNum++;  //As long as the OS background running thread is not DEAD  //and it is running do:  while ((threadTable.getTCBState(mythreadNum) != DEAD))  {  if (threadTable.getTCBState(mythreadNum) == RUNNING)  {  //Prints the pthread\_t of the running thread.  cout << "Thread Num# :" << mythreadNum  << " Pthread\_self() #: " << pthread\_self()  << endl;  //Print all information over the Scheduler class.  dump();  //Call Scheduler::yield() so that current thread can  //voluntarily give up the CPU to the next ready task.  yield();  }  else  {  //If not, just relinquish the CPU by the calling thread  //, insuring for example that the calling thread of  //a createTask() function call does not hold the CPU forever.  pthread\_yield();  }  }  }  //----------------------------------------  //start() - Initiate scheduling mechanism by  // setting the first task in thread table  // to running.  void Scheduler::start()  {  cout << "............." << endl;  cout << "............. STARTING SCHEDULING" << endl;  cout << ".............\n"  << endl;  TCB \*tmp = threadTable.getTCBHead();  //Find the first READY process  //by jumping through DEAD processes  while (tmp->threadState == DEAD)  tmp = tmp->next;  //Set the first found READY process to state RUNNING, then begin scheduling  threadTable.setTCBStartTime(tmp->threadID, clock());  threadTable.setTCBState(tmp->threadID, RUNNING);  currentThread = tmp->threadID;  setQuantum(1000 / NO\_OF\_THREADS); // Set quantum to 1sec/number of threads  wasteTime(2);  }  //----------------------------------------  //yield() - runs the thread with set RUNNING state until  // its quantum runs out. When quantum runs out,  // set current RUNNING to READY then finds the next  // READY thread then set its state to RUNNING. Finally,  // puts the OS thread current running in background to sleep  // so that the next can have its turn to run.  void Scheduler::yield()  {  cout << "-----------------------------------------------------------\n"  << endl;  int counter = 0;  cout << "Current Task # " << currentThread << " is trying to yield." << endl;  cout << "Current Quantum : " << currentQuantum << endl;  // calculate elapsed\_time since the thread last started to run.  clock\_t elapsed\_time = clock() - threadTable.getTCBStartTime(currentThread);  cout << "Elapsed time: " << elapsed\_time << endl;  if (elapsed\_time >= currentQuantum) // if quantum has run out  {  cout << "yielding....(Switching from task #" << currentThread << " to next ready task)." << endl;  // if current thread is RUNNING make it READY (its quantum has run out)  if (threadTable.getTCBState(currentThread) == RUNNING)  threadTable.setTCBState(currentThread, READY);  // now find the next READY thread and make it running  // watch out for deadlocks. (no ready processes)  currentThread = (currentThread + 1) % NO\_OF\_THREADS;  if (threadTable.findTCB(currentThread) == false)  currentThread = (currentThread + 1) % NO\_OF\_THREADS;  while (threadTable.getTCBState(currentThread) != READY && counter < NO\_OF\_THREADS - 1)  {  currentThread = (currentThread + 1) % NO\_OF\_THREADS;  if (threadTable.findTCB(currentThread) == false)  currentThread = (currentThread + 1) % NO\_OF\_THREADS;  }  // if we find a READY threads re-set the quantum and set the task to running  if (counter < NO\_OF\_THREADS - 1 && threadTable.getTCBState(currentThread) == READY)  {  threadTable.setTCBStartTime(currentThread, clock()); // restart the quantum  threadTable.setTCBState(currentThread, RUNNING);  cout << "Started Running task # " << currentThread << endl;  //Sleep(1) in here, insures synchronization between the  //threads and the yield function.  sleep(1);  }  else  cout << "----- P O S S I B L E D E A D L O C K -----" << endl;  }  else  cout << "---- N O Y I E L D -----" << endl;  }  //----------------------------------------  //setQuantum(long quantum) - Assign a passed value  // to currentQuantum.  void Scheduler::setQuantum(long quantum)  {  currentQuantum = quantum;  }  //----------------------------------------  //getQuantum() - Return currentQuantum value  long Scheduler::getQuantum()  {  return (currentQuantum);  }  //----------------------------------------  //setState(int,STATE) - Set the state of a specific task to  // a specific state.  void Scheduler::setState(int task\_id, STATE the\_state)  {  threadTable.setTCBState(task\_id, the\_state);  }  //----------------------------------------  //getTaskID() - Return currentThread ID.  int Scheduler::getCurrTaskID()  {  return currentThread;  }  //----------------------------------------  //getTCB() - Return a pointer to TCB searched  // by threadID.  TCB \*Scheduler::getTCB(int task\_id)  {  TCB \*tmp = threadTable.getTCBHead();  while (tmp)  {  if (task\_id == tmp->threadID)  return tmp;  tmp = tmp->next;  }  cout << "getTCB - TCB not found! Return exit(-1)." << endl;  exit(-1);  }  //----------------------------------------  //dump() - Output important messages over the Scheduler.  void Scheduler::dump()  {  cout << "---------------------- PROCESS TABLE ----------------------" << endl;  clock\_t elapsed\_time = clock() - threadTable.getTCBStartTime(currentThread);  cout << "Quantum = " << currentQuantum << endl;  TCB \*tmp = threadTable.getTCBHead();  while (tmp != NULL)  {  cout << " task id: " << tmp->threadID;  cout << " process id" << tmp->processID;  cout << " task state: ";  switch (tmp->threadState)  {  case 0:  cout << "READY";  break;  case 1:  cout << "RUNNING";  break;  case 2:  cout << "BLOCKED";  break;  case 3:  cout << "DEAD";  break;  }  if (tmp->threadID == currentThread)  cout << " <-- CURRENT PROCESS";  cout << endl;  tmp = tmp->next;  }  cout << "-----------------------------------------------------------\n"  << endl;  }  //----------------------------------------  //wasteTime(int) - runs a for loop for (x \*  // 10240) amount of times.  void Scheduler::wasteTime(int x)  {  unsigned long long Int64 = 0;  for (unsigned short i = 0; i < 10240 \* x; ++i)  {  for (unsigned short j = i; j > 0; --j)  Int64 += j + i;  }  }  //----------------------------------------  //garbageCollector() - Goes through the thread  // table finding all the  // state DEAD threads,  // after finding it, removes  // its TCB from the thread table.  void Scheduler::garbageCollector()  {  cout << "Garbage collector " << endl;  TCB \*tmp = threadTable.getTCBHead();  while (tmp != NULL)  {  if (tmp->threadState == DEAD)  {  threadTable.removeATCB(tmp->threadID);  }  tmp = tmp->next;  }  }  //----------------------------------------  //destroyTask(int) - Sets the state of a task  // to DEAD, then calls a pthread  // library function pthread\_cancel  // to cancel the running task in the  // background.  void Scheduler::destroyTask(int task\_id)  {  threadTable.setTCBState(task\_id, DEAD);  pthread\_cancel(\*threadTable.getTCBProcessID(task\_id));  }  //----------------------------------------  #endif |

### 

### Semaphore.h

|  |
| --- |
| /\*  Filename: Semaphore.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To implement a binary semaphore.  \*/  #ifndef Semaphore\_H  #define Semaphore\_H  #include <iostream>  #include <string>  using namespace std;  #include "Queue.h"  #include "Scheduler.h"  class Semaphore  {  private:  string resourceName; // The name of the resource being managed.  int semaValue; // 0 or 1 in the case of a binary Semaphore.  int luckyTask; // Preserve the task-id of the task that got the resource (for debugging).  pthread\_mutex\_t mux; // Critical section lock mechanism.  Queue<int> semaQueue;  Scheduler \*schedPtr;  public:  Semaphore(int starting\_value, string name, Scheduler \*theScheduler);  ~Semaphore();  void down(int taskID);  void up();  void dump(int level);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor  Semaphore::Semaphore(int starting\_value, string name, Scheduler \*theScheduler)  {  semaValue = starting\_value;  resourceName = name;  luckyTask = -1;  schedPtr = theScheduler;  pthread\_mutex\_init(&mux,NULL);  }  //---------------------------------------------  //Destructor  Semaphore::~Semaphore()  {  while(semaQueue.size() > 0)  semaQueue.dequeue();  semaValue = -99;  resourceName = "";  luckyTask = -1;  pthread\_mutex\_destroy(&mux);  }  //---------------------------------------------  //down() - Get the hold of the resource or get queued.  void Semaphore::down(int taskID)  {  pthread\_mutex\_lock(&mux);  if (taskID == luckyTask)  {  cout << "Task # " << luckyTask << " already has the resource! Ignore the request." << endl;  dump(1);  }  else  {  if (semaValue >= 1)  {  semaValue--;  luckyTask = taskID; // preserve the task-id who got the resource  dump(1);  }  else  {  semaQueue.enqueue(taskID);  schedPtr->setState(taskID, BLOCKED);  dump(1);  schedPtr->yield();  dump(1);  }  }  pthread\_mutex\_unlock(&mux);  }  //---------------------------------------------  //up() - Release the hold of the resource.  void Semaphore::up()  {  pthread\_mutex\_lock(&mux);  int task\_id;  cout << "TaskID : " << schedPtr->getCurrTaskID() << ", LuckyID : " << luckyTask << endl;  if (schedPtr->getCurrTaskID() == luckyTask) // check to see if the correct task is doing the up()  {  if (semaQueue.empty())  {  semaValue++;  luckyTask = -1;  dump(1);  }  else  {  task\_id = semaQueue.top()->data; // Remove from queue and unblock  semaQueue.dequeue();  schedPtr->setState(task\_id, READY); // set the task to READY  cout << "UnBlock : " << task\_id << " and release from the queue." << endl;  luckyTask = task\_id; //  cout << "Luck Task = " << luckyTask << endl;  dump(1);  schedPtr->yield();  dump(1);  }  }  else  {  cout << "Invalid Semaphore UP(). TaskID : " << schedPtr->getCurrTaskID() << " does not own the resource." << endl;  dump(1);  }  pthread\_mutex\_unlock(&mux);  }  //---------------------------------------------  //dump() - Output information over the Semaphore.  void Semaphore::dump(int level)  {  cout << "-------Semaphore DUMP-------" << endl;  switch (level)  {  case 0:  cout << "Sema\_Value: " << semaValue << endl;  cout << "Sema\_Name : " << resourceName << endl;  cout << "Obtained by Task-ID: " << luckyTask << endl;  break;  case 1:  cout << "Sema\_Value: " << semaValue << endl;  cout << "Sema\_Name : " << resourceName << endl;  cout << "Obtained by Task-ID: " << luckyTask << endl;  cout << "Sema-Queue: ";  semaQueue.dumpQueue();  break;  default:  cout << "ERROR in Semaphore DUMP level";  }  cout << "----------------------------" << endl;  }  #endif |

### 

### ThreadTable.h

|  |
| --- |
| /\*  Filename: ThreadTable.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To implement a thread table structure alongside with functions to maneuver it.  \*/  #ifndef THREAD\_TABLE\_H  #define THREAD\_TABLE\_H  #include <iostream>  using namespace std;  #ifndef THREAD\_STATES  #define THREAD\_STATES  enum STATE  {  READY,  RUNNING,  BLOCKED,  DEAD  };  #endif  struct TCB  {  enum STATE threadState;  int threadID;  pthread\_t processID;  clock\_t threadStartTime;  TCB \*next;  };  class ThreadTable  {  private:  TCB \*head;  TCB \*tail;  int threadTableSize;  public:  ThreadTable();  ~ThreadTable();  bool empty();  void setTCBStartTime(int mythreadID, clock\_t myStartTime);  clock\_t getTCBStartTime(int mythreadID);  void insertTCB(int mythreadID, clock\_t myStartTime, STATE mythreadState);  void removeATCB(int mythreadID);  void removeLastTCB();  void removeFirstTCB();  bool findTCB(int mythreadID);  TCB \*getTCB(int mythreadID);  pthread\_t \*getTCBProcessID(int mythreadID);  TCB \*getTCBHead();  TCB \*getTCBTail();  int getThreadTableSize();  void setTCBState(int mythreadID, STATE myState);  STATE getTCBState(int mythreadID);  void dumpThreadTable();    };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor.  ThreadTable::ThreadTable()  {  head = NULL;  threadTableSize = 0;  }  //----------------------------------------  //Destructor.  ThreadTable::~ThreadTable()  {  TCB \*tmp = head;  while (tmp)  {  tmp = tmp->next;  delete (head);  head = tmp;  }  head = tail = NULL;  threadTableSize = 0;  }  //----------------------------------------  //empty() - Return if the thread table is empty.  bool ThreadTable::empty()  {  if (threadTableSize == 0)  return true;  else  return false;  }  //----------------------------------------  //setTCBStartTime(int,clock\_t) - Set the value of the thread  // start time to a specified time.  void ThreadTable::setTCBStartTime(int mythreadID, clock\_t myStartTime)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  {  tmp->threadStartTime = myStartTime;  break;  }  tmp = tmp->next;  }  }  //----------------------------------------  //getTCBStartTime(int) - Get the start time of a specified  // thread number.  clock\_t ThreadTable::getTCBStartTime(int mythreadID)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  return tmp->threadStartTime;  tmp = tmp->next;  }  }  //----------------------------------------  //insertTCB(int, clock\_t, STATE) - Insert a new TCB block at  // the end of thread Table.  void ThreadTable::insertTCB(int mythreadID, clock\_t myStartTime, STATE mythreadState)  {  TCB \*newBlock = new TCB;  newBlock->threadID = mythreadID;  newBlock->threadStartTime = myStartTime;  newBlock->threadState = mythreadState;  newBlock->next = NULL;  if (empty())  head = tail = newBlock;  else  {  tail->next = newBlock;  tail = newBlock;  }  threadTableSize++;  }  //----------------------------------------  //removeATCB(int) - Remove a specific TCB of the thread table.  void ThreadTable::removeATCB(int threadID)  {  if (head->threadID == threadID)  removeFirstTCB();  else if (tail->threadID == threadID)  removeLastTCB();  else  {  TCB \*tmp = head;  TCB \*prev;  while (tmp->next && tmp->threadID != threadID)  {  prev = tmp;  tmp = tmp->next;  }  if (tmp->threadID == threadID)  {  prev->next = tmp->next;  delete tmp;  threadTableSize--;  }  else  exit(-1);  }  }  //----------------------------------------  //removeLastTCB() - Remove last TCB block of thread table.  void ThreadTable::removeLastTCB()  {  if (head->next == NULL)  delete (head);  else  {  TCB \*tmp = head;  while (tmp->next->next != NULL)  tmp = tmp->next;  delete (tmp->next);  tmp->next = NULL;  tail = tmp;  }  threadTableSize--;  }  //----------------------------------------  //removeFirstTCB() - Remove first TCB block of thread table.  void ThreadTable::removeFirstTCB()  {  if (!empty())  {  TCB \*tmp = head->next;  delete (head);  head = tmp;  }  threadTableSize--;  }  //----------------------------------------  //findTCB(int) - Return a thread specific TCB.  bool ThreadTable::findTCB(int mythreadID)  {  TCB \*tmp = head;  while (tmp)  {  if (mythreadID == tmp->threadID)  return true;  tmp = tmp->next;  }  return false;  }  //----------------------------------------  //getTCB() - Return a pointer to TCB searched by threadID.  TCB \*ThreadTable::getTCB(int mythreadID)  {  TCB \*tmp = head;  while (tmp)  {  if (mythreadID == tmp->threadID)  return tmp;  tmp = tmp->next;  }  cout << "getTCB - TCB not found! Return exit(-1)." << endl;  exit(-1);  }  //----------------------------------------  //getTCBProcessID(int) - Return a process ID pthread\_t of a TCB.  pthread\_t \*ThreadTable::getTCBProcessID(int mythreadID)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  return &tmp->processID;  tmp = tmp->next;  }  cout << "getTCBProcessID - ProcessID not found! Return exit(-1)." << endl;  exit(-1);  }  //getTCBHead() - Return a pointer to the first TCB in the thread table.  TCB \*ThreadTable::getTCBHead()  {  return head;  }  //----------------------------------------  //getTCBTail() - Return a pointer to the last TCB in the thread table.  TCB \*ThreadTable::getTCBTail()  {  return tail;  }  //----------------------------------------  //getThreadTableSize() - Return thread table size.  int ThreadTable::getThreadTableSize()  {  return threadTableSize;  }  //----------------------------------------  //setTCBState(int,STATE) - Set TCB state of a specified threadID.  void ThreadTable::setTCBState(int mythreadID, STATE myState)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  {  tmp->threadState = myState;  break;  }  tmp = tmp->next;  }  }  //----------------------------------------  //getTCBState(int) - Get a specified TCB state.  STATE ThreadTable::getTCBState(int mythreadID)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  {  return tmp->threadState;  }  tmp = tmp->next;  }  cout << "GetTCBState - TCB not found! Return exit(-1)." << endl;  exit(-1);  }  //----------------------------------------  //dumpThreadTable() - Output everything from the thread table.  void ThreadTable::dumpThreadTable()  {  cout << "..... START ThreadTable Dump ..... " << endl;  TCB \*tmp = head;  if (!empty())  {  while (tmp != NULL)  {  cout << "TaskNUM: " << tmp->threadID << endl;  cout << "TaskID: " << tmp->threadID << endl;  cout << "TaskState: ";  switch (tmp->threadState)  {  case 0:  cout << "READY" << endl;  break;  case 1:  cout << "RUNNING" << endl;  break;  case 2:  cout << "BLOCKED" << endl;  break;  case 3:  cout << "DEAD" << endl;  break;  }  cout << "..............................." << endl;  tmp = tmp->next;  }  cout << "..... END ThreadTable Dump ..... " << endl;  }  else  {  cout << "EMPTY TCB - RETURN EXIT(-1)" << endl;  exit(-1);  }  }  //----------------------------------------  #endif |

### 

## 

## Output

## 

|  |
| --- |
|  |
|  |

## 

## Testing Strategy

During the implementation of Phase I, software testing techniques were used such as Unit Testing. The objective was to guarantee the proper functioning of new and old components of the project. Furthermore, small proof of concepts were implemented separately of the project itself for study, design, and analysis.

To exemplify, small proof of concepts over the Thread Table class was implemented such that each member function of the class would be tested to guarantee its proper functionality. Moreover, the Queue and Linked List data structures were similarly tested.

Besides, the Scheduler mechanism as well as the Semaphore were equally tested as in the lab 6 of the C435 Operating System course taught by Dr. Hakizamdeh. Moreover, scenarios such as doing a Down on the Semaphore but not an up to release the resource to analyse the behavior of the program were tested to ensure that the class Semaphore were correctly releasing and locking its resources by the right running task.

Finally, after applying the Unit Testing approach, phase I was correctly functioning specially the Scheduler mechanism that presented issues in the past submission.

## 

## Discussion

After the first phase I implementation, much effort was made to correct the Scheduler class alongside to make an algorithm that would allow Scheduling in a Round Robin way. Moreover, the assessment of static variables for some variables such as a static mutex variable, and a static Thread Table variable were made.

The implementation of the Linked List, and Queue data structures remained the same as well as the format of the Thread Table. As before, the Thread Table class is similar to a regular Linked List class, however, with some specialties to it. Differently from a Linked List, the Thread Table class contains member functions that are special in its return types by for example returning a pthread\_t type when getting the ID of a thread.

The Scheduler class passed for major adjustments taking into account lab 6 provided during the C435 Operating System class where it implements a small framework for what could be Ultima 2.0 and lab 5 where it demonstrates the implementation and maneuver of tasks created by the C pthread library. The first major change was with the worker function followed by changes in the yield function.

The worker function is responsible to follow the creation of a task by the C pthread library, its purpose is to find the current running thread’s information that is stored in the Thread Table data structure as a node then as long as it is task state is not set to dead, the task should run by calling the function yield. Thus, the yield function should run the task as long as there is quantum available. Moreover, if a task runs out of quantum, the task should give up the CPU by calling the sleep function in which the current running task suspends its execution from a given number of seconds allowing the next task to run. Previously, Phase I tried to implement the Scheduling of tasks by using special functions from the pthread library such as pthread\_cond\_signal and cond\_wait, however, due to its complexity a busy waiting algorithm was used instead.

# Phase II

## Message Passing (IPC)

April 4, 2020

Individual work

## 

## Phase Abstract

This phase is the second part of the ULTIMA 2.0 project for the course C435 – Operating Systems at Indiana University of South Bend. The purpose of this phase is to develop a message passing mechanism which will be responsible to send messages through running tasks during the execution of the program alongside the development of a Master Control Block which registers the objects of other fundamental mechanisms such as the Scheduler, Semaphore, and message passing mechanism (IPC). Furthermore, phase II required the implementation of two new classes: the IPC (Inter process communication), and the MCB (Master Control Block).

*Keywords*: IPC, MCB, message passing, master control block

## 

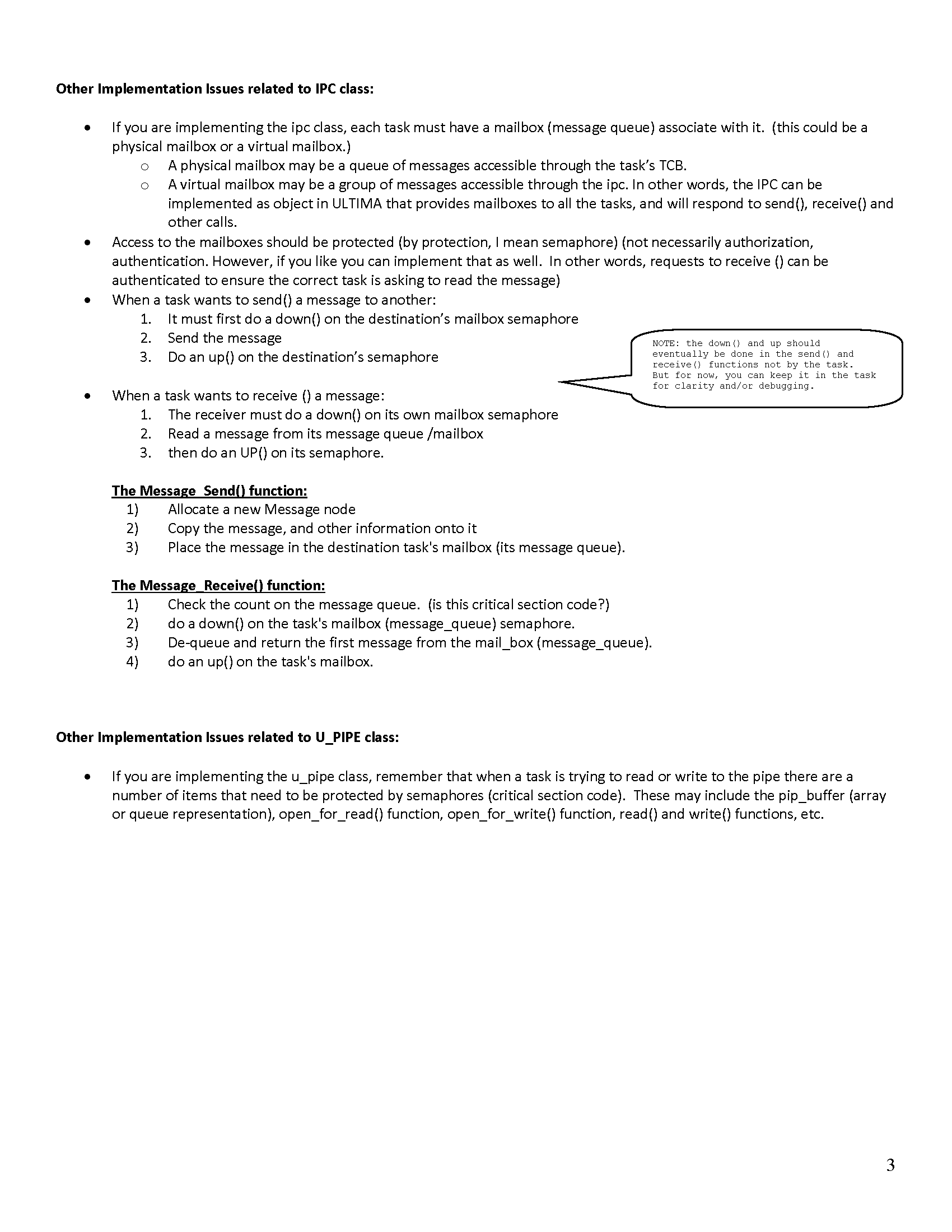
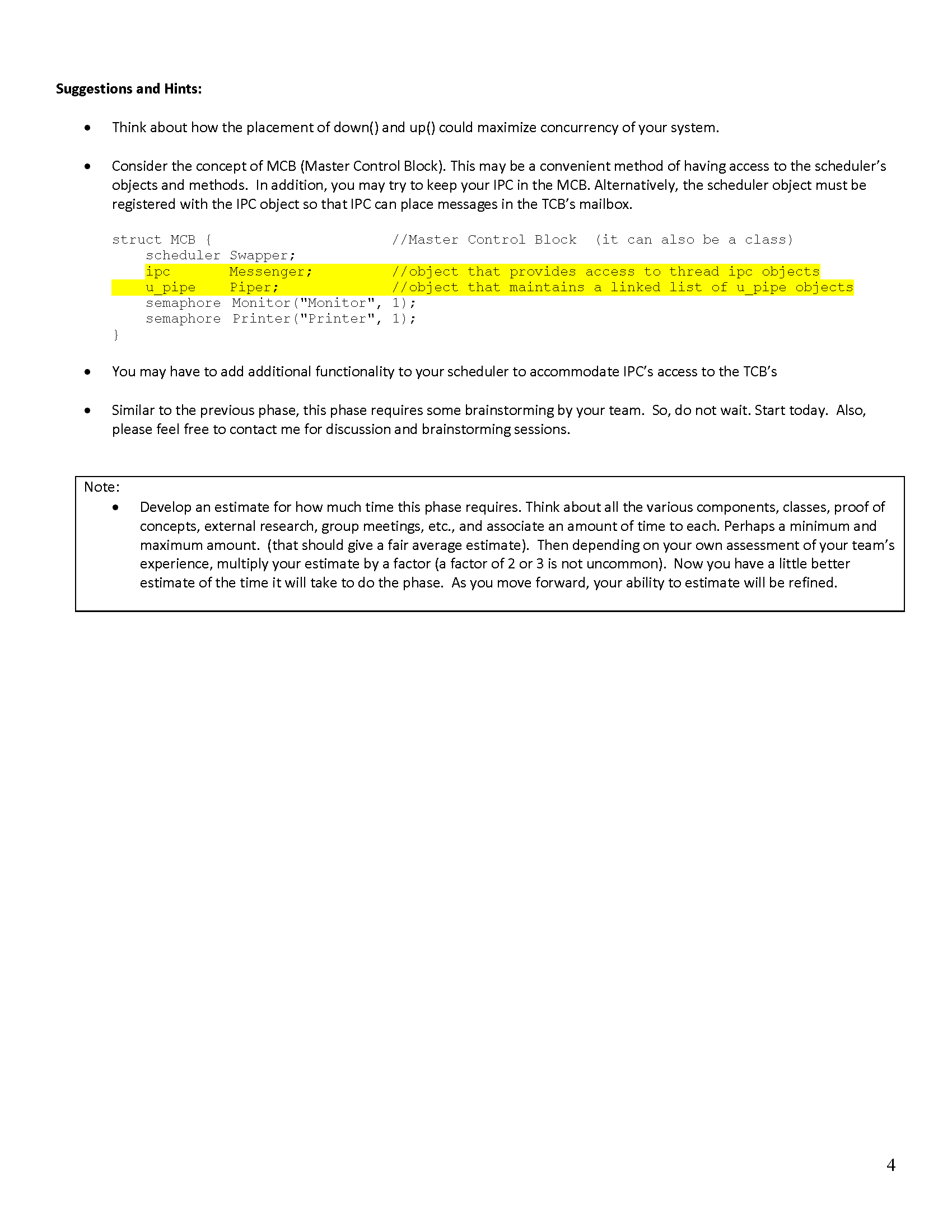
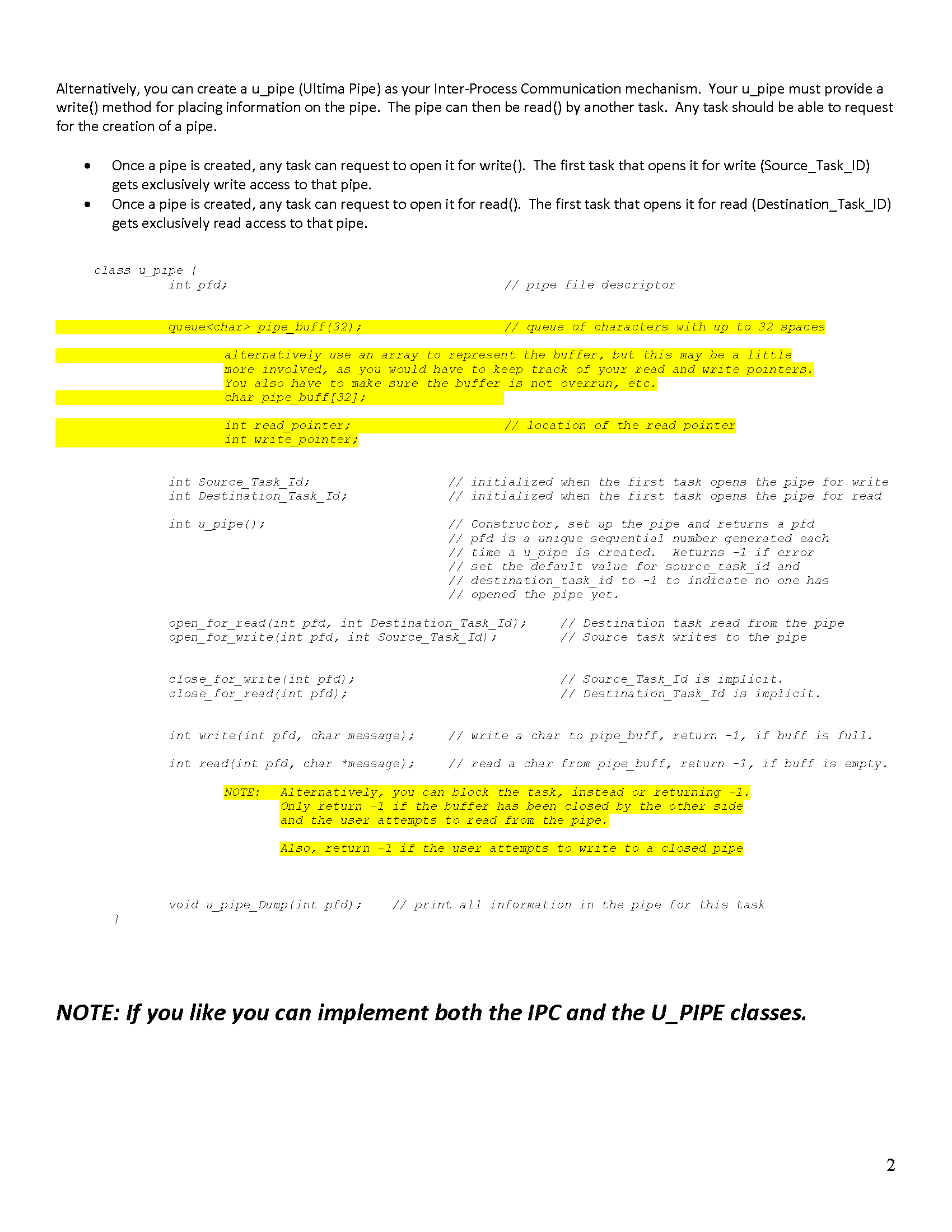
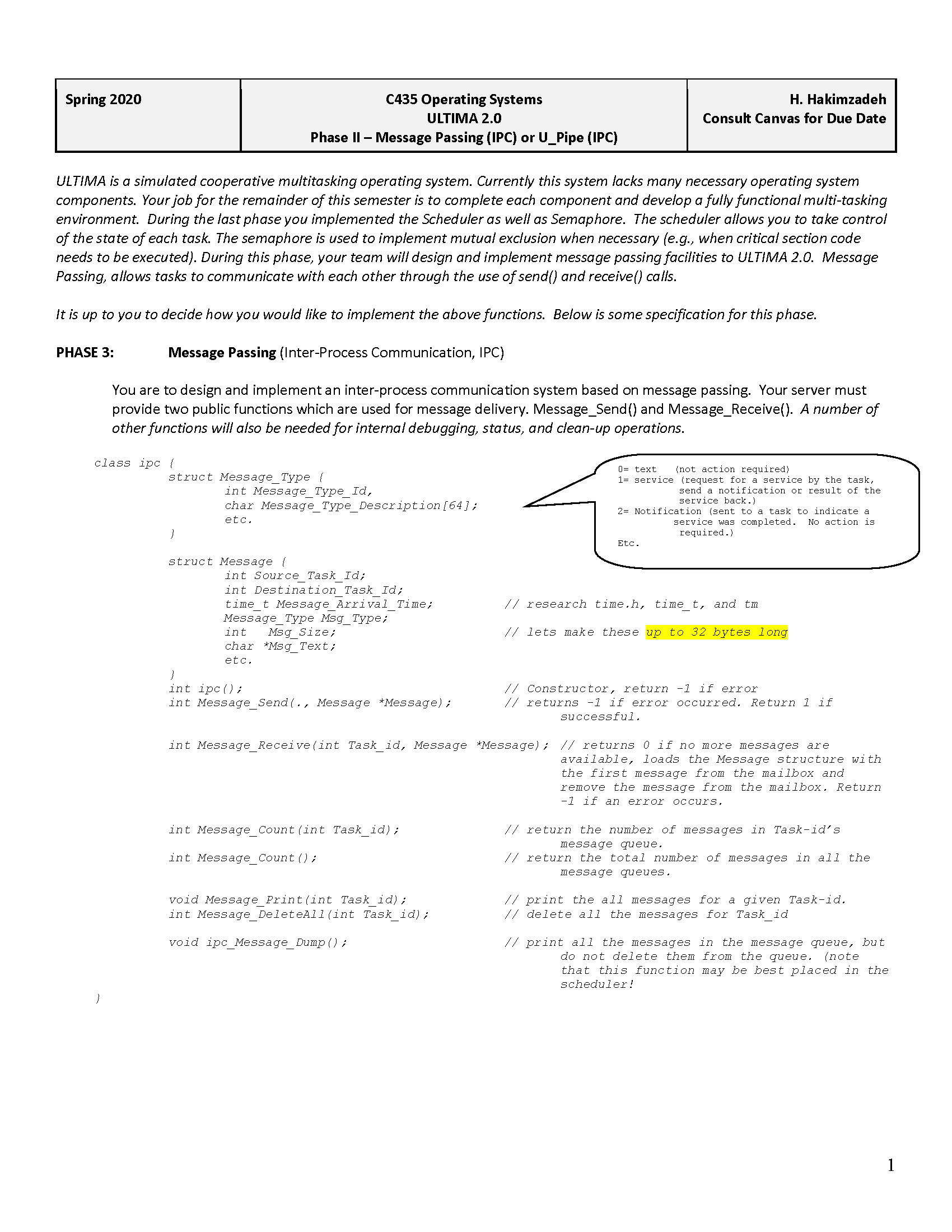
## 

