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

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
  - C++, Python, PHP, HTML, CSS, JavaScript, and MySQL.
  - Object-oriented programming.
  - OpenLayers, and Bootstrap frameworks.

- Operational system
  - 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 3

Table of Contents 6

Phase I 9
- Scheduler and Semaphore 9
- Phase Abstract 10
- Phase Description 11
- Design Documentation 15
- Source code 16
  - Main.cpp 17
  - LinkedList.h 20
  - Node.h 28
  - Queue.h 29
  - Scheduler.h 32
  - Semaphore.h 42
  - ThreadTable.h 46
- Output 55
- Testing Strategy 57
- Discussion 58

Phase II 60
- Message Passing (IPC) 60
- Phase Abstract 61
- Phase Description 62
- Design Documentation 68
- Source code 69
  - Main.cpp 70
  - IPC.h 74
  - Queue.h 79
  - LinkedList.h 84
  - MCB.h 92
  - Node.h 94
  - Scheduler.h 95
  - Semaphore.h 104
  - ThreadTable.h 109
- Output 118
- Demonstration of message sent 119
- Demonstration of IPC being Dump 120
- Testing Strategy 121
- Discussion 122

Phase III 123
Memory Management 123
Phase Abstract 124
Phase Description 125
Design Documentation 129
Source code 130
Main.cpp 131
IPC.h 137
LinkedList.h 144
MCB.h 154
MemoryMgmt.h 156
Node.h 171
Queue.h 172
Scheduler.h 177
Semaphore.h 188
ThreadTable.h 193
Window.h 203
Output 208
Demonstration of Curses Library 209
Demonstration of Memory Management 210
Testing Strategy 212
Discussion 213
Brainstorm Ideas 214
Project Schedule 219

Phase IV 220
File system 220
Phase Abstract 221
Phase Description 222
Design Documentation 229
Source code 230
Main.cpp 231
IPC.h 238
LinkedList.h 246
MCB.h 257
MemoryMgmt.h 259
Node.h 275
Queue.h 276
Scheduler.h 282
Semaphore.h 293
ThreadTable.h 299
Window.h 309

Ufs.h 315
Inode.h 338
Output 339
Demonstration of new Curses Library 340
Demonstration of File system 341
Demonstration of datafile 343
Testing Strategy 344
Some performed tests PREVIOUS MERGE: 345
Discussion 347
Project Schedule 348

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

Spring 2020 H. Hakimzadeh	C435 Operating Systems ULTIMA 2.0 PHASE I – Scheduler and Semaphore	30 points Consult Canvas for Due Date

ULTIMA is a simulated cooperative multitasking operating system. Currently this system lacks many necessary operating system components. Your job for the remainder of this semester is to complete each component and develop a fully functional multi-tasking environment.

PHASE 1: Scheduler and Semaphore

The current system lacks any kind of process management or resource synchronization. This problem is clear when two tasks try to use a shared resource such as the screen. In general, the operating system must provide a mechanism for controlling access to any shared resource, such as printer, memory or disk. In addition, we really don't know which task is actually running, which task has been blocked, etc.

Our job during this phase is to implement the notion of a binary SEMAPHORE. We must be able to support any number of semaphores. This is critical, since future phases of this project introduce new resources which will certainly need synchronization.

In addition to implementing the semaphore, we also need a scheduler with its own process table (more accurately a thread table). The thread table should be implemented dynamically for example a linked list of Task Control Block's (TCB). At this point, the TCB's are needed to preserve the STATE of each Task. For the purpose of this project, we will maintain 4 states: (RUNNING, READY, BLOCKED, DEAD). In the future phases, TCB's will maintain other information such as a pointer to memory allocated by each task, file descriptor, and task priority.

One way to implement the Thread Table is to create a TCB for each task when it is being created and insert the task ID, name, and other relevant information in the TCB. Then insert the TCB in a scheduling ring or linked list.

```
int Create_Task(char *task_name, etc...);     // return the T_id
```

In short, we need at least two classes. One that implements a scheduling ring and the associated algorithms and the other for implementing semaphores. Develop clear and concise test cases to show how and why your system works. Plan these cases as a group. Each class that you implement must contain one or more debugging functions which print the current state of that class. The overall phase will leverage these debugging functions to show that the phase is working.

```
Class scheduler {
        TCB *process_table;
        create_task();          // create appropriate data structures and calls coroutine()
        destroy_task();         // to kill a task (Set its status to DEAD)
        yield();                // strict round robin process switch.
        garbage_collect();      // remove dead task, free their resources, etc.

        dump(int level);        // debugging function with level indicating the verbosity of the dump include some
                                functions which will allow you to dump the contents of the process table in a readable
                                format.  See the expected output section (below) for suggestions.
}
```

```
Class semaphore {
        char resource_name [64];  // the name of the resource being managed
        int sema_value;           // 0 or 1 in the case of a binary semaphore
        queue *sema_queue;
        down();                   // get the resource or get queued!
        up()                      // release the resource

        dump(int level);          // include some functions which will allow you to dump the contents of the semaphore in
                                     a readable format.  See the expected output section (below) for suggestions.
}
```

Suggestions and Hints:

- Do not use a busy wait in Down (), instead block the task. (check the pThread library)
- Create a STATE parameter in the Task Control block (TCB) and have the scheduler check to see if a process is ready, blocked, or dead before scheduling.
- Be aware of deadlocks.

Expected Output:

The output from this phase must clearly show that the scheduler and process table work properly. In addition, you should design a test scenario which clearly shows that the semaphore is performing its job correctly. I suggest having two or three tasks try to acquire a resource (say write to a window) and show that one of them will get the resource and the others promptly get queued and their state changes to BLOCK or READY.

Sample Process Table Dump:

To show that your scheduler and process table operate correctly, develop a public dump() method which simply produces a table similar to the one below:

Task Name	Task ID	State	etc.
Task1	1	Ready	
Task2	2	Running	
Task3	3	Ready	

Sample Semaphore Dump:

To show that your semaphore is working properly, develop a public dump() method which shows the sema-value as well as the content of sema-queue.

```
Semaphore Dump:
Resource:     "Resource Name"        // i.e. printer, memory, etc.
Sema_value:   0                      // 0 or 1
Sema_queue:   T-id-->T-id -->....
```

Note:

- Develop an estimate for how much time this phase requires. Think about all the various components, classes, proof of concepts, external research, group meetings, etc, and associate an amount of time to each. Perhaps a minimum and maximum amount. (that should give a fair average estimate). Then depending on your own assessment of your team's experience, multiply your estimate by a factor (a factor of 2 or 3 is not uncommon). Now you have a little better estimate of the time it will take to do the phase. As you move forward, your ability to estimate will be refined.

Expected Output:

To show that your Schedule and Semaphore is working properly, you must show a number of tasks executing in various states (ready, running, blocked, dead). Along with screen shots showing what is happening to those tasks in their output window.

Spend some time to think about the best approach for testing this phase. Be prepared to demo your phase.

