Assignment 3

Part I: 20 points each

1. On edoras, create directory cs570/a03_threads. Create a small program, thread_demo, that does the following:
   a. From thread_demo.c's main function, create a loop that spawns 5 threads with integer arguments 0 through 4.
   b. The thread's entry point, in file worker.c, is function void worker(void *) where void * is expected to be a pointer to an int. The thread computes and prints the square of the argument, then returns.
   c. The main thread should wait until all threads have completed, then print "work complete."

   Be sure to create a makefile. You are expected to use the POSIX thread interface for this assignment. The FAQ has an example of creating POSIX threads and waiting for a thread to terminate can be done with pthread_join (see the man page or read the materials on threads on Blackboard). As always, turn in your code and output.
2. A periodic real time schedule has an operating system overhead of .05. Process oxygen monitor has a period of 2 Hz and requires 200 ms to operate. Process carbon monoxide monitor has a period of 1 Hz and requires 10 ms to operate. These processes are schedulable.
   a. Show that they are schedulable.
   b. Create a plot showing Hz vs required time indicating the limits of third process that could be scheduled.
3. A round robin scheduler has a time quantum of 10 ms. If a process has a CPU burst of 35 ms and is not preempted other than by a quantum expiration, how many times will it be scheduled before it completes?
4. Priority schedulers are prone to starvation, what modification can be made to the basic algorithm to prevent starvation.
5. Why do we typically not use admission schedulers on personal computing devices?

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, and 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. When we 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. Frame is the frame index which corresponds to the logical address. 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 automatically add nodes, they 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 rohan 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.

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).

When a page is not in the page table, assign a frame number (start at 0 and continue sequentially) to the page and create a new entry in the page table. Increment a miss counter.

Once all addresses have been handled, print the total number of addresses processed, the number of hits, misses and their corresponding percentages. Print the page size and the total number of bytes used by the tree (the C/C++ sizeof operator will come in handy for this).

When the address translation is specified, each logical address and the corresponding physical address should be output with one address pair per line. Write the numbers as 8 digit zero padded hexadecimal numbers separated by " `->` ". An example from the gcc_integ.tr file with 20 bits of page address :

```
60f74100 -> 00000100
0041f780 -> 00001780
0041f740 -> 00001740
11f5e2c0 -> 000022c0
05e78900 -> 00003900
```

To output the page table, form the page number for valid pages as if it was a 1 level table. As an example from a 3 level page table, if level 0 is 4 bits, level 1 is 8 bits and level 2 is 4 bits, an entry for level 0 page 0x3, level 1 0xCC, level 2 page D which maps to frame 3 would be:

```
00003CCD -> 00000003
```

Note that we zero pad page 3CCD and frame 3 to our standard 8 hexadecimal digits.

Here is a sample invocation:
> ./pagetable ~/lib/cs570/trace.sample.tr 5 7
Page table size: 1048576
Hits 224077 (99.83%), Misses 372 (0.17%) # Addresses 224449
Bytes used: 15024

IMPORTANT NOTE: The number of bytes used depends upon the size of the data structures used. However, you should get a rough estimate of how many bytes might be feasible from the number reported here.

Test Cases and Output

Create reports from 3 runs and leave them in your account, testing a one level page table with 20 bits, and two three level page tables with 4 8 8 and 4 4 12 bits. Leave in your account the page table dump (-p) , the logical to physical translations (-t) and the report file. Use the following template:

./pagetable -p table_4_8_8.txt -t logical_to_physical_4_8_8.txt
~mroch/lib/cs570/trace_sample.tr >report_4_8_8.txt

with the filename modified to reflect the page structure.

Answer the following questions (10 points each):

1. Compare the memory consumption between the runs. Which was the best? Will it always be the best?
2. If you implemented your program correctly, question 1 and 3 should have the same hit/miss rate. Will this always be the case regardless of the memory access pattern?

What to turn in

Pair programmers should turn in a *single package* containing separately answered questions from Part I and the common items for Part II.
You must turn in:

1. Questions from part I.
2. Answers to questions in Test Cases and Output section and the summary (don't include the address translation) for the requested runs. Note that you do not need to take screenshots, you can download the files and print the relevant parts.
3. Paper copy of your work including the program, Makefile, and the three reports.
4. Electronic copy of programs must be in your edoras account in directory cs570/a03_pagetable. Save the page table dump to page_dump.txt and the logical to physical address translation to logical_to_physical.txt.
5. All students must fill out and sign either the single or pair affidavit and attach it to your work. A grade of zero will be assigned if the affidavit is not turned in.

Remember that all assignments are due at the beginning of class, and the policy on late assignments is described in the syllabus.