## 

## 

## 

## Phase Description



## Design Documentation

## 

## Source code

### Main.cpp

|  |
| --- |
| /\*  Filename: main.cpp  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To implement the main class for Ultima 2.0.Furthermore,  this main should call some implemented functions in Phase II  so that its functionality is demonstrated.  \*/  #include <iostream>  #include <sched.h>  #include <time.h>  #include <pthread.h>  #include "Semaphore.h"  #include "Scheduler.h"  using namespace std;  #include "MCB.h"  //StartMCB() starts the Master Control Block  //object by registering its variables to the  //correspondent class objects.  void startMCB()  {  Scheduler \*newScheduler = new Scheduler();  Semaphore \*newSemaphore = new Semaphore(1,"resource1");  IPC \*newIPC = new IPC();  mcb = new MCB(newScheduler,newSemaphore,newIPC);  mcb->mysch->setMCB(mcb);  mcb->mysema->setMCB(mcb);  }  int main()  {  int t\_id;  startMCB();  t\_id = mcb->mysch->create\_task();  t\_id = mcb->mysch->create\_task();  t\_id = mcb->mysch->create\_task();  t\_id = mcb->mysch->create\_task();  //Uncomment the below statements to  //test the Scheduler::destroyTask(int)  //and garbageCollector() functions.  //mcb->mysch->destroyTask(2);  //mcb->mysch->garbageCollector();  mcb->mysch->start();  cout << endl << endl;  //Task 0 sends a message to Task 1  int err = mcb->myIPC->sendMessage(0,1,TEXT,"Testing my main...");  if(err == 0)  cout << "Sending message from Task 0 to Task 1" << endl;  else  cout << "Impossible to send the message!" << endl;  cout << endl << endl;  //Bellow statements test the implemented class in Phase I: the  //Binary Semaphore  //---------- DOWN  t\_id = mcb->mysch->getCurrTaskID();  cout << "Task " << t\_id << " is trying to obtain the semaphore (Resource1)" << endl;  mcb->mysema->down(t\_id); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  cout << "Task " << t\_id << " is trying to obtain the semaphore (Resource1)" << endl;  mcb->mysema->down(t\_id); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  cout << "Task " << t\_id << " is trying to obtain the semaphore (Resource1)" << endl;  mcb->mysema->down(t\_id); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  //---------- UP  cout << "START OF MY UP-----------------------" << endl<< endl;  t\_id = mcb->mysch->getCurrTaskID();  cout << "Task " << t\_id << " is trying to obtain the semaphore (Resource1)" << endl;  mcb->mysema->up(); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  cout << "Task " << t\_id << " is trying to obtain the semaphore (Resource1)" << endl;  mcb->mysema->up(); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  //Uncomment the below statements so that a second message is  //sent by Task 0 to Task 2  //mcb->myIPC->receiveMessage(1);  //mcb->myIPC->sendMessage(0,2,TEXT,"Testing my main2...");  t\_id = mcb->mysch->getCurrTaskID();  cout << "Task " << t\_id << " is trying to obtain the semaphore (Resource1)" << endl;  mcb->mysema->up(); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();    //Dumping entire IPC();  mcb->myIPC->dumpIPC();    //Uncomment the below statement to  //dump an specific mailbox  //mcb->myIPC->dumpMailbox(1);  return 0;  } |

## 

### IPC.h

|  |
| --- |
| /\*  Filename: IPC.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To define and create an inter process communication mechanism (IPC).  \*/  #ifndef IPC\_H  #define IPC\_H  #include<iostream>  #include<string>  #include<ctime> //So that arrival time is output pretty  #include "LinkedList.h"  #include "Queue.h"  using namespace std;  #ifndef MSG\_TYPE  #define MSG\_TYPE  enum MSGTYPE  {  TEXT, //No action required.  SERVICE, //Request for a service by the task.  NOTIFICATION// Sent to a task to indicate a service was completed.  };  #endif  struct Message  {  int src\_id;  int dst\_id;  string arrived\_time;  MSGTYPE msg\_type;  string msg\_content;  };  class IPC  {  public:  LinkedList<Queue<Message>> mailboxes;  int createMailbox(int task\_id);  int sendMessage(int src\_id, int dst\_id, MSGTYPE msg\_type, string msg\_content);  void receiveMessage(int task\_id);  int messageCount(int task\_id);  void removeAMailbox(int task\_id);  void deleteAllMailboxes();  void dumpIPC();  void dumpMailbox(int task\_id);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //createMailbox(int) - Create a mailbox queue inside the  // linked list Mailboxes so that a  // Scheduler::createTask thread can  // have its own mailbox "address".  int IPC::createMailbox(int task\_id)  {  Queue<Message> newBox;  mailboxes.insertBack(newBox,task\_id);  //Return 0 if successful created the box, or -1 if fails  if(mailboxes.size() > 0)  return 0;  else  return -1;  }  //----------------------------------------  //sendMessage() - Sends a message from a thread source  // to a thread destination. The message  // is enqueued in the destination thread's  // mailbox.  int IPC::sendMessage(int src\_id, int dst\_id, MSGTYPE msg\_type,string msg\_content)  {  time\_t now = time(0);  Message newMsg;  newMsg.arrived\_time = asctime(localtime(&now)); //https://www.tutorialspoint.com/cplusplus/cpp\_date\_time.htm  newMsg.dst\_id = dst\_id;  newMsg.src\_id = src\_id;  newMsg.msg\_type = msg\_type;  newMsg.msg\_content = msg\_content;  //Check if there the destination's mailbox exist  if(mailboxes.isNode(dst\_id))  {  mailboxes.searchNode(dst\_id)->data.enqueue(newMsg,dst\_id);  return 0;  }  else  {  cout << "Impossible to send message, mailbox does not exist!" << endl;  return -1;  }  }  //----------------------------------------  //receiveMessage(int) - Goes into a specific thread mailbox,  // retrieves its first message from its  // queue by displaying it, and removing  // it from its mailbox. This insures  // that the most recent message is read.  void IPC::receiveMessage(int task\_id)  {  //Check if there a mailbox for the task\_id  if(mailboxes.isNode(task\_id))  {  cout << "\n\n----MESSAGE---" << endl;  cout << "Source ID: " << mailboxes.searchNode(task\_id)->data.top()->data.src\_id << endl;  cout << "Dest ID: " << mailboxes.searchNode(task\_id)->data.top()->data.dst\_id << endl;  cout << "Arrival Time: " << mailboxes.searchNode(task\_id)->data.top()->data.arrived\_time;  cout << "Message Type: ";  switch(mailboxes.searchNode(task\_id)->data.top()->data.msg\_type)  {  case 0: cout << "TEXT" << endl; break;  case 1: cout << "SERVICE" << endl; break;  case 2: cout << "NOTIFICATION" << endl; break;  }  cout << "Content: " << mailboxes.searchNode(task\_id)->data.top()->data.msg\_content << endl;  cout << "----END MESSAGE---\n\n" << endl;  mailboxes.searchNode(task\_id)->data.dequeue(); //if yes, reads the first message and removes it  }  else  {  cout << "Impossible to obtain message, mailbox does not exist!" << endl;  }  }  //----------------------------------------  //messageCount(int) - Return how many messages there is  // in a thread's mailbox.  int IPC::messageCount(int task\_id)  {  if(mailboxes.isNode(task\_id))  return mailboxes.searchNode(task\_id)->data.size();  else  return -1;  }  //----------------------------------------  //removeAMailbox(int) - Delete a specific thread mailbox.  void IPC::removeAMailbox(int task\_id)  {  if(mailboxes.isNode(task\_id))  {  //Remove all messages from the mailbox.  while(mailboxes.searchNode(task\_id)->data.size() > 0 )  mailboxes.searchNode(task\_id)->data.dequeue();  }  mailboxes.removeNode(task\_id);  }  //deleteAllMailboxes() - Delete all thread's mailboxes.  void IPC::deleteAllMailboxes()  {  while(mailboxes.size() > 0)  mailboxes.removeBack();  }  //----------------------------------------  //dumpIPC() - Output all mailboxes within IPC.  void IPC::dumpIPC()  {  for(int i = 0; i < mailboxes.size(); i++)  {  dumpMailbox(i);  cout << "--------------------------------------------------" << endl;  }  }  //----------------------------------------  //dumpMailbox(int) - Output a thread specific mailbox.  void IPC::dumpMailbox(int task\_id)  {  cout << "---- MAILBOX Thread # " << task\_id << " -----" << endl;  Node<Message> \*tmp = mailboxes.searchNode(task\_id)->data.top();  for(int j = 0; j < mailboxes.searchNode(task\_id)->data.size(); j++)  {  cout << "\n\n----MESSAGE---" << endl;  cout << "Source ID: " << tmp->data.src\_id << endl;  cout << "Dest ID: " << tmp->data.dst\_id << endl;  cout << "Arrival Time: " << tmp->data.arrived\_time;  cout << "Message Type: ";  switch(tmp->data.msg\_type)  {  case 0: cout << "TEXT" << endl; break;  case 1: cout << "SERVICE" << endl; break;  case 2: cout << "NOTIFICATION" << endl; break;  }  cout << "Content: " << tmp->data.msg\_content << endl;  cout << "----END MESSAGE---\n\n" << endl;  tmp = tmp->next;  }  cout << "--------------------------------------------------" << endl;  }  //----------------------------------------  #endif |

### Queue.h

|  |
| --- |
| /\*  Filename: Queue.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To create a queue data structure.  \*/  #ifndef QUEUE\_LIST\_H  #define QUEUE\_LIST\_H  #include "LinkedList.h"  #include <iostream>  using namespace std;  #include "Node.h"  template <class T>  class Queue  {  private:  int queueSize;  Node <T> \*head;  public:  Queue();  ~Queue();  void enqueue(T data, int num = 0);  void dequeue();  bool empty();  Node<T>\* top();  int size();  void dumpQueue();  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor  template <class T>  Queue<T>::Queue()  {  head = NULL;  queueSize = 0;  }  //----------------------------------------  //Destructor  template <class T>  Queue<T>::~Queue()  {  Node<T> \*tmp = head;  while(tmp)  {  tmp = tmp->next;  delete(head);  head = tmp;  }  head = NULL;  queueSize = 0;  }  //----------------------------------------  //empty() - Returns if Queue is empty or not.  template <class T>  bool Queue<T>::empty()  {  if(size() > 0) return false;  else return true;  }  //----------------------------------------  //enqueue(T,int) - Insert node at the begin of  // the queue list.  template <class T>  void Queue<T>::enqueue(T data, int num)  {  Node<T> \*newNode = new Node<T>;  newNode->data = data;  newNode->num = num;  newNode->next = NULL;  if(empty())  {  head = newNode;  }  else  {  newNode->next = head;  head = newNode;  }  queueSize++;  }  //----------------------------------------  //dequeue() - Remove node at the beginning of the list.  template <class T>  void Queue<T>::dequeue()  {  if(!empty())  {  Node<T> \*tmp = head->next;  delete(head);  head = tmp;  }  queueSize--;  }  //----------------------------------------  //top() - Return the last node of the linked list.  template <class T>  Node<T>\* Queue<T>::top()  {  if(head)  return head;  else  {  cout << "Not found!" << endl;  }    }  //----------------------------------------  //size() - Return size of linked list.  template <class T>  int Queue<T>::size()  {  return queueSize;  }  //----------------------------------------  //dumpQueue() - Output all node in Queue.  template <class T>  void Queue<T>::dumpQueue()  {  Node<T> \*tmp = head;  if(!empty())  {  while(tmp!= NULL)  {  cout << tmp->data << " | " ;  tmp = tmp->next;  }  cout << endl;  }else cout << "---- EMPTY ----" << endl;  }  //----------------------------------------  #endif |

## 

### LinkedList.h

|  |
| --- |
| /\*  Filename: LinkedList.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To create a linked list data structure.  \*/  #ifndef LINK\_LIST\_H  #define LINK\_LIST\_H  #include <iostream>  using namespace std;  #include "Node.h"  template <class T>  class LinkedList  {  private:  Node<T> \*head;  Node<T> \*tail;  int listSize;  public:  LinkedList();  ~LinkedList();  void insertFront(T data, int num = 0);  void insertBack(T data, int num = 0);  void removeFront();  void removeBack();  void removeNode(T data);  void removeNode(int num);  Node<T> \*front();  Node<T> \*back();  int size();  bool empty();  Node<T> \*searchNode(T data);  Node<T> \*searchNode(int data);  bool isNode(int data);  void dumpList();  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor.  template <class T>  LinkedList<T>::LinkedList()  {  head = tail = NULL;  listSize = 0;  }  //----------------------------------------  //Destructor.  template <class T>  LinkedList<T>::~LinkedList()  {  Node<T> \*tmp = head;  while (tmp)  {  tmp = tmp->next;  delete (head);  head = tmp;  }  head = tail = NULL;  listSize = 0;  }  //----------------------------------------  //insertFront(T, int) - Insert node at the begin  // of the linked list.  template <class T>  void LinkedList<T>::insertFront(T data, int num)  {  Node<T> \*newNode = new Node<T>;  newNode->data = data;  newNode->num = num;  newNode->next = NULL;  if (empty())  {  head = tail = newNode;  }  else  {  newNode->next = head;  head = newNode;  }  listSize++;  }  //----------------------------------------  //insertBack(T,int) - Insert node at the end of the linked list.  template <class T>  void LinkedList<T>::insertBack(T data, int num)  {  Node<T> \*newNode = new Node<T>;  newNode->data = data;  newNode->num = num;  newNode->next = NULL;  if (empty())  head = tail = newNode;  else  {  tail->next = newNode;  tail = newNode;  }  listSize++;  }  //----------------------------------------  //removeFront() - Remove node at the beginning of the list.  template <class T>  void LinkedList<T>::removeFront()  {  if (!empty())  {  Node<T> \*tmp = head->next;  delete (head);  head = tmp;  }  listSize--;  }  //----------------------------------------  //removeBack() - Remove node at the end of the list.  template <class T>  void LinkedList<T>::removeBack()  {  if (head->next == NULL)  delete (head);  else  {  Node<T> \*tmp = head;  while (tmp->next->next != NULL)  tmp = tmp->next;  delete (tmp->next);  tmp->next = NULL;  tail = tmp;  }  listSize--;  }  //----------------------------------------  //removeNode() - Remove node by specification type T.  template <class T>  void LinkedList<T>::removeNode(T data)  {  if (head->data == data)  removeFront();  else if (tail->data == data)  removeBack();  else  {  Node<T> \*tmp = head;  Node<T> \*prev;  while (tmp->next && tmp->data != data)  {  prev = tmp;  tmp = tmp->next;  }  if (tmp->data == data)  {  prev->next = tmp->next;  delete tmp;  listSize--;  }  else  exit(-1);  }  }  //----------------------------------------  //removeNode(int) - Remove node by specification type int.  template <class T>  void LinkedList<T>::removeNode(int num)  {  if (num == 0)  removeFront();  else if (listSize == num)  removeBack();  else  {  Node<T> \*tmp = head;  Node<T> \*prev;  int count = 0;  while (tmp->next && count != num)  {  prev = tmp;  tmp = tmp->next;  count++;  }  if (count == num)  {  prev->next = tmp->next;  delete tmp;  listSize--;  }  else  exit(-1);  }  }  //----------------------------------------  //front() - Return head/first node of the linked list.  template <class T>  Node<T>\*LinkedList<T>::front()  {  return head;  }  //back() - Return the last node of the linked list.  template <class T>  Node<T> \*LinkedList<T>::back()  {  return tail;  }  //----------------------------------------  //size() - Return size of linked list.  template <class T>  int LinkedList<T>::size()  {  return listSize;  }  //----------------------------------------  //empty() - Return if the linked list is empty or not.  template <class T>  bool LinkedList<T>::empty()  {  if (listSize > 0)  return false;  else  return true;  }  //----------------------------------------  //searchNode(T) - Search node by specification type T.  template <class T>  Node<T> \*LinkedList<T>::searchNode(T data)  {  Node<T> \*tmp = new Node<T>;  tmp = head;  while (tmp)  {  if (data == tmp->data)  return tmp;  tmp = tmp->next;  }  exit(-1);  }  //----------------------------------------  //searchNode(int) - Search node by specification type int.  template <class T>  Node<T> \*LinkedList<T>::searchNode(int data)  {  Node<T> \*tmp = new Node<T>;  tmp = head;  while (tmp)  {  if (data == tmp->num)  return tmp;  tmp = tmp->next;  }  exit(-1);  }  //----------------------------------------  //isNode(int) - Return if there is such a node in the list.  template <class T>  bool LinkedList<T>::isNode(int data)  {  Node<T> \*tmp = new Node<T>;  tmp = head;  while (tmp)  {  if (data == tmp->num)  return true;  tmp = tmp->next;  }  return false;  }  //----------------------------------------  //dumpList() - Output the linked list if there is one,  // if not exit with error.  template <class T>  void LinkedList<T>::dumpList()  {  Node<T> \*tmp = head;  if (!empty())  {  while (tmp != NULL)  {  cout << tmp->data << " ";  tmp = tmp->next;  }  cout << endl;  }  else  cout << "---- EMPTY ----" << endl;  }  //----------------------------------------  #endif |

### 

### MCB.h

|  |
| --- |
| /\*  Filename: MCB.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To create a Master Control Block mechanism in which registers  other mechanisms such as IPC, Scheduler, Semaphore...  Furthermore, its registration allows the usage of such mechanisms  without controlling access issues.  \*/  #ifndef MCB\_H  #define MCB\_H  #include<iostream>  #include "Scheduler.h"  #include "Semaphore.h"  #include "IPC.h"  class Semaphore;  class Scheduler;  class IPC;  class MCB  {  public:  Scheduler \*mysch;  Semaphore \*mysema;  IPC \*myIPC;  MCB(Scheduler \*theScheduler, Semaphore \*theSemaphore, IPC \*theIPC);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor  MCB::MCB(Scheduler \*theScheduler, Semaphore \*theSemaphore, IPC \*theIPC)  {  this->myIPC = theIPC;  this->mysch = theScheduler;  this->mysema = theSemaphore;  }  //----------------------------------------  //----------------------------------------  //---- G L O B A L O B J E C T ----  //----------------------------------------  MCB \*mcb;  //----------------------------------------  #endif |

### 

### Node.h

|  |
| --- |
| /\*  Filename: Node.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To define the structure of a Node for a linked list, or queue.  \*/  #ifndef NODE\_H  #define NODE\_H  #include <iostream>  using namespace std;  template <class T>  struct Node  {  T data;  int num;  Node <T> \*next;  };  #endif |

### 

### Scheduler.h

|  |
| --- |
| /\*  Filename: IPC.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To implement a Round Robin scheduler mechanism.  \*/  #ifndef SCHEDULER\_H  #define SCHEDULER\_H  #ifndef THREAD\_STATES  #define THREAD\_STATES  enum STATE  {  READY,  RUNNING,  BLOCKED,  DEAD  };  #endif  const int NO\_OF\_THREADS = 3;  #include <pthread.h>  #include <iostream>  #include <unistd.h>  #include <assert.h>  #include "ThreadTable.h"  #include "MCB.h"  #include "IPC.h"  using namespace std;  class Scheduler  {  private:  int currentThread;  long currentQuantum;  int nextAvailableThreadID;  ThreadTable threadTable;  MCB \*mcb;  public:  Scheduler();  ~Scheduler();  int create\_task();  void \*worker(void \*arguments);  void start();  void yield();  void setQuantum(long quantum);  long getQuantum();  void setState(int task\_id, STATE the\_state);  int getCurrTaskID();  TCB\* getTCB(int task\_id);  void dump();  void wasteTime(int x);  void destroyTask(int task\_id);  void garbageCollector();  void setMCB(MCB \*mcb);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor  Scheduler::Scheduler()  {  currentThread = -1;//No active process.  nextAvailableThreadID = 0;  currentQuantum = 300; //default quantum 300ms.  }  //----------------------------------------  //Destructor  Scheduler::~Scheduler()  {  currentThread = -1;//No active process.  nextAvailableThreadID = 0;  currentQuantum = 300; //default quantum 300ms.  }  //----------------------------------------  //CreateTask() -- Create an operating system process.  int Scheduler::create\_task()  {  if (nextAvailableThreadID < NO\_OF\_THREADS)  {  cout << "Creating task # " << nextAvailableThreadID << endl;  threadTable.insertTCB(nextAvailableThreadID, clock(), READY);  //Source: https://thispointer.com/c-how-to-pass-class-member-function-to-pthread\_create/  typedef void \*(\*THREADFUNCPTR)(void \*);  int err = pthread\_create(threadTable.getTCBProcessID(nextAvailableThreadID), NULL, (THREADFUNCPTR)&Scheduler::worker, this); assert(!err);  mcb->myIPC->createMailbox(nextAvailableThreadID);  nextAvailableThreadID++;  return (nextAvailableThreadID - 1);  }  else  {  cout << "Create task FAILED: Number of available threads exceeded the limit.." << endl;  return (-1);// return error in case cannot create a task.  }  }  //----------------------------------------  //\*worker() - Context swapping function.  void \*Scheduler::worker(void \*arguments)  {  int mythreadNum = 0;  //Find the TCB related to the thread running in the OS background.  while (\*threadTable.getTCBProcessID(mythreadNum) != pthread\_self())  mythreadNum++;  //As long as the OS background running thread is not DEAD  //and it is running do:  while ((threadTable.getTCBState(mythreadNum) != DEAD))  {  if (threadTable.getTCBState(mythreadNum) == RUNNING)  {  //Prints the pthread\_t of the running thread.  cout << "Thread Num# :" << mythreadNum  << " Pthread\_self() #: " << pthread\_self()  << endl;  //Print all information over the Scheduler class.  dump();  //Call Scheduler::yield() so that current thread can  //voluntarily give up the CPU to the next ready task.  yield();  }  else  {  //If not, just relinquish the CPU by the calling thread  //, insuring for example that the calling thread of  //a createTask() function call does not hold the CPU forever.  pthread\_yield();  }  }  }  //----------------------------------------  //start() - Initiate scheduling mechanism by  // setting the first task in thread table  // to running.  void Scheduler::start()  {  cout << "............." << endl;  cout << "............. STARTING SCHEDULING" << endl;  cout << ".............\n"  << endl;  TCB \*tmp = threadTable.getTCBHead();  //Find the first READY process  //by jumping through DEAD processes  while (tmp->threadState == DEAD)  tmp = tmp->next;  //Set the first found READY process to state RUNNING, then begin scheduling  threadTable.setTCBStartTime(tmp->threadID, clock());  threadTable.setTCBState(tmp->threadID, RUNNING);  currentThread = tmp->threadID;  setQuantum(1000 / NO\_OF\_THREADS); // Set quantum to 1sec/number of threads  wasteTime(2);  }  //----------------------------------------  //yield() - runs the thread with set RUNNING state until  // its quantum runs out. When quantum runs out,  // set current RUNNING to READY then finds the next  // READY thread then set its state to RUNNING. Finally,  // puts the OS thread current running in background to sleep  // so that the next can have its turn to run.  void Scheduler::yield()  {  cout << "-----------------------------------------------------------\n"  << endl;  int counter = 0;  cout << "Current Task # " << currentThread << " is trying to yield." << endl;  cout << "Current Quantum : " << currentQuantum << endl;  // calculate elapsed\_time since the thread last started to run.  clock\_t elapsed\_time = clock() - threadTable.getTCBStartTime(currentThread);  cout << "Elapsed time: " << elapsed\_time << endl;  if (elapsed\_time >= currentQuantum) // if quantum has run out  {  cout << "yielding....(Switching from task #" << currentThread << " to next ready task)." << endl;  // if current thread is RUNNING make it READY (its quantum has run out)  if (threadTable.getTCBState(currentThread) == RUNNING)  threadTable.setTCBState(currentThread, READY);  // now find the next READY thread and make it running  // watch out for deadlocks. (no ready processes)  currentThread = (currentThread + 1) % NO\_OF\_THREADS;  if (threadTable.findTCB(currentThread) == false)  currentThread = (currentThread + 1) % NO\_OF\_THREADS;  while (threadTable.getTCBState(currentThread) != READY && counter < NO\_OF\_THREADS - 1)  {  currentThread = (currentThread + 1) % NO\_OF\_THREADS;  if (threadTable.findTCB(currentThread) == false)  currentThread = (currentThread + 1) % NO\_OF\_THREADS;  }  // if we find a READY threads re-set the quantum and set the task to running  if (counter < NO\_OF\_THREADS - 1 && threadTable.getTCBState(currentThread) == READY)  {  threadTable.setTCBStartTime(currentThread, clock()); // restart the quantum  threadTable.setTCBState(currentThread, RUNNING);  cout << "Started Running task # " << currentThread << endl;  //Sleep(1) in here, insures synchronization between the  //threads and the yield function.  sleep(1);  }  else  cout << "----- P O S S I B L E D E A D L O C K -----" << endl;  }  else  cout << "---- N O Y I E L D -----" << endl;  }  //----------------------------------------  //setQuantum(long quantum) - Assign a passed value  // to currentQuantum.  void Scheduler::setQuantum(long quantum)  {  currentQuantum = quantum;  }  //----------------------------------------  //getQuantum() - Return currentQuantum value  long Scheduler::getQuantum()  {  return (currentQuantum);  }  //----------------------------------------  //setState(int,STATE) - Set the state of a specific task to  // a specific state.  void Scheduler::setState(int task\_id, STATE the\_state)  {  threadTable.setTCBState(task\_id, the\_state);  }  //----------------------------------------  //getTaskID() - Return currentThread ID.  int Scheduler::getCurrTaskID()  {  return currentThread;  }  //----------------------------------------  //getTCB() - Return a pointer to TCB searched  // by threadID.  TCB \*Scheduler::getTCB(int task\_id)  {  TCB \*tmp = threadTable.getTCBHead();  while (tmp)  {  if (task\_id == tmp->threadID)  return tmp;  tmp = tmp->next;  }  cout << "getTCB - TCB not found! Return exit(-1)." << endl;  exit(-1);  }  //----------------------------------------  //dump() - Output important messages over the Scheduler.  void Scheduler::dump()  {  cout << "---------------------- PROCESS TABLE ----------------------" << endl;  clock\_t elapsed\_time = clock() - threadTable.getTCBStartTime(currentThread);  cout << "Quantum = " << currentQuantum << endl;  TCB \*tmp = threadTable.getTCBHead();  while (tmp != NULL)  {  cout << " task id: " << tmp->threadID;  cout << " process id" << tmp->processID;  cout << " task state: ";  switch (tmp->threadState)  {  case 0:  cout << "READY";  break;  case 1:  cout << "RUNNING";  break;  case 2:  cout << "BLOCKED";  break;  case 3:  cout << "DEAD";  break;  }  if (tmp->threadID == currentThread)  cout << " <-- CURRENT PROCESS";  cout << endl;  tmp = tmp->next;  }  cout << "-----------------------------------------------------------\n"  << endl;  }  //----------------------------------------  //wasteTime(int) - runs a for loop for (x \*  // 10240) amount of times.  void Scheduler::wasteTime(int x)  {  unsigned long long Int64 = 0;  for (unsigned short i = 0; i < 10240 \* x; ++i)  {  for (unsigned short j = i; j > 0; --j)  Int64 += j + i;  }  }  //----------------------------------------  //garbageCollector() - Goes through the thread  // table finding all the  // state DEAD threads,  // after finding it, removes  // its TCB from the thread table.  void Scheduler::garbageCollector()  {  cout << "Garbage collector " << endl;  TCB \*tmp = threadTable.getTCBHead();  while (tmp != NULL)  {  if (tmp->threadState == DEAD)  {  threadTable.removeATCB(tmp->threadID);  mcb->myIPC->removeAMailbox(tmp->threadID);  }  tmp = tmp->next;  }  }  //----------------------------------------  //destroyTask(int) - Sets the state of a task  // to DEAD, then calls a pthread  // library function pthread\_cancel  // to cancel the running task in the  // background.  void Scheduler::destroyTask(int task\_id)  {  threadTable.setTCBState(task\_id, DEAD);  pthread\_cancel(\*threadTable.getTCBProcessID(task\_id));  }  //----------------------------------------  //setMCB(MCB) - Register the Master Control Block mechanism  // within the scheduler so that Scheduler  // can access other mechanisms such as IPC,Semaphore,  // ...  void Scheduler::setMCB(MCB \*mcb)  {  this->mcb = mcb;  }  //----------------------------------------  #endif |

### Semaphore.h

|  |
| --- |
| /\*  Filename: Semaphore.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To implement a binary semaphore.  \*/  #ifndef Semaphore\_H  #define Semaphore\_H  #include <iostream>  #include <string>  using namespace std;  #include "Queue.h"  #include "Scheduler.h"  #include "MCB.h"  class Semaphore  {  private:  string resourceName; // The name of the resource being managed.  int semaValue; // 0 or 1 in the case of a binary Semaphore.  int luckyTask; // Preserve the task-id of the task that got the resource (for debugging).  pthread\_mutex\_t mux; // Critical section lock mechanism.  Queue<int> semaQueue;  MCB \*mcb;  public:  Semaphore(int starting\_value, string name);  ~Semaphore();  void down(int taskID);  void up();  void dump(int level);  void setMCB(MCB\* mcb);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor  Semaphore::Semaphore(int starting\_value, string name)  {  semaValue = starting\_value;  resourceName = name;  luckyTask = -1;  pthread\_mutex\_init(&mux,NULL);  }  //---------------------------------------------  //Destructor  Semaphore::~Semaphore()  {  while(semaQueue.size() > 0)  semaQueue.dequeue();  semaValue = -99;  resourceName = "";  luckyTask = -1;  pthread\_mutex\_destroy(&mux);  }  //---------------------------------------------  //down() - Get the hold of the resource or get queued.  void Semaphore::down(int taskID)  {  pthread\_mutex\_lock(&mux);  if (taskID == luckyTask)  {  cout << "Task # " << luckyTask << " already has the resource! Ignore the request." << endl;  dump(1);  }  else  {  if (semaValue >= 1)  {  semaValue--;  luckyTask = taskID; // preserve the task-id who got the resource  dump(1);  }  else  {  semaQueue.enqueue(taskID);  mcb->mysch->setState(taskID, BLOCKED);  dump(1);  mcb->mysch->yield();  dump(1);  }  }  pthread\_mutex\_unlock(&mux);  }  //---------------------------------------------  //up() - Release the hold of the resource.  void Semaphore::up()  {  pthread\_mutex\_lock(&mux);  int task\_id;  cout << "TaskID : " << mcb->mysch->getCurrTaskID() << ", LuckyID : " << luckyTask << endl;  if (mcb->mysch->getCurrTaskID() == luckyTask) // check to see if the correct task is doing the up()  {  if (semaQueue.empty())  {  semaValue++;  luckyTask = -1;  dump(1);  }  else  {  task\_id = semaQueue.top()->data; // Remove from queue and unblock  semaQueue.dequeue();  mcb->mysch->setState(task\_id, READY); // set the task to READY  cout << "UnBlock : " << task\_id << " and release from the queue." << endl;  luckyTask = task\_id; //  cout << "Luck Task = " << luckyTask << endl;  dump(1);  mcb->mysch->yield();  dump(1);  }  }  else  {  cout << "Invalid Semaphore UP(). TaskID : " << mcb->mysch->getCurrTaskID() << " does not own the resource." << endl;  dump(1);  }  pthread\_mutex\_unlock(&mux);  }  //---------------------------------------------  //dump() - Output information over the Semaphore.  void Semaphore::dump(int level)  {  cout << "-------Semaphore DUMP-------" << endl;  switch (level)  {  case 0:  cout << "Sema\_Value: " << semaValue << endl;  cout << "Sema\_Name : " << resourceName << endl;  cout << "Obtained by Task-ID: " << luckyTask << endl;  break;  case 1:  cout << "Sema\_Value: " << semaValue << endl;  cout << "Sema\_Name : " << resourceName << endl;  cout << "Obtained by Task-ID: " << luckyTask << endl;  cout << "Sema-Queue: ";  semaQueue.dumpQueue();  break;  default:  cout << "ERROR in Semaphore DUMP level";  }  cout << "----------------------------" << endl;  }  //setMCB(MCB) - Register the Master Control Block  // mechanism so that Semaphore class  // have access to other mechanisms  // such as the IPC, Scheduler...  void Semaphore::setMCB(MCB\* mcb)  {  this->mcb = mcb;  }  //---------------------------------------------  #endif |