What to hand in:

* Each group should put together two binders (one for you and one for me) with the following information:
    * Cover page with your name, course # and name, project name, date, group member names, e-mails, etc.
    * Project Abstract (You will add this abstract with every new phase)
    * Table of content for the Project.
    * Resume from each group member.
    * For each phase of the project include the following:
        * See below for **"Guideline for each phase of the project"**
        * Self and Peer evaluation form. (Fill them out but do not include it in your binder.) These are anonymous and should be returned to me directly. (via paper or electronically)

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

```
ggorski@ggorski: ~/Desktop/Gorski, ULTIMA-P1
File  Edit  View  Search  Terminal  Help
Thread Num# :0 Pthread_self() #: 140058923788032
---------------------- PROCESS TABLE ----------------------
Quantum = 333
 task id: 0 process id140058923788032 task state: RUNNING <-- CURRENT PROCESS
 task id: 1 process id140058915395328 task state: READY
 task id: 2 process id140058907002624 task state: READY
-------------------------------------------------------------


-------------------------------------------------------------

Current Task # 0 is trying to yield.
Current Quantum : 333
Elapsed time: 349
yielding....(Switching from task #0 to next ready task).
Started Running task # 1
Task 1 is trying to obtain the semaphore (Resource1)
-------Semaphore DUMP-------
Sema_Value: 0
Sema_Name : resource1
Obtained by Task-ID: 0
Sema-Queue: 1 |
----------------------------
-------------------------------------------------------------

Current Task # 1 is trying to yield.
Current Quantum : 333
Elapsed time: 163
---- N O   Y I E L D -----
-------Semaphore DUMP-------
Sema_Value: 0
Sema_Name : resource1
Obtained by Task-ID: 0
Sema-Queue: 1 |
----------------------------
-------------------------------------------------------------

Current Task # 1 is trying to yield.
Current Quantum : 333
Elapsed time: 2865260
yielding....(Switching from task #1 to next ready task).
Started Running task # 2
Thread Num# :2 Pthread_self() #: 140058907002624
```

```
ggorski@ggorski: ~/Desktop/Gorski, ULTIMA-P1
File  Edit  View  Search  Terminal  Help
Current Task # 2 is trying to yield.
Current Quantum : 333
Elapsed time: 999914
yielding....(Switching from task #2 to next ready task).
Started Running task # 0
Thread Num# :0 Pthread_self() #: 140058923788032
---------------------- PROCESS TABLE ----------------------
Quantum = 333
 task id: 0 process id140058923788032 task state: RUNNING <-- CURRENT PROCESS
 task id: 1 process id140058915395328 task state: BLOCKED
 task id: 2 process id140058907002624 task state: BLOCKED
-----------------------------------------------------------


-----------------------------------------------------------

Current Task # 0 is trying to yield.
Current Quantum : 333
Elapsed time: 560
yielding....(Switching from task #0 to next ready task).
Started Running task # 0
-------Semaphore DUMP-------
Sema_Value: 0
Sema_Name : resource1
Obtained by Task-ID: 0
Sema-Queue: 2 | 1 |
----------------------------
Thread Num# :0 Pthread_self() #: 140058923788032
---------------------- PROCESS TABLE ----------------------
Quantum = 333
 task id: 0 process id140058923788032 task state: RUNNING <-- CURRENT PROCESS
 task id: 1 process id140058915395328 task state: BLOCKED
 task id: 2 process id140058907002624 task state: BLOCKED
-----------------------------------------------------------


-----------------------------------------------------------

Current Task # 0 is trying to yield.
Current Quantum : 333
Elapsed time: 1999895
yielding....(Switching from task #0 to next ready task).
Started Running task # 0
Thread Num# :0 Pthread_self() #: 140058923788032
```

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

Spring 2020	C435 Operating Systems ULTIMA 2.0 Phase II – Message Passing (IPC) or U_Pipe (IPC)	H. Hakimzadeh Consult Canvas for Due Date

ULTIMA is a simulated cooperative multitasking operating system. Currently this system lacks many necessary operating system components. Your job for the remainder of this semester is to complete each component and develop a fully functional multi-tasking environment. During the last phase you implemented the Scheduler as well as Semaphore. The scheduler allows you to take control of the state of each task. The semaphore is used to implement mutual exclusion when necessary (e.g., when critical section code needs to be executed). During this phase, your team will design and implement message passing facilities to ULTIMA 2.0. Message Passing, allows tasks to communicate with each other through the use of send() and receive() calls.

It is up to you to decide how you would like to implement the above functions. Below is some specification for this phase.

PHASE 3: **Message Passing** (Inter-Process Communication, IPC)

You are to design and implement an inter-process communication system based on message passing. Your server must provide two public functions which are used for message delivery. Message_Send() and Message_Receive(). *A number of other functions will also be needed for internal debugging, status, and clean-up operations.*

```
class ipc {
        struct Message_Type {
                int Message_Type_Id,
                char Message_Type_Description[64];
                etc.
        }

        struct Message {
                int Source_Task_Id;
                int Destination_Task_Id;
                time_t Message_Arrival_Time;          // research time.h, time_t, and tm
                Message_Type Msg_Type;
                int   Msg_Size;                       // lets make these up to 32 bytes long
                char *Msg_Text;
                etc.
        }
        int ipc();                                    // Constructor, return -1 if error
        int Message_Send(., Message *Message);        // returns -1 if error occurred. Return 1 if
                                                             successful.

        int Message_Receive(int Task_id, Message *Message); // returns 0 if no more messages are
                                                             available, loads the Message structure with
                                                             the first message from the mailbox and
                                                             remove the message from the mailbox. Return
                                                             -1 if an error occurs.

        int Message_Count(int Task_id);               // return the number of messages in Task-id's
                                                             message queue.
        int Message_Count();                          // return the total number of messages in all the
                                                             message queues.

        void Message_Print(int Task_id);              // print the all messages for a given Task-id.
        int Message_DeleteAll(int Task_id);           // delete all the messages for Task_id

        void ipc_Message_Dump();                      // print all the messages in the message queue, but
                                                             do not delete them from the queue. (note
                                                             that this function may be best placed in the
                                                             scheduler!
}
```

```
0= text   (not action required)
1= service (request for a service by the task,
           send a notification or result of the
           service back.)
2= Notification (sent to a task to indicate a
          service was completed.  No action is
           required.)
Etc.
```

Alternatively, you can create a u_pipe (Ultima Pipe) as your Inter-Process Communication mechanism. Your u_pipe must provide a write() method for placing information on the pipe. The pipe can then be read() by another task. Any task should be able to request for the creation of a pipe.

- Once a pipe is created, any task can request to open it for write(). The first task that opens it for write (Source_Task_ID) gets exclusively write access to that pipe.
- Once a pipe is created, any task can request to open it for read(). The first task that opens it for read (Destination_Task_ID) gets exclusively read access to that pipe.

```
class u_pipe {
        int pfd;                                    // pipe file descriptor

        queue<char> pipe_buff(32);                  // queue of characters with up to 32 spaces

                alternatively use an array to represent the buffer, but this may be a little
                more involved, as you would have to keep track of your read and write pointers.
                You also have to make sure the buffer is not overrun, etc.
                char pipe_buff[32];

                int read_pointer;                   // location of the read pointer
                int write_pointer;


        int Source_Task_Id;                 // initialized when the first task opens the pipe for write
        int Destination_Task_Id;            // initialized when the first task opens the pipe for read

        int u_pipe();                       // Constructor, set up the pipe and returns a pfd
                                            // pfd is a unique sequential number generated each
                                            // time a u_pipe is created.  Returns -1 if error
                                            // set the default value for source_task_id and
                                            // destination_task_id to -1 to indicate no one has
                                            // opened the pipe yet.

        open_for_read(int pfd, int Destination_Task_Id);     // Destination task read from the pipe
        open_for_write(int pfd, int Source_Task_Id);         // Source task writes to the pipe


        close_for_write(int pfd);                            // Source_Task_Id is implicit.
        close_for_read(int pfd);                             // Destination_Task_Id is implicit.


        int write(int pfd, char message);   // write a char to pipe_buff, return -1, if buff is full.

        int read(int pfd, char *message);   // read a char from pipe_buff, return -1, if buff is empty.

                NOTE:   Alternatively, you can block the task, instead or returning -1.
                        Only return -1 if the buffer has been closed by the other side
                        and the user attempts to read from the pipe.

                        Also, return -1 if the user attempts to write to a closed pipe


        void u_pipe_Dump(int pfd);    // print all information in the pipe for this task
}
```

NOTE: If you like you can implement both the IPC and the U_PIPE classes.

Suggestions and Hints:

- Think about how the placement of down() and up() could maximize concurrency of your system.

- Consider the concept of MCB (Master Control Block). This may be a convenient method of having access to the scheduler's objects and methods. In addition, you may try to keep your IPC in the MCB. Alternatively, the scheduler object must be registered with the IPC object so that IPC can place messages in the TCB's mailbox.

```
struct MCB {                        //Master Control Block  (it can also be a class)
    scheduler Swapper;
    ipc       Messenger;            //object that provides access to thread ipc objects
    u_pipe    Piper;                //object that maintains a linked list of u_pipe objects
    semaphore Monitor("Monitor", 1);
    semaphore Printer("Printer", 1);
}
```

- You may have to add additional functionality to your scheduler to accommodate IPC's access to the TCB's

- Similar to the previous phase, this phase requires some brainstorming by your team. So, do not wait. Start today. Also, please feel free to contact me for discussion and brainstorming sessions.

