CECS 326 - 03

Operating Systems

Mayra Sanchez (ID: 016654974)

Assignment 1

Due Date: 9/17/2019

Submission Date: 9/17/2019

Program Description

This program simulates a simplified version of how memory management works in computer operating systems. This program utilizes structures, queues, arraylists, functions, random number generation, and dynamic memory allocation. Throughout this program the user will prompt the computer to check if there is sufficient memory for the amount of blocks the random number will occupy, if there are not enough consecutive blocks, then the computer will return an error message. If enough consecutive blocks are found then the memory space will be occupied. The program all does the same for deallocating memory such that a new PID is created everytime a process is initiated. When the user wants to deallocate a PID, then the amount of blocks the PID contains is wiped and the PID is cleared. Once the program is ended, the ready queue is cleared and the MBT table is set to it's default status with only the first 32 blocks being occupied.

```
//
//  main.cpp
//  CECS 326 - 03 - Assignment 1
//
//  Created by Mayra Sanchez (016654974) on 9/3/19.
//  Copyright © 2019 Mayra Sanchez. All rights reserved.
//

#include <iostream>
#include <cstring>
#include <cstdlib>
#include <queue>
#include <deque>
using namespace std;

void createMBT(bool *mbt);
void initiate();
void printMBT();
void printSystemState();
void editPID();
void endProgram();
//create a simple struct with 2 items (1st is an integer that reccords how many free blocks)
//boolean array

struct PCB {
    int pid;
    int size; //size of page table
    int *pointerPT; //page table pointer
    int firstVal;
```

```
    int PageTable[]; //arraylist of page table


    PCB(int pid, int size, int *pointerPT, int PageTable[], int firstVal) {//argument constructor
        this->pid = pid;
        this->size = size;
        this->pointerPT = pointerPT;
        this->firstVal = firstVal;
    }
    PCB() {}

    ~PCB() {//destructor
        //cout << "Destructor Called" << endl;
       //delete []pointerPT;
        //delete PageTable;
    }
};  //end of struct

const int arraySize = 1024; //creates array of size 1024
static bool mbt[arraySize]; //array (MBT) created with name 'list'
deque<PCB> Queue; //ready queue
int pid = 1; //starting point for PID
int firstVal;
bool running = true;


int main() {

    createMBT(mbt);
```

```
while (running) {
    cout << "1) Initiate a Process" << endl
    << "2) Print System State" << endl
    << "3) Terminate process with a specific PID" << endl
    << "4) Exit" << endl;

    int userin;
    cin >> userin;

    switch(userin) {
        case 1:
            cout << "Option 1 Selected" << endl;
            initiate();
            break;
        case 2:
            cout << "Option 2 Selected" << endl;
            printSystemState();
            printMBT();
            break;
        case 3:
            cout << "Option 3 Selected" << endl;
            editPID();
            break;
        case 4:
            cout << "Option 4 Selected" << endl;
            endProgram();
            break;
        default:
```

```
            cout << "Error. Invalid Option. Please try again"<<endl;
            break;
        }
    }
    return 0;
}

void createMBT(bool *mbt) {
    for(int i = 0; i < 32; i++) { //makes first 32 spaces not available
        mbt[i] = false;
    }
    for(int j = 32; j < arraySize; j++) { //rest of the variables are available
        mbt[j] = true;
    }
    cout << "MBT has been created." << endl;
    printMBT();
}

void initiate() {
    //generates random number from 10 to 250
    srand(time(0));
    int r = rand() % (241) + 10;

    //counts TOTAL amount of free spaces (doesn't matter if not consecutive
    int count = 0; //counts to see how many free spaces there are
    for (int i = 0; i < arraySize; i++) {
        if(mbt[i] == 1) {//counts each there is an empty space available
            count++;
        }
```

```
    }
    cout<< "There are " << count << " blocks available." << endl; //counts free blocks
    cout << "Your random number is: " << r << endl;

    //creating page table
    int PageTable[r]; //creates page table the with the same size as the random variable

    //checking consecutive amount of free spaces
    int consecutive1 = 0;
    int placeHolder = -1;
    bool changeMade = false;
    for (int i = 0; i < arraySize; i++) {
      if(mbt[i] == 0) {//if space is occupied, then consecutive 1 resets and the placeholder is
dismissed
        consecutive1 = 0;
        placeHolder = -1;
      }
      else {
        if(placeHolder == -1) {//place holder will not update if it is not -1
          firstVal = i+1;
          placeHolder = i;
        }
        consecutive1++;
        if(r == consecutive1) {