### 

### ThreadTable.h

|  |
| --- |
| /\*  Filename: ThreadTable.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To implement a thread table structure alongside with functions to maneuver it.  \*/  #ifndef THREAD\_TABLE\_H  #define THREAD\_TABLE\_H  #include <iostream>  using namespace std;  #ifndef THREAD\_STATES  #define THREAD\_STATES  enum STATE  {  READY,  RUNNING,  BLOCKED,  DEAD  };  #endif  struct TCB  {  enum STATE threadState;  int threadID;  pthread\_t processID;  clock\_t threadStartTime;  TCB \*next;  };  class ThreadTable  {  private:  TCB \*head;  TCB \*tail;  int threadTableSize;  public:  ThreadTable();  ~ThreadTable();  bool empty();  void setTCBStartTime(int mythreadID, clock\_t myStartTime);  clock\_t getTCBStartTime(int mythreadID);  void insertTCB(int mythreadID, clock\_t myStartTime, STATE mythreadState);  void removeATCB(int mythreadID);  void removeLastTCB();  void removeFirstTCB();  bool findTCB(int mythreadID);  TCB \*getTCB(int mythreadID);  pthread\_t \*getTCBProcessID(int mythreadID);  TCB \*getTCBHead();  TCB \*getTCBTail();  int getThreadTableSize();  void setTCBState(int mythreadID, STATE myState);  STATE getTCBState(int mythreadID);  void dumpThreadTable();    };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor.  ThreadTable::ThreadTable()  {  head = NULL;  threadTableSize = 0;  }  //----------------------------------------  //Destructor.  ThreadTable::~ThreadTable()  {  TCB \*tmp = head;  while (tmp)  {  tmp = tmp->next;  delete (head);  head = tmp;  }  head = tail = NULL;  threadTableSize = 0;  }  //----------------------------------------  //empty() - Return if the thread table is empty.  bool ThreadTable::empty()  {  if (threadTableSize == 0)  return true;  else  return false;  }  //----------------------------------------  //setTCBStartTime(int,clock\_t) - Set the value of the thread  // start time to a specified time.  void ThreadTable::setTCBStartTime(int mythreadID, clock\_t myStartTime)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  {  tmp->threadStartTime = myStartTime;  break;  }  tmp = tmp->next;  }  }  //----------------------------------------  //getTCBStartTime(int) - Get the start time of a specified  // thread number.  clock\_t ThreadTable::getTCBStartTime(int mythreadID)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  return tmp->threadStartTime;  tmp = tmp->next;  }  }  //----------------------------------------  //insertTCB(int, clock\_t, STATE) - Insert a new TCB block at  // the end of thread Table.  void ThreadTable::insertTCB(int mythreadID, clock\_t myStartTime, STATE mythreadState)  {  TCB \*newBlock = new TCB;  newBlock->threadID = mythreadID;  newBlock->threadStartTime = myStartTime;  newBlock->threadState = mythreadState;  newBlock->next = NULL;  if (empty())  head = tail = newBlock;  else  {  tail->next = newBlock;  tail = newBlock;  }  threadTableSize++;  }  //----------------------------------------  //removeATCB(int) - Remove a specific TCB of the thread table.  void ThreadTable::removeATCB(int threadID)  {  if (head->threadID == threadID)  removeFirstTCB();  else if (tail->threadID == threadID)  removeLastTCB();  else  {  TCB \*tmp = head;  TCB \*prev;  while (tmp->next && tmp->threadID != threadID)  {  prev = tmp;  tmp = tmp->next;  }  if (tmp->threadID == threadID)  {  prev->next = tmp->next;  delete tmp;  threadTableSize--;  }  else  exit(-1);  }  }  //----------------------------------------  //removeLastTCB() - Remove last TCB block of thread table.  void ThreadTable::removeLastTCB()  {  if (head->next == NULL)  delete (head);  else  {  TCB \*tmp = head;  while (tmp->next->next != NULL)  tmp = tmp->next;  delete (tmp->next);  tmp->next = NULL;  tail = tmp;  }  threadTableSize--;  }  //----------------------------------------  //removeFirstTCB() - Remove first TCB block of thread table.  void ThreadTable::removeFirstTCB()  {  if (!empty())  {  TCB \*tmp = head->next;  delete (head);  head = tmp;  }  threadTableSize--;  }  //----------------------------------------  //findTCB(int) - Return a thread specific TCB.  bool ThreadTable::findTCB(int mythreadID)  {  TCB \*tmp = head;  while (tmp)  {  if (mythreadID == tmp->threadID)  return true;  tmp = tmp->next;  }  return false;  }  //----------------------------------------  //getTCB() - Return a pointer to TCB searched by threadID.  TCB \*ThreadTable::getTCB(int mythreadID)  {  TCB \*tmp = head;  while (tmp)  {  if (mythreadID == tmp->threadID)  return tmp;  tmp = tmp->next;  }  cout << "getTCB - TCB not found! Return exit(-1)." << endl;  exit(-1);  }  //----------------------------------------  //getTCBProcessID(int) - Return a process ID pthread\_t of a TCB.  pthread\_t \*ThreadTable::getTCBProcessID(int mythreadID)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  return &tmp->processID;  tmp = tmp->next;  }  cout << "getTCBProcessID - ProcessID not found! Return exit(-1)." << endl;  exit(-1);  }  //getTCBHead() - Return a pointer to the first TCB in the thread table.  TCB \*ThreadTable::getTCBHead()  {  return head;  }  //----------------------------------------  //getTCBTail() - Return a pointer to the last TCB in the thread table.  TCB \*ThreadTable::getTCBTail()  {  return tail;  }  //----------------------------------------  //getThreadTableSize() - Return thread table size.  int ThreadTable::getThreadTableSize()  {  return threadTableSize;  }  //----------------------------------------  //setTCBState(int,STATE) - Set TCB state of a specified threadID.  void ThreadTable::setTCBState(int mythreadID, STATE myState)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  {  tmp->threadState = myState;  break;  }  tmp = tmp->next;  }  }  //----------------------------------------  //getTCBState(int) - Get a specified TCB state.  STATE ThreadTable::getTCBState(int mythreadID)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  {  return tmp->threadState;  }  tmp = tmp->next;  }  cout << "GetTCBState - TCB not found! Return exit(-1)." << endl;  exit(-1);  }  //----------------------------------------  //dumpThreadTable() - Output everything from the thread table.  void ThreadTable::dumpThreadTable()  {  cout << "..... START ThreadTable Dump ..... " << endl;  TCB \*tmp = head;  if (!empty())  {  while (tmp != NULL)  {  cout << "TaskNUM: " << tmp->threadID << endl;  cout << "TaskID: " << tmp->threadID << endl;  cout << "TaskState: ";  switch (tmp->threadState)  {  case 0:  cout << "READY" << endl;  break;  case 1:  cout << "RUNNING" << endl;  break;  case 2:  cout << "BLOCKED" << endl;  break;  case 3:  cout << "DEAD" << endl;  break;  }  cout << "..............................." << endl;  tmp = tmp->next;  }  cout << "..... END ThreadTable Dump ..... " << endl;  }  else  {  cout << "EMPTY TCB - RETURN EXIT(-1)" << endl;  exit(-1);  }  }  //----------------------------------------  #endif |

### 

### 

## 

## 

## 

## 

## 

## Output

## Demonstration of message sent

|  |
| --- |
|  |

## 

## Demonstration of IPC being Dump

|  |
| --- |
|  |

## 

## Testing Strategy

Similarly to Phase I, during the implementation of Phase II software testing techniques were used such as Unit Testing. The objective was to guarantee the proper functioning of new and old components of the project. Furthermore, small proof of concepts were implemented separately of the project itself for study, design, and analysis.

To exemplify such strategy, during the implementation of the Inter process communication class (IPC class), a separate main function was implemented so that each IPC’s member function could be tested individually. Moreover, this approach provided a good study on how to create a Linked List of Queues as well as how to maneuver such a data structure. Later, the Linked List of Queues were used to store messages that were sent from one task to another forming our IPC class.

Finally, proof of concepts over the MCB were implemented as an effort to analyse, and solve a Circular dependency problem which was encountered during the development phase. By applying some approaches learned during proof of concept, the problem of Circular dependency was overcome.

## Discussion

After the correction of Phase I, Phase II implemented two new classes: the MCB, and the IPC. The MCB, which stands for Master Control Block, is a class that contains pointers to an Scheduler object, to a Binary Semaphore object, and to a process communication object. Moreover, the MCB allows some mechanisms such as the Scheduler to access the message passing mechanism so that messages can be sent from one task to another. To better explain, at the beginning of the main function of the project (main.cpp), the function MCB start is called so that MCB set member functions are executed, thus, registering each object with the MCB.

The inter process communication (IPC), allows tasks to communicate with each other by the sender creating a new node in the recipient's mailbox( or Queue). For this, the data structure chosen was a Linked List where each of its nodes is a Queue symbolizing a task’s mailbox.In addition, by using a Queue as a mailbox, new messages are stored at the front as a node.

Finally, the new classes contain dump functions to demonstrate the proper functionality of each new implemented mechanism as well as demonstrate all sent and received messages.

## 

# Phase III

## Memory Management

April 13, 2020

Individual work

## 

## Phase Abstract

This phase is the third part of the ULTIMA 2.0 project for the course C435 – Operating Systems at Indiana University of South Bend. The purpose of this phase is to develop a memory management mechanism which allows the tasks that are running to request memory so that one can write or read if the same owns the memory location. Altogether, additional functions for memory management were implemented such as burp, coalesce, and memory free. Furthermore, a new class Window is implemented to provide a small GUI for Ultima to easily debug and navigate.

*Keywords*: Windows,GUI, memory management, Curses

## Phase Description

## 

## 



## Design Documentation

## Source code

### Main.cpp

|  |
| --- |
| #include <iostream>  #include <sched.h>  #include <time.h>  #include <pthread.h>  #include "Semaphore.h"  #include "Scheduler.h"  #include "MemoryMgmt.h"  using namespace std;  #include "MCB.h"  void startMCB()  {  Scheduler \*newScheduler = new Scheduler();  Semaphore \*newSemaphore = new Semaphore(1, "resourceTest");  Semaphore \*newSemaphore2 = new Semaphore(1, "memoryManager");  IPC \*newIPC = new IPC();  Window \*newWin = new Window();  MemoryManager \*newMemo = new MemoryManager();  mcb = new MCB(newScheduler, newSemaphore, newSemaphore2, newIPC, newWin, newMemo);  mcb->mysch->setMCB(mcb);  mcb->mysema->setMCB(mcb);  mcb->mysema2->setMCB(mcb);  mcb->mymemo->setMCB(mcb);  mcb->myIPC->setMCB(mcb);  }  int main()  {  startMCB();  char buff[256];  int input = -1;  int CPU\_Quantum = 1;  int t\_id;  t\_id = mcb->mysch->create\_task();  t\_id = mcb->mysch->create\_task();  t\_id = mcb->mysch->create\_task();  int error;  //mcb->mysch->destroyTask(2);  //mcb->mysch->garbageCollector();  mcb->mysch->start();  //---------------------------------------------------------------------------------  while (input != 'q')  {  input = wgetch(mcb->mywin->Console\_Win);  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  switch (input)  {  case '0':  case '1':  case '2':  if (input == '0')  mcb->mysch->destroyTask(0);  else if (input == '1')  mcb->mysch->destroyTask(1);  else  mcb->mysch->destroyTask(2);  sprintf(buff, " %c\n", input);  mcb->mywin->write\_window(mcb->mywin->Console\_Win, buff);  sprintf(buff, " Kill = %c\n", input);  mcb->mywin->write\_window(mcb->mywin->Console\_Win, buff);  wclear(mcb->mywin->Log\_Win);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  sleep(4);  wclear(mcb->mywin->Console\_Win);  mcb->mywin->write\_window(mcb->mywin->Console\_Win, 1, 1, "Ultima # ");  break;  case 'c':  refresh(); //clear the console window  wclear(mcb->mywin->Console\_Win);  mcb->mywin->write\_window(mcb->mywin->Console\_Win, 1, 1, "Ultima # ");  break;  case 'g':  mcb->mysch->garbageCollector();  break;  case 'h':  case 'b':  mcb->mywin->display\_help(mcb->mywin->Console\_Win);  mcb->mywin->write\_window(mcb->mywin->Console\_Win, 8, 1, "Ultima # ");  break;  case 'i':  error = mcb->myIPC->sendMessage(0, 1, TEXT, "Testing my main...");  if (error == 0)  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " Sending message...\n");  mcb->myIPC->dumpIPC();  break;  case 'n':  mcb->mywin->cont\_help(mcb->mywin->Console\_Win);  mcb->mywin->write\_window(mcb->mywin->Console\_Win, 8, 1, "Ultima # ");  break;  case 'm':  //Allocation test  wclear(mcb->mywin->Misc\_Table);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " ....Allocation test\n");  error = mcb->mymemo->Mem\_alloc(128, 1);  if (error == 0)  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " Block allocated with success!\n");  else  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " Error allocating block!\n");  //Writing test  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " ....Writing test\n");  error = mcb->mymemo->Mem\_write(0, 0, 16, "this is task one", 1);    //Reading test  char \*ch;  mcb->mymemo->Mem\_read(0,ch,1);  mcb->mymemo->Mem\_read(0,0,3,ch,1);  //Free test  //mcb->mymemo->Mem\_free(0,1);  //Memory Dump  mcb->mymemo->Mem\_dump();  break;  case 's':  //---------- DOWN  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to obtain semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema->down(t\_id); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to obtain semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema->down(t\_id); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to obtain semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema->down(t\_id); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  //---------- UP  t\_id = mcb->mysch->getCurrTaskID();  mcb->mysema->up(); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  mcb->mysema->up(); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  mcb->mysema->up(); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  break;  case 'q':  //end the loop, and end the program.  case ERR:  //if wgetch() return ERR, that means no keys were pressed  //earlier we enabled non-blocking input using nodelay() see above  //this allows the program to continue to inspect the keyboard without  //having to wait for the key to be pressed;  break;  default:  sprintf(buff, " %c\n", input);  mcb->mywin->write\_window(mcb->mywin->Console\_Win, buff);  mcb->mywin->write\_window(mcb->mywin->Console\_Win, " - Invalid Command\n");  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, " - Invalid Command\n");  mcb->mywin->write\_window(mcb->mywin->Console\_Win, " Ultima #");  break;  }  //sleep(1);  CPU\_Quantum++;  }  endwin(); //End the curses window  return 0;  } |

### 

### IPC.h

|  |
| --- |
| /\*  Filename: IPC.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To define create a inter process communication mechanism (IPC).  \*/  #ifndef IPC\_H  #define IPC\_H  #include <iostream>  #include <string>  #include <ctime> //So that arrival time is output pretty  #include "LinkedList.h"  #include "Queue.h"  #include "MCB.h"  using namespace std;  #ifndef MSG\_TYPE  #define MSG\_TYPE  enum MSGTYPE  {  TEXT, //No action required.  SERVICE, //Request for a service by the task.  NOTIFICATION // Sent to a task to indicate a service was completed.  };  #endif  struct Message  {  int src\_id;  int dst\_id;  string arrived\_time;  MSGTYPE msg\_type;  string msg\_content;  };  class IPC  {  private:  MCB \*mcb;  char buffer[256];  public:  LinkedList<Queue<Message>> mailboxes;  int createMailbox(int task\_id);  int sendMessage(int src\_id, int dst\_id, MSGTYPE msg\_type, string msg\_content);  void receiveMessage(int task\_id);  int messageCount(int task\_id);  void removeAMailbox(int task\_id);  void deleteAllMailboxes();  void dumpIPC();  void dumpMailbox(int task\_id);  void setMCB(MCB \*mcb);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //createMailbox(int) - Create a mailbox queue inside the  // linked list Mailboxes so that a  // Scheduler::createTask thread can  // have its own mailbox "address".  int IPC::createMailbox(int task\_id)  {  Queue<Message> newBox;  mailboxes.insertBack(newBox, task\_id);  //Return 0 if successful created the box, or -1 if fails  if (mailboxes.size() > 0)  return 0;  else  return -1;  }  //----------------------------------------  //sendMessage() - Sends a message from a thread source  // to a thread destination. The message  // is enqueued in the destination thread's  // mailbox.  int IPC::sendMessage(int src\_id, int dst\_id, MSGTYPE msg\_type, string msg\_content)  {  time\_t now = time(0);  Message newMsg;  newMsg.arrived\_time = asctime(localtime(&now)); //https://www.tutorialspoint.com/cplusplus/cpp\_date\_time.htm  newMsg.dst\_id = dst\_id;  newMsg.src\_id = src\_id;  newMsg.msg\_type = msg\_type;  newMsg.msg\_content = msg\_content;  //Check if there the destination's mailbox exist  if (mailboxes.isNode(dst\_id))  {  mailboxes.searchNode(dst\_id)->data.enqueue(newMsg, dst\_id);  return 0;  }  else  {  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " Impossible to send message, mailbox does not exist!\n");  return -1;  }  }  //----------------------------------------  //receiveMessage(int) - Goes into a specific thread mailbox,  // retrieves its first message from its  // queue by displaying it, and removing  // it from its mailbox. This insures  // that the most recent message is read.  void IPC::receiveMessage(int task\_id)  {  //Check if there a mailbox for the task\_id  if (mailboxes.isNode(task\_id))  {  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, "\n\n----MESSAGE---\n");  sprintf(buffer, " Source ID: %d\n", mailboxes.searchNode(task\_id)->data.top()->data.src\_id);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, "Dest ID: %d", mailboxes.searchNode(task\_id)->data.top()->data.dst\_id);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " Arrival Time: \n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  strcpy(buffer, mailboxes.searchNode(task\_id)->data.top()->data.arrived\_time.c\_str());  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " Message Type: \n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  switch (mailboxes.searchNode(task\_id)->data.top()->data.msg\_type)  {  case 0:  sprintf(buffer, " TEXT\n");  break;  case 1:  sprintf(buffer, " SERVICE\n");  break;  case 2:  sprintf(buffer, " NOTIFICATION\n");  break;  }  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " Content: \n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  strcpy(buffer, mailboxes.searchNode(task\_id)->data.top()->data.msg\_content.c\_str());  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " ----END MESSAGE---\n\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  mailboxes.searchNode(task\_id)->data.dequeue(); //if yes, reads the first message and removes it  }  else  {  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " Impossible to obtain message, mailbox do not exist!\n");  }  }  //----------------------------------------  //messageCount(int) - Return how many messages there is  // in a thread's mailbox.  int IPC::messageCount(int task\_id)  {  if (mailboxes.isNode(task\_id))  return mailboxes.searchNode(task\_id)->data.size();  else  return -1;  }  //----------------------------------------  //removeAMailbox(int) - Delete a specific thread mailbox.  void IPC::removeAMailbox(int task\_id)  {  if (mailboxes.isNode(task\_id))  {  //Remove all messages from the mailbox.  while (mailboxes.searchNode(task\_id)->data.size() > 0)  mailboxes.searchNode(task\_id)->data.dequeue();  }  mailboxes.removeNode(task\_id);  }  //deleteAllMailboxes() - Delete all thread's mailboxes.  void IPC::deleteAllMailboxes()  {  while (mailboxes.size() > 0)  mailboxes.removeBack();  }  //----------------------------------------  //dumpIPC() - Output all mailboxes within IPC.  void IPC::dumpIPC()  {  for (int i = 0; i < mailboxes.size(); i++)  {  dumpMailbox(i);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, "\n-------------------------\n");  }  }  //----------------------------------------  //dumpMailbox(int) - Output a thread specific mailbox.  void IPC::dumpMailbox(int task\_id)  {  sprintf(buffer, " ---- MAILBOX Thread # %d -----\n", task\_id);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  Node<Message> \*tmp = mailboxes.searchNode(task\_id)->data.top();  for (int j = 0; j < mailboxes.searchNode(task\_id)->data.size(); j++)  {  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " \n\n ----MESSAGE---\n");  sprintf(buffer, " Source ID: %d \n", tmp->data.src\_id);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " Dest ID: %d \n", tmp->data.dst\_id);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " Arrival Time: \n");  sprintf(buffer, " ");  strcat(buffer, tmp->data.arrived\_time.c\_str());  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " Message Type: \n");  switch (tmp->data.msg\_type)  {  case 0:  sprintf(buffer, " TEXT\n");  break;  case 1:  sprintf(buffer, " SERVICE\n");  break;  case 2:  sprintf(buffer, " NOTIFICATION\n");  break;  }  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " Content: \n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " ");  strcat(buffer, tmp->data.msg\_content.c\_str());  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, "\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " ----END MESSAGE---\n\n");  tmp = tmp->next;  }  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, "-----------------------\n");  }  //----------------------------------------  //setMCB(MCB) - Register the Master Control Block  // mechanism so that Semaphore class  // have access to other mechanisms  // such as the IPC, Scheduler...  void IPC::setMCB(MCB \*mcb)  {  this->mcb = mcb;  }  //----------------------------------------  #endif |

### LinkedList.h

|  |
| --- |
| /\*  Filename: LinkedList.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To create a linked list data structure.  \*/  #ifndef LINK\_LIST\_H  #define LINK\_LIST\_H  #include <iostream>  #include <curses.h>  using namespace std;  #include "Node.h"  template <class T>  class LinkedList  {  private:  Node<T> \*head;  Node<T> \*tail;  int listSize;  pthread\_mutex\_t mux;  char buffer[256];  public:  LinkedList();  ~LinkedList();  void insertFront(T data, int num = 0);  void insertBack(T data, int num = 0);  void removeFront();  void removeBack();  void removeNode(T data);  void removeNode(int num);  Node<T> \*front();  Node<T> \*back();  int size();  virtual bool empty();  Node<T> \*searchNode(T data);  Node<T> \*searchNode(int data);  bool isNode(int data);  void dumpList();  void dumpList(WINDOW \*win);  void write\_window(WINDOW \*Win, const char \*text);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor.  template <class T>  LinkedList<T>::LinkedList()  {  head = tail = NULL;  listSize = 0;  pthread\_mutex\_init(&mux, NULL);  }  //----------------------------------------  //Destructor.  template <class T>  LinkedList<T>::~LinkedList()  {  Node<T> \*tmp = head;  while (tmp)  {  tmp = tmp->next;  delete (head);  head = tmp;  }  head = tail = NULL;  listSize = 0;  pthread\_mutex\_destroy(&mux);  }  //----------------------------------------  //insertFront(T, int) - Insert node at the begin  // of the linked list.  template <class T>  void LinkedList<T>::insertFront(T data, int num)  {  Node<T> \*newNode = new Node<T>;  newNode->data = data;  newNode->num = num;  newNode->next = NULL;  if (empty())  {  head = tail = newNode;  }  else  {  newNode->next = head;  head = newNode;  }  listSize++;  }  //----------------------------------------  //insertBack(T,int) - Insert node at the end of linked list.  template <class T>  void LinkedList<T>::insertBack(T data, int num)  {  Node<T> \*newNode = new Node<T>;  newNode->data = data;  newNode->num = num;  newNode->next = NULL;  if (empty())  head = tail = newNode;  else  {  tail->next = newNode;  tail = newNode;  }  listSize++;  }  //----------------------------------------  //removeFront() - Remove node at the begin of the list.  template <class T>  void LinkedList<T>::removeFront()  {  if (!empty())  {  Node<T> \*tmp = head->next;  delete (head);  head = tmp;  }  listSize--;  }  //----------------------------------------  //removeBack() - Remove node at the end of the list.  template <class T>  void LinkedList<T>::removeBack()  {  if (head->next == NULL)  delete (head);  else  {  Node<T> \*tmp = head;  while (tmp->next->next != NULL)  tmp = tmp->next;  delete (tmp->next);  tmp->next = NULL;  tail = tmp;  }  listSize--;  }  //----------------------------------------  //removeNode() - Remove node by specification type T.  template <class T>  void LinkedList<T>::removeNode(T data)  {  if (head->data == data)  removeFront();  else if (tail->data == data)  removeBack();  else  {  Node<T> \*tmp = head;  Node<T> \*prev;  while (tmp->next && tmp->data != data)  {  prev = tmp;  tmp = tmp->next;  }  if (tmp->data == data)  {  prev->next = tmp->next;  delete tmp;  listSize--;  }  else  exit(-1);  }  }  //----------------------------------------  //removeNode(int) - Remove node by specification type int.  template <class T>  void LinkedList<T>::removeNode(int num)  {  if (num == 0)  removeFront();  else if (listSize == num)  removeBack();  else  {  Node<T> \*tmp = head;  Node<T> \*prev;  int count = 0;  while (tmp->next && count != num)  {  prev = tmp;  tmp = tmp->next;  count++;  }  if (count == num)  {  prev->next = tmp->next;  delete tmp;  listSize--;  }  else  exit(-1);  }  }  //----------------------------------------  //front() - Return head/first node of the linked list.  template <class T>  Node<T> \*LinkedList<T>::front()  {  return head;  }  //back() - Return the last node of the linked list.  template <class T>  Node<T> \*LinkedList<T>::back()  {  return tail;  }  //----------------------------------------  //size() - Return size of linked list.  template <class T>  int LinkedList<T>::size()  {  return listSize;  }  //----------------------------------------  //empty() - Return if linked list is empty or not.  template <class T>  bool LinkedList<T>::empty()  {  if (listSize > 0)  return false;  else  return true;  }  //----------------------------------------  //searchNode(T) - Search node by specification type T.  template <class T>  Node<T> \*LinkedList<T>::searchNode(T data)  {  Node<T> \*tmp = new Node<T>;  tmp = head;  while (tmp)  {  if (data == tmp->data)  return tmp;  tmp = tmp->next;  }  exit(-1);  }  //----------------------------------------  //searchNode(int) - Search node by specification type int.  template <class T>  Node<T> \*LinkedList<T>::searchNode(int data)  {  Node<T> \*tmp = new Node<T>;  tmp = head;  while (tmp)  {  if (data == tmp->num)  return tmp;  tmp = tmp->next;  }  exit(-1);  }  //----------------------------------------  //isNode(int) - Return if there is such a node in list.  template <class T>  bool LinkedList<T>::isNode(int data)  {  Node<T> \*tmp = new Node<T>;  tmp = head;  while (tmp)  {  if (data == tmp->num)  return true;  tmp = tmp->next;  }  return false;  }  //----------------------------------------  //dumpList() - Output the linked list if there is one,  // if not exit with error.  template <class T>  void LinkedList<T>::dumpList()  {  Node<T> \*tmp = head;  if (!empty())  {  while (tmp != NULL)  {  cout << tmp->data << " ";  tmp = tmp->next;  }  cout << endl;  }  else  cout << "---- EMPTY ----\n";  }  //dumpList() - Output the linked list if there is one,  // if not exit with error.  template <class T>  void LinkedList<T>::dumpList(WINDOW \*win)  {  Node<T> \*tmp = head;  if (!empty())  {  while (tmp != NULL)  {  sprintf(buffer, "%d ", tmp->data);  write\_window(win, buffer);  tmp = tmp->next;  }  write\_window(win, "\n");  }  else  write\_window(win, " ---- EMPTY ----\n");  }  //----------------------------------------  //write\_window() - To display things at specific  // curses window.  template <class T>  void LinkedList<T>::write\_window(WINDOW \*Win, const char \*text)  {  pthread\_mutex\_lock(&mux);  wprintw(Win, text);  box(Win, 0, 0);  wrefresh(Win); // draw the window  pthread\_mutex\_unlock(&mux);  }  #endif |

### 

### MCB.h

|  |
| --- |
| /\*  Filename: MCB.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To create a Master Control Block mechanism in which registers  other mechanisms such as IPC, Scheduler, Semaphore...  Furthermore, its registration allows the usage of such mechanisms  without controlling access issues.  \*/  #ifndef MCB\_H  #define MCB\_H  #include <iostream>  #include "Scheduler.h"  #include "Semaphore.h"  #include "Window.h"  class Semaphore;  class Scheduler;  class IPC;  class Window;  class MemoryManager;  class MCB  {  public:  Scheduler \*mysch;  Semaphore \*mysema;  Semaphore \*mysema2;  IPC \*myIPC;  Window \*mywin;  MemoryManager \*mymemo;  MCB(Scheduler \*theScheduler, Semaphore \*theSemaphore, Semaphore \*theSemaphore2, IPC \*theIPC, Window \*theWIN, MemoryManager \*theMemo);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor  MCB::MCB(Scheduler \*theScheduler, Semaphore \*theSemaphore, Semaphore \*theSemaphore2, IPC \*theIPC, Window \*theWIN, MemoryManager \*theMemo)  {  this->myIPC = theIPC;  this->mysch = theScheduler;  this->mysema = theSemaphore;  this->mywin = theWIN;  this->mymemo = theMemo;  this->mysema2 = theSemaphore2;  }  //----------------------------------------  //----------------------------------------  //---- G L O B A L O B J E C T ----  //----------------------------------------  MCB \*mcb;  //----------------------------------------  #endif |