Note:

- Develop an estimate for how much time this phase requires. Think about all the various components, classes, proof of concepts, external research, group meetings, etc., and associate an amount of time to each. Perhaps a minimum and maximum amount. (that should give a fair average estimate). Then depending on your own assessment of your team's experience, multiply your estimate by a factor (a factor of 2 or 3 is not uncommon). Now you have a little better estimate of the time it will take to do the phase. As you move forward, your ability to estimate will be refined.

Other Implementation Issues related to IPC class:

- If you are implementing the ipc class, each task must have a mailbox (message queue) associate with it. (this could be a physical mailbox or a virtual mailbox.)
    - A physical mailbox may be a queue of messages accessible through the task's TCB.
    - A virtual mailbox may be a group of messages accessible through the ipc. In other words, the IPC can be implemented as object in ULTIMA that provides mailboxes to all the tasks, and will respond to send(), receive() and other calls.
- Access to the mailboxes should be protected (by protection, I mean semaphore) (not necessarily authorization, authentication. However, if you like you can implement that as well. In other words, requests to receive () can be authenticated to ensure the correct task is asking to read the message)
- When a task wants to send() a message to another:
    1. It must first do a down() on the destination's mailbox semaphore
    2. Send the message
    3. Do an up() on the destination's semaphore

```
NOTE: the down() and up should
eventually be done in the send() and
receive() functions not by the task.
But for now, you can keep it in the task
for clarity and/or debugging.
```

- When a task wants to receive () a message:
    1. The receiver must do a down() on its own mailbox semaphore
    2. Read a message from its message queue /mailbox
    3. then do an UP() on its semaphore.

<u>The Message_Send() function:</u>

1) Allocate a new Message node
2) Copy the message, and other information onto it
3) Place the message in the destination task's mailbox (its message queue).

<u>The Message_Receive() function:</u>

1) Check the count on the message queue. (is this critical section code?)
2) do a down() on the task's mailbox (message_queue) semaphore.
3) De-queue and return the first message from the mail_box (message_queue).
4) do an up() on the task's mailbox.

Other Implementation Issues related to U_PIPE class:

- If you are implementing the u_pipe class, remember that when a task is trying to read or write to the pipe there are a number of items that need to be protected by semaphores (critical section code). These may include the pip_buffer (array or queue representation), open_for_read() function, open_for_write() function, read() and write() functions, etc.

Expected Output:

To show that your **ipc** system is working properly, you must show a number of tasks which send, and receive messages. The mailbox for a specific task, (or all tasks) should be shown in each step of the way to ensure the IPC is functioning properly.

Spend some time to think about the best approach for testing this phase. Note that you want your test to be cumulative. In other words, you still want to occasionally dump your scheduler to show that which tasks are running, ready, blocked, etc., who has access to what semaphores, who is in which semaphore queues, etc.

Possible output of the Messenger (MailBox)

Task Number: 1
Message Count: 3
Mail Box:

Source Task-id	Destination Task-id	Message Content	Message Size	Message Type	Message Arrival Time
2	1	Message from task 2 to task 1.	30	Text	xx
3	1	lpr file1	9	Service	xx
1	3	Got your print service request	30	Notification	xx

Alternatively, if you implement **u_pipe**, you must show one or more task creating pipes, opening them, write to them, read from them, etc.

Possible output of the Piper (Streams)

Number of pipes created: 3

pfd	Writer/ Source Task-ID	Reader Destination Task-ID	Write Status	Read Status	Pipe_Buffer
0	2	1	Open	Closed	Hello, this is a message from task 2 to task 1.
1	1	2	Open	Open	Message from 1 to 2
2	2	3	Closed	Closed	Stream of data going from 2 to 3

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

```
ggorski@ggorski: ~/Desktop/ULTIMA
File  Edit  View  Search  Terminal  Help

Current Task # 2 is trying to yield.
Current Quantum : 333
Elapsed time: 193
---- N O   Y I E L D -----
Thread Num# :2 Pthread_self() #: 140304207734528
---------------------- PROCESS TABLE ----------------------
Quantum = 333
 task id: 0 process id140304224519936 task state: READY
 task id: 1 process id140304216127232 task state: READY
 task id: 2 process id140304207734528 task state: RUNNING <-- CURRENT PROCESS
-----------------------------------------------------------

-----------------------------------------------------------

Current Task # 2 is trying to yield.
Current Quantum : 333
Elapsed time: 362
yielding....(Switching from task #2 to next ready task).
Started Running task # 0


Sending message from Task 0 to Task 1


Task 0 is trying to obtain the semaphore (Resource1)
-------Semaphore DUMP-------
Sema_Value: 0
Sema_Name : resource1
Obtained by Task-ID: 0
Sema-Queue: ---- EMPTY ----
----------------------------
Thread Num# :0 Pthread_self() #: 140304224519936
---------------------- PROCESS TABLE ----------------------
Quantum = 333
 task id: 0 process id140304224519936 task state: RUNNING <-- CURRENT PROCESS
 task id: 1 process id140304216127232 task state: READY
 task id: 2 process id140304207734528 task state: READY
-----------------------------------------------------------

-----------------------------------------------------------

Current Task # 0 is trying to yield.
Current Quantum : 333
Elapsed time: 999202
yielding....(Switching from task #0 to next ready task).
```

Demonstration of IPC being Dump

```
ggorski@ggorski: ~/Desktop/ULTIMA
File  Edit  View  Search  Terminal  Help
------------------------------------------------------------

------------------------------------------------------------

Current Task # 1 is trying to yield.
Current Quantum : 333
Elapsed time: 323
---- N O   Y I E L D -----
Thread Num# :1 Pthread_self() #: 139964739913472
---------------------- PROCESS TABLE ----------------------
Quantum = 333
 task id: 0 process id139964748306176 task state: READY
 task id: 1 process id139964739913472 task state: RUNNING <-- CURRENT PROCESS
 task id: 2 process id139964731520768 task state: READY
------------------------------------------------------------

------------------------------------------------------------

Current Task # 1 is trying to yield.
Current Quantum : 333
Elapsed time: 377
yielding....(Switching from task #1 to next ready task).
Started Running task # 2
---- MAILBOX Thread # 0 -----
Not found!
--------------------------------------------------
--------------------------------------------------
---- MAILBOX Thread # 1 -----


----MESSAGE---
Source ID: 0
Dest ID: 1
Arrival Time: Sun Apr  5 12:15:27 2020
Message Type: TEXT
Content: Testing my main...
----END MESSAGE---


--------------------------------------------------
--------------------------------------------------
---- MAILBOX Thread # 2 -----
Not found!
--------------------------------------------------
--------------------------------------------------
ggorski@ggorski:~/Desktop/ULTIMA$
```

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

Spring 2020	C435 Operating Systems ULTIMA 2.0 Phase III – Memory Management	H. Hakimzadeh Consult Canvas for Due Date

ULTIMA is a simulated cooperative multitasking operating system. Currently this system lacks many necessary operating system components. Your job for the remainder of this semester is to complete each component and develop a fully functional multi-tasking environment. During the last phase two phases you implemented the Scheduler for process management, a Semaphore for managing critical section code and access monitoring and synchronization. During this phase, your team will design and implement memory management for ULTIMA 2.0. Memory management allows tasks to dynamically ask for or free memory.