          int temp = 0; //temp variable to keep random number count
          cout << "You have sufficient memory" << endl;
          for(int j = placeHolder; j <= arraySize; j++){
            //occupies the memory space
            if (mbt[j] == 1 && temp < r) {
```

```
            PageTable[temp] = j; //allocates memory to page table (new)
            mbt[j] = 0; //occupies the blocks
            temp++;
        }
    }

    //determines size of PCB table
    int sizePT = 0;
    for (int i : PageTable) {
        sizePT++;
    }

    PCB newPCB; //creates a new PCB
    int* pt = PageTable; //pointer allocating memory to the page table
    newPCB = {pid, sizePT, pt, PageTable, firstVal}; //add firstVal
    Queue.push_back(newPCB); //adds to the queue
    cout << "Stores in PID # " << pid << endl;
    cout <<  "Starting from: " << firstVal << endl;
    pid++;//keeps track of the index of the PID in the PCB

    //shows new updated array
    cout << "MBT memory has been updated." << endl;
    changeMade = true;
    printMBT();
    cout << endl;
    break;//break statement here so it doesn't occupy the memory more than once
  }
 }
}
```

```
    if(changeMade == false) {//amount of consecutive 1s is not enough
        cout << "Error. You do not have sufficient memory." << endl << endl;
    }
}

//prints out MBT
void printMBT() {
    int count = 0;
    cout << "Printing MBT..." << endl;
    cout  << "LEGEND: 1 = Available      0 = Occupied" <<endl;
    for (int k = 0; k < arraySize; k++) {
        cout <<mbt[k];
        count++;
        if (count == 128) {
            cout << endl;
            count = 0;
        }
    }
}
//prints out PIDs and their data
void printSystemState() { //needs to access Page Table and print the content
    for (int i = 0; i < Queue.size(); i++) {
        cout << "PID: " << Queue[i].pid << endl;
        cout << "Size: " << Queue[i].size << endl;
        cout << "Block Numbers: " <<endl;
        //printBlocks(Queue[i].size);
        //prints index of the blocks the PID has
        int count = 0;
        for (int j = Queue[i].firstVal; j < Queue[i].size + Queue[i].firstVal; j++) {
```

```
            if( j == 33 || j < 100) {
                cout << " ";
            }
            cout << j << "|";
            count++;

            if(count == 23) {
                cout << endl;
                count = 0;
            }
        }
        cout << endl;
    }
}


//deletes PID and the blocks it occupied are set to available once again
void editPID() {
    for (int k = 0; k < Queue.size(); k++) {
        cout << "PID: " << Queue[k].pid << endl;
    }
    cout << "Enter the PID you would like to edit:  " << endl;
    int userin;
    cin >> userin;
    bool checkPID = false;

    for(int i = 0; i <= Queue.size(); i++) {
        if(userin == Queue[i].pid) {
            checkPID = true;
```

```
            cout <<"Deallocating the data in PID #"<< userin <<endl;
            for(int j = Queue[i].firstVal; j < Queue[i].size + Queue[i].firstVal; j++) {//change
starting position to firstVal //add value of firstval to the size
                mbt[j] = true; //blocks are now available
            }
            Queue.erase(Queue.begin()+i);//pid selected is deleted
            //delete pointerPT;
        }
    }
    if(Queue.size() == 0) {
        for(int i = 0; i < 32; i++) { //makes first 32 spaces not available
            mbt[i] = false;
        }
        for(int j = 32; j < arraySize; j++) { //rest of the variables are available
            mbt[j] = true;
        }   }
    if (checkPID == false) {
        cout << "That PID does not exist." << endl;
    }
    printMBT();

}


void endProgram() {
    if(Queue.size() != 0) {
        cout <<"Printing PID in ready queue..." << endl;
        //prints all PIDS in ready queue
        for(int i = 0; i < Queue.size(); i++) {
```

```
            cout << "PID: " << Queue[i].pid << endl;
        }
        cout <<"Are you sure you want to exit? Enter 0 to exit. Enter 1 to continue." << endl;
        int confirmQ;
        cin >> confirmQ;
        if (confirmQ == 0) {
            Queue.clear(); //clears ready queue

            for(int i = 32; i < arraySize; i++) {
                mbt[i] = true; //clears mbt memory to be set to default
            }
            cout <<"MBT has been set to default" << endl;
            cout << "Program will now terminate.\nThank you for participating!" << endl;
            //delete mbt;
            //delete PCB;
            running = false;
        }
        else if (confirmQ == 1) {
            cout << "Program will continue to run." << endl;
        }
        else {
            cout << "That was not a valid option. Program will continue to run." << endl;
        }
    }
    else {
        cout << "Ready Queue is empty. Program will quit affirmatively." <<endl;
        running = false;
    }
}
```