### 

### MemoryMgmt.h

|  |
| --- |
| /\*  Filename: MemoryMgmt.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To create a memory management mechanism.  Phase III  \*/  #ifndef MEMORY\_MGMT\_H  #define MEMORY\_MGMT\_H  #include <iostream>  #include <fstream>  using namespace std;  #include "LinkedList.h"  #include "MCB.h"  class MCB;  enum MEMORY\_LABEL  {  FREE,  USED  };  struct Segment  {  int base; //Start of the memory segment.  int limit; //End of the memory segment.  int size; //Number of bytes requested by task.  int ownerID; //ID of the owner task.  int handler; //set by memory manager class.  MEMORY\_LABEL status; //0 - free, 1 - used.  int write\_cursor;  int read\_cursor;  };  class MemoryManager  {  private:  int nextAvailableHandler;  int freeSpace; //How much free memory I have.  LinkedList<Segment> MemoManager;  char \*memory; //Actually memory resource.  MCB \*mcb;  char buffer[2024];  int Mem\_left();  int Mem\_largest();  int Mem\_smallest();  int Mem\_coalesce();  int Mem\_burp();  public:  MemoryManager();  ~MemoryManager();  int Mem\_dump();  int Mem\_alloc(int size, int taskID);  int Mem\_free(int memory\_handle, int taskID);  int Mem\_read(int memory\_handle, char \*ch, int taskID);  int Mem\_write(int memory\_handle, char ch, int taskID);  //Overloaded multi-byte read and write  int Mem\_read(int memory\_handle, int offset\_from\_beg, int text\_size, char \*text, int taskID);  int Mem\_write(int memory\_handle, int offset\_from\_beg, int text\_size, char \*text, int taskID);  void setMCB(MCB \*mcb) { this->mcb = mcb; }  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor - Initialize some basic variables  //as well as insert first segment in memory which  //is equivalent of the whole memory as free status  //.i.e. a block of status free from 0 to 1023.  MemoryManager::MemoryManager()  {  //Initialize basic variables.  nextAvailableHandler = 0;  freeSpace = 1024;  //Initialize actual memory by  //setting capacity to 1024 bytes  //and overwritting with dots.  memory = new char[1024];  for (int i = 0; i < 1024; i++)  memory[i] = '.';  //First segment of the manager should display  //the entire empty memory.  Segment newSeg;  newSeg.base = 0;  newSeg.limit = 1023;  newSeg.ownerID = -1;  newSeg.handler = -1;  newSeg.size = 1024;  newSeg.status = FREE;  newSeg.write\_cursor = newSeg.read\_cursor = 0;  MemoManager.insertFront(newSeg, newSeg.handler);  }  //----------------------------------------  //Destructor.  MemoryManager::~MemoryManager()  {  nextAvailableHandler = 0;  freeSpace = 1024;  MemoManager.~LinkedList();  for (int i = 0; i < 1024; i++)  {  memory[i] = '.';  buffer[i]=' ';  }  mcb = NULL;  }  //----------------------------------------  //Mem\_alloc() - Uses first fit approach.  //Finds the first hole big enough then allocate,  //if not then throw -1 for error.  int MemoryManager::Mem\_alloc(int size, int taskID)  {  if (freeSpace < size)  return -1; //Segment fault. No available memory.  else  {  Node<Segment> \*tmp = MemoManager.back();  //Create new segments at the from of the linked  //list manager.  Segment newSeg;  newSeg.base = tmp->data.base;  if (MemoManager.size() == 0)  newSeg.base = 0;  else  newSeg.limit = newSeg.base + 127;  newSeg.ownerID = taskID;  newSeg.handler = nextAvailableHandler;  newSeg.size = size;  newSeg.status = USED;  newSeg.write\_cursor = newSeg.read\_cursor = newSeg.base;  //Adjust the last node which would be the free  //location by setting new values for base and size.  tmp->data.base = newSeg.limit + 1;  tmp->data.size = tmp->data.limit - tmp->data.base;  tmp->data.write\_cursor = tmp->data.base;  tmp->data.read\_cursor = tmp->data.base;  freeSpace = tmp->data.size;  MemoManager.insertFront(newSeg);  //Handle increment  nextAvailableHandler++;  return 0; //Success allocating memory.  }  }  //----------------------------------------  //Mem\_free() - Check if there is such segment in memory  //and if requesting task owns the segment then if  //yes free the respective segment. If not of those  //apply, throw -1 for error.  int MemoryManager::Mem\_free(int memory\_handle, int taskID)  {  //Check if there is such segment in memory.  if (MemoManager.isNode(memory\_handle))  {  Node<Segment> \*tmp = MemoManager.front();  while (tmp)  {  if (tmp->data.handler == memory\_handle)  break;  tmp = tmp->next;  }  //Check if taskID is the owner(have permission).  if (tmp->data.ownerID == taskID)  {  //Start of free up memory:  for (int i = tmp->data.base; i <= tmp->data.limit; i++)  memory[i] = '#';  //Reset cursor's location to point to the beginning of the block.  tmp->data.write\_cursor = tmp->data.read\_cursor = tmp->data.base;  tmp->data.status = FREE;  tmp->data.size = 0;  freeSpace += (tmp->data.limit - tmp->data.base);    int err = Mem\_coalesce();  if (err == -1)  {  sprintf(buffer, " No coalesce!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  }    return 0; //Success.  }  else  return -1; //Access denied, task does not own the memory block.  }  else  return -1; //Segment not found.  }  //----------------------------------------  //Mem\_read() - Overloaded function / Single Byte Read  //The below reads ONE character then increments read\_cursor location  //(current\_location adjustment).  int MemoryManager::Mem\_read(int memory\_handle, char \*ch, int taskID)  {  mcb->mysema2->down(taskID);  //Check if there is such segment in memory.  if (MemoManager.isNode(memory\_handle))  {  Node<Segment> \*tmp = MemoManager.front();  while (tmp)  {  if (tmp->data.handler == memory\_handle)  break;  tmp = tmp->next;  }  //Check if taskID is the owner(have permission).  if (tmp->data.ownerID == taskID)  {  ch = &memory[tmp->data.read\_cursor]; //Read single character.  tmp->data.read\_cursor++;  sprintf(buffer, " -----------------\n Content of Read: %c\n-----------------\n", \*ch);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  //Checks if the new read\_cursor position goes beyond  //the writer\_cursor, if yes then set read\_cursor to the base  //of the block.  if (tmp->data.read\_cursor >= tmp->data.limit)  {  tmp->data.read\_cursor = tmp->data.base;  }  mcb->mysema2->up();  return 0; //Success reading.  }  else  {  mcb->mysema2->up();  return -1; //Access denied, task does not own the memory block.  }  }  else  {  mcb->mysema2->up();  return -1; //Segment not found.  }  }  //----------------------------------------  //Mem\_write() - Overloaded function / Single Byte Write  //The below writes ONE character then increments write\_cursor location.  int MemoryManager::Mem\_write(int memory\_handle, char ch, int taskID)  {  mcb->mysema2->down(taskID);  //Check if there is such segment in memory.  if (MemoManager.isNode(memory\_handle))  {  Node<Segment> \*tmp = MemoManager.front();  while (tmp)  {  if (tmp->data.handler == memory\_handle)  break;  tmp = tmp->next;  }  //Check if taskID is the owner(have permission).  if (tmp->data.ownerID == taskID)  {  //Check if write\_cursor reached the limit of the segment block.  if (tmp->data.write\_cursor == tmp->data.limit)  {  mcb->mysema2->up();  return -1; //Segment is full.  }  else  {  memory[tmp->data.write\_cursor] = ch; //Write in it.  tmp->data.write\_cursor++; //adjust write\_cursor.  mcb->mysema2->up();  return 0; //Success writing.  }  }  else  {  mcb->mysema2->up();  return -1; //Access denied, task does not own the memory block.  }  }  else  {  mcb->mysema2->up();  return -1; //Segment not found  }  }  //----------------------------------------  //Overloaded multi-byte read and write  //Mem\_read() - Read a large memory location.  int MemoryManager::Mem\_read(int memory\_handle, int offset\_from\_beg, int text\_size, char \*text, int taskID)  {  mcb->mysema2->down(taskID);  //Check if there is such segment in memory.  if (MemoManager.isNode(memory\_handle))  {  Node<Segment> \*tmp = MemoManager.front();  while (tmp)  {  if (tmp->data.handler == memory\_handle)  break;  tmp = tmp->next;  }  //Check if TaskID owns the block  if (tmp->data.ownerID == taskID)  {  //Check if what I am reading goes beyond my block limit  if (tmp->data.limit < offset\_from\_beg + tmp->data.read\_cursor + text\_size)  {  mcb->mysema2->up();  return -1; //Trying to reach more than it should.  }  else  {  text = new char[offset\_from\_beg + 1];  text[offset\_from\_beg + 1] = '\0';  for (int i = 0; i < text\_size; i++)  {  text[i] = memory[tmp->data.read\_cursor];  tmp->data.read\_cursor++;  }  mcb->mywin->write\_window(mcb->mywin->Misc\_Table," -----------------\n Content of read: \n");  strcpy(buffer,text);  strcat(buffer,"\n-----------------\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,buffer);  mcb->mysema2->up();  return 0; //Success reading.  }  }  else{  mcb->mysema2->up();  return -1; //Access denied, task does not own the memory block.  }  }  else  {  mcb->mysema2->up();  return -1; //Segment not found.  }  }  //----------------------------------------  //Mem\_write() - Write a large text into memory.  int MemoryManager::Mem\_write(int memory\_handle, int offset\_from\_beg, int text\_size, char \*text, int taskID)  {  mcb->mysema2->down(taskID);  //Check if there is such segment in memory.  if (MemoManager.isNode(memory\_handle))  {  Node<Segment> \*tmp = MemoManager.front();  while (tmp)  {  if (tmp->data.handler == memory\_handle)  break;  tmp = tmp->next;  }  //Check if TaskID owns the block  if (tmp->data.ownerID == taskID)  {  //Check if what I am writing goes beyond my block limit  if (tmp->data.limit < offset\_from\_beg + tmp->data.write\_cursor + text\_size)  return -1; //Trying to reach more than it should.  else  {  for (int i = 0; i < text\_size; i++)  {  memory[tmp->data.write\_cursor + offset\_from\_beg] = text[i];  tmp->data.write\_cursor++;  }  tmp->data.write\_cursor += offset\_from\_beg;  mcb->mysema2->up();  return 0; //Success writing.  }  }  else  {  mcb->mysema2->up();  return -1; //Access denied, task does not own the memory block.  }  }  else  {  mcb->mysema2->up();  return -1; //Segment not found.  }  }  //----------------------------------------  //Mem\_left() - Return how much free space is  //still available.  int MemoryManager::Mem\_left()  {  return freeSpace;  }  //Mem\_largest - Return largest block in size.  int MemoryManager::Mem\_largest()  {  Node<Segment> \*tmp = MemoManager.front();  int largest = tmp->data.size; //Set the lasgest to be the first block found.  for (int i = 0; i < MemoManager.size(); i++)  {  if (tmp->data.size > largest && tmp->data.status == FREE)  largest = tmp->data.size;  tmp = tmp->next;  }  return largest;  }  //----------------------------------------  //Mem\_smallest() - Return smallest block in size.  int MemoryManager::Mem\_smallest()  {  Node<Segment> \*tmp = MemoManager.front();  int smallest = tmp->data.size; //Set the lasgest to be the first block found.  for (int i = 0; i < MemoManager.size(); i++)  {  if (tmp->data.size > smallest && tmp->data.status == FREE)  smallest = tmp->data.size;  tmp = tmp->next;  }  return smallest;  }  //Mem\_coalesce() - Merge two adjacent free blocks of memory.  int MemoryManager::Mem\_coalesce()  {  Mem\_dump();  Node<Segment> \*curr = MemoManager.front();  Node<Segment> \*prev;  int i = -1;  while (curr)  {  if (prev->data.status == FREE && curr->data.status == FREE)  {  for (int i = prev->data.base; i <= prev->data.limit; i++)  memory[i] = '.';  for (int i = curr->data.base; i <= curr->data.limit; i++)  memory[i] = '.';  curr->data.base = prev->data.base;  curr->data.size = 0;  curr->data.write\_cursor = curr->data.read\_cursor = 0;  MemoManager.removeNode(i);  Mem\_dump();  return 0; //Success coalesce.  }  i++;  prev = curr;  curr = curr->next;  }  return -1; //Error coalesce.  }  //----------------------------------------  //Mem\_burp() - Move empty holes to the end of MemoryManager list.  int MemoryManager::Mem\_burp()  {  //Check if there are empty holes.  if (freeSpace > 0)  {  Node<Segment> \*tmp = MemoManager.front();  for (int i = 0; i < MemoManager.size(); i++)  {  //Check if is an empty hole.  if (tmp->data.status == FREE)  {  //Copy the empty hole  Segment newSeg;  newSeg.base = tmp->data.base;  newSeg.handler = tmp->data.handler;  newSeg.limit = tmp->data.limit;  newSeg.ownerID = tmp->data.ownerID;  newSeg.read\_cursor = tmp->data.read\_cursor;  newSeg.size = tmp->data.size;  newSeg.status = tmp->data.status;  newSeg.write\_cursor = tmp->data.write\_cursor;  //Remove respective empty hole  MemoManager.removeNode(i);  //Add it at the end of MemoryManager list.  MemoManager.insertBack(newSeg);  }  tmp = tmp->next;  }  return 0; //Success in moving holes.  }  else  return -1; //No empty holes.  }  //----------------------------------------  //Mem\_dump() - Display all information over the memory manager.  int MemoryManager::Mem\_dump()  {  Node<Segment> \*tmp = MemoManager.front();  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, "\n------------------\n ---- Memory Dump ----\n");  for (int i = 0; i < MemoManager.size(); i++)  {  sprintf(buffer, " Handler: %d\n", tmp->data.handler);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " Owner: %d\n", tmp->data.ownerID);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " Base: %d\n", tmp->data.base);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " Limit: %d\n", tmp->data.limit);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " Size: %d\n", tmp->data.size);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " Status: ");  if (tmp->data.status == FREE)  strcat(buffer, " FREE\n");  else  strcat(buffer, " USED\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " Read\_cursor: %d\n", tmp->data.read\_cursor);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " Write\_cursor: %d\n", tmp->data.write\_cursor);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, "---------------------------\n");  tmp = tmp->next;  }  strcpy(buffer," ");  strcat(buffer,memory);  strcat(buffer,"\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  }  #endif |

## 

### Node.h

|  |
| --- |
| /\*  Filename: Node.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To define the structure of a Node for a linked list, or queue.  \*/  #ifndef NODE\_H  #define NODE\_H  #include <iostream>  using namespace std;  template <class T>  struct Node  {  T data;  int num;  Node<T> \*next;  };  #endif |

### 

### Queue.h

|  |
| --- |
| /\*  Filename: Queue.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To create a queue data structure.  \*/  #ifndef QUEUE\_LIST\_H  #define QUEUE\_LIST\_H  #include "LinkedList.h"  #include <iostream>  using namespace std;  #include "Node.h"  #include <curses.h>  template <class T>  class Queue  {  private:  int queueSize;  Node<T> \*head;  pthread\_mutex\_t mux;  char buffer[256];  public:  Queue();  ~Queue();  void enqueue(T data, int num = 0);  void dequeue();  bool empty();  Node<T> \*top();  Node<T> \*top(WINDOW \*win);  int size();  void dumpQueue();  void dumpQueue(WINDOW \*win);  void write\_window(WINDOW \*Win, const char \*text);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor  template <class T>  Queue<T>::Queue()  {  head = NULL;  queueSize = 0;  pthread\_mutex\_init(&mux, NULL);  }  //----------------------------------------  //Destructor  template <class T>  Queue<T>::~Queue()  {  Node<T> \*tmp = head;  while (tmp)  {  tmp = tmp->next;  delete (head);  head = tmp;  }  head = NULL;  queueSize = 0;  pthread\_mutex\_destroy(&mux);  }  //----------------------------------------  //empty() - Returns if Queue is empty or not.  template <class T>  bool Queue<T>::empty()  {  if (size() > 0)  return false;  else  return true;  }  //----------------------------------------  //enqueue(T,int) - Insert node at the begin of  // the queue list.  template <class T>  void Queue<T>::enqueue(T data, int num)  {  Node<T> \*newNode = new Node<T>;  newNode->data = data;  newNode->num = num;  newNode->next = NULL;  if (empty())  {  head = newNode;  }  else  {  newNode->next = head;  head = newNode;  }  queueSize++;  }  //----------------------------------------  //dequeue() - Remove node at the begin of the list.  template <class T>  void Queue<T>::dequeue()  {  if (!empty())  {  Node<T> \*tmp = head->next;  delete (head);  head = tmp;  }  queueSize--;  }  //----------------------------------------  //top() - Return the last node of the linked list.  template <class T>  Node<T> \*Queue<T>::top()  {  if (head)  return head;  else  {  cout << "Not found!";  }  }  template <class T>  Node<T> \*Queue<T>::top(WINDOW \*win)  {  if (head)  return head;  else  {  write\_window(win, "Not found!\n");  }  }  //----------------------------------------  //size() - Return size of linked list.  template <class T>  int Queue<T>::size()  {  return queueSize;  }  //----------------------------------------  //dumpQueue() - Output all node in Queue.  template <class T>  void Queue<T>::dumpQueue()  {  Node<T> \*tmp = head;  if (!empty())  {  while (tmp != NULL)  {  cout << tmp->data << " | ";  tmp = tmp->next;  }  cout << endl;  }  else  cout << "---- EMPTY ----" << endl;  }  //dumpQueue() - Output all node in Queue.  template <class T>  void Queue<T>::dumpQueue(WINDOW \*win)  {  Node<T> \*tmp = head;  if (!empty())  {  while (tmp != NULL)  {  sprintf(buffer, " %d \n", tmp->data);  write\_window(win, buffer);  tmp = tmp->next;  }  }  else  write\_window(win, " ---- EMPTY ----\n");  }  //----------------------------------------  //write\_window() - To display things at specific curses  // window.  template <class T>  void Queue<T>::write\_window(WINDOW \*Win, const char \*text)  {  pthread\_mutex\_lock(&mux);  wprintw(Win, text);  box(Win, 0, 0);  wrefresh(Win); // draw the window  pthread\_mutex\_unlock(&mux);  }  //----------------------------------------  #endif |

### 

### Scheduler.h

|  |
| --- |
| /\*  Filename: IPC.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To implement a Round Robin scheduler mechanism.  \*/  #ifndef SCHEDULER\_H  #define SCHEDULER\_H  #ifndef THREAD\_STATES  #define THREAD\_STATES  enum STATE  {  READY,  RUNNING,  BLOCKED,  DEAD  };  #endif  const int NO\_OF\_THREADS = 3;  #include <pthread.h>  #include <iostream>  #include <unistd.h>  #include <assert.h>  #include <string.h>  #include "ThreadTable.h"  #include "MCB.h"  #include "IPC.h"  using namespace std;  class Scheduler  {  private:  int currentThread;  long currentQuantum;  int nextAvailableThreadID;  ThreadTable threadTable;  MCB \*mcb;  char buffer[256];  pthread\_mutex\_t mux;  public:  Scheduler();  ~Scheduler();  int create\_task();  void \*worker(void \*arguments);  void start();  void yield();  void setQuantum(long quantum);  long getQuantum();  void setState(int task\_id, STATE the\_state);  int getCurrTaskID();  TCB \*getTCB(int task\_id);  void dump();  void wasteTime(int x);  void destroyTask(int task\_id);  void garbageCollector();  void setMCB(MCB \*mcb);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor  Scheduler::Scheduler()  {  currentThread = -1; //No active process.  nextAvailableThreadID = 0;  currentQuantum = 300; //default quantum 300ms.  }  //----------------------------------------  //Destructor  Scheduler::~Scheduler()  {  currentThread = -1; //No active process.  nextAvailableThreadID = 0;  currentQuantum = 300; //default quantum 300ms.  }  //----------------------------------------  //CreateTask() -- Create an operating system process.  int Scheduler::create\_task()  {  char buffer[256];  if (nextAvailableThreadID < NO\_OF\_THREADS)  {  sprintf(buffer, "Creating task # %d\n", nextAvailableThreadID);  mcb->mywin->write\_window(mcb->mywin->Heading\_Win, 5 + nextAvailableThreadID, 6, buffer);  WINDOW \*newTaskWIN = mcb->mywin->create\_window(15, 27, 15, 2 + (threadTable.getThreadTableSize() \* 28));  threadTable.insertTCB(nextAvailableThreadID, clock(), READY, newTaskWIN);  //Source: https://thispointer.com/c-how-to-pass-class-member-function-to-pthread\_create/  typedef void \*(\*THREADFUNCPTR)(void \*);  int err = pthread\_create(threadTable.getTCBProcessID(nextAvailableThreadID), NULL, (THREADFUNCPTR)&Scheduler::worker, this);  assert(!err);  mcb->myIPC->createMailbox(nextAvailableThreadID);  nextAvailableThreadID++;  return (nextAvailableThreadID - 1);  }  else  {  //sprintf(buffer,"Create task FAILED: Number of available threads exceeded the limit..\n");  return (-1); // return error in case cannot create a task.  }  }  //----------------------------------------  //\*worker() - Context swapping function.  void \*Scheduler::worker(void \*arguments)  {  int mythreadNum = 0;  char buffer[256];  //Find the TCB related to the thread running in the OS background.  while (\*threadTable.getTCBProcessID(mythreadNum) != pthread\_self())  mythreadNum++;  //As long as the OS background running thread is not DEAD  //and it is running do:  while ((threadTable.getTCBState(mythreadNum) != DEAD))  {  if (threadTable.getTCBState(mythreadNum) == RUNNING)  {  //Prints the pthread\_t of the running thread.  sprintf(buffer, " Task#%d running..\n", currentThread);  mcb->mywin->write\_window(threadTable.getTCBWIN(currentThread), buffer);  //Print all information over the Scheduler class.  dump();  //Call Scheduler::yield() so that current thread can  //voluntary give up the CPU to the next ready task.  yield();  }  else  {  //If not, just relinquish the CPU by the calling thread  //, insuring for example that the calling thread of  //a createTask() function call do not hold the CPU forever.  pthread\_yield();  }  }  }  //----------------------------------------  //start() - Initiate scheduling mechanism by  // setting the first task in thread table  // to running.  void Scheduler::start()  {  sprintf(buffer, " ... STARTING SCHEDULING...\n");  mcb->mywin->write\_window(mcb->mywin->Heading\_Win, 4, 1, buffer);  TCB \*tmp = threadTable.getTCBHead();  //Find the first READY process  //by jumping through DEAD processes  while (tmp->threadState == DEAD)  tmp = tmp->next;  //Set the first found READY process to state RUNNING, then begin scheduling  threadTable.setTCBStartTime(tmp->threadID, clock());  threadTable.setTCBState(tmp->threadID, RUNNING);  currentThread = tmp->threadID;  setQuantum(1000 / NO\_OF\_THREADS); // Set quantum to 1sec/number of threads  wasteTime(2);  }  //----------------------------------------  //yield() - runs the thread with set RUNNING state until  // its quantum runs out. When quantum runs out,  // set current RUNNING to READY then finds the next  // READY thread then set its state to RUNNING. Finally,  // puts the OS thread current running in background to sleep  // so that the next can have its turn to run.  void Scheduler::yield()  {  sprintf(buffer, " ---------------------\n");  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  int counter = 0;  sprintf(buffer, " Current Task # %d is trying to yield.\n", currentThread);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  sprintf(buffer, " Current Quantum : %li\n", currentQuantum);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  // calculate elapsed\_time since the thread last started to run.  clock\_t elapsed\_time = clock() - threadTable.getTCBStartTime(currentThread);  sprintf(buffer, " Elapsed time: %li\n", elapsed\_time);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  if (elapsed\_time >= currentQuantum) // if quantum has run out  {  sprintf(buffer, " yielding....(Switching from task #%d to next ready task).\n", currentThread);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  // if current thread is RUNNING make it READY (its quantum has run out)  if (threadTable.getTCBState(currentThread) == RUNNING)  threadTable.setTCBState(currentThread, READY);  // now find the next READY thread and make it running  // watch out for deadlocks. (no ready processes)  currentThread = (currentThread + 1) % NO\_OF\_THREADS;  if (threadTable.findTCB(currentThread) == false)  currentThread = (currentThread + 1) % NO\_OF\_THREADS;  while (threadTable.getTCBState(currentThread) != READY && counter < NO\_OF\_THREADS - 1)  {  currentThread = (currentThread + 1) % NO\_OF\_THREADS;  if (threadTable.findTCB(currentThread) == false)  currentThread = (currentThread + 1) % NO\_OF\_THREADS;  }  // if we find a READY threads re-set the quantum and set the task to running  if (counter < NO\_OF\_THREADS - 1 && threadTable.getTCBState(currentThread) == READY)  {  threadTable.setTCBStartTime(currentThread, clock()); // restart the quantum  threadTable.setTCBState(currentThread, RUNNING);  sprintf(buffer, " Started Running task # %d \n", currentThread);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  //Sleep(1) in here, insures syncronization between the  //threads and the yield function  sleep(1);  }  else  {  sprintf(buffer, "----- P O S S I B L E D E A D L O C K -----");  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  }  }  else  {  sprintf(buffer, "---- N O Y I E L D -----\n");  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  }  }  //----------------------------------------  //setQuantum(long quantum) - Assign a passed value  // to currentQuantum.  void Scheduler::setQuantum(long quantum)  {  currentQuantum = quantum;  }  //----------------------------------------  //getQuantum() - Return currentQuantum value  long Scheduler::getQuantum()  {  return (currentQuantum);  }  //----------------------------------------  //setState(int,STATE) - Set the state of a specific task to  // a specific state.  void Scheduler::setState(int task\_id, STATE the\_state)  {  threadTable.setTCBState(task\_id, the\_state);  }  //----------------------------------------  //getTaskID() - Return currentThread ID.  int Scheduler::getCurrTaskID()  {  return currentThread;  }  //----------------------------------------  //getTCB() - Return a pointer to TCB searched  // by threadID.  TCB \*Scheduler::getTCB(int task\_id)  {  TCB \*tmp = threadTable.getTCBHead();  while (tmp)  {  if (task\_id == tmp->threadID)  return tmp;  tmp = tmp->next;  }  //sprintf(buffer,"getTCB - TCB not found! Return exit(-1).");  //mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  exit(-1);  }  //----------------------------------------  //dump() - Ouput important messages over the Scheduler.  void Scheduler::dump()  {  sprintf(buffer, " -----PROCESS Table -----\n");  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  clock\_t elapsed\_time = clock() - threadTable.getTCBStartTime(currentThread);  sprintf(buffer, " Quantum = %li \n", currentQuantum);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  TCB \*tmp = threadTable.getTCBHead();  while (tmp != NULL)  {  if (tmp->threadState == READY)  sprintf(buffer, " process id %li task state: READY ", tmp->processID);  else if (tmp->threadState == RUNNING)  sprintf(buffer, " process id %li task state: RUNNING ", tmp->processID);  else if (tmp->threadState == BLOCKED)  sprintf(buffer, " process id %li task state: BLOCKED ", tmp->processID);  else if (tmp->threadState == DEAD)  sprintf(buffer, " process id %li task state: DEAD ", tmp->processID);  char curr[] = "<--CURRENT\n";  if (tmp->threadID == currentThread)  strcat(buffer, curr);  else  strcat(buffer, "\n");  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  tmp = tmp->next;  }  }  //----------------------------------------  //wasteTime(int) - runs a for loop for (x \*  // 10240) amount of times.  void Scheduler::wasteTime(int x)  {  unsigned long long Int64 = 0;  for (unsigned short i = 0; i < 10240 \* x; ++i)  {  for (unsigned short j = i; j > 0; --j)  Int64 += j + i;  }  }  //----------------------------------------  //garbageCollector() - Goes through the thread  // table finding all the  // state DEAD threads,  // after finding it, removes  // its TCB from thread table.  void Scheduler::garbageCollector()  {  //sprintf(buffer,"Garbage collector ");  TCB \*tmp = threadTable.getTCBHead();  while (tmp != NULL)  {  if (tmp->threadState == DEAD)  {  threadTable.removeATCB(tmp->threadID);  mcb->myIPC->removeAMailbox(tmp->threadID);  }  tmp = tmp->next;  }  }  //----------------------------------------  //destroyTask(int) - Sets the state of a task  // to DEAD, then calls a pthread  // library function pthread\_cancel  // to cancel the running task in the  // background.  void Scheduler::destroyTask(int task\_id)  {  threadTable.setTCBState(task\_id, DEAD);  pthread\_cancel(\*threadTable.getTCBProcessID(task\_id));  }  //----------------------------------------  //setMCB(MCB) - Register the Master Control Block mechanism  // within the scheduler so that Scheduler  // can access other mechanisms such as IPC,Semaphore,  // ...  void Scheduler::setMCB(MCB \*mcb)  {  this->mcb = mcb;  }  //----------------------------------------  #endif |

### 

### Semaphore.h

|  |
| --- |
| /\*  Filename: Semaphore.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To implement a binary semaphore.  \*/  #ifndef Semaphore\_H  #define Semaphore\_H  #include <iostream>  #include <string>  using namespace std;  #include "Queue.h"  #include "Scheduler.h"  #include "MCB.h"  class Semaphore  {  private:  string resourceName; // The name of the resource being managed.  int semaValue; // 0 or 1 in the case of a binary Semaphore.  int luckyTask; // Preserve the task-id of the task that got the resource (for debbugging).  pthread\_mutex\_t mux; // Critical section lock mechanism.  Queue<int> semaQueue;  MCB \*mcb;  char buffer[256];  public:  Semaphore(int starting\_value, string name);  ~Semaphore();  void down(int taskID);  void up();  void dump(int level);  void setMCB(MCB\* mcb);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor  Semaphore::Semaphore(int starting\_value, string name)  {  semaValue = starting\_value;  resourceName = name;  luckyTask = -1;  pthread\_mutex\_init(&mux,NULL);  }  //---------------------------------------------  //Destructor  Semaphore::~Semaphore()  {  while(semaQueue.size() > 0)  semaQueue.dequeue();  semaValue = -99;  resourceName = "";  luckyTask = -1;  pthread\_mutex\_destroy(&mux);  }  //---------------------------------------------  //down() - Get the hold of the resource or get queued.  void Semaphore::down(int taskID)  {  pthread\_mutex\_lock(&mux);  if (taskID == luckyTask)  {  sprintf(buffer," Task# %d own the resource. REQUEST IGNORED!\n", luckyTask);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  dump(1);  }  else  {  if (semaValue >= 1)  {  semaValue--;  luckyTask = taskID; // preserve the task-id who got the resource  dump(1);  }  else  {  semaQueue.enqueue(taskID);  mcb->mysch->setState(taskID, BLOCKED);  dump(1);  mcb->mysch->yield();  dump(1);  }  }  pthread\_mutex\_unlock(&mux);  }  //---------------------------------------------  //up() - Release the hold of the resource.  void Semaphore::up()  {  pthread\_mutex\_lock(&mux);  int task\_id;  sprintf(buffer," TaskID: %d LuckyID: %d \n", mcb->mysch->getCurrTaskID(), luckyTask);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  if (mcb->mysch->getCurrTaskID() == luckyTask) // check to see if the correct task is doing the up()  {  if (semaQueue.empty())  {  semaValue++;  luckyTask = -1;  dump(1);  }  else  {  task\_id = semaQueue.top()->data; // Remove from queue and unblock  semaQueue.dequeue();  mcb->mysch->setState(task\_id, READY); // set the task to READY  sprintf(buffer," UnBlock: %d nd release from the queue.\n", task\_id);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  luckyTask = task\_id; //  sprintf(buffer," Luck Task = %d \n", luckyTask);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  dump(1);  mcb->mysch->yield();  dump(1);  }  }  else  {  sprintf(buffer," Invalid UP() - TaskID : %d does not own the resource.\n", mcb->mysch->getCurrTaskID());  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  dump(1);  }  pthread\_mutex\_unlock(&mux);  }  //---------------------------------------------  //dump() - Output information over the Semaphore.  void Semaphore::dump(int level)  {  sprintf(buffer," \n-------Semaphore DUMP-------\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  switch (level)  {  case 0:  sprintf(buffer," Sema\_Value: %d\n", semaValue);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);    sprintf(buffer," Sema\_Name: ");  strcat(buffer,resourceName.c\_str());  strcat(buffer,"\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer," Obtained by Task-ID: %d \n", luckyTask);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  break;  case 1:  sprintf(buffer," Sema\_Value: %d\n", semaValue);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);    sprintf(buffer," Sema\_Name: ");  strcat(buffer,resourceName.c\_str());  strcat(buffer,"\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer," Obtained by Task-ID: %d \n", luckyTask);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  //semaQueue.dumpQueue(mcb->mywin->Misc\_Table);  break;  default:  sprintf(buffer," ERROR in Semaphore DUMP level\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  }  sprintf(buffer,"----------------------------\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  }  //setMCB(MCB) - Register the Master Control Block  // mechanism so that Semaphore class  // have access to other mechanisms  // such as the IPC, Scheduler...  void Semaphore::setMCB(MCB\* mcb)  {  this->mcb = mcb;  }  //---------------------------------------------  #endif |

### 

### ThreadTable.h

|  |
| --- |
| /\*  Filename: ThreadTable.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To implement a thread table structure alongside with functions to maneuver it.  \*/  #ifndef THREAD\_TABLE\_H  #define THREAD\_TABLE\_H  #include <iostream>  #include <curses.h>  using namespace std;  #ifndef THREAD\_STATES  #define THREAD\_STATES  enum STATE  {  READY,  RUNNING,  BLOCKED,  DEAD  };  #endif  struct TCB  {  enum STATE threadState;  int threadID;  pthread\_t processID;  clock\_t threadStartTime;  WINDOW \*threadWIN;  TCB \*next;  };  class ThreadTable  {  private:  TCB \*head;  TCB \*tail;  int threadTableSize;  public:  ThreadTable();  ~ThreadTable();  bool empty();  void setTCBStartTime(int mythreadID, clock\_t myStartTime);  clock\_t getTCBStartTime(int mythreadID);  void insertTCB(int mythreadID, clock\_t myStartTime, STATE mythreadState, WINDOW \*myThreadWIN);  void removeATCB(int mythreadID);  void removeLastTCB();  void removeFirstTCB();  bool findTCB(int mythreadID);  TCB \*getTCB(int mythreadID);  pthread\_t \*getTCBProcessID(int mythreadID);  TCB \*getTCBHead();  TCB \*getTCBTail();  int getThreadTableSize();  void setTCBState(int mythreadID, STATE myState);  STATE getTCBState(int mythreadID);  void dumpThreadTable();  WINDOW \*getTCBWIN(int mythreadID);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor.  ThreadTable::ThreadTable()  {  head = NULL;  threadTableSize = 0;  }  //----------------------------------------  //Destructor.  ThreadTable::~ThreadTable()  {  TCB \*tmp = head;  while (tmp)  {  tmp = tmp->next;  delete (head);  head = tmp;  }  head = tail = NULL;  threadTableSize = 0;  }  //----------------------------------------  //empty() - Return if thread table is empty.  bool ThreadTable::empty()  {  if (threadTableSize == 0)  return true;  else  return false;  }  //----------------------------------------  //setTCBStartTime(int,clock\_t) - Set the value of the thread  // start time to a specified time.  void ThreadTable::setTCBStartTime(int mythreadID, clock\_t myStartTime)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  {  tmp->threadStartTime = myStartTime;  break;  }  tmp = tmp->next;  }  }  //----------------------------------------  //getTCBStartTime(int) - Get the start time of a specified  // thread number.  clock\_t ThreadTable::getTCBStartTime(int mythreadID)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  return tmp->threadStartTime;  tmp = tmp->next;  }  }  //----------------------------------------  //insertTCB(int, clock\_t, STATE) - Insert a new TCB block at  // the end of thread Table.  void ThreadTable::insertTCB(int mythreadID, clock\_t myStartTime, STATE mythreadState, WINDOW \*myThreadWIN)  {  TCB \*newBlock = new TCB;  newBlock->threadID = mythreadID;  newBlock->threadStartTime = myStartTime;  newBlock->threadState = mythreadState;  newBlock->threadWIN = myThreadWIN;  newBlock->next = NULL;  if (empty())  head = tail = newBlock;  else  {  tail->next = newBlock;  tail = newBlock;  }  threadTableSize++;  }  //----------------------------------------  //removeATCB(int) - Remove a specific TCB of the thread table.  void ThreadTable::removeATCB(int threadID)  {  if (head->threadID == threadID)  removeFirstTCB();  else if (tail->threadID == threadID)  removeLastTCB();  else  {  TCB \*tmp = head;  TCB \*prev;  while (tmp->next && tmp->threadID != threadID)  {  prev = tmp;  tmp = tmp->next;  }  if (tmp->threadID == threadID)  {  prev->next = tmp->next;  delete tmp;  threadTableSize--;  }  else  exit(-1);  }  }  //----------------------------------------  //removeLastTCB() - Remove last TCB block of thread table.  void ThreadTable::removeLastTCB()  {  if (head->next == NULL)  delete (head);  else  {  TCB \*tmp = head;  while (tmp->next->next != NULL)  tmp = tmp->next;  delete (tmp->next);  tmp->next = NULL;  tail = tmp;  }  threadTableSize--;  }  //----------------------------------------  //removeFirstTCB() - Remove first TCB block of thread table.  void ThreadTable::removeFirstTCB()  {  if (!empty())  {  TCB \*tmp = head->next;  delete (head);  head = tmp;  }  threadTableSize--;  }  //----------------------------------------  //findTCB(int) - Return a thread specific TCB.  bool ThreadTable::findTCB(int mythreadID)  {  TCB \*tmp = head;  while (tmp)  {  if (mythreadID == tmp->threadID)  return true;  tmp = tmp->next;  }  return false;  }  //----------------------------------------  //getTCB() - Return a pointer to TCB searched by threadID.  TCB \*ThreadTable::getTCB(int mythreadID)  {  TCB \*tmp = head;  while (tmp)  {  if (mythreadID == tmp->threadID)  return tmp;  tmp = tmp->next;  }  cout << "getTCB - TCB not found! Return exit(-1)." << endl;  exit(-1);  }  //----------------------------------------  //getTCBProcessID(int) - Return a process ID pthread\_t of a TCB.  pthread\_t \*ThreadTable::getTCBProcessID(int mythreadID)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  return &tmp->processID;  tmp = tmp->next;  }  cout << "getTCBProcessID - ProcessID not found! Return exit(-1)." << endl;  exit(-1);  }  //getTCBHead() - Return a pointer to first TCB in thread table.  TCB \*ThreadTable::getTCBHead()  {  return head;  }  //----------------------------------------  //getTCBTail() - Return a pointer to last TCB in thread table.  TCB \*ThreadTable::getTCBTail()  {  return tail;  }  //----------------------------------------  //getThreadTableSize() - Return thread table size.  int ThreadTable::getThreadTableSize()  {  return threadTableSize;  }  //----------------------------------------  //setTCBState(int,STATE) - Set TCB state of a specified threadID.  void ThreadTable::setTCBState(int mythreadID, STATE myState)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  {  tmp->threadState = myState;  break;  }  tmp = tmp->next;  }  }  //----------------------------------------  //getTCBState(int) - Get a specified TCB state.  STATE ThreadTable::getTCBState(int mythreadID)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  {  return tmp->threadState;  }  tmp = tmp->next;  }  cout << "GetTCBState - TCB not found! Return exit(-1)." << endl;  exit(-1);  }  //----------------------------------------  //dumpThreadTable() - Output everything from over thread table.  void ThreadTable::dumpThreadTable()  {  cout << "..... START ThreadTable Dump ..... " << endl;  TCB \*tmp = head;  if (!empty())  {  while (tmp != NULL)  {  cout << "TaskNUM: " << tmp->threadID << endl;  cout << "TaskID: " << tmp->threadID << endl;  cout << "TaskState: ";  switch (tmp->threadState)  {  case 0:  cout << "READY" << endl;  break;  case 1:  cout << "RUNNING" << endl;  break;  case 2:  cout << "BLOCKED" << endl;  break;  case 3:  cout << "DEAD" << endl;  break;  }  cout << "..............................." << endl;  tmp = tmp->next;  }  cout << "..... END ThreadTable Dump ..... " << endl;  }  else  {  cout << "EMPTY TCB - RETURN EXIT(-1)" << endl;  exit(-1);  }  }  //----------------------------------------  WINDOW \*ThreadTable::getTCBWIN(int mythreadID)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  {  return tmp->threadWIN;  }  tmp = tmp->next;  }  cout << "GetTCBWIN - TCB not found! Return exit(-1)." << endl;  exit(-1);  }  #endif |

