CPSC 457 S17 - Assignment 4

Due date: **Sunday, June 25, 2017 at 11:59pm**.
Individual assignment. No group work allowed.
Weight: 8% of the final grade.

Link to the assignment web page:

https://sites.google.com/site/pfederlcpsc457spring2017/assignments/assignment-4

Q1 – Written question (5 marks)

Provide a simple example of a system in an unsafe state where the processes could finish without entering a deadlock. Show the state, and two sequences of execution: one leading to a deadlock and the other leading to the completion of all processes.

Hint: you should not need more than 3 processes and one resource type (with multiple instances).

Q2 – Written question (5 marks)

In a system consisting of four processes and five resource types, the current allocation and maximum needs are as follows:

Process	Allocation	Maximum	Available
P0	1 0 2 1 1	1 1 2 1 3	0 0 x 1 2
P1	2 0 1 1 0	2 2 2 1 0	
P2	1 1 0 1 0	2 1 3 1 0	
P3	1 1 1 1 0	1 1 2 2 1	

What is the smallest value of 'x' that can keep the system in a safe state? Once you find the 'x' value, find a safe execution order using the Safety Algorithm. Include a step-by-step walk-through of the Safety Algorithm.

Q3 – Written question (5 marks)

Assume an OS has five free memory partitions of 100KB, 500KB, 200KB, 300KB and 600KB:

free 100KB	P10 30KB	free 500KB	P11 30KB	free 200KB	P12 30KB	free 300KB	P13 30KB	free 600KB

The OS needs to place 4 new processes in memory in the following order: P1 of 212KB, P2 of 417KB, P3 of 112KB and P4 of 426KB. Draw the diagrams of the partitions after the OS has placed the processes using 4 different algorithms: first-fit, best-fit, worst-fit and next fit. The resulting diagrams must show the size of each partition, and the status of each partition. If a process cannot be placed, indicate that below the diagram. Please start the placement algorithms from the first partition.

Q4 – Written question (5 marks)

Consider a system with 1KB (1024 bytes) page size. What are the page numbers and offsets for the following addresses?

Address	Page number	Offset
2375		
19366		
30000		
256		
16385		

Q5 – Written question (5 marks)

Consider a system with a 32-bit logical address space and 4KB page size. The system supports up to 512MB of physical memory. How many entries are there in each of the following?

a) A conventional single-level page table.
b) An inverted page table.

Show your calculations.

Q6 – Written question (5 marks)

Consider a system where a direct memory reference takes 200ns.

a) If we add a single-level page table stored in memory to this system, how much time would it take to locate and reference a page in memory?
b) If we also add a TLB, and 75% of all page-table references are found in the TLB, what is the effective access time ? Assume that searching TLB takes 10ns.

Show your calculations.

Q7 – Written question (5 marks)

Consider the following page reference string:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

Assume there are 3 frames in the physical memory and all frames are initially empty. Illustrate how pages are placed into the frames according to the LRU and the optimal replacement algorithms. How many page faults would occur for each algorithm? Show your work.

Q8 - Programming question (20 marks)

For this question you will write a program that implements the Banker's Algorithm. Your program will determine whether there is a safe execution sequence for a given set of processes. The process information will be provided in a configuration file, which is passed in as the command-line argument. The configuration file will contain the number of processes (`numProc`), the number of resource types (`numResourceTypes`), currently available resources

(`available`), current resource allocation for all processes, the maximum resources required by each process, and a request (`request`) by process 'i'. For instance, the example we discussed in Lecture 9 on slide #40, can be translated into the following configuration file (`config1.txt`):

```
numProc = 5
numResourceTypes = 3
available = <3 3 2>
P0 <0 1 0> <7 5 3>
P1 <2 0 0> <3 2 2>
P2 <3 0 2> <9 0 2>
P3 <2 1 1> <2 2 2>
P4 <0 0 2> <4 3 3>
request 1 = <1 0 2>
```

The last line represents a request by process #1 for resources (1,0,2). If you run your program on the above configuration file, the output should look as follows:

```
$ ./banker config1.txt
Grant request <1 0 2> from P1.
Sequence: P1, P3, P0, P2, P4.
```

If the request can be granted, your program needs to output a possible execution sequence, as above. If the request cannot be satisfied, the output of your program should look like this:

```
$ ./banker config2.txt
Reject request <1 0 2> from P1.
Reason: request would result in unsafe state.
```

If the request cannot be granted, the program should output the reason for rejection. Possible reasons for rejecting a request include:

- request would result in an unsafe state,
- request is invalid (exceeding declared max for process),
- not enough resources available.

