Assignment 3

Part I Questions – This part must be done on your own. Each question is worth 20 points. Remember, you must show your work for quantitative questions.

1. A psychologist is designing an experiment to measure startle responses. The system generates a loud noise and a camera watches the subject for reactions. The goal is to determine how fast people habituate to stimuli. The subject has a "abort" button that stops the experiment if they become too stressed. It is important that the system stops the stimulus quickly if they abort. Which scheduling algorithm metric should be considered if this experiment is to be conducted effectively?
2. Process P_{107} is currently executing with a priority of 5 (lower is better). Process P_{59} has just completed its I/O burst and has a priority of 4. The ready queue contains P_{86} and P_{95}, with priorities of 5 and 8 respectively. Draw a queueing diagram showing the state of the system after the I/O completion interrupt of P_{59} is handled (queue heads should be on the right of your diagram).
3. A process has CPU bursts of 5, 3, 6, and 7 ms. If the average CPU burst time in a system is 6 ms, show what the shortest process next algorithm would predict for these four bursts when α=0.5.
4. A hard real time system that monitors shipping lanes for whales to avoid whale strikes has the following characteristics:

Task	Frequency (Hz)	Time (ms)
Analyze audio for whales	10	80
Send telemetry to shore	1	50
Notify nearby vessels of whales	5	20

The operating system overhead is 0.2. Can these tasks be scheduled?

5. A physical memory has an access time of 10 ns. A four-level page table is managed by the MMU with a TLB that has a 2 ns access time and a hit rate of .96. What is the effective access time?

6. A physical memory system has divided a small embedded system memory of 16 KiB into blocks of 0x400 (1024_{10}) bytes that are numbered 0x0 through 0xF. Three processes are currently in memory:

Process	First block	Last block
P_7	0x2	0x7
P_9	0x9	0xC
P2	0xD	0xE

   a. Show a bitmap for memory allocation where 1 is allocated and 0 is free. Block 0 should be placed in the low-order bit of the bitmap.

   b. Draw a linked list representation of the memory allocation showing both processes and holes.

   c. Revise a. and b. to show the state of each data structure if P_9 exits.

Part II Programming – Multilevel page table (150 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. 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 Canvas 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. You can include these files in your compilation and use the functionality to process the tracefile. The file trace.sample.tr is a sample of the trace of an executing process. The file samp_read.c provides a demonstration of how to use the BYU tracefile. You are welcome to cannibalize portions of the demonstration program in your assignment. 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. **Implementing this interface correctly is critical to earning points.** Several functions are provided to help you put output in the correct format and are listed next to each of the output modes. These are located in output_mode_helpers.c, which will compile in C or C++. The function to use for each output mode is listed in parentheses after the mode explanation. See the function documentation for details.

Optional arguments:

-n N — Process only the first N memory references. Processes all addresses if not present.

-o mode — output mode. Mode is a string that specifies what is output:

bitmasks – Write out the bitmasks for each level starting with the highest level, one per line. You do not need to actually process any addresses. Program prints bitmasks and exits. (Use **report_bitmasks**.)

logical2physical – Each logical address is mapped to a physical address. (Use **report_logical2physical**.)

page2frame – The page numbers for each level are shown followed by the frame number. (Use **report_pagemap**.)

offset – Show the mapping between logical addresses and their offset. (Use **report_logical2offset**.)

summary – Show summary statistics. This is the default argument if -o is not specified. (Use **report_summary**.) Statistics reported include the page size, number of addresses processed, hit and miss rates, bytes required for the page table and number of frames allocated.

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 ~/lib/cs570/trace/trace.sample.tr 8 12

> Constructs a 2 level page table with 8 bits for level 0, and 12 bits for level 1. Process the entire file and output the summary.

./pagetable -n 10000 -o page2frame ~/lib/cs570/trace/trace.sample.tr 8 7 4

> Constructs a 3 level page table with 8 bits for level 0, 7 bits for level 1, and 4 bits for level 2. Processes only the first 10,000 memory references, and output mappings between page numbers and frame numbers.

Additional examples and correct outputs can be seen in file sample_output.txt that is provided in the same directory as the sample trace and other code.

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 access, 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.

Output is dependent on the output mode specified as discussed above.

What to turn in

Part 1 should be completed solo and submitted via Canvas. Part II may be completed with a pair programmer and is submitted to gradescope. Turn in your code, Makefile, and a text file output.txt showing the summary result of running on the tracefile with arguments 4 4 8 and 16 (3 vs 1 level page table). Affidavits are required as usual.

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:o:")) != -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 'o':     /* produce map of pages */
      // optarg contains the output method…
      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;
```