### 

### Window.h

|  |
| --- |
| /\*  Filename: Semaphore.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To implement a small GUI for  Ultima using curses library.  \*/  #ifndef WINDOW\_H  #define WINDOW\_H  #include <curses.h>  #include <iostream>  #include <curses.h>  #include <stdarg.h>  #include <fcntl.h>  #include "MCB.h"  class MCB;  using namespace std;  class Window  {  private:  MCB \*mcb;  pthread\_mutex\_t mux\_Win;  public:  WINDOW \*Heading\_Win;  WINDOW \*Log\_Win;  WINDOW \*Console\_Win;  WINDOW \*Misc\_Table;  Window();  ~Window();  WINDOW \*create\_window(int height, int width, int starty, int startx);  void write\_window(WINDOW \*Win, const char \*text);  void write\_window(WINDOW \*Win, int x, int y, const char \*text);  void display\_help(WINDOW \*Win);  void setMCB(MCB \*mcb);  void cont\_help(WINDOW \*Win);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  // create a windows to display thread data in  // create a new windows: WINDOW \* win = newwin(nlines, ncols, y0, x0);  //Constructor  Window::Window()  {  //Initializing mutex  pthread\_mutex\_init(&mux\_Win, NULL);  // Start nCurses  initscr();  //Enable terminal color  start\_color();  init\_pair(1, COLOR\_BLACK, COLOR\_WHITE);  wbkgd(Misc\_Table, COLOR\_PAIR(1));  //End of terminal color.  refresh();  int Y, X;  int Max\_X, Max\_Y;  getmaxyx(stdscr, Max\_Y, Max\_X); //Get screen size  wprintw(stdscr, "Current Y = %d , Current X = %d\nGiovanna Gorski - C435 Operating Systems", Y, X);  refresh();  //Starting Heading Window  Heading\_Win = create\_window(12, 83, 3, 2);  write\_window(Heading\_Win, 2, 28, "ULTIMA 2.0 (Spring 2020)");  write\_window(Heading\_Win, 9, 2, "Press 'q' or Ctrl-C to exit the program...");  //end of Heading Window  //Starting Log\_Win  Log\_Win = create\_window(15, 60, 30, 2);  write\_window(Log\_Win, 1, 5, ".......Log....\n");  //----------------------------------------  Console\_Win = create\_window(15, 23, 30, 62);  write\_window(Console\_Win, 1, 1, "....Console....\n");  write\_window(Console\_Win, 2, 1, "Ultima # ");  Misc\_Table = create\_window(42, 83, 3, 85);  write\_window(Misc\_Table, 1, 1, "...Miscellaneous...\n");  cbreak();  noecho();  nodelay(Console\_Win, true);  }  //----------------------------------------  //Destructor  Window::~Window()  {  endwin(); //End the curses window  pthread\_mutex\_destroy(&mux\_Win);  }  //----------------------------------------  //create\_window() - To create a box window.  WINDOW \*Window::create\_window(int height, int width, int starty, int startx)  {  WINDOW \*Win;  Win = newwin(height, width, starty, startx);  scrollok(Win, TRUE); // Allow scrolling of the window  pthread\_mutex\_lock(&mux\_Win);  scroll(Win); // scroll the window  box(Win, 0, 0); // 0, 0 gives default characters  // for the vertical and horizontal lines  wrefresh(Win); // draw the window  wclear(Win);  pthread\_mutex\_unlock(&mux\_Win);  return Win;  }  //----------------------------------------  //write\_window() - To write text in specific window.  void Window::write\_window(WINDOW \*Win, const char \*text)  {  pthread\_mutex\_lock(&mux\_Win);  wprintw(Win, text);  box(Win, 0, 0);  wrefresh(Win); // draw the window  pthread\_mutex\_unlock(&mux\_Win);  }  //----------------------------------------  //write\_window() - To write text in specific window location.  void Window::write\_window(WINDOW \*Win, int x, int y, const char \*text)  {  pthread\_mutex\_lock(&mux\_Win);  mvwprintw(Win, x, y, text);  box(Win, 0, 0);  wrefresh(Win); // draw the window  pthread\_mutex\_unlock(&mux\_Win);  }  //----------------------------------------  //display\_help() - To construct a well formated  // help menu in console window.  void Window::display\_help(WINDOW \*Win)  {  wclear(Win);  write\_window(Win, 1, 1, "...Help...");  write\_window(Win, 2, 1, "1= Kill 1");  write\_window(Win, 3, 1, "2= Kill 2");  write\_window(Win, 4, 1, "3= Kill 3");  write\_window(Win, 5, 1, "n= Next Menu");  write\_window(Win, 6, 1, "c= clear screen");  write\_window(Win, 7, 1, "h= help screen");  write\_window(Win, 8, 1, "q= Quit");  }  //------------------------------------------------------  //cont\_help() - To construct a well formated  // help menu in console window.  void Window::cont\_help(WINDOW \*Win)  {  wclear(Win);  write\_window(Win, 1, 1, "...Next Menu...");  write\_window(Win, 2, 1, "s= Semaphore");  write\_window(Win, 3, 1, "i= IPC");  write\_window(Win, 4, 1, "g= Garbage Collect");  write\_window(Win, 5, 1, "m= Memory Manager");  write\_window(Win, 6, 1, "b= Back to previous");  }  //----------------------------------------  //setMCB(MCB) - Register the Master Control Block  // mechanism so that Semaphore class  // have access to other mechanisms  // such as the IPC, Scheduler...  void Window::setMCB(MCB \*mcb)  {  this->mcb = mcb;  }  //----------------------------------------  #endif |

### 

### 

## 

## Output

## Demonstration of Curses Library

## 

## Demonstration of Memory Management

## 

## Testing Strategy

To guarantee the success of this phase, phase 3 was separated into two parts in which the first part was designed and implemented a class Window so that a small graphic user interface was created for feasibility of debugging and testing of the Ultima 2.0. The second part follows the development of phase 3 by creating a memory manager class that acts as a memory management mechanism. Moreover, by dividing the phase into two separate parts, each new implemented class could be tested individually by using unit testing techniques. The objective of this approach is to guarantee the correctness of each small implemented member function of each developed class.

For the first part, a new copy of Ultima phase 2 was created and modified so that all the output statements(cout) would be displayed in their respective window. By doing this, it was noticed the necessity of registering a Window object with the Master Control Block so that any mechanism inside Ultima could utilize some important function such as write\_window() which is responsible for drawing and displaying texts within a created window.

Similarly, for the second part another new copy of Ultima phase 2 was created and modified so that the concept of memory and memory management could exist in Ultima. At the end both parts were combined to become phase 3.

## Discussion

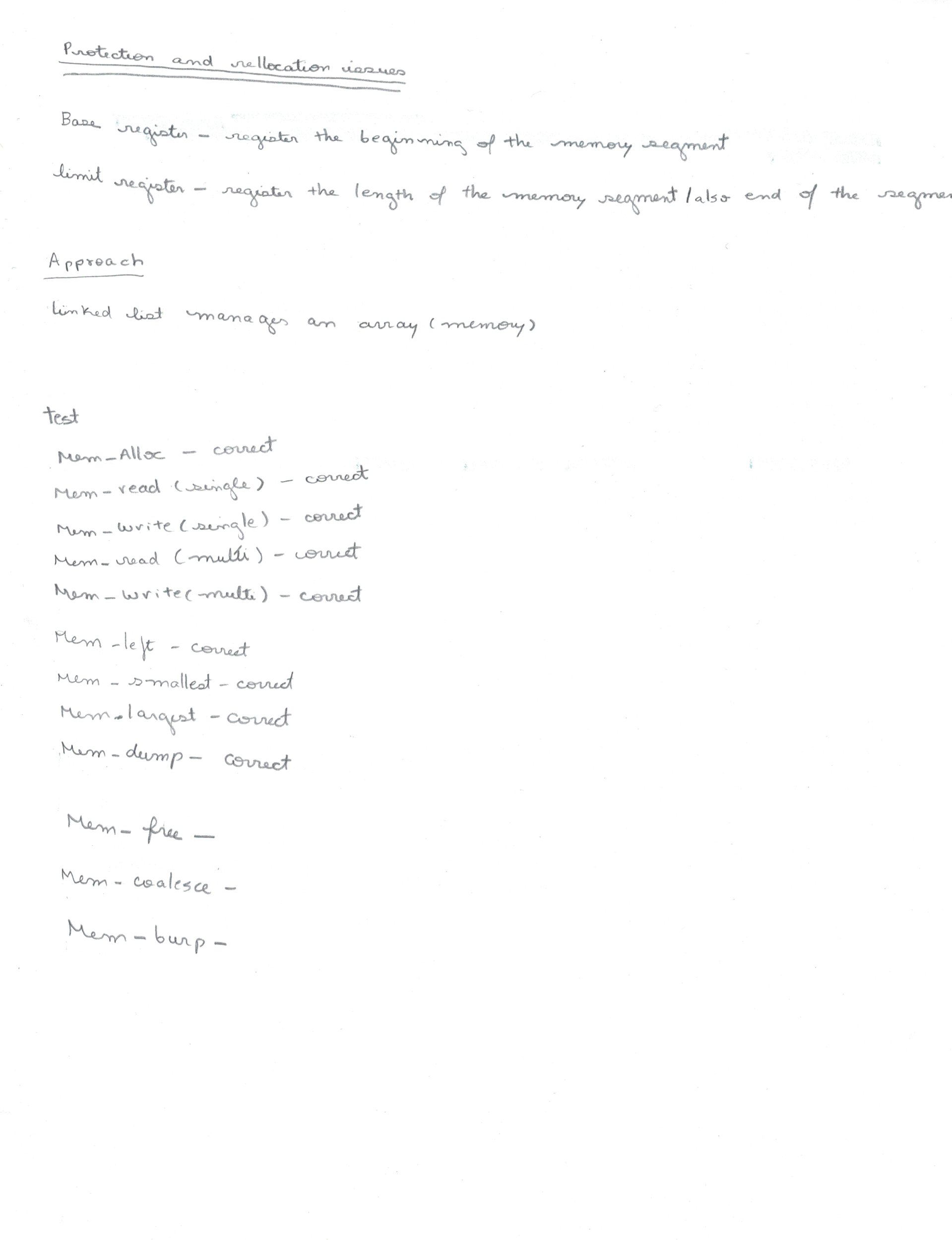
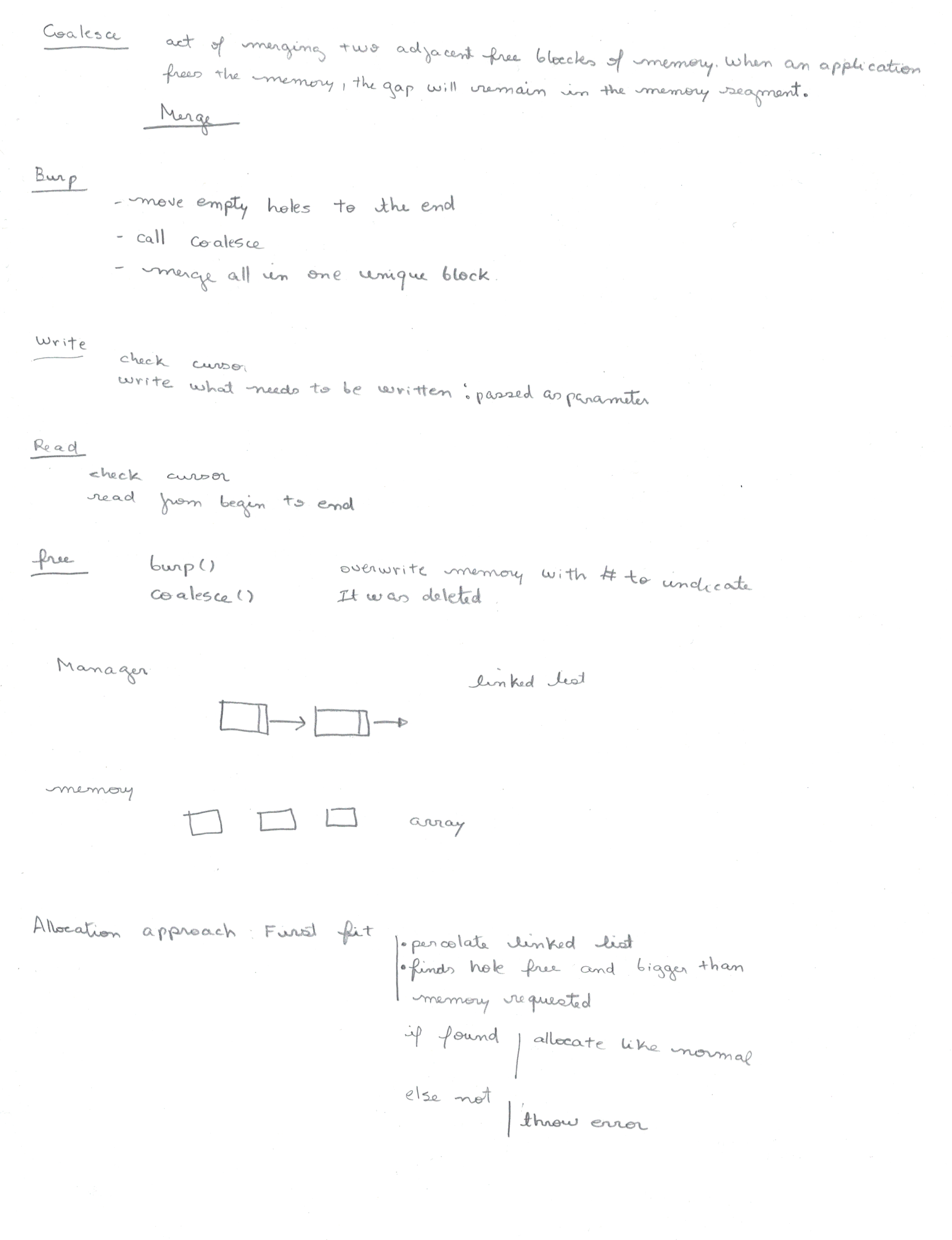
For this phase, much time was spent during design than implementation. Furthermore, the implementation of phase 3 was divided into two parts: curses phase, and memory management phase. During week 1, a big effort was made so that a small user interface was created by using some functions available in the Curses library.

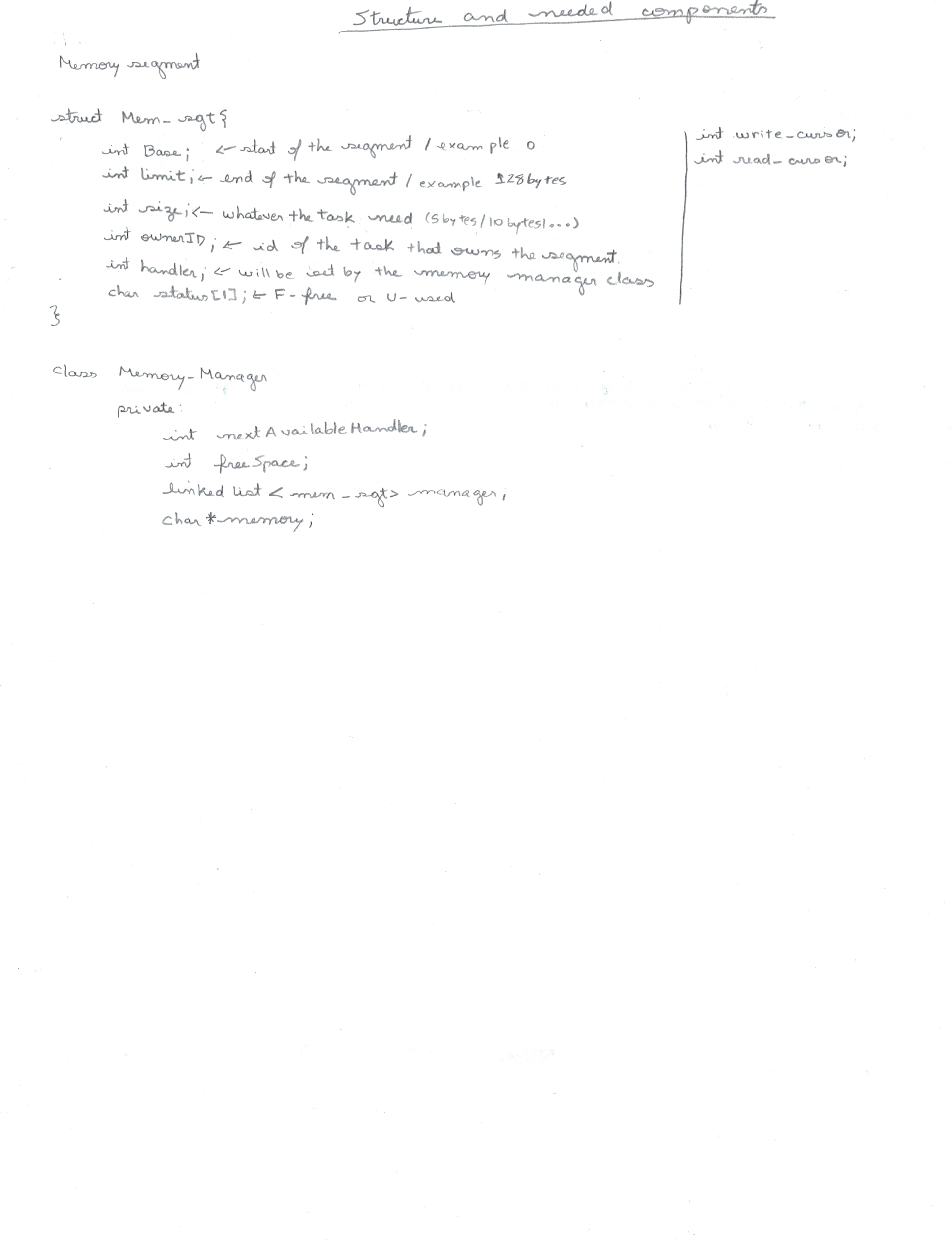
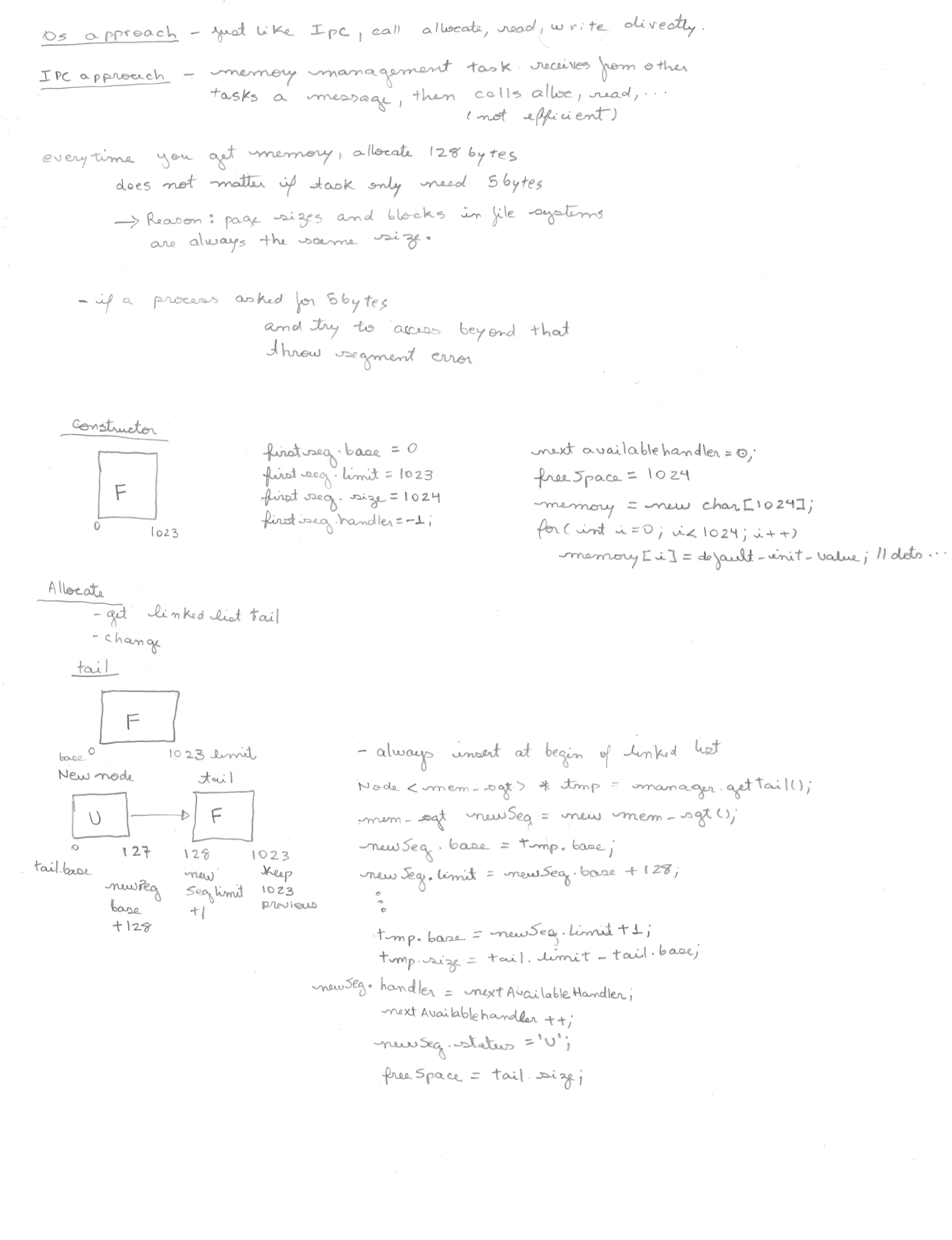
By using lab 5 and lab 2 from the course C435 Operating System, it was possible to develop a class Window in which member functions such as create\_window(), write\_window(), display\_help(), and cont\_help() were created. At the start of the main program, a Window class object is created and registered with the Master Control block so that other mechanisms can use some member functions such as write\_window() to write things in a specific window passed as parameter in the function.

During week 2, much time was spent during design than implementation of the class Memory Manager. The class memory manager contains important member functions capable of gerencing a shared resource which is a character array that acts as a memory. Furthermore, the available memory is 1024 bytes, and when a task requests for memory, the task becomes the owner of a location of size 128 bytes. All these requests alongside writing and reading from memory are protected by a Semaphore set specific for memory management, the semaphore guarantees that a task is not writing or reading a memory location being worked on by another task. In addition, ownership is honored by checking if the request for a location comes from the owner task ID of the block being requested.

Below are some brainstorming ideas previous the implementation of the class memory manager:

## Brainstorm Ideas





## Project Schedule

# Phase IV

## File system

April 27, 2020

Individual work

## Phase Abstract

This phase is the fourth and last part of the ULTIMA 2.0 project for the course C435 – Operating Systems at Indiana University of South Bend. The purpose of this phase is to develop a file system mechanism which allows the tasks that are running to create files so that one can write or read according to permissions set during creation. Moreover, a file system allows information stored to be persistent which means that it can be preserved as well as it can be loaded back in the next program execution or Ultima execution. Altogether, additional functions for file systems were implemented such as directory dumps, datafile (or disk) dump, read character, and write character. Furthermore, some touch ups were accomplished in the GUI for feasibility of debugging and testing.

*Keywords*: Windows,GUI,File system, Curses, inodes

## Phase Description

## 

## 

## 

## Design Documentation

## 

## Source code

### Main.cpp

|  |
| --- |
| #include <iostream>  #include <sched.h>  #include <time.h>  #include <pthread.h>  #include "Semaphore.h"  #include "Scheduler.h"  #include "MemoryMgmt.h"  #include "Ufs.h"  #include "IPC.h"  using namespace std;  #include "MCB.h"  void startMCB();  int main()  {  startMCB();  char buff[256];  int input = -1;  int CPU\_Quantum = 1;  int t\_id;  t\_id = mcb->mysch->create\_task();  t\_id = mcb->mysch->create\_task();  t\_id = mcb->mysch->create\_task();  //t\_id = mcb->mysch->create\_task(); //It should display error if uncomment  int error;  mcb->mysch->start();  //---------------------------------------------------------------------------------  while (input != 'q')  {  input = wgetch(mcb->mywin->Console\_Win);  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  switch (input)  {  case '0':  case '1':  case '2':  if (input == '0')  mcb->mysch->destroyTask(0);  else if (input == '1')  mcb->mysch->destroyTask(1);  else  mcb->mysch->destroyTask(2);  sprintf(buff, " %c\n", input);  mcb->mywin->write\_window(mcb->mywin->Console\_Win, buff);  sprintf(buff, " Kill = %c\n", input);  mcb->mywin->write\_window(mcb->mywin->Console\_Win, buff);  wclear(mcb->mywin->Log\_Win);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  sleep(4);  wclear(mcb->mywin->Console\_Win);  mcb->mywin->write\_window(mcb->mywin->Console\_Win, 1, 1, "Ultima # ");  break;  case 'c':  refresh(); //clear the console window  wclear(mcb->mywin->Console\_Win);  mcb->mywin->write\_window(mcb->mywin->Console\_Win, 1, 1, "Ultima # ");  break;  case 'g':  mcb->mysch->garbageCollector();  break;  case 'h':  case 'b':  mcb->mywin->display\_help(mcb->mywin->Console\_Win);  mcb->mywin->write\_window(mcb->mywin->Console\_Win, 8, 1, "Ultima # ");  break;  case 'i':  error = mcb->myIPC->sendMessage(0, 1, TEXT, "Testing my main...");  if (error == 0)  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " Sending message...\n");  mcb->myIPC->dumpIPC();  //mcb->myIPC->receiveMessage(1);  break;  case 'n':  mcb->mywin->cont\_help(mcb->mywin->Console\_Win);  mcb->mywin->write\_window(mcb->mywin->Console\_Win, 8, 1, "Ultima # ");  break;  case 'm':  //Allocation test  wclear(mcb->mywin->Misc\_Table);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " ....Allocation test\n");  error = mcb->mymemo->Mem\_alloc(128, 1);  if (error == 0)  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " Block allocated with success!\n");  else  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " Error allocating block!\n");  //Writing test  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " ....Writing test\n");  error = mcb->mymemo->Mem\_write(0, 0, 16, "this is task one", 1);    //Reading test  char \*ch;  mcb->mymemo->Mem\_read(0,ch,1);  mcb->mymemo->Mem\_read(0,0,3,ch,1);  //Free test  //mcb->mymemo->Mem\_free(0,1);  //Memory Dump  mcb->mymemo->Mem\_dump();  break;  case 's':  //---------- DOWN  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to obtain semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema->down(t\_id); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to obtain semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema->down(t\_id); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to obtain semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema->down(t\_id); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  //---------- UP  t\_id = mcb->mysch->getCurrTaskID();  mcb->mysema->up(); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  mcb->mysema->up(); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  mcb->mysema->up(); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  break;  case 'u':  mcb->mywin->write\_window(mcb->mywin->Misc\_Table," ...Testing filesystem...\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table," ...Creating a file...\n");  mcb->myufs->Create\_file(0,"f1",75,"1100");  mcb->myufs->Create\_file(1,"f2",130,"1101");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table," ...Dump Metada file...\n");    mcb->myufs->Dir();    //Uncomment below to dump data file.  //mcb->mywin->write\_window(mcb->mywin->Misc\_Table,"...Dump Data file...\n");  //mcb->myufs->dump();    //Uncomment below to test access permission.  /\* int f\_id;  mcb->mywin->write\_window(mcb->mywin->Misc\_Table," ...Testing Permissions... \n");  //mcb->myufs->Dir(0);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table," Testing USER READ \n");  f\_id = mcb->myufs->Open(0,0,"f1",READ);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table," Testing USER WRITE \n");  f\_id = mcb->myufs->Open(0,0,"f1",WRITE);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table," Testing OTHERS READ \n");  f\_id = mcb->myufs->Open(1,0,"f1",READ);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table," Testing OTHERS WRITE \n");  f\_id = mcb->myufs->Open(1,0,"f1",WRITE); \*/  //Uncomment below to dump data file.  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,"...Dump Data file...\n");  mcb->myufs->dump();  //cout << newFS->Change\_permission(0,"f1","0000") << endl;  char b[256];  sprintf(b," Open f\_id: %d\n",mcb->myufs->Open(1,1,"f2",WRITE));  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b," Open f\_id: %d\n",mcb->myufs->Open(0,0,"f1",READ));  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,"...Write char...\n");    mcb->myufs->Write\_char(1,0,'H');  mcb->myufs->Write\_char(1,0,'i');  //Uncomment below to dump data file.  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,"...Dump Data file...\n");  mcb->myufs->dump();  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,"...Read char...\n");  mcb->myufs->Read\_char(0,1,ch);  break;  case 'q':  //end the loop, and end the program.  case ERR:  //if wgetch() return ERR, that means no keys were pressed  //earlier we enabled non-blocking input using nodelay() see above  //this allows the program to continue to inspect the keyboard without  //having to wait for the key to be pressed;  break;  default:  sprintf(buff, " %c\n", input);  mcb->mywin->write\_window(mcb->mywin->Console\_Win, buff);  mcb->mywin->write\_window(mcb->mywin->Console\_Win, " - Invalid Command\n");  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, " - Invalid Command\n");  mcb->mywin->write\_window(mcb->mywin->Console\_Win, " Ultima #");  break;  }  //sleep(1);  CPU\_Quantum++;  }  endwin(); //End the curses window  return 0;  }  void startMCB()  {  Scheduler \*newScheduler = new Scheduler();  Semaphore \*newSemaphore = new Semaphore(1, "resourceTest");  Semaphore \*newSemaphore1 = new Semaphore(1,"IpC");  Semaphore \*newSemaphore2 = new Semaphore(1, "memoryManager");  Semaphore \*newSemaphore3 = new Semaphore(1,"FileSystem");  IPC \*newIPC = new IPC();  Window \*newWin = new Window();  MemoryManager \*newMemo = new MemoryManager();      Ufs \*newUfs = new Ufs("root",16,128,'^');    mcb = new MCB(newScheduler, newSemaphore,newSemaphore1, newSemaphore2, newSemaphore3, newIPC, newWin, newMemo,newUfs);  mcb->mysch->setMCB(mcb); //Scheduler  mcb->mysema->setMCB(mcb); //Regular semaphore.  mcb->mysema1->setMCB(mcb); //IPC semaphore.  mcb->mysema2->setMCB(mcb); //Memory Manager semaphore  mcb->mysema3->setMCB(mcb); //Filesystem semaphore.  mcb->mymemo->setMCB(mcb); //Memory manager.  mcb->myIPC->setMCB(mcb); //Inter process communication.  mcb->myufs->setMCB(mcb); //Filesystem.  } |