You are free to use the `banker.cpp` code in the Appendix 1 as a starting point. If you do, all you have to do is implement the `Banker::isSafe()` method.

Q9 - Programming question (20 marks)

Write a simulator (`pagesim.c`) of three page replacement algorithms: optimal, LRU and clock. Your simulator will read in a reference string from standard input, and then run a simulation using all three algorithms. At the end of the simulation your program will output the following statistics for each algorithm:

1. the contents of the frames; and
2. the number of page faults.

For the clock algorithm you can use the single refererence bit implementation. At the beginning of the simulation all frames are empty. The number of frames in physical nemory will be specified on the command line.

Example input file `test1.txt`:

```
1 2 3 4 1 2 5 1 2 3 4 5
```

Sample output:

```
$ ./pagesim 4 < test1.txt
Optimal:
  - frames: 4 2 3 5
  - page faults: 6
LRU:
  - frames: 5 2 4 3
  - page faults: 8
Clock:
  - frames: 4 5 2 3
  - page faults: 10
```

You can make the following assumptions:

- Number of frames will be between 1 and 20 (inclusive).
- Number of entries in the reference string will be at most 5000.
- Frame numbers will be non-negative integers smaller than 100.

Submission

You should submit 4 files for this assignment:

- Answers to the written questions combined into a single file, called either `report.txt` or `report.pdf`. Do not use any other file formats.
- Your solutions to Q8 and Q9 in files called `banker.c/cpp, pagesim.c/cpp.`

Since D2L will be configured to accept only a single file, you will need to submit an archive, eg. `assignment4.tgz`. To create such an archive, you could use a command similar to this:

```
$ tar czvf assignment4.tgz report.pdf banker.cpp pagesim.cpp
```

General information about all assignments:

- Due time: All assignments are due at 23:59 on the due date listed on the assignment. Late assignments or components of assignments will not be accepted for marking without approval for an extension beforehand. What you have submitted in D2L as of the due date is what will be marked.
- Extensions may be granted for reasonable cases, but only by the course instructor, and only with the receipt of the appropriate documentation (e.g. a doctor's note). Typical examples of reasonable cases for an extension include: illness or a death in the family. Cases where extensions will not be granted include situations that are typical of student life, such as having multiple due dates, work commitments, etc. Forgetting to hand in your assignment on time is not a valid reason for getting an extension.
- After you submit your work to D2L, make sure that you check the content of your submission. It's your responsibility to do this, so make sure that you submit your

assignment with enough time before it is due so that you can double-check your upload, and possibly re-upload the assignment.

- All assignments should include contact information, including full name, student ID and tutorial section, at the very top of each file submitted.
- Assignments must reflect individual work. Group work is not allowed in this class nor can you copy the work of others. For further information on plagiarism, cheating and other academic misconduct, check the information at this link: http://www.ucalgary.ca/pubs/calendar/current/k-5.html.
- You can and should submit many times before the due date. D2L will simply overwrite previous submissions with newer ones. It's better to submit incomplete work for a chance of getting partial marks, than not to submit anything.
- Only one file can be submitted per assignment. If you need to submit multiple files, you can put them into a single container. The container types supported will be ZIP and TAR. No other formats will be accepted.
- Assignments will be marked by your TAs. If you have questions about assignment marking, contact your TA first. If you still have questions after you have talked to your TA then you can contact your instructor.