PHASE 3: Memory Management Unit

Assuming you have 1024 bytes of RAM available in your system, develop a class and series of functions which act as a memory manage. The mem_mgr may be implemented as an independent thread (serving memory to its clients) or it may be implemented as part of the main thread ("the OS"). Your mem_mgr must provide at least two public functions for allocation and deallocation of memory. Mem_Alloc() and Mem_Free(). Several other functions will also be needed for internal debugging, dump, and clean-up operations. Think about implementing Mem_Read() and Mem_Write() if you want to implement relocation and protection using BASE and LIMIT registers.

```
Class mem_mgr {
    Mem_mgr(int size, char default_initial_value);   // allocate 1024 unsigned chars and initialize the entire
                                                     memory with "." dots.
    int Mem_Alloc(int size);                         // returns a unique integer memory_handle or -1 if not enough
                                                     memory is available. Set the Current_Location for this
                                                     memory segment. (beginning of the allocated area)
    int Mem_Free(int memory_handle);                 // place "#" in the memory freed. Return -1 if error occurs
    int Mem_Read(int memory_handle, char *ch);       // read a character from current location in memory and bring
                                                     it back in "ch", return a -1 if at end of bounds. Keep track of
                                                     the "Current Location" or the location of the next char to be
                                                     read.
    int Mem_Write(int memory_handle, char ch);       // write a character to the current_location in memory, return
                                                     a -1 if at end of bounds.

    //Overloaded multi-byte read and write.
    int Mem_Read(int memory_handle, int offset_from_beg, int text_size, char *text )
    int Mem_Write((int memory_handle, int offset_from_beg, int text_size, char *text )

    Your Memory Manager must also include the following private functions for internal debugging, status, and clean-
    up operations.

    int Mem_Left();                                  // return the amount of core memory left in the OS
    int Mem_Largest();                               // return the size of largest available memory segment.
    int Mem_Smallest();                              // return the size of smallest available memory segment.
    int Mem_Coalesce();                              // combine two or more contiguous blocks of free space, and
                                                     place "." Dots in the coalesced memory.
    int Mem_Dump(int starting_from, int num_bytes);  // dump the contents of memory
}
```

Other Implementation Issues:

- When processes ask for and later free up memory, you must create and maintain a corresponding linked list of free and used blocks of memory.
- Access to the memory should be protected. Memory is a shared resource and therefore you should be careful when you try to allocate it. However, once you allocate it, it belongs to you and you can write to your memory concurrently with other

tasks. (since you are not interfering with each other). Of course, if you try to access someone else's memory space (accidentally or maliciously), that should not be allowed, and you should generate a segmentation fault. In other words, if your handle/task-id combination is not valid, or if you try to read beyond the end of your ending_location (LIMIT), you should get an error.

- When a task wants to mem_read() one or more characters we must remember the following:
  - Single read of a character should occur from the "Current_Location" of that memory handle and bring it back in "ch", return a -1 if at end of bounds. Once the character is read, the "Current_Location" must be adjusted.
  - Multi-character read should occur from (starting_location+offset_from_beg) and continue for (text_size). The result should be returned in (text) container.
  - Review the concept of base and limit registers. (relocation and protection)
- When a task wants to mem_write(), similar concerns must be observed.
- In order for the Memory Management to work properly, you must implement a MEMORY SEMAPHORE. Otherwise, corruption of the core memory or the linked list structures that hold information about which task has which memory blocks can occur. (i.e. two task may end up pointing to the same memory space.)
- When freeing the memory consider having the Mem_Free() function Un-Block all other processes waiting for memory, allowing them to once again compete for the available memory. This method is often referred to as "Shark Pond" memory allocation technique. (Watch out for the possibility for race conditions)
- Allocate memory in 128-byte blocks.
- Use first fit memory allocation strategy.
- The Mem_Coalesce() should be called every time the Mem_Free() function is called. However, you should be able to pause the OS, dump your memory before and after mem_coalesce(). This allows you to see a deleted segment represented as "#########", followed by coalesced segment represented as "……"
- If you are using the concept of MCB, then you may want to add your Memory manager to the structure

```
struct MCB {                        // Master Control Block
    scheduler     Swapper;
    ipc           Messenger;
    u_pipe        piper;
    mem_mgr       MMU(1024, '.');
    semaphore     Monitor("Monitor", 1);
    semaphore     Printer("Printer", 1);
    semaphore     Core("Core Memory", 1);

}
```

- You may have to add additional functionality to your scheduler to accommodate the Memory Manager. (for example, the notion of "current location", "memory_handle", etc may have to be preserved in the task's TCB. Brainstorm!
- Like the previous phases, this phase requires some brainstorming by your team. So, do not wait. Start today. Also, please feel free to contact me for discussion and brainstorming sessions.

Note:

- Develop an estimate for how much time this phase requires. Use an approach like the one we discussed in class. Think about all the various components, classes, proof of concepts, external research, group meetings, etc, and associate an amount of time to each. Perhaps a minimum and maximum amount. (that should give a fair average estimate). Then depending on your own assessment of your experience, multiply your estimate by a factor (a factor of 2 or 3 is not uncommon). Now you have a little better estimate of the time it will take to do the phase. As you more forward your ability to estimate will be refined.

Expected Output:

- To show that your memory manager is working properly, you must clearly show that several tasks ask for and receive memory.
- You must also show what will happen if a task asks for but cannot receive memory.
- You will need to show the linked list of nodes which maintain the free and used spaces, as well as the core itself.

Possible output of the Memory Usage (info in the linked list):

Status	Memory Handle	Starting Location	Ending Location	Size/ Bytes	Current Location	Task-ID
Used	1	0	127	128	17	1
Free	2	128	1023	896	NA	MM

MM = Memory Manager

Memory CORE Dump:

```
this is task one.....................................................................................
.....................................................................................................
.....................................................................................................
.....................................................................................................
.....................................................................................................
.....................................................................................................
.....................................................................................................
.......................
```

- Spend some time to think about the best approach for testing this phase. Note that you want your test to be cumulative. In other words, you still want to occasionally dump your scheduler to show that which tasks are running, ready, blocked, etc., who has access to what semaphores, who is in which semaphore queues, who has sent messages to who, and of course who allocated what memory and has read/written what.

What to hand in:

- Maintain a cumulative binder (basically everything from previous phases, with any modification/corrections)
    1. Cover page with your name, course # and name, project name, group members their e-mail, phone, date, etc.
    2. Project Abstract
    3. Table of content for the Project. (it would be good to add TABs at this point, since there are multiple phases)
    4. Phase Description
    5. Design documentation (data structures and their relationships, class diagram, UML, etc.) (note, that the design diagrams for the previous phase may be changed to accommodate the new phase. So, those changes should be documented as new design diagrams. (don't lose them)
    6. Header files
    7. Source files
    8. Output - Your test plan for the phase (Screen shots showing how your OS is running, and specifically how this phase of the project is implemented). Basically, if I am not able to tell from your documentation if your phase is working or not, I may ask that you meet with me to demo the phase)
- Resume from each group member.
- Self and Peer evaluation form. (Fill them out but do not include it in your binder.) These are anonymous and should be returned to me directly.

