Assignment 4

Part I - Answer the following questions (20 points each)

Question 1:
Suppose three processes A, B, C arrive in the system at the same time in that order, each has a series of estimated alternating CPU bursts and I/O bursts as follows:

A: 11ms (CPU), 2ms (I/O), 6ms (CPU)
B: 3ms (CPU), 8ms (I/O), 7ms (CPU)
C: 2ms (CPU), 1ms(I/O), 4ms (CPU), 1ms (I/O), 3ms (CPU)

Calculate the **turnaround (completion) and wait times** of all processes and the average of these times using each of the following scheduling algorithms:

Round Robin (RR) with 5ms time quantum,
First Come First Serve (FCFS).

You must show the steps of your calculation.

NOTE the WAIT time is the wait time each process spends in the process READY queue**.**
And for simplicity, assume context switch time and other overheads are comparatively negligible.

Please give your calculated numbers of times in the following format:

	Turnaround Time			Wait Time	
	RR	FCFS		RR	FCFS
A					
B					
C					
Average					

Use notation process(e, s, r) (e.g., A(5, 5, t)) to track CPU execution of a process over the time:

e - the total CPU time that has been executed for a particular process,
s - the total system time passed from the beginning of executing the first process,

r - the reason why the process is stopped along the execution (either suspended or completed)

- t - time quantum expired during RR,
- i - process initiated an I/O,
- c - process completed execution

Question 2:

Consider a 32-bit system with a virtual memory space of 1 GB. Each word / memory address has 4 bytes. Set each page size to 4KB, and part of the page table entries is given as below.

	Page Table	
Virtual Page Number	Valid	Physical Frame Number
0x0	1	0xE
0x1	0	0x2
0x2	1	0x20B
0x3	1	0xA2
0x4	1	0x6
0x5	0	0x30
0x6	1	0x725

Based on the above information, use binary arithmetic to translate the following virtual addresses to their corresponding physical addresses in 32-bit hexadecimal format.

1) Virtual Address: 0x00000A96
2) Virtual Address: 0x00006813
3) Virtual Address: 0x00004715

You must show the steps of your address translation.

Question 3

A hard real-time system has been developed to for a fly-by-wire aviation system. It has sensors for the following pilot interface systems:

System	Sampling Frequency (Hz)	CPU time required (ms)
Yoke	20	5
Rudder pedals	15	2
Throttle	10	1

Overhead of the hard real-time system on the architecture selected for the plane is .21. Can these tasks be scheduled? Show your work. Regardless of whether or not the tasks are schedulable, what should a designer do if they are not.

Question 4

On a specific architecture, memory access time in 10 ns. Suppose a TLB access requires 1 ns and has a miss rate of 0.01. If the MMU is capable of accessing a 2 level page table with negligible overhead other than the memory access times, what is the effective access time?

Part II: 120 points

You will construct a simulation of an N-level page tree. Your program will read a file consisting of memory accesses for a single process, construct the tree, and assign frame indices to each page. Once all of the addresses have been specified, the pages which are in use will be printed in order. You will assume an infinite number of frames are available and need not worry about a page replacement algorithm. See section functionality for details.

Specification

A 32 bit logical address space is assumed. The user will indicate how many bits are to be used for each of the page table levels, and a user-specified file containing hexadecimal addresses will be used to construct a page table.

Mandatory interfaces and structures:

- unsigned int LogicalToPage(unsigned int LogicalAddress, unsigned int Mask, unsigned int Shift) - Given a logical address, apply the given bit mask and shift right by the given number of bits. Returns the page number. This function can be used to access the page number of any level by supplying the appropriate parameters. Example: Suppose the level two pages occupied bits 22 through 27, and we wish to extract the second level page number of address 0x3c654321. LogicalToPage(0x3c654321, 0x0FC00000, 22) should return 0x31 (decimal 49). Remember, this is computed by taking the bitwise and of 0x3c654321 and 0x0FC00000, which is 0x0C400000. We then shift right by 22 bits. The last five hexadecimal zeros take up 20 bits, and the bits higher than this are 1100 0110 (C6). We shift by two more bits to have the 22 bits, leaving us with 11 0001, or 0x31.

- PAGETABLE – Top level descriptor describing attributes of the N level page table and containing a pointer to the level 0 page structure.

- LEVEL – An entry for an arbitrary level, this is the structure (or class) which represents one of the sublevels.

