System Software – Assignment Report

Rafi Rahman – u3191010

Task 1

A. The thread has not completed its execution after line 1 as it continues in line 3 after the other thread is executed in line 2.

B. There could be several reasons for this:

I. tid2 was interrupted and put in queue to let the other processes running on the computer finish, while tid1 was allowed to execute.
II. tid2's multiplication operation did not complete as fast as tid1's (or rather, wasn't given as high of a priority as tid1), and it was slowed down as a result, allowing tid1 complete first.
III. A combination of I and II.

C. The computer may have freed up its resources, and because of that, tid2 was allowed to execute alongside tid1 without being put on hold.

D. If we are assuming the process has a certain time limit to run, it is likely that the effective execution output of each of the two threads will be higher. On the other hand, as an example, tid2 might find itself battling tid1 for a timesharing slot (whether this occurs is dependent on the OS and hardware), so tid2's effective execution output might be reduced, though even in that situation, it is still likely to be higher than tid2's effective execution output would be if the computer had other processes running.

E. It should be possible, though very unlikely, to produce exactly the same output with multiple execution instances of the program, as it is stated that the execution path of the threads is complete unpredictable. Because of that, it is also possible to have O's printed before X's, as the CPU may decide to put tid1 on hold after its creation due to resources not being available, do the same for tid2, but then allow tid2 to run before tid1 when resources are available for just one thread.

Task 2

A. The reason the counter value is smaller than the number of visits likely has to do with the threads that are created by the program not being mutually excluded from one another. Because the variable cnt is shared by all these threads, if these threads are running at the same time and making changes to cnt at the same time, the value of cnt after the program executes will not come out being the value that is expected. The threads are executed a certain way by the CPU, but that is dependent on how many resources the CPU has available, and as a result, it is possible for the value of cnt to be 50 or 70, rather than a number around 60.

B. See appendix.

Task 3

A. There are two types of memory addresses in operating systems: logical and physical. Logical memory addresses are generated by the CPU while a process runs. They are virtual addresses, as they do not exist physically. Physical memory addresses identify a physical location of required data in a memory. A logical address can be used as a reference by the CPU to access its corresponding physical address and the data within that address. The process image of an application occupies the logical level of the memory. Paging is a memory management technique in which process address space is broken into blocks of the same size by the OS to manage how a computer's memory resources are shared. These blocks are called pages. Paging allows the OS to address more memory than what is physically available. The main memory is divided into frames. When a process is running, pages are loaded into available non-contiguous frames. When a frame is allocated to a page, the system links the logical address to a physical address on the main memory and the link is stored in a data structure called page table. If there is not enough memory available, the system will move any idle pages into the virtual memory, created using secondary storage (e.g. HDD), to free up the main memory for the current process to use, and when the pages are required, it is brought back from the virtual memory.

B. In a paging memory management system, pages are of fixed size and a process may not acquire the entire block size. As a result, internal fragmentation can occur at both the physical memory level and the process image level.

C. A normal application running on the user level on a computer with a paging memory management system cannot directly access the physical memory. This is because in order to directly access the physical memory, a process must be running on the kernel level, or it can indirectly access the physical memory using the MMU.

D. Defragmentation applications are able to locate the contiguous data stored in memory and organise the data into smaller contiguous frames, allowing the OS to access the data faster. However, the application has to use the MMU to access the physical memory, as applications cannot directly access the physical memory.

Task 4

A. The program stopped when it used 3851.6 MB. It was running on a computer with $2^{33} – 1$ bytes of physical address space and $2^{64} – 1$ bytes of logical address space, and 8 GB of RAM and a 64-bit OS.

B. Both virtual and physical memory is exhausted when there is a memory leak problem with a process.

C. When a process starts consuming more and more memory, other processes will be pushed to secondary storage and they will not be able to run until they are brought back into the main memory. This will also slow down the performance of the whole system.

D. Memory leak occurs because dynamically allocated memory becomes unreachable. A garbage collection mechanism attempts to reclaim memory that is no longer in use by the program and can therefore find and reclaim unreachable memory. As a result, the memory leak problem can be avoided on a platform with a built-in garbage collection mechanism.

Appendix

Code for task 2 – B:

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

// Repeat 100 times to mimic 100 random visits to the page
#define RPT 100

// Web page visit counter
int cnt = 0;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void* counter() {
    int cntLocalCopy;
    float r;

    pthread_mutex_lock(&mutex);
    cntLocalCopy = cnt;
    // Mimicking the work of the sever in serving the page to
    // the browser
    r = rand()%2000;
    usleep(r);
    cnt = cntLocalCopy + 1;
    pthread_mutex_unlock(&mutex);
}

int main () {
    int i;
    float r;
    pthread_t tid[RPT];

    // Seed the random number sequence
    srand(time(NULL));

    for (i=0; i < RPT; i++) {
        // Mimicking the random access to the web page
        r = rand()%2000;
        usleep(r);

        // A thread to respond to a connection request
        pthread_create(&tid[i], NULL, &counter, NULL);
    }

    for (i = 0; i < RPT; i++) {
        // Wait till current thread completes before creating more threads
        pthread_join(tid[i], NULL);
    }

    // Print out the counter value and the number of mimicked visits
    // the 2 values should be the same if the program is written
    // properly
    printf("cnt = %d, repeat = %d\n", cnt, RPT);
}
```

Code and output for task 4:

```
#include <iostream>

int main() {
    while (true) {
        int *x = new int[10];

        std::cout << "x";
    }

    return 0;
}
```

The output is a series of x’s. It will eventually stop running.