Ultima Phase II
LinkedList
- *head: Node<T>
- *tail: Node<T>
- listSize:int
- mux: pthread_mutex_t
- buffe[256]: char
+ LinkedList();
+ ~LinkedList();
+ insertFront(T, int): void
+ insertBack(T, int): void
+ removeFront(): void
+ removeBack(): void
+ removeNode(T: void
+ removeNode(int): void
+ front(): Node<T> *
+ back(): Node<T> *
+ size(): int
+ empty(): bool
+ searchNode(T): Node<T> *
+ searchNode(int): Node<T> *
+ isNode(int): bool
+ dumpList(): void
+ dumpList(WINDOW*);
+ write_window(WINDOW*,const char*): void
Semaphore
- resourceName: string
- semaValue: int
- luckyTask: int
- mux: pthread_mutex_t
- semaQueue: Queue<int>
- schedPtr: Scheduler *
- mcb: MCB*
- buffer[256]: char
+ Semaphore(int, string, Scheduler)
+ ~Semaphore()
+ down(int taskID): void
+ up(): void
+ dump(int level): void
Main
+ t_id: int
+ MCB
+ buff[256]: char
+ input: int
+ CPU_Quantum: int
+ error: int
+ ch: char*
startMCB(): void
IPC
- mcb : *MCB
- buffer[256]: char
+ mailboxes: LinkedList<;Queue<Message>>
+ createMailbox(int): int
+ sendMessage(int, int, MSGTYPE,string): int
+ receiveMessage(int): void
+ messageCount(int): int
+ removeAMailbox(int): void
+ deleteAllMailboxes(): void
+ dumpIPC(): void
+ dumpMailbox(int): void
+ setMCB(MCB*): void
<<struct>>Message
+ src_id: int
+ dst_id: int
+ msg_type: MSGTYPE
+ arrived_time: string
+ msg_content: string
MCB
+ mysch : Scheduler
+ mysema: Semaphore
+ myIPC: IPC
+ mywin: Window
+ mysema2: Semaphore
+ mymemo: MemoryManager
+ MCB()
ThreadTable
- head: TCB *
- tail: TCB *
- threadTableSize: int
+ ThreadTable()
+ ~ThreadTable()
+ empty() : bool
+ setTCBStartTime(int, clock_t): void
+ cgetTCBStartTime(int): clock_t
+ insertTCB(int, clock_t, STATE): void
+ removeATCB(int): void
+ removeLastTCB(): void
+ removeFirstTCB(): void
+ findTCB(int mythreadID): bool
+ getTCB(int mythreadID): TCB *
+ getTCBProcessID(int mythreadID): pthread_t *
+ getTCBHead(): TCB *
+ getTCBTail(): TCB *
+ getThreadTableSize(): int
+ setTCBState(int, STATE): void
+ getTCBState(int mythreadID): STATE
+ dumpThreadTable(): void
+ getTCBWIN(int): *WINDOW
Scheduler
- currentThread: int
- currentQuantum: long
- nextAvailableThreadID: int
- threadTable: ThreadTable
- mcb: MCB*
- buffer[256]: char
- mux: pthread_mux_t
+ Scheduler()
+ ~Scheduler()
+ create_task(): int
+ worker(void*): void *
+ start(): void
+ yield(): void
+ setQuantum(long): void
+ getQuantum(): long
+ setState(int, STATE): void
+ getCurrTaskID(): int
+ getTCB(int): TCB *
+ dump(): void
+ wasteTime(int): void
+ destroyTask(int): void
+ garbageCollector(): void
+ setMCB(MCB*): void
<<struct>>Node
-data T
-num: int
-next: Node <T> *
Queue
- queueSize: int
- head: Node <T> *
- mux: pthread_mutex_t
- buffer[256]: char
+ Queue();
+ ~Queue();
+ enqueue(T, int): void
+ dequeue(): void
+ empty(): bool
+ top(): Node<T>*
+ size(): int
+ dumpQueue(): void
+ dumpQueue(WINDOW*): void
+ write_window(WINDOW*,const char*)
<<struct>>TCB
+threadState: enum STATE
+threadID: int
+processID: pthread_t
+threadStartTime: clock_t
+next: TCB *
Window
- mcb: MCB*
- mux_Win: pthread_mutex_t
+ Heading_Win: WINDOW*
+ Log_Win: WINDOW*
+ Console_Win: WINDOW*
+ Misc_Table: WINDOW*
+ Window()
+~Window()
+ create_window(int, int, int, int): WINDOW*
+ write_window(WINDOW*, const char*): void
+ write_window(WINDOW*, int, int, const char): void
+ display_help(WINDOW*): void
+ setMCB(MCB*): void
+ cont_help(WINDOW*): void
MemoryManager
- nextAvailableHandler: int
- freeSpace: int
- MemoManager: LinkedList
- memory: char*
- mcb: MCB*
- buffer[2024]: char*
- Mem_left(): int
- Mem_largest(): int
- Mem_smallest(): int
- Mem_coalesce(): int
- Mem_burp(): int
+ MemoryManager()
+ ~MemoryManager()
+ Mem_dump(): int
+ Mem_alloc(int, int): int
+ Mem_free(int, int): int
+ Mem_read(int, char*, int): int
+ Mem_write(int, char, int): int
+ Mem_read(int, int, int, char*, int): int
+ Mem_write(int, int, int, char*, int)
+ setMCB(MCB*): void
<<struct>>Segment
+ base: int
+ limit: int
+ size: int
+ ownerID: int
+ handler: int
+ status: MEMORY_LABEL
+ write_cursor: int
+ read_cursor: int

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

```
ggorski@ggorski: ~/Desktop/ULTIMA-P3,window
File  Edit  View  Search  Terminal  Help
Current Y = -796720488 , Current X = 32575
Giovanna Gorski - C435 Operating Systems

                        ULTIMA 2.0 (Spring 2020)

 ... STARTING SCHEDULING...
     Creating task # 0
     Creating task # 1
     Creating task # 2

 Press 'q' or Ctrl-C to exit the program...

Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..

Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..

Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..

process id 139912028288768 task state: RUNNING <--CURRENT
---------------------
Current Task # 2 is trying to yield.
Current Quantum : 333
Elapsed time: 954
yielding....(Switching from task #2 to next ready task).
Started Running task # 0
--------------------
Current Task # 0 is trying to yield.
Current Quantum : 333
Elapsed time: 104197
yielding....(Switching from task #0 to next ready task).
Started Running task # 1

...Help...
1= Kill 1
2= Kill 2
3= Kill 3
n= Next Menu
c= clear screen
h= help screen
Ultima #

...Miscellaneous...
```

```
ggorski@ggorski: ~/Desktop/ULTIMA-P3,window
File  Edit  View  Search  Terminal  Help
Current Y = -796720488 , Current X = 32575
Giovanna Gorski - C435 Operating Systems

                        ULTIMA 2.0 (Spring 2020)

 ... STARTING SCHEDULING...
     Creating task # 0
     Creating task # 1
     Creating task # 2

 Press 'q' or Ctrl-C to exit the program...

Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..

Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..

Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..

yielding....(Switching from task #2 to next ready task).
Started Running task # 0
-----PROCESS Table -----
Quantum = 333
process id 139912045074176 task state: RUNNING <--CURRENT
process id 139912036681472 task state: READY
process id 139912028288768 task state: READY
---------------------
Current Task # 0 is trying to yield.
Current Quantum : 333
Elapsed time: 1060
yielding....(Switching from task #0 to next ready task).
Started Running task # 1

...Next Menu...
s= Semaphore
i= IPC
g= Garbage Collect
m= Memory Manager
b= Back to previous

Ultima #

...Miscellaneous...
```

Demonstration of Memory Management

```
ggorski@ggorski: ~/Desktop/ULTIMA-P3,window
File  Edit  View  Search  Terminal  Help
Current Y = -796720488 , Current X = 32575
Giovanna Gorski - C435 Operating Systems

                        ULTIMA 2.0 (Spring 2020)

 ... STARTING SCHEDULING...
     Creating task # 0
     Creating task # 1
     Creating task # 2

 Press 'q' or Ctrl-C to exit the program...

Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..
Task#0 running..

Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..
Task#1 running..

Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..
Task#2 running..

yielding....(Switching from task #2 to next ready task).
Started Running task # 0
-----PROCESS Table -----
Quantum = 333
process id 139912045074176 task state: RUNNING <--CURRENT
process id 139912036681472 task state: READY
process id 139912028288768 task state: READY
---------------------
Current Task # 0 is trying to yield.
Current Quantum : 333
Elapsed time: 874
yielding....(Switching from task #0 to next ready task).
Started Running task # 1

...Next Menu...
s= Semaphore
i= IPC
g= Garbage Collect
m= Memory Manager
b= Back to previous

Ultima #

------Semaphore DUMP-------
Sema_Value: 0
Sema_Name: memoryManager
Obtained by Task-ID: 1
--------------------------

----------------
---- Memory Dump ----
Handler: 0
Owner:  1
Base: 0
Limit: 127
Size: 128
Status:  USED
Read_cursor: 4
Write_cursor: 16
-------------------------
Handler: -1
Owner:  -1
Base: 128
Limit: 1023
Size: 895
Status:  FREE
Read_cursor: 128
Write_cursor: 128
-------------------------
this is task one.....................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................
```

```
------Semaphore DUMP-------
Sema_Value: 0
Sema_Name: memoryManager
Obtained by Task-ID: 1
---------------------------

----------------
---- Memory Dump ----
Handler: 0
Owner:  1
Base: 0
Limit: 127
Size: 128
Status:  USED
Read_cursor: 4
Write_cursor: 16
--------------------------
Handler: -1
Owner:  -1
Base: 128
Limit: 1023
Size: 895
Status:  FREE
Read_cursor: 128
Write_cursor: 128
--------------------------
this is task one.................................................................
.................................................................................
.................................................................................
.................................................................................
.................................................................................
.................................................................................
.................................................................................
.................................................................................
.................................................................................
.................................................................................
.................................................................................
.............................
```

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

<u>Coalesce</u> act of merging two adjacent free blocks of memory. When an application frees the memory, the gap will remain in the memory segment.