- MAP – A structure containing information about the mapping of a page to a frame, used in leaf nodes of the tree.

You are given considerable latitude as to how you choose to design these data structures. A sample data structure and advice on how it might be used is given on the Blackboard assignment page

- MAP * PageLookup(PAGETABLE *PageTable, unsigned int LogicalAddress) - Given a page table and a logical address, return the appropriate entry of the page table. You must have an appropriate return value for when the page is not found (e.g. NULL if this is the first time the

page has been seen). Note that if you use a different data structure than the one proposed, this may return a different type, but the function name and idea should be the same. Similarly, If PageLookup was a method of the C++ class PAGETABLE, the function signature could change in an expected way: MAP * PAGETABLE::PageLookup(unsigned int LogicalAddress). This advice should be applied to other page table functions as appropriate.

- void PageInsert(PAGETABLE *PageTable, unsigned int LogicalAddress, unsigned int Frame) - Used to add new entries to the page table when we have discovered that a page has not yet been allocated (PageLookup returns NULL). Frame is the frame index which corresponds to the page number of the logical address. Use a frame number of 0 the first time this is called, and increment the frame index by 1 each time a new page→frame map is needed. If you wish, you may replace void with int or bool and return an error code if unable to allocate memory for the page table. HINT: If you are inserting a page, you do not always add nodes at every level. The Map structure may already exist at some or all of the levels.

- All other interfaces may be developed as you see fit.

Your assignment must be split into multiple files (you may group by functionality) and you must have a Makefile which compiles the program when the user types "make". Typing make in your directory MUST generate an executable file called "**pagetable**".

The traces were collected from a Pentium II running Windows 2000 and are courtesy of the Brigham Young University Trace Distribution Center. The files byutr.h and byu_tracereader.c implement a small program to read and print trace files. Note that the cache was enabled during this run, so CPU cache hits will not be seen in the trace. Cannibalize these functions to read the trace files in your program. The file trace.sample.tr is a sample of the trace of an executing process. All files can be found on edoras in directory ~mroch/lib/cs570/trace.

User Interface

When invoked, your simulator should accept the following optional arguments and have two or more arguments.

Optional arguments:

-n N	Process only the first N memory references
-p filename	Print valid entries of the page table to the specified file. The format of the pagefile is **very important** as it will be compared to the solution automatically. See the functionality section below for details.
-t	Show the logical to physical address translation for each memory reference.

The first mandatory argument is the name of the address file to parse. The remaining arguments are the number of bits to be used for each level.

Sample invocations:

pagetable trace_file 8 12 – Construct a 2 level page table with 8 bits for level 0, and 12 bits for level 1. Process the entire file.

pagetable –n 3000000 –p page.txt trace_file 8 7 4 – Construct a 3 level page table with 8 bits for level 0, 7 bits for level 1, and 4 bits for level 2. Process only the first 3 million memory references, and write the mappings of valid pages to file page.txt.

See next page for a hint on parsing command line arguments.

User interface hint: Parsing optional arguments can be made easier with the use of standard libraries. One such library that works with both C and C++ is getopt. Here's an example of using it:

```
#include <unistd.h>

int main(int argc, char **argv) {

  // other stuff (e.g. declarations)

  /*
   * This example uses the standard C library getopt (man -s3 getopt)
   * It relies on several external variables that are defined when
   * the getopt.h routine is included.  On POSIX2 compliant machines
   * <getopt.h> is included in <unistd.h> and only the unix standard
   * header need be included.
   */
  while ( (Option = getopt(argc, argv, "n:p:t")) != -1) {
    switch (Option) {
    case 'n': /* Number of addresses to process */
      // optarg will contain the string following -n
      // Process appropriately (e.g. convert to integer atoi(optarg))
      break;
    case 'p':   /* produce map of pages */
      // optarg contains name of page file…
      break;
    case 't': /* Show address translation */
      // No argument this time, just set a flag
      break;
    default:
      // print something about the usage and die…
      exit(BADFLAG);  // BADFLAG is an error # defined in a header
    }
 }

 /* first mandatory argument, optind is defined by getopt */
 idx = optind;
```

Functionality

Upon start, you should create an empty page table (only the level 0 node should be allocated). The program should read addresses one at a time from the input trace. For each address, the page table should be searched. If the page is in the table, increment a hit counter (note that this is hit/miss for the page table, not for a translation lookaside buffer which we are not simulating).