### IPC.h

|  |
| --- |
| /\*  Filename: IPC.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To define create a inter process communication mechanism (IPC).  PHASE II  \*/  #ifndef IPC\_H  #define IPC\_H  #include <iostream>  #include <string>  #include <ctime> //So that arrival time is output pretty  #include "LinkedList.h"  #include "Queue.h"  #include "MCB.h"  using namespace std;  #ifndef MSG\_TYPE  #define MSG\_TYPE  enum MSGTYPE  {  TEXT, //No action required.  SERVICE, //Request for a service by the task.  NOTIFICATION // Sent to a task to indicate a service was completed.  };  #endif  struct Message  {  int src\_id;  int dst\_id;  string arrived\_time;  MSGTYPE msg\_type;  string msg\_content;  };  class IPC  {  private:  MCB \*mcb;  char buffer[256];  public:  LinkedList<Queue<Message>> mailboxes;  int createMailbox(int task\_id);  int sendMessage(int src\_id, int dst\_id, MSGTYPE msg\_type, string msg\_content);  void receiveMessage(int task\_id);  int messageCount(int task\_id);  void removeAMailbox(int task\_id);  void deleteAllMailboxes();  void dumpIPC();  void dumpMailbox(int task\_id);  void setMCB(MCB \*mcb);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //createMailbox(int) - Create a mailbox queue inside the  // linked list Mailboxes so that a  // Scheduler::createTask thread can  // have its own mailbox "address".  int IPC::createMailbox(int task\_id)  {  Queue<Message> newBox;  mailboxes.insertBack(newBox, task\_id);  //Return 0 if successful created the box, or -1 if fails  if (mailboxes.size() > 0)  return 0;  else  return -1;  }  //----------------------------------------  //sendMessage() - Sends a message from a thread source  // to a thread destination. The message  // is enqueued in the destination thread's  // mailbox.  int IPC::sendMessage(int src\_id, int dst\_id, MSGTYPE msg\_type, string msg\_content)  {  time\_t now = time(0);  Message newMsg;  newMsg.arrived\_time = asctime(localtime(&now)); //https://www.tutorialspoint.com/cplusplus/cpp\_date\_time.htm  newMsg.dst\_id = dst\_id;  newMsg.src\_id = src\_id;  newMsg.msg\_type = msg\_type;  newMsg.msg\_content = msg\_content;  //Check if there the destination's mailbox exist  if (mailboxes.isNode(dst\_id))  {  //mcb->mysema1->down(src\_id);  mailboxes.searchNode(dst\_id)->data.enqueue(newMsg, dst\_id);  //mcb->mysema1->up();  return 0;  }  else  {  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " Impossible to send message, mailbox does not exist!\n");  return -1;  }  }  //----------------------------------------  //receiveMessage(int) - Goes into a specific thread mailbox,  // retrieves its first message from its  // queue by displaying it, and removing  // it from its mailbox. This insures  // that the most recent message is read.  void IPC::receiveMessage(int task\_id)  {  //Check if there a mailbox for the task\_id  if (mailboxes.isNode(task\_id))  {  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " ...Receiving message...\n");  Node<Message> \*tmp = mailboxes.searchNode(task\_id)->data.top();  sprintf(buffer, " \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " | | SourceID | DestID | Arrived Time \t| MessageType\t| Content \t|\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " ------------------------------------------------------------------------------------------------------\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  //Treating date so that \n is removed.  string date;  for (int i = 0; i < tmp->data.arrived\_time.size(); i++)  if (tmp->data.arrived\_time[i] != '\n')  date[i] = tmp->data.arrived\_time[i];  //--------  //Treating message type so that display pretty.  switch (tmp->data.msg\_type)  {  case 0:  sprintf(buffer, " | | %d \t| %d \t | %s \t| TEXT \t\t| %s \t\t|\n", tmp->data.src\_id, tmp->data.dst\_id, date.c\_str(), tmp->data.msg\_content.c\_str());  break;  case 1:  sprintf(buffer, " | | %d \t| %d \t | %s \t| SERVICE \t\t| %s \t\t|\n", tmp->data.src\_id, tmp->data.dst\_id, date.c\_str(), tmp->data.msg\_content.c\_str());  break;  case 2:  sprintf(buffer, " | | %d \t| %d \t | %s \t| NOTIFICATION \t\t| %s \t\t|\n", tmp->data.src\_id, tmp->data.dst\_id, date.c\_str(), tmp->data.msg\_content.c\_str());  break;  }  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  //---------------  sprintf(buffer, " ------------------------------------------------------------------------------------------------------\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  //mcb->mysema1->down(task\_id);  mailboxes.searchNode(task\_id)->data.dequeue(); //if yes, reads the first message and removes it  //mcb->mysema1->up();  }  else  {  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " Impossible to obtain message, mailbox do not exist!\n");  }  }  //----------------------------------------  //messageCount(int) - Return how many messages there is  // in a thread's mailbox.  int IPC::messageCount(int task\_id)  {  if (mailboxes.isNode(task\_id))  return mailboxes.searchNode(task\_id)->data.size();  else  return -1;  }  //----------------------------------------  //removeAMailbox(int) - Delete a specific thread mailbox.  void IPC::removeAMailbox(int task\_id)  {  if (mailboxes.isNode(task\_id))  {  //Remove all messages from the mailbox.  while (mailboxes.searchNode(task\_id)->data.size() > 0)  mailboxes.searchNode(task\_id)->data.dequeue();  }  mailboxes.removeNode(task\_id);  }  //deleteAllMailboxes() - Delete all thread's mailboxes.  void IPC::deleteAllMailboxes()  {  while (mailboxes.size() > 0)  mailboxes.removeBack();  }  //----------------------------------------  //dumpIPC() - Output all mailboxes within IPC.  void IPC::dumpIPC()  {  for (int i = 0; i < mailboxes.size(); i++)  {  dumpMailbox(i);  }  }  //----------------------------------------  //dumpMailbox(int) - Output a thread specific mailbox.  void IPC::dumpMailbox(int task\_id)  {  //mcb->mysema1->down(task\_id);  sprintf(buffer, "\n ---- MAILBOX Thread # %d -----\n", task\_id);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  Node<Message> \*tmp = mailboxes.searchNode(task\_id)->data.top();  sprintf(buffer, " \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " | # | SourceID | DestID | Arrived Time \t| MessageType\t| Content \t|\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " ------------------------------------------------------------------------------------------------------\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  for (int j = 0; j < mailboxes.searchNode(task\_id)->data.size(); j++)  {  //Treating date so that \n is removed.  string date;  for (int i = 0; i < tmp->data.arrived\_time.size(); i++)  if (tmp->data.arrived\_time[i] != '\n')  date[i] = tmp->data.arrived\_time[i];  //--------  //Treating message type so that display pretty.  switch (tmp->data.msg\_type)  {  case 0:  sprintf(buffer, " | %d | %d \t| %d \t | %s \t| TEXT \t\t| %s \t\t|\n", j, tmp->data.src\_id, tmp->data.dst\_id, date.c\_str(), tmp->data.msg\_content.c\_str());  break;  case 1:  sprintf(buffer, " | %d | %d \t| %d \t | %s \t| SERVICE \t\t| %s \t\t|\n", j, tmp->data.src\_id, tmp->data.dst\_id, date.c\_str(), tmp->data.msg\_content.c\_str());  break;  case 2:  sprintf(buffer, " | %d | %d \t| %d \t | %s \t| NOTIFICATION \t\t| %s \t\t|\n", j, tmp->data.src\_id, tmp->data.dst\_id, date.c\_str(), tmp->data.msg\_content.c\_str());  break;  }  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  //---------------  sprintf(buffer, " ------------------------------------------------------------------------------------------------------\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  tmp = tmp->next;  }  sprintf(buffer, " \n ------------------------------------------------------------------------------------------------------\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  //mcb->mysema1->up();  }  //----------------------------------------  //setMCB(MCB) - Register the Master Control Block  // mechanism so that Semaphore class  // have access to other mechanisms  // such as the IPC, Scheduler...  void IPC::setMCB(MCB \*mcb)  {  this->mcb = mcb;  }  //----------------------------------------  #endif |

### LinkedList.h

|  |
| --- |
| /\*  Filename: LinkedList.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To create a linked list data structure.  \*/  #ifndef LINK\_LIST\_H  #define LINK\_LIST\_H  #include <iostream>  #include <curses.h>  using namespace std;  #include "Node.h"  template <class T>  class LinkedList  {  private:  Node<T> \*head;  Node<T> \*tail;  int listSize;  pthread\_mutex\_t mux;  char buffer[256];    public:  LinkedList();  ~LinkedList();  void insertFront(T data, int num = 0);  void insertBack(T data, int num = 0);  void removeFront();  void removeBack();  void removeNode(T data);  void removeNode(int num);  Node<T> \*front();  Node<T> \*back();  int size();  virtual bool empty();  Node<T> \*searchNode(T data);  Node<T> \*searchNode(int data);  bool isNode(int data);  void dumpList();  void dumpList(WINDOW \*win);  void write\_window(WINDOW \*Win, const char \*text);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor.  template <class T>  LinkedList<T>::LinkedList()  {  head = tail = NULL;  listSize = 0;  pthread\_mutex\_init(&mux, NULL);  }  //----------------------------------------  //Destructor.  template <class T>  LinkedList<T>::~LinkedList()  {  Node<T> \*tmp = head;  while (tmp)  {  tmp = tmp->next;  delete (head);  head = tmp;  }  head = tail = NULL;  listSize = 0;  pthread\_mutex\_destroy(&mux);  }  //----------------------------------------  //insertFront(T, int) - Insert node at the begin  // of the linked list.  template <class T>  void LinkedList<T>::insertFront(T data, int num)  {  Node<T> \*newNode = new Node<T>;  newNode->data = data;  newNode->num = num;  newNode->next = NULL;  if (empty())  {  head = tail = newNode;  }  else  {  newNode->next = head;  head = newNode;  }  listSize++;  }  //----------------------------------------  //insertBack(T,int) - Insert node at the end of linked list.  template <class T>  void LinkedList<T>::insertBack(T data, int num)  {  Node<T> \*newNode = new Node<T>;  newNode->data = data;  newNode->num = num;  newNode->next = NULL;  if (empty())  head = tail = newNode;  else  {  tail->next = newNode;  tail = newNode;  }  listSize++;  }  //----------------------------------------  //removeFront() - Remove node at the begin of the list.  template <class T>  void LinkedList<T>::removeFront()  {  if (!empty())  {  Node<T> \*tmp = head->next;  delete (head);  head = tmp;  }  listSize--;  }  //----------------------------------------  //removeBack() - Remove node at the end of the list.  template <class T>  void LinkedList<T>::removeBack()  {  if (head->next == NULL){  delete (head);  head = tail = NULL;  }  else  {  Node<T> \*tmp = head;  while (tmp->next->next != NULL)  tmp = tmp->next;  delete (tmp->next);  tmp->next = NULL;  tail = tmp;  }  listSize--;  }  //----------------------------------------  //removeNode() - Remove node by specification type T.  template <class T>  void LinkedList<T>::removeNode(T data)  {  if (head->data == data)  removeFront();  else if (tail->data == data)  removeBack();  else  {  Node<T> \*tmp = head;  Node<T> \*prev;  while (tmp->next && tmp->data != data)  {  prev = tmp;  tmp = tmp->next;  }  if (tmp->data == data)  {  prev->next = tmp->next;  delete tmp;  listSize--;  }  else  exit(-1);  }  }  //----------------------------------------  //removeNode(int) - Remove node by specification type int.  template <class T>  void LinkedList<T>::removeNode(int num)  {  if (num == 0)  removeFront();  else if (listSize == num)  removeBack();  else  {  Node<T> \*tmp = head;  Node<T> \*prev;  int count = 0;  while (tmp->next && count != num)  {  prev = tmp;  tmp = tmp->next;  count++;  }  if (count == num)  {  prev->next = tmp->next;  delete tmp;  listSize--;  }  else  exit(-1);  }  }  //----------------------------------------  //front() - Return head/first node of the linked list.  template <class T>  Node<T> \*LinkedList<T>::front()  {  return head;  }  //back() - Return the last node of the linked list.  template <class T>  Node<T> \*LinkedList<T>::back()  {  return tail;  }  //----------------------------------------  //size() - Return size of linked list.  template <class T>  int LinkedList<T>::size()  {  return listSize;  }  //----------------------------------------  //empty() - Return if linked list is empty or not.  template <class T>  bool LinkedList<T>::empty()  {  if (listSize > 0)  return false;  else  return true;  }  //----------------------------------------  //searchNode(T) - Search node by specification type T.  template <class T>  Node<T> \*LinkedList<T>::searchNode(T data)  {  Node<T> \*tmp = new Node<T>;  tmp = head;  while (tmp)  {  if (data == tmp->data)  return tmp;  tmp = tmp->next;  }  exit(-1);  }  //----------------------------------------  //searchNode(int) - Search node by specification type int.  template <class T>  Node<T> \*LinkedList<T>::searchNode(int data)  {  Node<T> \*tmp = new Node<T>;  tmp = head;  while (tmp)  {  if (data == tmp->num)  return tmp;  tmp = tmp->next;  }  exit(-1);  }  //----------------------------------------  //isNode(int) - Return if there is such a node in list.  template <class T>  bool LinkedList<T>::isNode(int data)  {  Node<T> \*tmp = new Node<T>;  tmp = head;  while (tmp)  {  if (data == tmp->num)  return true;  tmp = tmp->next;  }  return false;  }  //----------------------------------------  //dumpList() - Output the linked list if there is one,  // if not exit with error.  template <class T>  void LinkedList<T>::dumpList()  {  Node<T> \*tmp = head;  if (!empty())  {  while (tmp != NULL)  {  cout << tmp->data << " ";  tmp = tmp->next;  }  cout << endl;  }  else  cout << "---- EMPTY ----\n";  }  //dumpList() - Output the linked list if there is one,  // if not exit with error.  template <class T>  void LinkedList<T>::dumpList(WINDOW \*win)  {  Node<T> \*tmp = head;  if (!empty())  {  while (tmp != NULL)  {  sprintf(buffer, "%d ", tmp->data);  write\_window(win, buffer);  tmp = tmp->next;  }  write\_window(win, "\n");  }  else  write\_window(win, " ---- EMPTY ----\n");  }  //----------------------------------------  //write\_window() - To display things at specific  // curses window.  template <class T>  void LinkedList<T>::write\_window(WINDOW \*Win, const char \*text)  {  pthread\_mutex\_lock(&mux);  wprintw(Win, text);  box(Win, 0, 0);  wrefresh(Win); // draw the window  pthread\_mutex\_unlock(&mux);  }  #endif |

### 

### MCB.h

|  |
| --- |
| /\*  Filename: MCB.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To create a Master Control Block mechanism in which registers  other mechanisms such as IPC, Scheduler, Semaphore...  Furthermore, its registration allows the usage of such mechanisms  without controlling access issues.  PHASE II  \*/  #ifndef MCB\_H  #define MCB\_H  #include <iostream>  #include "Scheduler.h"  #include "Semaphore.h"  #include "Window.h"  class Semaphore;  class Scheduler;  class IPC;  class Window;  class MemoryManager;  class Ufs;  class MCB  {  public:  Scheduler \*mysch;  Semaphore \*mysema; //Regular semaphore.  Semaphore \*mysema1; //IPC semaphore.  Semaphore \*mysema2; //Memory Management semaphore.  Semaphore \*mysema3; //Filesystem semaphore.  IPC \*myIPC;  Window \*mywin;  MemoryManager \*mymemo;  Ufs \*myufs;  MCB(Scheduler \*theScheduler, Semaphore \*theSemaphore, Semaphore \*theSemaphore1,Semaphore \*theSemaphore2, Semaphore \*theSemaphore3, IPC \*theIPC, Window \*theWIN, MemoryManager \*theMemo, Ufs \*theUfs);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor  MCB::MCB(Scheduler \*theScheduler, Semaphore \*theSemaphore, Semaphore \*theSemaphore1,Semaphore \*theSemaphore2, Semaphore \*theSemaphore3, IPC \*theIPC, Window \*theWIN, MemoryManager \*theMemo, Ufs \*theUfs)  {  this->myIPC = theIPC;  this->mysch = theScheduler;  this->mywin = theWIN;  this->mymemo = theMemo;  this->myufs = theUfs;  this->mysema = theSemaphore; //Regular semaphore.  this->mysema1 = theSemaphore1; //IPC semaphore.  this->mysema2 = theSemaphore2; //Memory Management semaphore.  this->mysema3 = theSemaphore3; //Filesystem semaphore.  }  //----------------------------------------  //----------------------------------------  //---- G L O B A L O B J E C T ----  //----------------------------------------  MCB \*mcb;  //----------------------------------------  #endif |

### 

### MemoryMgmt.h

|  |
| --- |
| /\*  Filename: MemoryMgmt.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To create a memory management mechanism.  Phase III  \*/  #ifndef MEMORY\_MGMT\_H  #define MEMORY\_MGMT\_H  #include <iostream>  #include <fstream>  using namespace std;  #include "LinkedList.h"  #include "MCB.h"  class MCB;  enum MEMORY\_LABEL  {  FREE,  USED  };  struct Segment  {  int base; //Start of the memory segment.  int limit; //End of the memory segment.  int size; //Number of bytes requested by task.  int ownerID; //ID of the owner task.  int handler; //set by memory manager class.  MEMORY\_LABEL status; //0 - free, 1 - used.  int write\_cursor;  int read\_cursor;  };  class MemoryManager  {  private:  int nextAvailableHandler;  int freeSpace; //How much free memory I have.  LinkedList<Segment> MemoManager;  char \*memory; //Actually memory resource.  MCB \*mcb;  char buffer[2024];  int Mem\_left();  int Mem\_largest();  int Mem\_smallest();  int Mem\_coalesce();  int Mem\_burp();  public:  MemoryManager();  ~MemoryManager();  int Mem\_dump();  int Mem\_alloc(int size, int taskID);  int Mem\_free(int memory\_handle, int taskID);  int Mem\_read(int memory\_handle, char \*ch, int taskID);  int Mem\_write(int memory\_handle, char ch, int taskID);  //Overloaded multi-byte read and write  int Mem\_read(int memory\_handle, int offset\_from\_beg, int text\_size, char \*text, int taskID);  int Mem\_write(int memory\_handle, int offset\_from\_beg, int text\_size, char \*text, int taskID);  void setMCB(MCB \*mcb) { this->mcb = mcb; }  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor - Initialize some basic variables  //as well as insert first segment in memory which  //is equivalent of the whole memory as free status  //.i.e. a block of status free from 0 to 1023.  MemoryManager::MemoryManager()  {  //Initialize basic variables.  nextAvailableHandler = 0;  freeSpace = 1024;  //Initialize actual memory by  //setting capacity to 1024 bytes  //and overwritting with dots.  memory = new char[1024];  for (int i = 0; i < 1024; i++)  memory[i] = '.';  //First segment of the manager should display  //the entire empty memory.  Segment newSeg;  newSeg.base = 0;  newSeg.limit = 1023;  newSeg.ownerID = -1;  newSeg.handler = -1;  newSeg.size = 1024;  newSeg.status = FREE;  newSeg.write\_cursor = newSeg.read\_cursor = 0;  MemoManager.insertFront(newSeg, newSeg.handler);  }  //----------------------------------------  //Destructor.  MemoryManager::~MemoryManager()  {  nextAvailableHandler = 0;  freeSpace = 1024;  MemoManager.~LinkedList();  for (int i = 0; i < 1024; i++)  {  memory[i] = '.';  buffer[i]=' ';  }  mcb = NULL;  }  //----------------------------------------  //Mem\_alloc() - Uses first fit approach.  //Finds the first hole big enough then allocate,  //if not then throw -1 for error.  int MemoryManager::Mem\_alloc(int size, int taskID)  {  if (freeSpace < size)  return -1; //Segment fault. No available memory.  else  {  Node<Segment> \*tmp = MemoManager.back();  //Create new segments at the from of the linked  //list manager.  Segment newSeg;  newSeg.base = tmp->data.base;  if (MemoManager.size() == 0)  newSeg.base = 0;  else  newSeg.limit = newSeg.base + 127;  newSeg.ownerID = taskID;  newSeg.handler = nextAvailableHandler;  newSeg.size = size;  newSeg.status = USED;  newSeg.write\_cursor = newSeg.read\_cursor = newSeg.base;  //Adjust the last node which would be the free  //location by setting new values for base and size.  tmp->data.base = newSeg.limit + 1;  tmp->data.size = tmp->data.limit - tmp->data.base;  tmp->data.write\_cursor = tmp->data.base;  tmp->data.read\_cursor = tmp->data.base;  freeSpace = tmp->data.size;  MemoManager.insertFront(newSeg);  //Handle increment  nextAvailableHandler++;  return 0; //Success allocating memory.  }  }  //----------------------------------------  //Mem\_free() - Check if there is such segment in memory  //and if requesting task owns the segment then if  //yes free the respective segment. If not of those  //apply, throw -1 for error.  int MemoryManager::Mem\_free(int memory\_handle, int taskID)  {  //Check if there is such segment in memory.  if (MemoManager.isNode(memory\_handle))  {  Node<Segment> \*tmp = MemoManager.front();  while (tmp)  {  if (tmp->data.handler == memory\_handle)  break;  tmp = tmp->next;  }  //Check if taskID is the owner(have permission).  if (tmp->data.ownerID == taskID)  {  //Start of free up memory:  for (int i = tmp->data.base; i <= tmp->data.limit; i++)  memory[i] = '#';  //Reset cursor's location to point to the beginning of the block.  tmp->data.write\_cursor = tmp->data.read\_cursor = tmp->data.base;  tmp->data.status = FREE;  tmp->data.size = 0;  freeSpace += (tmp->data.limit - tmp->data.base);    int err = Mem\_coalesce();  if (err == -1)  {  sprintf(buffer, " No coalesce!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  }    return 0; //Success.  }  else  return -1; //Access denied, task does not own the memory block.  }  else  return -1; //Segment not found.  }  //----------------------------------------  //Mem\_read() - Overloaded function / Single Byte Read  //The below reads ONE character then increments read\_cursor location  //(current\_location adjustment).  int MemoryManager::Mem\_read(int memory\_handle, char \*ch, int taskID)  {  mcb->mysema2->down(taskID);  //Check if there is such segment in memory.  if (MemoManager.isNode(memory\_handle))  {  Node<Segment> \*tmp = MemoManager.front();  while (tmp)  {  if (tmp->data.handler == memory\_handle)  break;  tmp = tmp->next;  }  //Check if taskID is the owner(have permission).  if (tmp->data.ownerID == taskID)  {  ch = &memory[tmp->data.read\_cursor]; //Read single character.  tmp->data.read\_cursor++;  sprintf(buffer, " -----------------\n Content of Read: %c\n-----------------\n", \*ch);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  //Checks if the new read\_cursor position goes beyond  //the writer\_cursor, if yes then set read\_cursor to the base  //of the block.  if (tmp->data.read\_cursor >= tmp->data.limit)  {  tmp->data.read\_cursor = tmp->data.base;  }  mcb->mysema2->up();  return 0; //Success reading.  }  else  {  mcb->mysema2->up();  return -1; //Access denied, task does not own the memory block.  }  }  else  {  mcb->mysema2->up();  return -1; //Segment not found.  }  }  //----------------------------------------  //Mem\_write() - Overloaded function / Single Byte Write  //The below writes ONE character then increments write\_cursor location.  int MemoryManager::Mem\_write(int memory\_handle, char ch, int taskID)  {  mcb->mysema2->down(taskID);  //Check if there is such segment in memory.  if (MemoManager.isNode(memory\_handle))  {  Node<Segment> \*tmp = MemoManager.front();  while (tmp)  {  if (tmp->data.handler == memory\_handle)  break;  tmp = tmp->next;  }  //Check if taskID is the owner(have permission).  if (tmp->data.ownerID == taskID)  {  //Check if write\_cursor reached the limit of the segment block.  if (tmp->data.write\_cursor == tmp->data.limit)  {  mcb->mysema2->up();  return -1; //Segment is full.  }  else  {  memory[tmp->data.write\_cursor] = ch; //Write in it.  tmp->data.write\_cursor++; //adjust write\_cursor.  mcb->mysema2->up();  return 0; //Success writing.  }  }  else  {  mcb->mysema2->up();  return -1; //Access denied, task does not own the memory block.  }  }  else  {  mcb->mysema2->up();  return -1; //Segment not found  }  }  //----------------------------------------  //Overloaded multi-byte read and write  //Mem\_read() - Read a large memory location.  int MemoryManager::Mem\_read(int memory\_handle, int offset\_from\_beg, int text\_size, char \*text, int taskID)  {  mcb->mysema2->down(taskID);  //Check if there is such segment in memory.  if (MemoManager.isNode(memory\_handle))  {  Node<Segment> \*tmp = MemoManager.front();  while (tmp)  {  if (tmp->data.handler == memory\_handle)  break;  tmp = tmp->next;  }  //Check if TaskID owns the block  if (tmp->data.ownerID == taskID)  {  //Check if what I am reading goes beyond my block limit  if (tmp->data.limit < offset\_from\_beg + tmp->data.read\_cursor + text\_size)  {  mcb->mysema2->up();  return -1; //Trying to reach more than it should.  }  else  {  text = new char[offset\_from\_beg + 1];  text[offset\_from\_beg + 1] = '\0';  for (int i = 0; i < text\_size; i++)  {  text[i] = memory[tmp->data.read\_cursor];  tmp->data.read\_cursor++;  }  mcb->mywin->write\_window(mcb->mywin->Misc\_Table," -----------------\n Content of read: \n");  strcpy(buffer,text);  strcat(buffer,"\n-----------------\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,buffer);  mcb->mysema2->up();  return 0; //Success reading.  }  }  else{  mcb->mysema2->up();  return -1; //Access denied, task does not own the memory block.  }  }  else  {  mcb->mysema2->up();  return -1; //Segment not found.  }  }  //----------------------------------------  //Mem\_write() - Write a large text into memory.  int MemoryManager::Mem\_write(int memory\_handle, int offset\_from\_beg, int text\_size, char \*text, int taskID)  {  mcb->mysema2->down(taskID);  //Check if there is such segment in memory.  if (MemoManager.isNode(memory\_handle))  {  Node<Segment> \*tmp = MemoManager.front();  while (tmp)  {  if (tmp->data.handler == memory\_handle)  break;  tmp = tmp->next;  }  //Check if TaskID owns the block  if (tmp->data.ownerID == taskID)  {  //Check if what I am writing goes beyond my block limit  if (tmp->data.limit < offset\_from\_beg + tmp->data.write\_cursor + text\_size)  return -1; //Trying to reach more than it should.  else  {  for (int i = 0; i < text\_size; i++)  {  memory[tmp->data.write\_cursor + offset\_from\_beg] = text[i];  tmp->data.write\_cursor++;  }  tmp->data.write\_cursor += offset\_from\_beg;  mcb->mysema2->up();  return 0; //Success writing.  }  }  else  {  mcb->mysema2->up();  return -1; //Access denied, task does not own the memory block.  }  }  else  {  mcb->mysema2->up();  return -1; //Segment not found.  }  }  //----------------------------------------  //Mem\_left() - Return how much free space is  //still available.  int MemoryManager::Mem\_left()  {  return freeSpace;  }  //Mem\_largest - Return largest block in size.  int MemoryManager::Mem\_largest()  {  Node<Segment> \*tmp = MemoManager.front();  int largest = tmp->data.size; //Set the lasgest to be the first block found.  for (int i = 0; i < MemoManager.size(); i++)  {  if (tmp->data.size > largest && tmp->data.status == FREE)  largest = tmp->data.size;  tmp = tmp->next;  }  return largest;  }  //----------------------------------------  //Mem\_smallest() - Return smallest block in size.  int MemoryManager::Mem\_smallest()  {  Node<Segment> \*tmp = MemoManager.front();  int smallest = tmp->data.size; //Set the lasgest to be the first block found.  for (int i = 0; i < MemoManager.size(); i++)  {  if (tmp->data.size > smallest && tmp->data.status == FREE)  smallest = tmp->data.size;  tmp = tmp->next;  }  return smallest;  }  //Mem\_coalesce() - Merge two adjacent free blocks of memory.  int MemoryManager::Mem\_coalesce()  {  Mem\_dump();  Node<Segment> \*curr = MemoManager.front();  Node<Segment> \*prev;  int i = -1;  while (curr)  {  if (prev->data.status == FREE && curr->data.status == FREE)  {  for (int i = prev->data.base; i <= prev->data.limit; i++)  memory[i] = '.';  for (int i = curr->data.base; i <= curr->data.limit; i++)  memory[i] = '.';  curr->data.base = prev->data.base;  curr->data.size = 0;  curr->data.write\_cursor = curr->data.read\_cursor = 0;  MemoManager.removeNode(i);  Mem\_dump();  return 0; //Success coalesce.  }  i++;  prev = curr;  curr = curr->next;  }  return -1; //Error coalesce.  }  //----------------------------------------  //Mem\_burp() - Move empty holes to the end of MemoryManager list.  int MemoryManager::Mem\_burp()  {  //Check if there are empty holes.  if (freeSpace > 0)  {  Node<Segment> \*tmp = MemoManager.front();  for (int i = 0; i < MemoManager.size(); i++)  {  //Check if is an empty hole.  if (tmp->data.status == FREE)  {  //Copy the empty hole  Segment newSeg;  newSeg.base = tmp->data.base;  newSeg.handler = tmp->data.handler;  newSeg.limit = tmp->data.limit;  newSeg.ownerID = tmp->data.ownerID;  newSeg.read\_cursor = tmp->data.read\_cursor;  newSeg.size = tmp->data.size;  newSeg.status = tmp->data.status;  newSeg.write\_cursor = tmp->data.write\_cursor;  //Remove respective empty hole  MemoManager.removeNode(i);  //Add it at the end of MemoryManager list.  MemoManager.insertBack(newSeg);  }  tmp = tmp->next;  }  return 0; //Success in moving holes.  }  else  return -1; //No empty holes.  }  //----------------------------------------  //Mem\_dump() - Display all information over the memory manager.  int MemoryManager::Mem\_dump()  {  Node<Segment> \*tmp = MemoManager.front();  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, "\n------------------\n ---- Memory Dump ----\n");  for (int i = 0; i < MemoManager.size(); i++)  {  sprintf(buffer, " Handler: %d\n", tmp->data.handler);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " Owner: %d\n", tmp->data.ownerID);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " Base: %d\n", tmp->data.base);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " Limit: %d\n", tmp->data.limit);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " Size: %d\n", tmp->data.size);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " Status: ");  if (tmp->data.status == FREE)  strcat(buffer, " FREE\n");  else  strcat(buffer, " USED\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " Read\_cursor: %d\n", tmp->data.read\_cursor);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  sprintf(buffer, " Write\_cursor: %d\n", tmp->data.write\_cursor);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, "---------------------------\n");  tmp = tmp->next;  }  strcpy(buffer," ");  strcat(buffer,memory);  strcat(buffer,"\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  }  #endif |

## 

### Node.h

|  |
| --- |
| /\*  Filename: Node.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To define the structure of a Node for a linked list, or queue.  \*/  #ifndef NODE\_H  #define NODE\_H  #include <iostream>  using namespace std;  template <class T>  struct Node  {  T data;  int num;  Node<T> \*next;  };  #endif |

### 

### Queue.h

|  |
| --- |
| /\*  Filename: Queue.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To create a queue data structure.  \*/  #ifndef QUEUE\_LIST\_H  #define QUEUE\_LIST\_H  #include "LinkedList.h"  #include <iostream>  using namespace std;  #include "Node.h"  #include <curses.h>  template <class T>  class Queue  {  private:  int queueSize;  Node<T> \*head;  pthread\_mutex\_t mux;  char buffer[256];  public:  Queue();  ~Queue();  void enqueue(T data, int num = 0);  void dequeue();  bool empty();  Node<T> \*top();  Node<T> \*top(WINDOW \*win);  int size();  void dumpQueue();  void dumpQueue(WINDOW \*win);  void write\_window(WINDOW \*Win, const char \*text);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor  template <class T>  Queue<T>::Queue()  {  head = NULL;  queueSize = 0;  pthread\_mutex\_init(&mux, NULL);  }  //----------------------------------------  //Destructor  template <class T>  Queue<T>::~Queue()  {  Node<T> \*tmp = head;  while (tmp)  {  tmp = tmp->next;  delete (head);  head = tmp;  }  head = NULL;  queueSize = 0;  pthread\_mutex\_destroy(&mux);  }  //----------------------------------------  //empty() - Returns if Queue is empty or not.  template <class T>  bool Queue<T>::empty()  {  if (size() > 0)  return false;  else  return true;  }  //----------------------------------------  //enqueue(T,int) - Insert node at the begin of  // the queue list.  template <class T>  void Queue<T>::enqueue(T data, int num)  {  Node<T> \*newNode = new Node<T>;  newNode->data = data;  newNode->num = num;  newNode->next = NULL;  if (empty())  {  head = newNode;  }  else  {  newNode->next = head;  head = newNode;  }  queueSize++;  }  //----------------------------------------  //dequeue() - Remove node at the begin of the list.  template <class T>  void Queue<T>::dequeue()  {  if (!empty())  {  Node<T> \*tmp = head->next;  delete (head);  head = tmp;  }  queueSize--;  }  //----------------------------------------  //top() - Return the last node of the linked list.  template <class T>  Node<T> \*Queue<T>::top()  {  if (head)  return head;  else  {  //cout << "Not found!";  }  }  template <class T>  Node<T> \*Queue<T>::top(WINDOW \*win)  {  if (head)  return head;  else  {  write\_window(win, "Not found!\n");  }  }  //----------------------------------------  //size() - Return size of linked list.  template <class T>  int Queue<T>::size()  {  return queueSize;  }  //----------------------------------------  //dumpQueue() - Output all node in Queue.  template <class T>  void Queue<T>::dumpQueue()  {  Node<T> \*tmp = head;  if (!empty())  {  while (tmp != NULL)  {  cout << tmp->data << " | ";  tmp = tmp->next;  }  cout << endl;  }  else  cout << "---- EMPTY ----" << endl;  }  //dumpQueue() - Output all node in Queue.  template <class T>  void Queue<T>::dumpQueue(WINDOW \*win)  {  Node<T> \*tmp = head;  if (!empty())  {  while (tmp != NULL)  {  sprintf(buffer, " %d \n", tmp->data);  write\_window(win, buffer);  tmp = tmp->next;  }  }  else  write\_window(win, " ---- EMPTY ----\n");  }  //----------------------------------------  //write\_window() - To display things at specific curses  // window.  template <class T>  void Queue<T>::write\_window(WINDOW \*Win, const char \*text)  {  pthread\_mutex\_lock(&mux);  wprintw(Win, text);  box(Win, 0, 0);  wrefresh(Win); // draw the window  pthread\_mutex\_unlock(&mux);  }  //----------------------------------------  #endif |