Appendix 1 – banker.cpp for Q9

```
/*
 * banker.cpp
 *
 * Student Name:
 * Student Number:
 *
 * Class: CPSC 457 Spring 2017
 * Instructor: Pavol Federl
 *
 * Copyright 2017 University of Calgary. All rights reserved.
 */

#include <iostream>
#include <fstream>
#include <sstream>
#include <stdlib.h>
#include <algorithm>

using namespace std;

class Banker
{
private:
    int numProc;      // the number of processes
    int numResources; // the number of resources
    int * available;  // number of available instances of each resource
    int ** max;       // the max demand of each process, e.g., max[i][j] = k
                      // means process i needs at most k instances
                      // of resource j
    int ** allocation;// number of resource instances already allocated
```

```
    int ** need;      // number of resource instances needed by each process

public:

    /* Initializing the vectors and matrixes for the Banker's Algorithm.
     * Takes ownership of all arrays.
     *
     * @param avail  The available vector
     * @param m      The max demand matrix
     * @param alloc  The allocation matrix
     * @param p      The number of processes
     * @param r      The number of resources
     */
    Banker (int * avail, int ** m, int ** alloc, int p, int r) {
        numProc = p;
        numResources = r;
        // Setup the available vector, the max matrix, and the
        // allocation matrix
        available = avail;
        max = m;
        allocation = alloc;
        // Initialize the need matrix
        need = new int*[numProc];
        for (int i = 0; i < numProc; i++)
            need[i] = new int[numResources];
    }

    /* Destroy the vectors and matrixes
     */
    ~Banker() {
        numProc = 0;
        numResources = 0;

        // Free all allocated memory space
        delete[] available;
        for (int i = 0; i < numProc; i++)
        {
            delete[] need[i];
            delete[] max[i];
            delete[] allocation[i];
        }
        delete[] need;
        delete[] max;
        delete[] allocation;
    }

    /* Check whether it is safe to grant the request
     * @param pid    The process that is making the request
     * @param req    The request
     * @param sequenceOrReason  The safe execution sequence returned
     *               by the algorithm or reason for rejection
     * @return Whether granting the request will lead to a safe state.
     */
    bool isSafe (int pid, int * req, string & sequenceOrReason) {
        sequenceOrReason = "Not implemented yet.";
        return false;
    }
```

```
};

int main (int argc, char * const argv[]) {
    ifstream config;       // Configuration file
    string conffile;       // The configuration file name
    int numProc;           // The number of processes
    int numResources;      // The number of resources
    string sequenceOrReason; // The execution sequence returned by
                             // the Banker's Algorithm
    int i, j, index;       // Indices for the vectors and matrixes
    int pid;               // The ID of the process that is making the
                           // request
    string reqStr;         // The request vector in string format

    // Read in the config file name from the commanda-line arguments
    if (argc < 2) {
        cout << "Usage: banker <config file>\n";
        return 0;
    }
    else {
        conffile = argv[1];
    }

    // Open the file
    config.open(conffile.c_str());

    // Get the number of process and the number of resources
    string line, var, equal; // strings for parsing lines in cfg. file
    getline(config, line);
    istringstream iss(line);
    iss >> var >> equal >> numProc;     // Get the number of processes
    iss.clear();

    getline(config, line);
    iss.str(line);
    iss >> var >> equal >> numResources;    // Get the number of resources
    iss.clear();

    // Create the available vector, the max matrix, and the allocation
    // matrix according to the number of processes and the number of
    // resources
    int * available = new int[numResources];
    int ** max = new int*[numProc];
    int ** allocation = new int*[numProc];
    for (int i = 0; i < numProc; i++)
    {
        max[i] = new int[numResources];
        allocation[i] = new int[numResources];
    }

    // Get the available vector
    getline(config, line);
    replace(line.begin(), line.end(), '<', ' ');  // Remove "<" and ">"
    replace(line.begin(), line.end(), '>', ' ');
    iss.str(line);
    iss >> var >> equal;
    for (j = 0; j < numResources; j++) // Read in the "available" vector
```

```
        iss >> available[j];
    iss.clear();

    // Get the max matrix and the allocation matrix
    for (i = 0; i < numProc; i++)
    {
        getline(config, line);
        replace(line.begin(), line.end(), '<', ' ');
        replace(line.begin(), line.end(), '>', ' ');
        iss.str(line);
        iss >> var;
        index = atoi(&var.at(1));            // Get the process ID
        if (index < 0 || index >= numProc)
        {
            cerr << "Invalid process ID: " << var << endl;
            return 0;
        }
        // Get the number of resources allocated to process "index".
        for (j = 0; j < numResources; j++)
            iss >> allocation[index][j];

        // Get the max allocation to process "index".
        for (j = 0; j < numResources; j++)
            iss >> max[index][j];
        iss.clear();
    }

    // Get the request vector
    int * request = new int[numResources];
    getline(config, line);
    reqStr = line.substr(line.find('<'),
                         line.find('>') - line.find('<') + 1);
    replace(line.begin(), line.end(), '<', ' ');
    replace(line.begin(), line.end(), '>', ' ');
    iss.str(line);
    iss >> var >> pid >> equal;
    for (j = 0; j < numResources; j++)  // Read in the "request" vector
        iss >> request[j];
    iss.clear();

    // Check the request using the Banker's algorithm.
    Banker * banker = new Banker(
             available, max, allocation, numProc, numResources);
    if (banker -> isSafe(pid, request, sequenceOrReason))
        cout << "Grant request " << reqStr << " from P" << pid << ".\n"
             << "Sequence: " << sequenceOrReason << ".\n";
    else
        cout << "Reject request " << reqStr << " from P" << pid << ".\n"
             << "Reason: " << sequenceOrReason << "\n";
}
```