<u>Merge</u>

<u>Burp</u>

- move empty holes to the end
- call Coalesce
- merge all in one unique block.

<u>Write</u>

check cursor
write what needs to be written : passed as parameter

<u>Read</u>

check cursor
read from begin to end

<u>free</u> burp() overwrite memory with # to indicate
Coalesce() It was deleted

Manager

linked list

memory

array

Allocation approach : First fit

- percolate linked list
- finds hole free and bigger than memory requested

if found | allocate like normal

else not | throw error

Protection and rellocation issues

Base register – register the beginning of the memory segment

limit register – register the length of the memory segment / also end of the segmen

Approach

linked list manages an array (memory)

test

Mem_Alloc – correct

Mem-read (single) – correct

Mem-write (single) – correct

Mem-read (multi) – correct

Mem-write (multi) – correct

Mem-left – correct

Mem-smallest – correct

Mem-largest – correct

Mem-dump – correct

Mem-free –

Mem-coalesce –

Mem-burp –

<u>OS approach</u> - just like IPC, call allocate, read, write directly.

<u>IPC approach</u> - memory management task receives from other tasks a message, then calls alloc, read, ...
(not efficient)

everytime you get memory, allocate 128 bytes
does not matter if task only need 5 bytes
→ Reason: page sizes and blocks in file systems are always the same size.

- if a process asked for 5 bytes
and try to access beyond that
throw segment error

<u>Constructor</u>

```
firstseg.base = 0
firstseg.limit = 1023
firstseg.size = 1024
firstseg.handler = -1;
```

```
next availablehandler = 0;
freeSpace = 1024
memory = new char[1024];
for( int i=0; i<1024; i++)
    memory[i] = default-init-value; // dots...
```

<u>Allocate</u>
- get linked list tail
- change

<u>tail</u>

```
- always insert at begin of linked list
Node <mem-sgt> * tmp = manager.getTail();
mem-sgt newSeg = new mem-sgt();
newSeg.base = tmp.base;
newSeg.limit = newSeg.base + 128;
:
tmp.base = newSeg.limit + 1;
tmp.size = tail.limit - tail.base;
newSeg.handler = nextAvailableHandler;
nextAvailablehandler ++;
newSeg.status = 'U';
freeSpace = tail.size;
```

Structure and needed components

Memory segment

```
struct Mem_sgt{
    int Base;  <- start of the segment / example 0
    int limit; <- end of the segment / example 128 bytes
    int size;  <- whatever the task need (5 bytes / 10 bytes / ...)
    int ownerID; <- id of the task that owns the segment.
    int handler; <- will be set by the memory manager class
    char status[1]; <- F - free  or U - used
}
```

```
int write_cursor;
int read_cursor;
```

```
Class Memory-Manager
    private:
        int nextAvailableHandler;
        int freeSpace;
        linked list < mem_sgt> manager;
        char *memory;
```

ULTIMA 2.0

C435 - Operating Systems

Spring 2020

Indiana University

Giovanna Gorski

Project Start: Mon, 2/10/2020

Display Week: 9

TASK	ASSIGNED TO	PROGRESS	START	END
Documentation		5%	4/4/20	4/5/20
Phase 3 - Memory Management				
Design		30%	4/1/20	4/9/20
Implementation		15%	4/1/20	4/10/20
Debugging		30%	4/10/20	4/12/20
Testing		20%	4/10/20	4/12/20
Documentation		5%	4/11/20	4/13/20
Phase 4 -				
Design		35%	4/16/20	4/21/20
Implementation		35%	4/16/20	4/25/20
Debugging		10%	4/16/20	4/27/20
Testing		15%	4/16/20	4/27/20
Documentation		5%	4/27/20	4/27/20

Insert new rows ABOVE this one

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

Spring 2020	C435 Operating Systems ULTIMA 2.0 Phase IV – File System	H. Hakimzadeh See Canvas for Due Date

ULTIMA is a simulated cooperative multitasking operating system. Currently this system lacks the ability to provide filesystem facilities to its users. Your job during this phase of the project is to build a new file system known as UFS.

PHASE 4: **Ultima File System** (UFS)

Create a new File System. Assume you have a 2K disk drive available (yes 2048 bytes). Then write a series of functions which allow our tasks to call upon the file system to open, read, write, close or check the directory of the file system. Your file system should accommodate persistent files and directories.

To simplify our design, we will create two separate files. The first file will simply emulate a 2K filesystem consisting of 16 blocks of 128 byte each. This is viewed as a set of data blocks. The second file will be used to maintain i-node structures (directory information). This file should accommodate up to 16 i-node structures. Each i-node is 40 bytes (8+4+4+4+4+16).

```
Class i-node {
        char filename[8];          // filenames can be up to 8 chars long.(including the null byte)
        int owner_task_id;         // owner's task_id (this should be user-id, but for now we can use task-id)
        int starting_block;        // starting block where the file is written.
        int size;                  // file size indicates the actual file size, not the number of blocks used by this
                                      file.
        char permission[4];        // rwrw (owner, others) (feel free to implement this as a bit pattern)
        unsigned int blocks[4];    // array of integers (block #) belonging to this file. (max file size= 4x128 bytes)
                                      you may also try to use a bitmap to represent the used blocks (for example a
                                      16 bit representation showing which blocks are used by which i-node (see
                                      example below):
                                      file-1:  1111 0000 0000 0000
                                      file-2:  0000 1110 0000 0000
                                      file-3:  0000 0000 0001 0000
                                      to figure out which blocks are still available we can OR (|) the bitmaps. All the
                                      zeroes are open blocks:
                                      free blocks:   1111 1110 0001 0000
                                      Alternatively, we can use a mechanism similar to our memory management
                                      (linked list of holes and processes)

        Optional attributes:
        time_t    Creation_time, last_modified_time;
}
```

```
Class ufs {

        String fs_name;
        int fs_block_size;
        int fs_number_of_blocks;
        int next_file_handle                          // ufs_Create_file gets the next handle.
        char initialization_char;

   // Filesystem operations:
        void ufs(fs_name, noBlocks, blockSize, init_char);   // Only used by Superuser (you), to create the file system,
                                                              format the virtual disk. (initialize with 0x5E or "^"), also wipe
                                                              the i-nodes.
        void format();                                // format the current file system

   // File operations:
        int Open(T_id, file_handle, filenm, mode);    // file handle must be valid, and belong to t-id, or file name
                                                         should be valid and permission should be set for others.
                                                         Mode = "R" or "W", returns a unique integer file_id or -1 for
                                                         error.  An error could be no such file name is created in the
                                                         system or such a file is not owned by this task_id or the
                                                         permissions set for this file do not allow it to be open.

        int Close(T_id, file_id);                     // close the file, returns a -1 if error occurs.

        Int Read_Char(T_id, file_handle, char *ch);   // read a char from file, return -1 if EOF. Keep track of the current file
                                                         location.
        int Write_Char(T_id, file_handle, char ch);   // write a char to the file, return -1 if error. Keep track of the current
                                                         file location.

        // optional: feel free to create multi-byte ufs_read() and ufs_write()


   // Directory operations:
        int Create_file(T_id, filenm, file_size, permission);

        Int Del_file(Task_id, file_name);             // check the file's ownership, delete the content of the file
                                                              from data blocks by writing "$" in it. Then delete
                                                              the file from the i-node table.

        int Change_Permission(Task_id, file_name, new_permission);  // rwrw (only the file's owner can change the
                                                                       permissions on the file)
        void Dir();                                   // show the directory, everyone's file name and permissions.
        void Dir(Task-id);                            // Overloaded method. Only show the file belonging to t-id.
        void dump();                                  // dump the ufs i-nodes as well as the ufs data blocks
}
```

Expected Output:

To show that your filesystem is working properly, you must show several tasks which create and then write-to and read-from their files. You must also show what happens if a task asks to create a file and it cannot, due to lack of disk space (data blocks, i-nodes). Also show what happens if we try to open a file that we don't have permission to open. Finally, show what happens if we try to read or write beyond the file size. Show the content of your directory structure

Directory:

File handle	File Name	Blocks used by this file				Size	Starting Block	Status	Permission	Owner ID	Create Time	Modified Time
1	File 1	0				75	0	open_read	rwrw	Tid-0		
2	File 2	1	2			145	1	open_write	rwr-	Tid-0		
3	File 3	3				19	3	Closed	rw--	Tid-2		

File System Dump:

File 1 .....
.....File 2.....
.....
.....File 3 .....
.....
.....
.....