### 

### Scheduler.h

|  |
| --- |
| /\*  Filename: IPC.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To implement a Round Robin scheduler mechanism.  \*/  #ifndef SCHEDULER\_H  #define SCHEDULER\_H  #ifndef THREAD\_STATES  #define THREAD\_STATES  enum STATE  {  READY,  RUNNING,  BLOCKED,  DEAD  };  #endif  const int NO\_OF\_THREADS = 3;  #include <pthread.h>  #include <iostream>  #include <unistd.h>  #include <assert.h>  #include <string.h>  #include "ThreadTable.h"  #include "MCB.h"  #include "IPC.h"  using namespace std;  class Scheduler  {  private:  int currentThread;  long currentQuantum;  int nextAvailableThreadID;  ThreadTable threadTable;  MCB \*mcb;  char buffer[256];  pthread\_mutex\_t mux;  public:  Scheduler();  ~Scheduler();  int create\_task();  void \*worker(void \*arguments);  void start();  void yield();  void setQuantum(long quantum);  long getQuantum();  void setState(int task\_id, STATE the\_state);  int getCurrTaskID();  TCB \*getTCB(int task\_id);  void dump();  void wasteTime(int x);  void destroyTask(int task\_id);  void garbageCollector();  void setMCB(MCB \*mcb);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor  Scheduler::Scheduler()  {  currentThread = -1; //No active process.  nextAvailableThreadID = 0;  currentQuantum = 300; //default quantum 300ms.  }  //----------------------------------------  //Destructor  Scheduler::~Scheduler()  {  currentThread = -1; //No active process.  nextAvailableThreadID = 0;  currentQuantum = 300; //default quantum 300ms.  }  //----------------------------------------  //CreateTask() -- Create an operating system process.  int Scheduler::create\_task()  {  if (nextAvailableThreadID < NO\_OF\_THREADS)  {  sprintf(buffer, " Creating task # %d\n", nextAvailableThreadID);  mcb->mywin->write\_window(mcb->mywin->Heading\_Win, 5 + nextAvailableThreadID, 6, buffer);  WINDOW \*newTaskWIN = mcb->mywin->create\_window(10, 27, 15, 2 + (threadTable.getThreadTableSize() \* 28));  threadTable.insertTCB(nextAvailableThreadID, clock(), READY, newTaskWIN);  //Source: https://thispointer.com/c-how-to-pass-class-member-function-to-pthread\_create/  typedef void \*(\*THREADFUNCPTR)(void \*);  int err = pthread\_create(threadTable.getTCBProcessID(nextAvailableThreadID), NULL, (THREADFUNCPTR)&Scheduler::worker, this);  assert(!err);  mcb->myIPC->createMailbox(nextAvailableThreadID);  nextAvailableThreadID++;  return (nextAvailableThreadID - 1);  }  else  {  sprintf(buffer," Create task FAILED: Number of available threads exceeded the limit..\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,buffer);  return (-1); // return error in case cannot create a task.  }  }  //----------------------------------------  //\*worker() - Context swapping function.  void \*Scheduler::worker(void \*arguments)  {  int mythreadNum = 0;  char buffer[256];  //Find the TCB related to the thread running in the OS background.  while (\*threadTable.getTCBProcessID(mythreadNum) != pthread\_self())  mythreadNum++;  //As long as the OS background running thread is not DEAD  //and it is running do:  while ((threadTable.getTCBState(mythreadNum) != DEAD))  {  if (threadTable.getTCBState(mythreadNum) == RUNNING)  {  //Prints the pthread\_t of the running thread.  sprintf(buffer, " Task#%d running...\n", currentThread);  mcb->mywin->write\_window(threadTable.getTCBWIN(currentThread), buffer);  //Print all information over the Scheduler class.  dump();  //Call Scheduler::yield() so that current thread can  //voluntary give up the CPU to the next ready task.  yield();  }  else  {  //If not, just relinquish the CPU by the calling thread  //, insuring for example that the calling thread of  //a createTask() function call do not hold the CPU forever.  pthread\_yield();  }  }  }  //----------------------------------------  //start() - Initiate scheduling mechanism by  // setting the first task in thread table  // to running.  void Scheduler::start()  {  sprintf(buffer, " ... STARTING SCHEDULING...\n");  mcb->mywin->write\_window(mcb->mywin->Heading\_Win, 4, 1, buffer);  TCB \*tmp = threadTable.getTCBHead();  //Find the first READY process  //by jumping through DEAD processes  while (tmp->threadState == DEAD)  tmp = tmp->next;  //Set the first found READY process to state RUNNING, then begin scheduling  threadTable.setTCBStartTime(tmp->threadID, clock());  threadTable.setTCBState(tmp->threadID, RUNNING);  currentThread = tmp->threadID;  setQuantum(1000 / NO\_OF\_THREADS); // Set quantum to 1sec/number of threads  wasteTime(2);  }  //----------------------------------------  //yield() - runs the thread with set RUNNING state until  // its quantum runs out. When quantum runs out,  // set current RUNNING to READY then finds the next  // READY thread then set its state to RUNNING. Finally,  // puts the OS thread current running in background to sleep  // so that the next can have its turn to run.  void Scheduler::yield()  {  int counter = 0;  sprintf(buffer, " Current Task # %d is trying to yield.\n", currentThread);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  sprintf(buffer, " Current Quantum : %li\n", currentQuantum);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  // calculate elapsed\_time since the thread last started to run.  clock\_t elapsed\_time = clock() - threadTable.getTCBStartTime(currentThread);  sprintf(buffer, " Elapsed time: %li\n", elapsed\_time);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  if (elapsed\_time >= currentQuantum) // if quantum has run out  {  sprintf(buffer, " Yielding....(Switching from task #%d to next ready task).\n", currentThread);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  // if current thread is RUNNING make it READY (its quantum has run out)  if (threadTable.getTCBState(currentThread) == RUNNING)  threadTable.setTCBState(currentThread, READY);  // now find the next READY thread and make it running  // watch out for deadlocks. (no ready processes)  currentThread = threadTable.getTCBIter()->threadID;  while (threadTable.getTCBState(currentThread) != READY && counter < NO\_OF\_THREADS - 1)  {  currentThread = threadTable.getTCBIter()->threadID;  counter++;  }  // if we find a READY threads re-set the quantum and set the task to running  if (counter < NO\_OF\_THREADS - 1 && threadTable.getTCBState(currentThread) == READY)  {  threadTable.setTCBStartTime(currentThread, clock()); // restart the quantum  threadTable.setTCBState(currentThread, RUNNING);  sprintf(buffer, " Started Running task # %d \n", currentThread);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  //Sleep(1) in here, insures syncronization between the  //threads and the yield function  sleep(1);  }  else  {  sprintf(buffer, " ----- P O S S I B L E D E A D L O C K -----\n");  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  }  }  else  {  sprintf(buffer, " ---- N O Y I E L D -----\n");  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  }  }  //----------------------------------------  //setQuantum(long quantum) - Assign a passed value  // to currentQuantum.  void Scheduler::setQuantum(long quantum)  {  currentQuantum = quantum;  }  //----------------------------------------  //getQuantum() - Return currentQuantum value  long Scheduler::getQuantum()  {  return (currentQuantum);  }  //----------------------------------------  //setState(int,STATE) - Set the state of a specific task to  // a specific state.  void Scheduler::setState(int task\_id, STATE the\_state)  {  threadTable.setTCBState(task\_id, the\_state);  }  //----------------------------------------  //getTaskID() - Return currentThread ID.  int Scheduler::getCurrTaskID()  {  return currentThread;  }  //----------------------------------------  //getTCB() - Return a pointer to TCB searched  // by threadID.  TCB \*Scheduler::getTCB(int task\_id)  {  TCB \*tmp = threadTable.getTCBHead();  while (tmp)  {  if (task\_id == tmp->threadID)  return tmp;  tmp = tmp->next;  }  /\* sprintf(buffer," getTCB - TCB not found! Return exit(-1).");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer); \*/  exit(-1);  }  //----------------------------------------  //dump() - Ouput important messages over the Scheduler.  void Scheduler::dump()  {  clock\_t elapsed\_time = clock() - threadTable.getTCBStartTime(currentThread);  sprintf(buffer, " TASK # \t PROCESS ID \t\t TASK STATE \n---------------------------------------------------------\n");  mcb->mywin->write\_window(mcb->mywin->Tb\_Win, 1, 1, buffer);  TCB \*tmp = threadTable.getTCBHead();  while (tmp != NULL)  {  if (tmp->threadState == READY)  sprintf(buffer, " \t%d \t|%li \t|READY ", tmp->threadID, tmp->processID);  else if (tmp->threadState == RUNNING)  sprintf(buffer, " \t%d \t|%li \t|RUNNING ", tmp->threadID, tmp->processID);  else if (tmp->threadState == BLOCKED)  sprintf(buffer, " \t%d \t|%li \t|BLOCKED ", tmp->threadID, tmp->processID);  else if (tmp->threadState == DEAD)  sprintf(buffer, " \t%d \t|%li \t|DEAD ", tmp->threadID, tmp->processID);  char curr[] = "<--CURRENT\n";  if (tmp->threadID == currentThread)  strcat(buffer, curr);  else  strcat(buffer, "\n");  mcb->mywin->write\_window(mcb->mywin->Tb\_Win, buffer);  tmp = tmp->next;  }  mcb->mywin->write\_window(mcb->mywin->Tb\_Win, "---------------------------------------------------------\n");  sprintf(buffer, " Quantum: %li \t| Elapsed Time: %d\t|\n", currentQuantum, elapsed\_time);  mcb->mywin->write\_window(mcb->mywin->Tb\_Win1, buffer);  }  //----------------------------------------  //wasteTime(int) - runs a for loop for (x \*  // 10240) amount of times.  void Scheduler::wasteTime(int x)  {  unsigned long long Int64 = 0;  for (unsigned short i = 0; i < 10240 \* x; ++i)  {  for (unsigned short j = i; j > 0; --j)  Int64 += j + i;  }  }  //----------------------------------------  //garbageCollector() - Goes through the thread  // table finding all the  // state DEAD threads,  // after finding it, removes  // its TCB from thread table.  void Scheduler::garbageCollector()  {  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " ...Cleaning Garbage..\n");  for (int j = 0; j < NO\_OF\_THREADS; j++)  {  if (threadTable.findTCB(j) && threadTable.getTCBState(j) == DEAD)  {  threadTable.removeATCB(j);  mcb->myIPC->removeAMailbox(j);  break;  }  }  }  //----------------------------------------  //destroyTask(int) - Sets the state of a task  // to DEAD, then calls a pthread  // library function pthread\_cancel  // to cancel the running task in the  // background.  void Scheduler::destroyTask(int task\_id)  {  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " ...Killing Task..\n");  pthread\_cancel(\*threadTable.getTCBProcessID(task\_id));  threadTable.setTCBState(task\_id, DEAD);  }  //----------------------------------------  //setMCB(MCB) - Register the Master Control Block mechanism  // within the scheduler so that Scheduler  // can access other mechanisms such as IPC,Semaphore,  // ...  void Scheduler::setMCB(MCB \*mcb)  {  this->mcb = mcb;  }  //----------------------------------------  #endif |

### 

### Semaphore.h

|  |
| --- |
| /\*  Filename: Semaphore.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To implement a binary semaphore.  \*/  #ifndef Semaphore\_H  #define Semaphore\_H  #include <iostream>  #include <string>  using namespace std;  #include "Queue.h"  #include "Scheduler.h"  #include "MCB.h"  class Semaphore  {  private:  string resourceName; // The name of the resource being managed.  int semaValue; // 0 or 1 in the case of a binary Semaphore.  int luckyTask; // Preserve the task-id of the task that got the resource (for debbugging).  pthread\_mutex\_t mux; // Critical section lock mechanism.  Queue<int> semaQueue;  MCB \*mcb;  char buffer[256];  public:  Semaphore(int starting\_value, string name);  ~Semaphore();  void down(int taskID);  void up();  void dump();  void setMCB(MCB\* mcb);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor  Semaphore::Semaphore(int starting\_value, string name)  {  semaValue = starting\_value;  resourceName = name;  luckyTask = -1;  pthread\_mutex\_init(&mux,NULL);  }  //---------------------------------------------  //Destructor  Semaphore::~Semaphore()  {  while(semaQueue.size() > 0)  semaQueue.dequeue();  semaValue = -99;  resourceName = "";  luckyTask = -1;  pthread\_mutex\_destroy(&mux);  }  //---------------------------------------------  //down() - Get the hold of the resource or get queued.  void Semaphore::down(int taskID)  {  pthread\_mutex\_lock(&mux);  if (taskID == luckyTask)  {  sprintf(buffer," Task# %d own the resource. REQUEST IGNORED!\n", luckyTask);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  dump();  }  else  {  if (semaValue >= 1)  {  semaValue--;  luckyTask = taskID; // preserve the task-id who got the resource  dump();  }  else  {  semaQueue.enqueue(taskID);  mcb->mysch->setState(taskID, BLOCKED);  dump();  mcb->mysch->yield();  dump();  }  }  pthread\_mutex\_unlock(&mux);  }  //---------------------------------------------  //up() - Release the hold of the resource.  void Semaphore::up()  {  pthread\_mutex\_lock(&mux);  int task\_id;  sprintf(buffer," TaskID: %d LuckyID: %d \n", mcb->mysch->getCurrTaskID(), luckyTask);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  if (mcb->mysch->getCurrTaskID() == luckyTask) // check to see if the correct task is doing the up()  {  if (semaQueue.empty())  {  semaValue++;  luckyTask = -1;  dump();  }  else  {  task\_id = semaQueue.top()->data; // Remove from queue and unblock  semaQueue.dequeue();  mcb->mysch->setState(task\_id, READY); // set the task to READY  sprintf(buffer," Unblock: %d ,and release from the queue.\n", task\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  luckyTask = task\_id; //  sprintf(buffer," Lucky Task = %d \n", luckyTask);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  dump();  mcb->mysch->yield();  dump();  }  }  else  {  sprintf(buffer," Invalid UP() - TaskID : %d does not own the resource.\n", mcb->mysch->getCurrTaskID());  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buffer);  dump();  }  pthread\_mutex\_unlock(&mux);  }  //---------------------------------------------  //dump() - Output information over the Semaphore.  void Semaphore::dump()  {  mcb->mywin->write\_window(mcb->mywin->Sem\_Win,1,1, " SEMAPHORE DUMP\n-------------------\n");  sprintf(buffer," Value: %d\n", semaValue);  mcb->mywin->write\_window(mcb->mywin->Sem\_Win, buffer);    sprintf(buffer," Name: ");  strcat(buffer,resourceName.c\_str());  strcat(buffer,"\n");  mcb->mywin->write\_window(mcb->mywin->Sem\_Win, buffer);  sprintf(buffer," Owned by Task#: %d \n", luckyTask);  mcb->mywin->write\_window(mcb->mywin->Sem\_Win, buffer);  sprintf(buffer," Queue: ");  if(semaQueue.size() > 0)  {  char buffer2[256];  Node<int> \*tmp = semaQueue.top();    while(tmp!=NULL)  {  sprintf(buffer2," %d | ",tmp->data);  strcat(buffer,buffer2);  tmp = tmp->next;  }  strcat(buffer,"\n");    }else  {  strcat(buffer," EMPTY\n");  }  mcb->mywin->write\_window(mcb->mywin->Sem\_Win, buffer);  }  //setMCB(MCB) - Register the Master Control Block  // mechanism so that Semaphore class  // have access to other mechanisms  // such as the IPC, Scheduler...  void Semaphore::setMCB(MCB\* mcb)  {  this->mcb = mcb;  }  //---------------------------------------------  #endif |

### 

### ThreadTable.h

|  |
| --- |
| /\*  Filename: ThreadTable.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To implement a thread table structure alongside with functions to maneuver it.  \*/  #ifndef THREAD\_TABLE\_H  #define THREAD\_TABLE\_H  #include <iostream>  #include <curses.h>  using namespace std;  #ifndef THREAD\_STATES  #define THREAD\_STATES  enum STATE  {  READY,  RUNNING,  BLOCKED,  DEAD  };  #endif  struct TCB  {  enum STATE threadState;  int threadID;  pthread\_t processID;  clock\_t threadStartTime;  WINDOW \*threadWIN;  TCB \*next;  };  class ThreadTable  {  private:  TCB \*head;  TCB \*tail;  int threadTableSize;  TCB \*iter;  public:  ThreadTable();  ~ThreadTable();  bool empty();  void setTCBStartTime(int mythreadID, clock\_t myStartTime);  clock\_t getTCBStartTime(int mythreadID);  void insertTCB(int mythreadID, clock\_t myStartTime, STATE mythreadState, WINDOW \*myThreadWIN);  void removeATCB(int mythreadID);  void removeLastTCB();  void removeFirstTCB();  bool findTCB(int mythreadID);  TCB \*getTCB(int mythreadID);  pthread\_t \*getTCBProcessID(int mythreadID);  TCB \*getTCBHead();  TCB \*getTCBTail();  int getThreadTableSize();  void setTCBState(int mythreadID, STATE myState);  STATE getTCBState(int mythreadID);  void dumpThreadTable();  WINDOW \*getTCBWIN(int mythreadID);  TCB \* getTCBIter();  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor.  ThreadTable::ThreadTable()  {  iter = head = tail = NULL;  threadTableSize = 0;  }  //----------------------------------------  //Destructor.  ThreadTable::~ThreadTable()  {  TCB \*tmp = head;  while (tmp)  {  tmp = tmp->next;  delete (head);  head = tmp;  }  iter = head = tail = NULL;  threadTableSize = 0;  }  //----------------------------------------  //empty() - Return if thread table is empty.  bool ThreadTable::empty()  {  if (threadTableSize == 0)  return true;  else  return false;  }  //----------------------------------------  //setTCBStartTime(int,clock\_t) - Set the value of the thread  // start time to a specified time.  void ThreadTable::setTCBStartTime(int mythreadID, clock\_t myStartTime)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  {  tmp->threadStartTime = myStartTime;  break;  }  tmp = tmp->next;  }  }  //----------------------------------------  //getTCBStartTime(int) - Get the start time of a specified  // thread number.  clock\_t ThreadTable::getTCBStartTime(int mythreadID)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  return tmp->threadStartTime;  tmp = tmp->next;  }  }  //----------------------------------------  //insertTCB(int, clock\_t, STATE) - Insert a new TCB block at  // the end of thread Table.  void ThreadTable::insertTCB(int mythreadID, clock\_t myStartTime, STATE mythreadState, WINDOW \*myThreadWIN)  {  TCB \*newBlock = new TCB;  newBlock->threadID = mythreadID;  newBlock->threadStartTime = myStartTime;  newBlock->threadState = mythreadState;  newBlock->threadWIN = myThreadWIN;  newBlock->next = NULL;  if (empty())  iter = head = tail = newBlock;  else  {  tail->next = newBlock;  tail = newBlock;  }  threadTableSize++;  }  //----------------------------------------  //removeATCB(int) - Remove a specific TCB of the thread table.  void ThreadTable::removeATCB(int threadID)  {  if(head->threadID == threadID)  removeFirstTCB();  else if(tail->threadID == threadID)  removeLastTCB();  else{  TCB \*curr = head;  TCB \*prev;  while (curr->next && curr->threadID != threadID)  {  prev = curr;  curr = curr->next;  }  if (curr->threadID == threadID)  {  prev->next = curr->next;  delete curr;  threadTableSize--;  }  }  }  //----------------------------------------  //removeLastTCB() - Remove last TCB block of thread table.  void ThreadTable::removeLastTCB()  {  if (head->next == NULL)  delete (head);  else  {  TCB \*tmp = head;  while (tmp->next->next != NULL)  tmp = tmp->next;  delete (tmp->next);  tmp->next = NULL;  tail = tmp;  }  threadTableSize--;  }  //----------------------------------------  //removeFirstTCB() - Remove first TCB block of thread table.  void ThreadTable::removeFirstTCB()  {  if(!empty())  {  TCB \*tmp = head->next;  delete(head);  head = tmp;  }  threadTableSize--;  }  //----------------------------------------  //findTCB(int) - Return a thread specific TCB.  bool ThreadTable::findTCB(int mythreadID)  {  TCB \*tmp = head;  while (tmp)  {  if (mythreadID == tmp->threadID)  return true;  tmp = tmp->next;  }  return false;  }  //----------------------------------------  //getTCB() - Return a pointer to TCB searched by threadID.  TCB \*ThreadTable::getTCB(int mythreadID)  {  TCB \*tmp = head;  while (tmp)  {  if (mythreadID == tmp->threadID)  return tmp;  tmp = tmp->next;  }  cout << "getTCB - TCB not found! Return exit(-1)." << endl;  exit(-1);  }  //----------------------------------------  //getTCBProcessID(int) - Return a process ID pthread\_t of a TCB.  pthread\_t \*ThreadTable::getTCBProcessID(int mythreadID)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  return &tmp->processID;  tmp = tmp->next;  }  cout << "getTCBProcessID - ProcessID not found! Return exit(-1)." << endl;  exit(-1);  }  //getTCBHead() - Return a pointer to first TCB in thread table.  TCB \*ThreadTable::getTCBHead()  {  return head;  }  //----------------------------------------  //getTCBTail() - Return a pointer to last TCB in thread table.  TCB \*ThreadTable::getTCBTail()  {  return tail;  }  //----------------------------------------  //getThreadTableSize() - Return thread table size.  int ThreadTable::getThreadTableSize()  {  return threadTableSize;  }  //----------------------------------------  //setTCBState(int,STATE) - Set TCB state of a specified threadID.  void ThreadTable::setTCBState(int mythreadID, STATE myState)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  {  tmp->threadState = myState;  break;  }  tmp = tmp->next;  }  }  //----------------------------------------  //getTCBState(int) - Get a specified TCB state.  STATE ThreadTable::getTCBState(int mythreadID)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  {  return tmp->threadState;  }  tmp = tmp->next;  }  }  //----------------------------------------  //dumpThreadTable() - Output everything from over thread table.  void ThreadTable::dumpThreadTable()  {  cout << "..... START ThreadTable Dump ..... " << endl;  TCB \*tmp = head;  if (!empty())  {  while (tmp != NULL)  {  cout << "TaskNUM: " << tmp->threadID << endl;  cout << "TaskID: " << tmp->threadID << endl;  cout << "TaskState: ";  switch (tmp->threadState)  {  case 0:  cout << "READY" << endl;  break;  case 1:  cout << "RUNNING" << endl;  break;  case 2:  cout << "BLOCKED" << endl;  break;  case 3:  cout << "DEAD" << endl;  break;  }  cout << "..............................." << endl;  tmp = tmp->next;  }  cout << "..... END ThreadTable Dump ..... " << endl;  }  else  {  cout << "EMPTY TCB - RETURN EXIT(-1)" << endl;  exit(-1);  }  }  //----------------------------------------  WINDOW \*ThreadTable::getTCBWIN(int mythreadID)  {  TCB \*tmp = head;  while (tmp != NULL)  {  if (mythreadID == tmp->threadID)  {  return tmp->threadWIN;  }  tmp = tmp->next;  }  cout << "GetTCBWIN - TCB not found! Return exit(-1)." << endl;  exit(-1);  }  //  TCB \* ThreadTable::getTCBIter()  {  if(iter->next!=NULL)  iter = iter->next;  else  iter = getTCBHead();  return iter;  }  #endif |

### 

### Window.h

|  |
| --- |
| /\*  Filename: Semaphore.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To implement a small GUI for  Ultima using curses library.  \*/  #ifndef WINDOW\_H  #define WINDOW\_H  #include <curses.h>  #include <iostream>  #include <curses.h>  #include <stdarg.h>  #include <fcntl.h>  #include "MCB.h"  class MCB;  using namespace std;  class Window  {  private:  MCB \*mcb;  pthread\_mutex\_t mux\_Win;  public:  WINDOW \*Heading\_Win;  WINDOW \*Log\_Win;  WINDOW \*Console\_Win;  WINDOW \*Misc\_Table;  WINDOW \*Tb\_Win;  WINDOW \*Tb\_Win1;  WINDOW \*Sem\_Win;  Window();  ~Window();  WINDOW \*create\_window(int height, int width, int starty, int startx);  void write\_window(WINDOW \*Win, const char \*text);  void write\_window(WINDOW \*Win, int x, int y, const char \*text);  void display\_help(WINDOW \*Win);  void setMCB(MCB \*mcb);  void cont\_help(WINDOW \*Win);  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  // create a windows to display thread data in  // create a new windows: WINDOW \* win = newwin(nlines, ncols, y0, x0);  //Constructor  Window::Window()  {  //Initializing mutex  pthread\_mutex\_init(&mux\_Win, NULL);  // Start nCurses  initscr();  //Enable terminal color  start\_color();  init\_pair(1, COLOR\_BLACK, COLOR\_WHITE);  wbkgd(Misc\_Table, COLOR\_PAIR(1));  //End of terminal color.  refresh();  int Y, X;  int Max\_X, Max\_Y;  getmaxyx(stdscr, Max\_Y, Max\_X); //Get screen size  wprintw(stdscr, "Current Y = %d , Current X = %d\nGiovanna Gorski - C435 Operating Systems", Y, X);  refresh();  //Starting Heading Window  Heading\_Win = create\_window(12, 83, 3, 2);  write\_window(Heading\_Win, 2, 28, "ULTIMA 2.0 (Spring 2020)");  write\_window(Heading\_Win, 9, 2, "Press 'q' or Ctrl-C to exit the program...");  //end of Heading Window  //Starting Log\_Win  Log\_Win = create\_window(15, 60, 25, 2);  write\_window(Log\_Win, 1, 5, ".......Log....\n");    Tb\_Win = create\_window(8, 60, 40, 2);  Tb\_Win1 = create\_window(3, 60, 47, 2);  write\_window(Tb\_Win, 1, 5, ".......Process Table....\n");  Console\_Win = create\_window(15, 23, 25, 62);  write\_window(Console\_Win, 1, 1, "....Console....\n");  write\_window(Console\_Win, 2, 1, "....h for help....\n");  Sem\_Win = create\_window(10, 23, 40, 62);  Misc\_Table = create\_window(47, 110, 3, 85);  write\_window(Misc\_Table, 1, 1, "...Miscellaneous...\n");  cbreak();  noecho();  nodelay(Console\_Win, true);  }  //----------------------------------------  //Destructor  Window::~Window()  {  endwin(); //End the curses window  pthread\_mutex\_destroy(&mux\_Win);  }  //----------------------------------------  //create\_window() - To create a box window.  WINDOW \*Window::create\_window(int height, int width, int starty, int startx)  {  WINDOW \*Win;  Win = newwin(height, width, starty, startx);  scrollok(Win, TRUE); // Allow scrolling of the window  pthread\_mutex\_lock(&mux\_Win);  scroll(Win); // scroll the window  box(Win, 0, 0); // 0, 0 gives default characters  // for the vertical and horizontal lines  wrefresh(Win); // draw the window  wclear(Win);  pthread\_mutex\_unlock(&mux\_Win);  return Win;  }  //----------------------------------------  //write\_window() - To write text in specific window.  void Window::write\_window(WINDOW \*Win, const char \*text)  {  pthread\_mutex\_lock(&mux\_Win);  wprintw(Win, text);  box(Win, 0, 0);  wrefresh(Win); // draw the window  pthread\_mutex\_unlock(&mux\_Win);  }  //----------------------------------------  //write\_window() - To write text in specific window location.  void Window::write\_window(WINDOW \*Win, int x, int y, const char \*text)  {  pthread\_mutex\_lock(&mux\_Win);  mvwprintw(Win, x, y, text);  box(Win, 0, 0);  wrefresh(Win); // draw the window  pthread\_mutex\_unlock(&mux\_Win);  }  //----------------------------------------  //display\_help() - To construct a well formated  // help menu in console window.  void Window::display\_help(WINDOW \*Win)  {  wclear(Win);  write\_window(Win, 1, 1, "...Help...");  write\_window(Win, 2, 1, "1= Kill 1");  write\_window(Win, 3, 1, "2= Kill 2");  write\_window(Win, 4, 1, "3= Kill 3");  write\_window(Win, 5, 1, "n= Next Menu");  write\_window(Win, 6, 1, "c= clear screen");  write\_window(Win, 7, 1, "h= help screen");  write\_window(Win, 8, 1, "q= Quit");  }  //------------------------------------------------------  //cont\_help() - To construct a well formated  // help menu in console window.  void Window::cont\_help(WINDOW \*Win)  {  wclear(Win);  write\_window(Win, 1, 1, "...Next Menu...");  write\_window(Win, 2, 1, "s= Semaphore");  write\_window(Win, 3, 1, "i= IPC");  write\_window(Win, 4, 1, "g= Garbage Collect");  write\_window(Win, 5, 1, "m= Memory Manager");  write\_window(Win, 6, 1, "u= Filesystem");  write\_window(Win, 7, 1, "b= Back to previous");  }  //----------------------------------------  //setMCB(MCB) - Register the Master Control Block  // mechanism so that Semaphore class  // have access to other mechanisms  // such as the IPC, Scheduler...  void Window::setMCB(MCB \*mcb)  {  this->mcb = mcb;  }  //----------------------------------------  #endif |