Other Implementation Issues:

- When processes ask to create_file(), *we must first check to see if there is an i-node available, then try to reserve data blocks. If enough contiguous file space is not available or there are no i-nodes available, return a -1. Changes to the i-note and corresponding data blocks should be atomic. Return file_handle if successful.*
- *File permission is 4 bytes (or bits) "rwrw" (similar to unix, the permission may be rw owner and rw for others)*
- *A file must be created, before it can be opened.* Access to the file should be protected. The owner of the file can access the file. Others can read or write the file if the read or write permission for "others" is set.
- Only the owner of the file can delete a file.
- **Blocks used by each i-node entry may be stored as:**
  1) As array of integers representing each block # used by the i-node:
  2) Bitmap of used blocks (assuming 16 possible blocks per i-node)

  file-1: 1000 0000 0000 0000
  file-2: 0110 0000 0000 0000
  file-3: 0001 0000 0000 0000

  To figure out which blocks are still available we can OR (|) the bitmaps together and then complement the results. All the

  All used blocks: 1111 0000 0000 0000
  All available blocks: 0000 1111 1111 1111

  Alternatively,

  We can use a linked list. Each node in the linked list maintains an i-node and all of its properties.

- If you are using the concept of MCB, then you may want to add your file system to the structure

```
struct MCB {                          // Master Control Block
    scheduler     Swapper;
    ipc           Messenger;
    u_pipe        Piper;
    mem_mgr       MMU(1024, '.');
    ufs           filesystem(root, 16, 128,'^');
    semaphore     Monitor("Monitor", 1);
    semaphore     Printer("Printer", 1);
    semaphore     Core("Core Memory", 1);

}
```

- You may have to add additional functionality to your scheduler to accommodate the filesystem. (for example, the notion of "current file location", "file_handle", etc may have to be preserved in the task's TCB. Brainstorm!
- Like the previous phases, this phase requires some brainstorming by your team. So, do not wait. Start today. Also, please feel free to contact me for discussion and brainstorming sessions.
- We may need a structure similar to the memory manager to determine which block have already been used, and which are free.

Note:
- Develop an estimate for how much time this phase requires. Use an approach like the one we discussed in class. Think about all the various components, classes, proof of concepts, external research, group meetings, etc, and associate an amount of time to each. Perhaps a minimum and maximum amount. (that should give a fair average estimate). Then depending on your own assessment of your experience, multiply your estimate by a factor (a factor of 2 or 3 is not uncommon). Now you have a little better estimate of the time it will take to do the phase. As you more forward your ability to estimate will be refined.

What to hand in:

- Maintain a cumulative binder (basically everything from previous phases, with any modification/corrections)
    1. Cover page with your name, course # and name, project name, group members their e-mail, phone, date, etc.
    2. Project Abstract
    3. Table of content for the Project. (it would be good to add TABs at this point, since there are multiple phases)
    4. Phase Description
    5. Design documentation (data structures and their relationships, class diagram, UML, etc.) (note, that the design diagrams for the previous phase may be changed to accommodate the new phase. So, those changes should be documented as new design diagrams. (don't lose them)
    6. Header files
    7. Source files
    8. Output - Your test plan for the phase (Screen shots showing how your OS is running, and specifically how this phase of the project is implemented). Basically, if I am not able to tell from your documentation if your phase is working or not, I may ask that you meet with me to demo the phase)
- Resume from each group member.
- Self and Peer evaluation form. (Fill them out but do not include it in your binder.) These are anonymous and should be returned to me directly.

Ultima Phase IV
LinkedList
- *head: Node<T>
- *tail: Node<T>
- listSize:int
- mux: pthread_mutex_t
- buffe[256]: char
+ LinkedList();
+ ~LinkedList();
+ insertFront(T, int): void
+ insertBack(T, int): void
+ removeFront(): void
+ removeBack(): void
+ removeNode(T: void
+ removeNode(int): void
+ front(): Node<T> *
+ back(): Node<T> *
+ size(): int
+ empty(): bool
+ searchNode(T): Node<T> *
+ searchNode(int): Node<T> *
+ isNode(int): bool
+ dumpList(): void
+ dumpList(WINDOW*);
+ write_window(WINDOW*,const char*): void
Semaphore
- resourceName: string
- semaValue: int
- luckyTask: int
- mux: pthread_mutex_t
- semaQueue: Queue<int>
- schedPtr: Scheduler *
- mcb: MCB*
- buffer[256]: char
+ Semaphore(int, string, Scheduler)
+ ~Semaphore()
+ down(int taskID): void
+ up(): void
+ dump(int level): void
Main
+ t_id: int
+ MCB
+ buff[256]: char
+ input: int
+ CPU_Quantum: int
+ error: int
+ ch: char*
startMCB(): void
IPC
- mcb : *MCB
- buffer[256]: char
+ mailboxes: LinkedList<;Queue<Message>>
+ createMailbox(int): int
+ sendMessage(int, int, MSGTYPE,string): int
+ receiveMessage(int): void
+ messageCount(int): int
+ removeAMailbox(int): void
+ deleteAllMailboxes(): void
+ dumpIPC(): void
+ dumpMailbox(int): void
+ setMCB(MCB*): void
<<struct>>Message
+ src_id: int
+ dst_id: int
+ msg_type: MSGTYPE
+ arrived_time: string
+ msg_content: string
MCB
+ mysch : Scheduler
+ mysema: Semaphore
+ myIPC: IPC
+ mywin: Window
+ mysema2: Semaphore
+ mymemo: MemoryManager
+ MCB()
ThreadTable
- head: TCB *
- tail: TCB *
- threadTableSize: int
+ ThreadTable()
+ ~ThreadTable()
+ empty() : bool
+ setTCBStartTime(int, clock_t): void
+ cgetTCBStartTime(int): clock_t
+ insertTCB(int, clock_t, STATE): void
+ removeATCB(int): void
+ removeLastTCB(): void
+ removeFirstTCB(): void
+ findTCB(int mythreadID): bool
+ getTCB(int mythreadID): TCB *
+ getTCBProcessID(int mythreadID): pthread_t *
+ getTCBHead(): TCB *
+ getTCBTail(): TCB *
+ getThreadTableSize(): int
+ setTCBState(int, STATE): void
+ getTCBState(int mythreadID): STATE
+ dumpThreadTable(): void
+ getTCBWIN(int): *WINDOW
Scheduler
- currentThread: int
- currentQuantum: long
- nextAvailableThreadID: int
- threadTable: ThreadTable
- mcb: MCB*
- buffer[256]: char
- mux: pthread_mux_t
+ Scheduler()
+ ~Scheduler()
+ create_task(): int
+ worker(void*): void *
+ start(): void
+ yield(): void
+ setQuantum(long): void
+ getQuantum(): long
+ setState(int, STATE): void
+ getCurrTaskID(): int
+ getTCB(int): TCB *
+ dump(): void
+ wasteTime(int): void
+ destroyTask(int): void
+ garbageCollector(): void
+ setMCB(MCB*): void
<<struct>>Node
-data T
-num: int
-next: Node <T> *
Queue
- queueSize: int
- head: Node <T> *
- mux: pthread_mutex_t
- buffer[256]: char
+ Queue();
+ ~Queue();
+ enqueue(T, int): void
+ dequeue(): void
+ empty(): bool
+ top(): Node<T>*
+ size(): int
+ dumpQueue(): void
+ dumpQueue(WINDOW*): void
+ write_window(WINDOW*,const char*)
<<struct>>TCB
+threadState: enum STATE
+threadID: int
+processID: pthread_t
+threadStartTime: clock_t
+next: TCB *
Window
- mcb: MCB*
- mux_Win: pthread_mutex_t
+ Heading_Win: WINDOW*
+ Log_Win: WINDOW*
+ Console_Win: WINDOW*
+ Misc_Table: WINDOW*
+ Window()
+~Window()
+ create_window(int, int, int, int): WINDOW*
+ write_window(WINDOW*, const char*): void
+ write_window(WINDOW*, int, int, const char): void
+ display_help(WINDOW*): void
+ setMCB(MCB*): void
+ cont_help(WINDOW*): void
MemoryManager
- nextAvailableHandler: int
- freeSpace: int
- MemoManager: LinkedList
- memory: char*
- mcb: MCB*
- buffer[2024]: char*
- Mem_left(): int
- Mem_largest(): int
- Mem_smallest(): int
- Mem_coalesce(): int
- Mem_burp(): int
+ MemoryManager()
+ ~MemoryManager()
+ Mem_dump(): int
+ Mem_alloc(int, int): int
+ Mem_free(int, int): int
+ Mem_read(int, char*, int): int
+ Mem_write(int, char, int): int
+ Mem_read(int, int, int, char*, int): int
+ Mem_write(int, int, int, char*, int)
+ setMCB(MCB*): void
<<struct>>Segment
+ base: int
+ limit: int
+ size: int
+ ownerID: int
+ handler: int
+ status: MEMORY_LABEL
+ write_cursor: int
+ read_cursor: int
Ufs
- fs_name: string
- fs_metafile_name: string
- fs_block_size:int
- fs_no_blocks: int
- next_handle: int
- init_char: char
- no_blocks_available: int
- fs_id: int
- opened_files:LinkedList<openedFile>
- inodes: inode *
- next_file_id: int
- mcb;MCB *
- b[256]: char
+ Ufs(string, int, int, char)
+ format(): void
+ Open(int, int, string, OPEN_MODE): int
+ Close(int, int): int
+ Read_char(int, int, char*): int
+ Write_char(int, int, char): int
+ Create_file(int, string, int, string): int
+ Del_file(int, string): int
+ Change_permission(int, string, string): int
+ Dir(): void
+ Dir(inttask_id): void
+ dump(): void
+ setMCB(MCB*):void
Inode
+ filename[8]: char
+ owner_task_id: int
+ starting_block: int
+ size: int
+ permission[4]: char
+ blocks[4]: int
+ handle: int
+ next_block: int
<<struct>>openedFile
+ filename[8]: char
+ permission[4]: char
+ mode: OPEN_MODE
+ fileID: int
+ ownerID: int
+ opened_by_id: int
+ write_cursor: int
+ read_cursor: int

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(Text, 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
```

Output

Demonstration of new Curses Library

```
ggorski@ggorski: ~/Desktop/p4
File  Edit  View  Search  Terminal  Help
Current Y = -908471656 , Current X = 32565
Giovanna Gorski - C435 Operating Systems

                        ULTIMA 2.0 (Spring 2020)

 ... STARTING SCHEDULING...
      Creating task # 0
      Creating task # 1
      Creating task # 2

 Press 'q' or Ctrl-C to exit the program...

Task#0 running...
Task#0 running...
Task#0 running...
Task#0 running...
Task#0 running...
Task#0 running...
Task#0 running...
Task#0 running...

Task#1 running...
Task#1 running...
Task#1 running...
Task#1 running...
Task#1 running...
Task#1 running...
Task#1 running...
Task#1 running...

Task#2 running...
Task#2 running...
Task#2 running...
Task#2 running...
Task#2 running...
Task#2 running...
Task#2 running...
Task#2 running...

Elapsed time: 6704
Yielding....(Switching from task #1 to next ready task).
Started Running task # 0
Current Task # 0 is trying to yield.
Current Quantum : 333
Elapsed time: 1129850
Yielding....(Switching from task #0 to next ready task).
Started Running task # 1
Current Task # 1 is trying to yield.
Current Quantum : 333
Elapsed time: 9505
Yielding....(Switching from task #1 to next ready task).
Started Running task # 0

...Help...
1= Kill 1
2= Kill 2
3= Kill 3
n= Next Menu
c= clear screen
h= help screen
Ultima #

...Miscellaneous...

     TASK #        PROCESS ID              TASK STATE
-----------------------------------------------------
       0       |139868983650048        |READY
       1       |139868975257344        |RUNNING <--CURRENT
       2       |139868966864640        |BLOCKED
-----------------------------------------------------

Quantum: 333   | Elapsed Time: 8620    |

 SEMAPHORE DUMP
------------------
Value: 0
Name: resourceTest
Owned by Task#: 1
Queue:  2 |
```

Demonstration of File system

Before	After
Inode.h IPC.h LinkedList.h main.cpp Makefile MCB.h MemoryMgmt.h Node.h Queue.h Scheduler.h Semaphore.h ThreadTable.h Ufs.h Window.h	a.out Inode.h IPC.h LinkedList.h main.cpp Makefile MCB.h MemoryMgmt.h metaroot.txt Node.h Queue.h root.txt Scheduler.h Semaphore.h ThreadTable.h Ufs.h Window.h

Demonstration of datafile

root - Notepad
File Edit Format View Help
^^^

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:

```
ggorski@ggorski: ~/Desktop/p4
File  Edit  View  Search  Terminal  Help
ggorski@ggorski:~/Desktop/p4$ ./a.out
 File already exists!

 File already exists!

Lets print what used inodes I have


Filename: f1
Owner ID: 0
Starting Block: 0
Size: 75
Permission: 1 | 1 | 0 | 0
Next block: -1
Used blocks: 0 | -1 | -1 | -1
Handle: 0
-----------
Filename: f2
Owner ID: 1
Starting Block: 1
Size: 130
Permission: 1 | 0 | 0 | 1
Next block: 2
Used blocks: 1 | 2 | -1 | -1
Handle: 1
-----------
Filename: f2
Owner ID: 1
Starting Block: 2
Size: 130
Permission: 1 | 0 | 0 | 1
Next block: -1
Used blocks: 1 | 2 | -1 | -1
Handle: 1
-----------
No of inodes available: 13 out of 16
ggorski@ggorski:~/Desktop/p4$
```

```
ggorski@ggorski: ~/Desktop/p4
File  Edit  View  Search  Terminal  Help
 -----------
 Filename: f2
 Owner ID: 1
 Starting Block: 2
 File size: 130
 Permission: r-x-w
 Next block: -1
 Used blocks: 1 | 2 | -1 | -1
 Handle: 1
 -----------


^^^
```

```
ggorski@ggorski: ~/Desktop/p4
File  Edit  View  Search  Terminal  Help
 -----------
 Filename: f2
 Owner ID: 1
 Starting Block: 2
 File size: 130
 Permission: r-x-w
 Next block: -1
 Used blocks: 1 | 2 | -1 | -1
 Handle: 1
 -----------


^^^
```

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

ULTIMA 2.0

C435 - Operating Systems

Spring 2020

Indiana University

Giovanna Gorski

Project Start: Mon, 2/10/2020

Display Week: 9

TASK	ASSIGNED TO	PROGRESS	START	END
Documentation		5%	4/4/20	4/5/20
Phase 3 - Memory Management				
Design		30%	4/1/20	4/9/20
Implementation		15%	4/1/20	4/10/20
Debugging		30%	4/10/20	4/12/20
Testing		20%	4/10/20	4/12/20
Documentation		5%	4/11/20	4/13/20
Phase 4 -				
Design		35%	4/16/20	4/21/20
Implementation		35%	4/16/20	4/25/20
Debugging		10%	4/16/20	4/27/20
Testing		15%	4/16/20	4/27/20
Documentation		5%	4/27/20	4/27/20

Insert new rows ABOVE this one