### 

### Ufs.h

|  |
| --- |
| /\*  Filename: Ufs.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To create a file system mechanism.  Phase IV.  \*/  #ifndef UFS\_H  #define UFS\_H  #include<iostream>  #include<fstream>  #include<string.h>  using namespace std;  #include "LinkedList.h"  #include "Inode.h"  #include "MCB.h"  class MCB;  //Modes for opening a file.  enum OPEN\_MODE  {  READ,  WRITE  };  const int MAX\_FILE\_SIZE = 4 \* 128; //4 blocks of 128 bytes each.  //Data structure to store  //information over openned  //files.  struct openedFile  {  char filename[8];  char permission[4];  OPEN\_MODE mode;  int fileID;  int ownerID;  int opened\_by\_id;  int write\_cursor;  int read\_cursor;  };  class Ufs  {  private:  string fs\_name;  string fs\_metafile\_name;  int fs\_block\_size; //For ultima it should be 128  int fs\_no\_blocks; //For ultima it should be 16  int next\_handle;  char init\_char; //For ultima it should be '^'  int no\_blocks\_available; //At begin should be 16, but as a file is created it should decrement;  int fs\_id;  LinkedList<openedFile> opened\_files;  inode \*inodes;  int next\_file\_id;  MCB \*mcb;  char b[256];  public:  Ufs(string theFilename, int theNoBlocks, int theBlockSize, char theInitChar);  void format();  int Open(int task\_id, int file\_handle, string filename, OPEN\_MODE mode);  int Close(int task\_id, int file\_id);  int Read\_char(int task\_id, int file\_id, char \*ch);  int Write\_char(int task\_id, int file\_id, char ch);  int Create\_file(int task\_id, string filename, int file\_size, string permission);  int Del\_file(int task\_id, string filename);  int Change\_permission(int task\_id,string filename,string new\_permission);  void Dir();  void Dir(int task\_id);  void dump();  void setMCB(MCB \*mcb) { this->mcb = mcb; }  };  //----------------------------------------  //---- M E M B E R F U N C T I O N S ----  //----------------------------------------  //Constructor - The Ufs constructor should locate a datafile and a metadata  // file, if not it will call the format() function so that  // both files are created.  Ufs::Ufs(string theFilename, int theNoBlocks, int theBlockSize, char theInitChar)  {  this->fs\_no\_blocks = theNoBlocks;  this->fs\_block\_size = theBlockSize;  this->init\_char = theInitChar;  this->next\_file\_id = 0;  this->next\_handle = 0;  this->no\_blocks\_available = theNoBlocks;  this->fs\_id = 0;  inodes = new inode[theNoBlocks];  //I have 2 files: one to keep metadata(inode) information  //, and one to keep actual data.  //Filenames: [fs\_name].txt - actual data.  // meta[fs\_name.txt] - metadata information.  this->fs\_name = theFilename + ".txt";    this->fs\_metafile\_name = "meta"+fs\_name;  fstream metadata(this->fs\_metafile\_name);  fstream datafile(this->fs\_name);  //If either of the files does not exist,  //call format to create them.  if(metadata.fail() || datafile.fail())  {  metadata.close();  datafile.close();  format();  }  else  {  //Files exist, read the metadata file into the  //inode[theNoBlocks] array.  datafile.close();  metadata.read((char\*)inodes,(sizeof(inode) \* theNoBlocks));  metadata.close();  //Updating to current amount of inodes available.  for(int i = 0 ; i < theNoBlocks; i++)  if(inodes[i].owner\_task\_id != -1)  no\_blocks\_available--;  }  }  //----------------------------------------  //Format() - The format function is an auxiliary function  // in case of the file system mechanism does not  // find a metadata file and data file.  void Ufs::format()  {  //Create inodes\*fs\_no\_blocks(16 inodes for ULTIMA).  for(int i = 0; i < fs\_no\_blocks; i++)  {  strcpy(inodes[i].filename,"0000000");  inodes[i].owner\_task\_id = -1;  inodes[i].starting\_block = i;  inodes[i].size = 0;  strcpy(inodes[i].permission,"0000");  inodes[i].next\_block = -1;  inodes[i].blocks[0] =  inodes[i].blocks[1] =  inodes[i].blocks[2] =  inodes[i].blocks[3] = -1;  inodes[i].handle = next\_handle;  }  //After creating 16 inodes, store them into a metadata file.  ofstream metadata(fs\_metafile\_name);  metadata.write((char\*)inodes,(sizeof(inode) \* fs\_no\_blocks));  metadata.close();  //Now creating datafile containing init character.  ofstream datafile(fs\_name);  for(int i = 0; i < (fs\_no\_blocks \* fs\_block\_size); i++)  datafile << init\_char;  datafile.close();  }  //----------------------------------------  //Open() - Open a file by checking some constraints such  // ownership, permission, and mode.  // The opening of a file is done by inserting a  // node in a linked list of opened files.  int Ufs::Open(int task\_id, int file\_handle, string filename, OPEN\_MODE mode)  {  mcb->mysema3->down(task\_id);  for(int i = 0; i < fs\_no\_blocks ; i++)  {  //If there is a filename.  if(strcmp(inodes[i].filename,filename.c\_str()) == 0)  {    //Check ownership, and check if file handle is valid.  if(inodes[i].owner\_task\_id == task\_id && inodes[i].handle == file\_handle)  {  //If owner, check if user read permission is set.  if((mode == READ) && (inodes[i].permission[0] == '0'))  {  sprintf(b, " Invalid, owner cannot read!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return -1;  }  else if((mode == WRITE) && (inodes[i].permission[1] == '0')) //Check user write.  {  sprintf(b, " Invalid, owner cannot write!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return -1;  }  }  else  {  //If owner, check if user read permission is set.  if((mode == READ) && (inodes[i].permission[2] == '0'))  {  sprintf(b, " Invalid, others cannot read!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return -1;  }  else if((mode == WRITE) && (inodes[i].permission[3] == '0')) //Check user write.  {  sprintf(b, " Invalid, others cannot write!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return -1;  }  }  //If file is ok, then open it by creating a new node  //a linked list of opened files.  openedFile newOpenedFile;  strcpy(newOpenedFile.filename,filename.c\_str());  strcpy(newOpenedFile.permission, inodes[i].permission);  newOpenedFile.mode = mode;  newOpenedFile.fileID = next\_file\_id;  newOpenedFile.ownerID = inodes[i].owner\_task\_id;  newOpenedFile.opened\_by\_id = task\_id;  newOpenedFile.read\_cursor = 0;  newOpenedFile.write\_cursor = 0;  opened\_files.insertBack(newOpenedFile,next\_file\_id);  next\_file\_id++;  mcb->mysema3->up();  return newOpenedFile.fileID;  }  }  //If no filename found at all, return -1 for error.  sprintf(b, " Invalid filename!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return -1;  }  //----------------------------------------  //Close() - Goes through the linked list of opened files  // check if there are such files or if the task  // in which is trying to close a file is the one  // that opened. If those constraints apply then  // remove from the opened files list and return  // 1, or if not return -1 for error.  int Ufs::Close(int task\_id, int file\_id)  {  mcb->mysema3->down(task\_id);  if(!opened\_files.isNode(file\_id)) //Check if there is such opened file.  {  sprintf(b, " Invalid filename or file not open!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return -1;  }  else  {  Node<openedFile> \*tmp = opened\_files.searchNode(file\_id);  if(tmp->data.opened\_by\_id != task\_id) //Check if the task trying to close if who opened.  {  sprintf(b, " Task does not own file!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return -1;  }  else  {  sprintf(b," File closed!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  opened\_files.removeNode(file\_id);  mcb->mysema3->up();  return 1;  }  }  }  //----------------------------------------  int Ufs::Read\_char(int task\_id, int file\_id, char \*ch)  {  mcb->mysema3->down(task\_id);  char c;  //Check if the file is open  if(!opened\_files.isNode(file\_id))  {  //If close.  sprintf(b, " File not open!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return -1;  }  else  {  //If open.  Node<openedFile> \*tmp = opened\_files.searchNode(file\_id);    //Check if the mode in which the file is were  //opened, is correct.  if(tmp->data.mode != READ)  {  sprintf(b, " File not open for READ!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return -1;  }  //Check if who opened is who wants to read.  if(tmp->data.opened\_by\_id != task\_id)  {  sprintf(b, " File not opened by task id!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return -1;  }  //Now lets try to read.  for(int i = 0; i < fs\_no\_blocks; i++)  {  //Look through the inodes for my file.  if(strcmp(tmp->data.filename, inodes[i].filename) == 0)  {  if(tmp->data.read\_cursor < fs\_block\_size && tmp->data.read\_cursor <= inodes[i].size) //If where I am reading is in the first block.  {  fstream datafile(fs\_name);  datafile.seekp(tmp->data.read\_cursor + (inodes[i].starting\_block \* fs\_block\_size));  datafile.get(c);  datafile.close();  sprintf(b, " Char: %c \n",c);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  tmp->data.read\_cursor = tmp->data.read\_cursor + 1 ;  mcb->mysema3->up();  return 1;  }  else if(tmp->data.read\_cursor > inodes[i].size) //If where I am reading is out of my filesize.  {  //If I am trying to read outside of my filesize boundaries  //I just set the cursor  tmp->data.read\_cursor = 0;  sprintf(b, " Read char outside of boundaries!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return -1;  }else // If where I am reading is in another block.  {  //Find corresponding block.  int no\_blocks\_needed = inodes[i].size % fs\_block\_size;  int block\_needed = inodes[i].blocks[no\_blocks\_needed-1];  fstream datafile(fs\_name);  datafile.seekp(block\_needed \* fs\_block\_size);  datafile.get(c);  datafile.close();  sprintf(b, " Char: %c \n",c);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  tmp->data.read\_cursor = tmp->data.read\_cursor + 1 ;  mcb->mysema3->up();  return 1;  }  }  }  }  mcb->mysema3->up();  return -1;  }  //----------------------------------------  int Ufs::Write\_char(int task\_id, int file\_id, char ch)  {  mcb->mysema3->down(task\_id);  //Check if the file is open  if(!opened\_files.isNode(file\_id))  {  //If close.  sprintf(b, " File not open!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return -1;  }  else  {  //If open.  Node<openedFile> \*tmp = opened\_files.searchNode(file\_id);    //Check if the mode in which the file is were  //opened, is correct.  if(tmp->data.mode != WRITE)  {  sprintf(b, " File not open for WRITE!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return -1;  }  //Check if who opened is who wants to read.  if(tmp->data.opened\_by\_id != task\_id)  {  sprintf(b, " File not opened by task id!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return -1;  }  //Now lets try to read.  for(int i = 0; i < fs\_no\_blocks; i++)  {  //Look through the inodes for my file.  if(strcmp(tmp->data.filename, inodes[i].filename) == 0)  {  if(tmp->data.write\_cursor < fs\_block\_size && tmp->data.write\_cursor <= inodes[i].size) //If where I am reading is in the first block.  {  fstream datafile(fs\_name);  datafile.seekp(tmp->data.write\_cursor + (inodes[i].starting\_block \* fs\_block\_size));  datafile.put(ch);  datafile.close();  tmp->data.write\_cursor = tmp->data.write\_cursor + 1 ;  mcb->mysema3->up();  return 1;  }  else if(tmp->data.write\_cursor > inodes[i].size) //If where I am reading is out of my filesize.  {  //If I am trying to read outside of my filesize boundaries  //I just set the cursor  tmp->data.write\_cursor = 0;  sprintf(b, " Read char outside of boundaries!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return -1;  }else // If where I am reading is in another block.  {  //Find corresponding block.  int no\_blocks\_needed = inodes[i].size % fs\_block\_size;  int block\_needed = inodes[i].blocks[no\_blocks\_needed-1];  fstream datafile(fs\_name);  datafile.seekp(block\_needed \* fs\_block\_size);  datafile.put(ch);  datafile.close();  tmp->data.write\_cursor = tmp->data.write\_cursor + 1 ;  mcb->mysema3->up();  return 1;  }  }  }  }  mcb->mysema3->up();  return -1;  }  //----------------------------------------  //Create\_file() - The function create file, creates a file according  // to the constraints passed as parameter.  // Moreover, it checks if file already exist,  // if file is bigger than space we got, and  // if filename is acceptable size wise.  int Ufs::Create\_file(int task\_id, string filename, int file\_size, string permission)  {  mcb->mysema3->down(task\_id);  //Check if there is such file already.  for(int i = 0 ; i < fs\_no\_blocks; i++)  {  if(task\_id == inodes[i].owner\_task\_id && strcmp(inodes[i].filename,filename.c\_str()) == 0)  {  sprintf(b, " File already exists!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return -1;  }  }    if(file\_size < 0 || file\_size > MAX\_FILE\_SIZE ) //Check if file size is valid.  {  sprintf(b, " Invalid file size!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return -1;  }  else if(strlen(filename.c\_str()) > 8)  {  sprintf(b, " Invalid file name!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return -1;  }  else  { //If file size only requires one data block.  if(file\_size < fs\_block\_size && no\_blocks\_available > 0)  {  for(int i = 0; i < fs\_no\_blocks; i++)  {  if(inodes[i].owner\_task\_id == -1)  {  strcpy(inodes[i].filename,filename.c\_str());  inodes[i].owner\_task\_id = task\_id;  inodes[i].starting\_block = i;  inodes[i].size = file\_size;  strcpy(inodes[i].permission, permission.c\_str());  inodes[i].next\_block = -1;  inodes[i].blocks[0] = i;  inodes[i].handle = next\_handle;    no\_blocks\_available--;  next\_handle++;  //Store to metadafile in right location.  fstream metadata(fs\_metafile\_name);  //Go to correct inode location.  metadata.seekp(sizeof(inode) \* i);  //Write on that inode location the new information.  metadata.write((char \*)&(inodes[i]),sizeof(inode));  metadata.close();  sprintf(b, " File created with success!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return inodes[i].handle;  }  }  }  else  {  //If file size requires more than one data block due to file size > 128.  int no\_blocks\_needed = file\_size / fs\_block\_size;    //If not enough space.  if(no\_blocks\_available < no\_blocks\_needed)  {  sprintf(b, " Not enough enough space!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return -1;  }  else  {  int count = 0;  inode \*previous\_inode\_block;  for(int i = 0; i < fs\_no\_blocks; i++)  {  if(inodes[i].owner\_task\_id == -1)  {  //Store to metadafile in right location.  fstream metadata(fs\_metafile\_name);  strcpy(inodes[i].filename,filename.c\_str());  inodes[i].owner\_task\_id = task\_id;  inodes[i].starting\_block = i;  inodes[i].size = file\_size;  inodes[i].handle = next\_handle;  strcpy(inodes[i].permission, permission.c\_str());  if(count == 0)  {  //Stores the first inode of the file  previous\_inode\_block=&inodes[i];  }  else if(count > 0)  {  //Goes back to previous inode,  //and assign the next\_block value  //and the used blocks array.  previous\_inode\_block->next\_block = i;  previous\_inode\_block->blocks[count] = i;  inodes[i].blocks[count-1] = previous\_inode\_block->blocks[count-1];  //Go to previous inode location to assign new next\_block value;  metadata.seekp(sizeof(inode) \* previous\_inode\_block->starting\_block);  //Write on that inode location the new information.  metadata.write((char \*)&(inodes[previous\_inode\_block->starting\_block]),sizeof(inode));  //Now current inode will be saved  //so that previous inode next\_block  //is set pointing to current inode created.  previous\_inode\_block = &inodes[i];  }    inodes[i].blocks[count] = i; //Sets block being used.  no\_blocks\_available--;    //Go to correct inode location.  metadata.seekp(sizeof(inode) \* i);  //Write on that inode location the new information.  metadata.write((char \*)&(inodes[i]),sizeof(inode));  metadata.close();  //If is the last block needed for  //the file, just set the next\_block  //to -1 meaning there is not a next  //one.  if(count == no\_blocks\_needed)  {  next\_handle++;  inodes[i].next\_block = -1;  sprintf(b, " File created with success!\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  mcb->mysema3->up();  return inodes[i].handle;  }  count++;  }  }  }  }  }  }  //----------------------------------------  int Ufs::Del\_file(int task\_id, string filename)  {  mcb->mysema3->down(task\_id);  int result = -1;  for(int i = 0 ; i < fs\_no\_blocks; i ++)  {  if(inodes[i].owner\_task\_id == task\_id && strcmp(inodes[i].filename,filename.c\_str()) == 0)  {    fstream datafile(fs\_name);  datafile.seekp(inodes[i].starting\_block \* fs\_block\_size);  for(int j = 0; j < fs\_block\_size; j++)  datafile << '$';    datafile.close();  strcpy(inodes[i].filename,"0000000");  inodes[i].owner\_task\_id = -1;  inodes[i].starting\_block = i;  inodes[i].size = 0;  strcpy(inodes[i].permission,"0000");  inodes[i].next\_block = -1;  inodes[i].blocks[0] =  inodes[i].blocks[1] =  inodes[i].blocks[2] =  inodes[i].blocks[3] = -1;  inodes[i].handle = next\_handle;  //Store to metadafile in right location.  fstream metadata(fs\_metafile\_name);  //Go to correct inode location.  metadata.seekp(sizeof(inode) \* i);  //Write on that inode location the new information.  metadata.write((char \*)&(inodes[i]),sizeof(inode));  metadata.close();  result = 1;  }  }  mcb->mysema3->up();  return result;  }  //----------------------------------------  int Ufs::Change\_permission(int task\_id,string filename,string new\_permission)  {  mcb->mysema3->down(task\_id);  int result = -1;  //Update all corresponding inodes  for(int i = 0; i < fs\_no\_blocks; i++)  {  if(inodes[i].owner\_task\_id == task\_id && strcmp(inodes[i].filename,filename.c\_str()) == 0)  {  strcpy(inodes[i].permission, new\_permission.c\_str());  fstream metadata(fs\_metafile\_name);  //Go to correct inode location.  metadata.seekp(sizeof(inode) \* i);  //Write on that inode location the new information.  metadata.write((char \*)&(inodes[i]),sizeof(inode));  metadata.close();  result = 1;  }  }  Node <openedFile> \*tmp = opened\_files.front();  while(tmp != NULL)  {  if(tmp->data.ownerID == task\_id && strcmp(tmp->data.filename, filename.c\_str()) == 0)  {  strcpy(tmp->data.permission, new\_permission.c\_str());  }  tmp = tmp->next;  }  mcb->mysema3->up();  return result;  }  //----------------------------------------  //Dir() - Show all everyone's file name and permissions.  void Ufs::Dir()  {  sprintf(b, " -----------\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  for(int i = 0; i < fs\_no\_blocks; i++)  {  if(inodes[i].owner\_task\_id != -1)  {  sprintf(b, " Filename: %s\n", inodes[i].filename);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, " Owner ID: %d\n", inodes[i].owner\_task\_id);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, " Starting Block: %d\n", inodes[i].starting\_block);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, " File size: %d\n" , inodes[i].size);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, " Permission: ");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  if(inodes[i].permission[0] == '1')  sprintf(b, "r");  else if(inodes[i].permission[0] == '0')  sprintf(b, "-");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  if(inodes[i].permission[1] == '1')  sprintf(b, "w");  else if(inodes[i].permission[1] == '0')  sprintf(b, "-");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, "x");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  if(inodes[i].permission[2] == '1')  sprintf(b, "r");  else if(inodes[i].permission[2] == '0')  sprintf(b, "-");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  if(inodes[i].permission[3] == '1')  sprintf(b, "w");  else if(inodes[i].permission[3] == '0')  sprintf(b, "-");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, "\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, " Next block: %d\n", inodes[i].next\_block);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, " Used blocks: %d | %d | %d | %d \n", inodes[i].blocks[0], inodes[i].blocks[1], inodes[i].blocks[2], inodes[i].blocks[3]);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, " Handle: %d\n",inodes[i].handle);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, " -----------\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  }  }  }  //Dir() - Show file belonging to an specific task id.  // The function should break out of the loop  // when the first is print by calling "break;".  // OVERLOADED.  void Ufs::Dir(int task\_id)  {  mcb->mysema3->down(task\_id);  sprintf(b, " -----------\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  for(int i = 0; i < fs\_no\_blocks; i++)  {  if(inodes[i].owner\_task\_id == task\_id)  {  sprintf(b, " Filename: %s\n", inodes[i].filename);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, " Owner ID: %d\n", inodes[i].owner\_task\_id);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, " Starting Block: %d\n", inodes[i].starting\_block);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, " File size: %d\n" , inodes[i].size);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, " Permission: ");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  if(inodes[i].permission[0] == '1')  sprintf(b, "r");  else if(inodes[i].permission[0] == '0')  sprintf(b, "-");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  if(inodes[i].permission[1] == '1')  sprintf(b, "w");  else if(inodes[i].permission[1] == '0')  sprintf(b, "-");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, "x");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  if(inodes[i].permission[2] == '1')  sprintf(b, "r");  else if(inodes[i].permission[2] == '0')  sprintf(b, "-");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  if(inodes[i].permission[3] == '1')  sprintf(b, "w");  else if(inodes[i].permission[3] == '0')  sprintf(b, "-");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, "\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, " Next block: %d\n", inodes[i].next\_block);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, " Used blocks: %d | %d | %d | %d \n", inodes[i].blocks[0], inodes[i].blocks[1], inodes[i].blocks[2], inodes[i].blocks[3]);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, " Handle: %d\n",inodes[i].handle);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  sprintf(b, " -----------\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,b);  break;  }  }  mcb->mysema3->up();  }  //Dump() - Dump datafile content.  void Ufs::dump()  {  ifstream datafile(fs\_name);  char mydata[(fs\_no\_blocks\*fs\_block\_size)+1];  datafile.read((char\*)mydata,(fs\_no\_blocks \* fs\_block\_size));  strcat(mydata,"\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, mydata);  datafile.close();  }  #endif |

### 

### Inode.h

|  |
| --- |
| /\*  Filename: Inode.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To create a inode structure.  Phase IV  \*/  #ifndef INODE\_H  #define INODE\_H  #include<iostream>  using namespace std;  class inode  {  public:  char filename[8]; //Name of the file created.  int owner\_task\_id; //Task ID which created the file.  int starting\_block; //Block address in which the filename data starts.  int size; //Total filesize.  char permission[4]; //permission[4] = {user\_read, user\_write, others\_read, others\_write}  int blocks[4]; //Max file size = 4 blocks of 128 bytes each.  int handle;  int next\_block; //Indicate the next block related to inode.  };  #endif |

### driver.h

|  |
| --- |
| /\*  Filename: driver.h  Author: Giovanna Gorski  Course: C435 - Operating Systems  Instructor: Dr. Hossein Hakimzadeh  Spring 2020  Purpose: To clean up main program as well as  to implement some driver function for  mechanism testing.  PHASE V  \*/  #include "Scheduler.h"  #include "Semaphore.h"  #include "Window.h"  #include "IPC.h"  #include "MemoryMgmt.h"  #include "Ufs.h"  #include "MCB.h"  //phI\_deadlock () - Phase 1 testing scenario:  // Two worker functions engage in circular wait.  // First task ask for resource1.down(), gets it  // and tries to ask for resource2.down(), but it does  // not get so it becomes state BLOCKED and has to wait  // for it be released. Second task asks for resource2.down()  // , gets it then asks for resource2.down() gets BLOCKED.  // Both blocked tasks should not run.  void phI\_deadlock()  {  int t\_id;  char buff[265];  //---------- DOWN  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to obtain semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema->down(t\_id); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to obtain semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema->down(t\_id); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to obtain semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema0->down(t\_id); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to obtain semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema0->down(t\_id); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  //---------- UP  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to release semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema0->up(); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to release semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema0->up(); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to release semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema0->up(); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to release semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema->up(); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to release semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema->up(); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to release semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema->up(); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  }  //semaphore\_run () - provides a simple Semaphore test.  void semaphore\_run()  {  int t\_id;  char buff[256];  //---------- DOWN  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to obtain semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema->down(t\_id); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to obtain semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema->down(t\_id); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to obtain semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema->down(t\_id); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  //---------- UP  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to release semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema->up(); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to release semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema->up(); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buff, " TaskID:%d is trying to release semaphore.\n", t\_id);  mcb->mywin->write\_window(mcb->mywin->Log\_Win, buff);  mcb->mysema->up(); // hold on to the resource  mcb->mysch->wasteTime(3);  mcb->mysch->yield();  }  //cleanWindows() - clean the Miscellaneous, Semaphore, and Console  // windows of the GUI.  void cleanWindows()  {  refresh(); //clear the console window  wclear(mcb->mywin->Console\_Win);  wclear(mcb->mywin->Misc\_Table);  wclear(mcb->mywin->Sem\_Win);  refresh(); //clear the console window  mcb->mywin->write\_window(mcb->mywin->Console\_Win, 1, 1, "Ultima # ");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, 1, 1, "...Miscellaneous...\n");  mcb->mywin->write\_window(mcb->mywin->Sem\_Win, 1, 1, "\n");  }  void killMenu(int input)  {  char buff[256];  if (input == '0')  mcb->mysch->destroyTask(0);  else if (input == '1')  mcb->mysch->destroyTask(1);  else if(input == '2')  mcb->mysch->destroyTask(2);  sprintf(buff, " Kill = %c\n", input);  mcb->mywin->write\_window(mcb->mywin->Console\_Win, buff);  }  void IPCTest()  {  srand (time(NULL));  int t\_id, error;  char buffer[256];  t\_id = mcb->mysch->getCurrTaskID();  for(int i = 0; i < 5 ; i++)  {  int dest = rand() % 3;  error = mcb->myIPC->sendMessage(t\_id, dest, TEXT, "Testing my main...");  if (error == 0)  {  sprintf(buffer," TaskID #%d - Sending message to TaskID #%d...\n",t\_id, dest);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  }  }  mcb->myIPC->dumpIPC();  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buffer, " TaskID #%d - Message count: %d messages\n",t\_id,mcb->myIPC->messageCount(t\_id));  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, buffer);  t\_id = mcb->mysch->getCurrTaskID();  sprintf(buffer, " TaskID #%d - Receiving message...\n",t\_id);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,buffer);  mcb->myIPC->receiveMessage(t\_id);  t\_id = mcb->mysch->getCurrTaskID();  mcb->myIPC->dumpMailbox(t\_id);  sprintf(buffer, " TaskID #%d - Delete all messages...\n",t\_id);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table,buffer);  mcb->myIPC->removeAMailbox(t\_id);  mcb->myIPC->dumpMailbox(t\_id);  }  void memoryTest()  {  int error;  error = mcb->mymemo->Mem\_alloc(128, 1);  if (error == 0)  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " Block allocated with success!\n");  else  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " Error allocating block!\n");  //Writing test  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " ....Writing test\n");  error = mcb->mymemo->Mem\_write(0, 0, 16, "this is task one", 1);    //Reading test  char \*ch;  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " ....Reading test\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " ....Char read\n");  mcb->mymemo->Mem\_read(0,ch,1);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " ....Multibyte read \n");  mcb->mymemo->Mem\_read(0,0,3,ch,1);  //Free test  //mcb->mymemo->Mem\_free(0,1);  //Memory Dump  mcb->mymemo->Mem\_dump();  }  void fileSysTestI()  {    mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " ...Testing filesystem...\n");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " ...Creating a file...\n");  mcb->myufs->Create\_file(0, "f1", 75, "1100");  mcb->myufs->Create\_file(1, "f2", 130, "1101");  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, " ...Dump Metada file...\n");  mcb->myufs->Dir();  }  void fileSysTestII()  {  char \*ch;  //Uncomment below to dump data file.  //mcb->mywin->write\_window(mcb->mywin->Misc\_Table,"...Dump Data file...\n");  //mcb->myufs->dump();  //Uncomment below to test access permission.  /\* int f\_id;  mcb->mywin->write\_window(mcb->mywin->Misc\_Table," ...Testing Permissions... \n");  //mcb->myufs->Dir(0);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table," Testing USER READ \n");  f\_id = mcb->myufs->Open(0,0,"f1",READ);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table," Testing USER WRITE \n");  f\_id = mcb->myufs->Open(0,0,"f1",WRITE);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table," Testing OTHERS READ \n");  f\_id = mcb->myufs->Open(1,0,"f1",READ);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table," Testing OTHERS WRITE \n");  f\_id = mcb->myufs->Open(1,0,"f1",WRITE); \*/  //Uncomment below to dump data file.  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, "...Dump Data file...\n");  mcb->myufs->dump();  //cout << newFS->Change\_permission(0,"f1","0000") << endl;  char b[256];  sprintf(b, " Open f\_id: %d\n", mcb->myufs->Open(1, 1, "f2", WRITE));  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, b);  sprintf(b, " Open f\_id: %d\n", mcb->myufs->Open(0, 0, "f1", READ));  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, b);  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, "...Write char...\n");  mcb->myufs->Write\_char(1, 0, 'H');  mcb->myufs->Write\_char(1, 0, 'i');  //Uncomment below to dump data file.  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, "...Dump Data file...\n");  mcb->myufs->dump();  mcb->mywin->write\_window(mcb->mywin->Misc\_Table, "...Read char...\n");  mcb->myufs->Read\_char(0, 1, ch);  }  void fileSysTestIII()  {  mcb->myufs->Dir();  } |

## 

## Output

## Demonstration of new Curses Library

## 

## Demonstration of File system

## 

## 

|  |  |
| --- | --- |
| Before | After |

## 

## Demonstration of datafile

## 

## 

## Testing Strategy

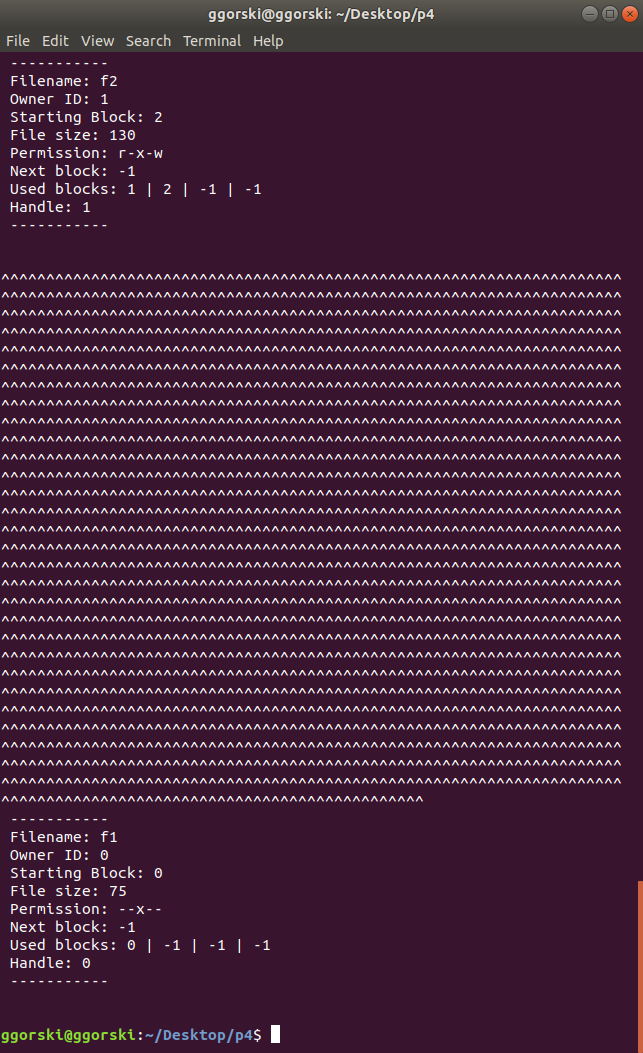
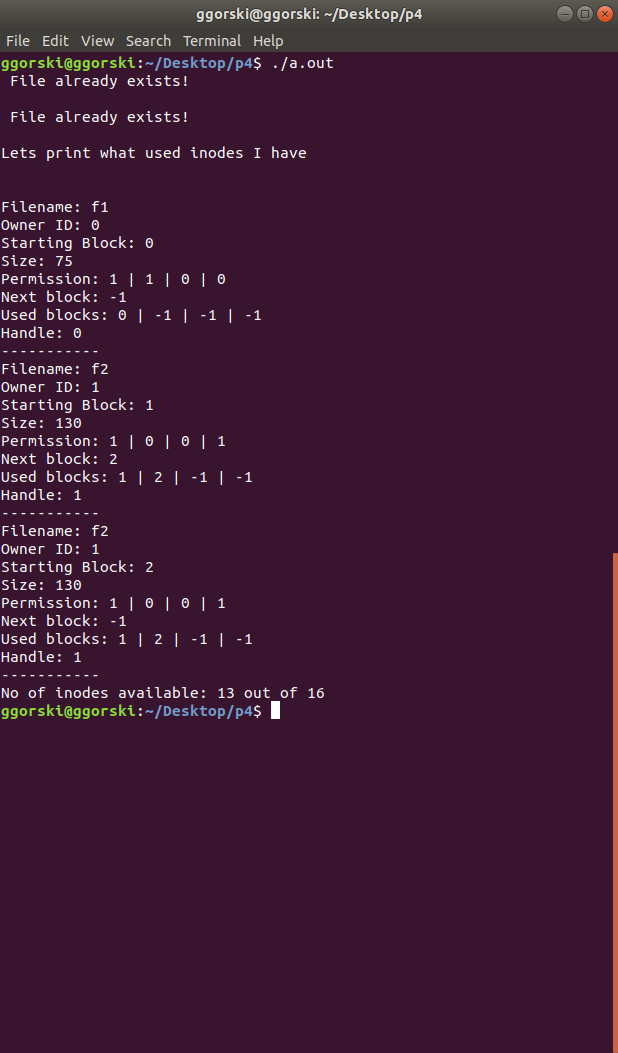
The testing strategy for phase 4 started by its implementation. To guarantee the success of the phase alongside with the correct functionality of each new member function developed, phase 4 was implemented as a separate project from the main ULTIMA OS.

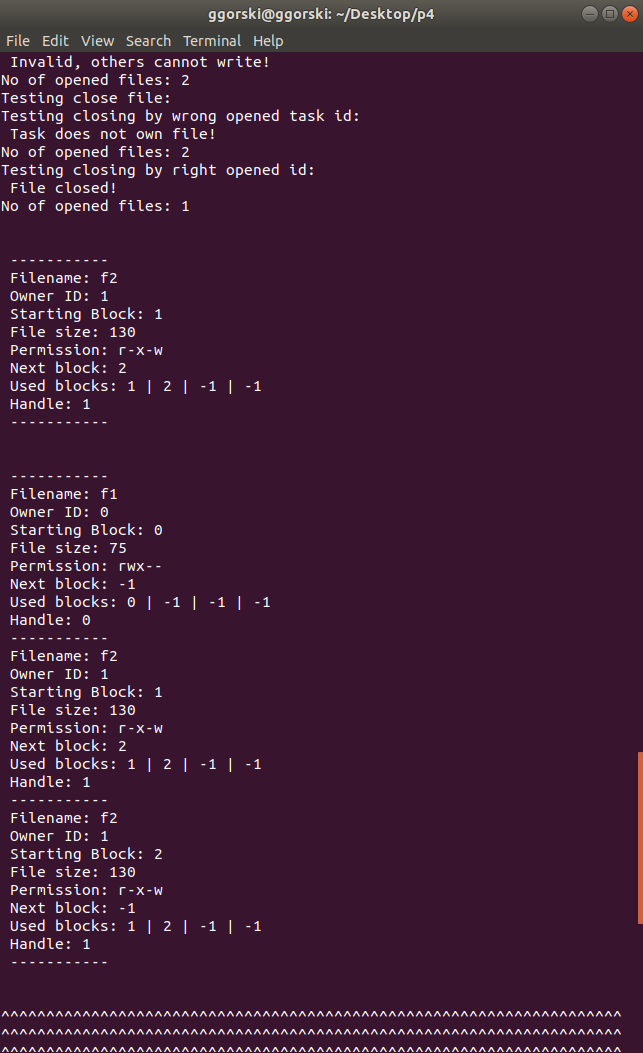
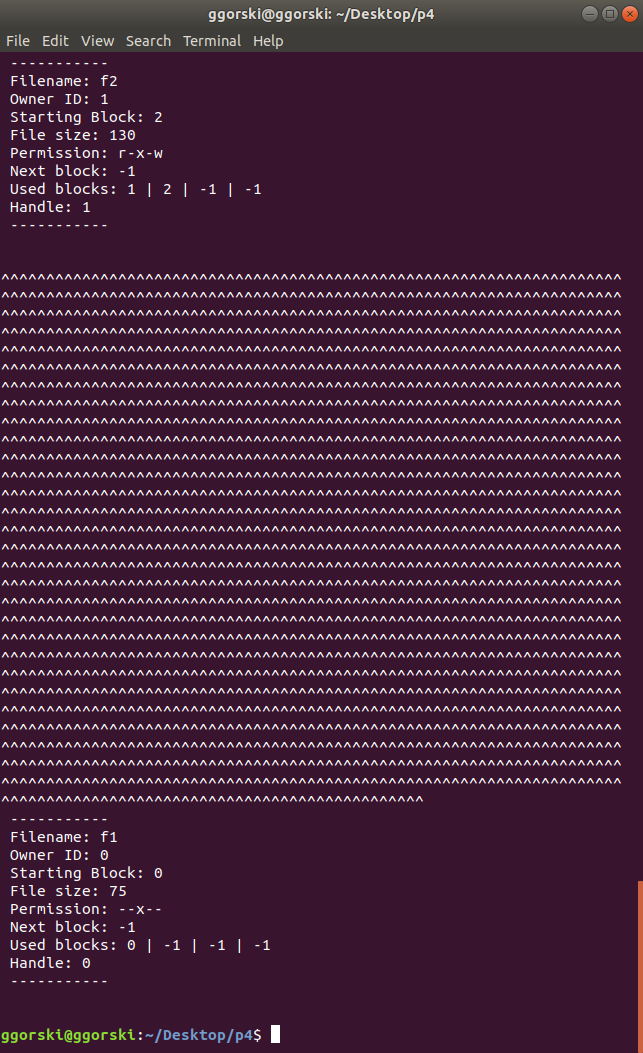
By doing so, different scenarios were tested. Some of the scenarios tested were first of all checking if such metadata file and data file were existent in the view of Ultima OS, if such files were found, the metadata file would be loaded into a data structure so that operations such as read character or write character could be accomplished by running task. Moreover, the testing of the file system was aided by the presence of dump functions which would display in the ULTIMA’s GUI the inodes from the created file, and a dump function which displays sonely the datafile (disk).

At the end of testing, a new phase started in which was the merge of this new mechanism with the Ultima OS. Furthermore, some adjustments were made to the new mechanism such as using the write to window functions available inside the Windows class, and the addition of a semaphore so that critical regions of code inside the new mechanism were protected.

Finally, the final product was successfully merged and some adjustments were made in Ultima's GUI so that the debugging and test phases are more feasible and visible in terms of what is happening.

## Some performed tests PREVIOUS MERGE:





## 

## Testing Ultima

## 

## Discussion

Similar to phase 3, for this phase much time was spent during design and understanding requirements. Furthermore, the implementation of phase 4 was at first by itself outside of ULTIMA OS’s world and then later when its member functions were properly functioning that the merge happened (addition of such a new mechanism to Ultima).

By mainly using the labs over random access files from the course C435 Operating System, it was possible to develop the whole new mechanism. We use important concepts of random access to not only write or read characters from our disk file (datafile), but also to update our meta dafile when a file is ,for example ,created by a task.

When this phase was first to be touched, much time was spent during design than implementation of the Ufs class. The class Ufs contains important member functions needed by a file system. Furthermore, the available physical memory(size of datafile) is 2048 bytes, and when a task requests for physical memory, the task becomes the owner of a location of size 128 bytes similarly to our previous memory management mechanism. All these requests alongside writing and reading from memory are protected by a Semaphore set specific for the file system class, the semaphore guarantees that a task is not writing or reading a memory location being worked on by another task.

Finally, ULTIMA OS in being a very challenging project since it not only test our comprehension over what we learn in the course C435 Operating System, but also it puts into test our programming skills.

## Project Schedule

## 