CS 4323 - DESIGN AND IMPLEMENTATION OF OPERATING SYSTEMS I

Phase II (Due: April 27)

For this phase of the project, you are to change the memory management function to a paged system. You are to simulate and measure the performance of a paged memory with 128 allocable page frames under a virtual memory management scheme. Each page frame is 256 bytes long. The memory allocation strategy used in this simulation is a proportional allocation scheme in which the number of allocated page frames will be 1/4 of the total number of pages required by a job (rounded up to the next higher integer).

(I) MAJOR DATA STRUCTURES

FFT: The free frame table keeps track of the free page frames. For our simulation, FFT is to be maintained "in memory" as a linked list, i.e., each free page frame will have a pointer field pointing to the next free frame.

PCB: The process context/control block of each job maintains, where necessary, such information as jobid, program counter value, size in bytes, size in pages, page table base address, number of page faults, number of replacements (clean, dirty), etc.

PT: The page table entries will have the following format.

v/i	resident	reference	modified	frame#

v/i bit: You will implement a fixed-size page table. Therefore, the page tables will each have 128 entries (to be able to address the entire memory). Thus, the v/i bit is necessary to indicate which portion of the page table is actually occupied by a job.

resident bit: Shows whether the page is currently in memory.

reference bit: Set every time a page is referenced.

modified bit: Set only if the page is referenced for a "write"

(II) SIMULATION INPUT/OUTPUT

INPUT: The input for this simulation will come from a set of files (called the arrivals file and the jb01 through jb23 reference string files) that will be made available in the /home/opsys/OS-I/SP17 directory, as explained below.

a) arrivals: This file contains job load requests in the following format.

ID (integer)	size in bytes (integer)	reference string file name (string of size 4)

The reference string file name will be used to open a file with that name in the aforementioned directory. This file pointer will be copied into the PC (program counter) field of the relevant PCB.

Note: Analogous to Phase I, if the ID and the size in bytes for a new request are 0, you will interpret it as having no new arrivals at that point in time.

b) jb01 through jb23: These files contain the reference strings of the jobs

in the following format. The entries in these files are the successive pages that are referenced as each job is executing its program instructions and as each job is accessing its data.

reference-code + page#
| |
p,w,r integer

These reference codes are defined later in this document.

Note: You are expected to do error checking for memory faults or addressing errors. When a fault occurs, output an "abnormal termination" message followed by statistics similar to a typical normal termination.

OUTPUT: Output specific to Phase II will be written to two files as explained below. Output specific to Phase I should not be included unless your Phase I output was mostly incomplete or erroneous.

(a) mem_stat file

i) After termination of each job, your simulation program will write a message indicating normal/abnormal termination, followed by statistics as described under MEM_MANAGER(release) below.

ii) Memory utilization info will also be written at set intervals (every 4th job delivered). This information will be written by the stats entry of the MEM_MANAGER.

(b) trace file

This file will contain the page placement and replacement trace, with one line of information per placed/replaced page, consisting of the following info: jobid, placed/replaced page, frame#, and page status (e.g., not referenced and not modified; referenced but not modified; etc.) if applicable.

Note: For enhancing the readability of your output files, use a tabular and columnar format with appropriate column headers.

(III) MAJOR COMPONENTS OF THE SYSTEM

(1) LOADER: This component or module is responsible for initializing new jobs and loading the "demanded" pages, as explained below.

i) Initializing New Jobs: You will proceed in the same fashion as in Phase I. The LOADER is also responsible for initializing page tables. The process of initializing a page table for a new job involves two steps:
a) Setting the v/i bit of as many page table entries as the job contains.
b) Clearing the resident, referenced, and modified bits of the page table.

ii) Loading the Demanded Pages: Once a page fault occurs, the system will obtain and allocate a free frame for the newly-referenced page. The LOADER is responsible for loading the new page in the designated frame (in this simulation, loading a page consists of writing a message in the trace file) and redefining the appropriate entry in the page table of the job and the FFT. Handling a page fault will take 10 virtual time units (a page fault is is considered disk I/O), during which time the respective process will be blocked.

(2) MEM_MANAGER: This component is responsible for maintaining the free

frame list (FFT) as well as allocating and releasing memory. The memory manager MEM_MANAGER in our simulation will have the following entries:

i) init: Invoked at the system start-up time, it initializes the free frame table (FFT).

ii) admit: Invoked by the LOADER, it will return a boolean value indicating whether a request to initialize a new job has been accepted. A false flag indicates that there is not enough memory available to load the current request. However, the LOADER will keep trying to initialize new requests until MEM_MANAGER sends a flag indicating that it has completely run out of memory. As in Phase I, you will define (with justification and documentation) your own limit as to when memory is "used up". This simulation involves a demand paging scheme and thus it is not necessary to load any pages into any frames at the time of load. Actual page loading into frames takes place (one page at a time) when a page is actually referenced.

iii) allocate: The SYSTEM will call MEM_MANAGER(allocate) for the purpose of obtaining a free frame. The allocate entry will return the address of the free frame at the head of the free frame table (FFT).

iv) release: Given a pointer that points to the page table of a job, MEM_MANAGER will release the allocated frames. Before memory is released, the following statistics must be output: jobid, job size, number of allocated page frames, internal fragmentation, length of the reference string for this job, total number of page faults, number of clean pages replaced, and number of dirty pages replaced.

v) stats: Invoked by the SYSTEM, this entry will give the utilization status of the memory in terms of the number of free frames divided by the total number of frames, and the number of allocated or occupied frames divided by the total number of frames.

(3) CPU: This component is in charge of fetching, decoding, and executing instructions and updating the reference bits according to the type of reference encountered. The CPU clock will be incremented by 2 time units for the execution of each instruction.

The reference codes are to be interpreted as defined below.

p : refers to a PC instruction access
r : read
w : write

Read/Write requests will cause a job to get blocked for I/O for a total of 12 virtual time units. Use the same scheme for blocked jobs as in Phase I.

When "executing" an instruction, the CPU will determine whether the page is resident. If so, the reference bit will be set according to the type of the reference. If a page is not resident, control is transferred to the SYSTEM. The SYSTEM will determine the validity of the referenced page. If the reference is valid, the SYSTEM will determine whether it owes any free frames to the job. If so, the SYSTEM will call MEM_MANAGER(allocate) to obtain the address of a free frame. It will then call the LOADER and pass to it the address of the page table, the address of the page to be loaded, and the address of the frame to load the page into.

If a replacement page is to be sought, the REPLACER will be invoked and will be given the address of the page table for the job. The REPLACER will return the address of the candidate for replacement (the replacement strategy is described below under REPLACER).

Once the SYSTEM finds a replacement page, it will call the LOADER and pass to it the page table address, the page number of the page to be loaded, and the page number of the replacement page. The LOADER is responsible for swapping the current page to the backing store, i.e., the disk, if the page is dirty (output a message to the trace file to keep track of such situations), loading the new page in the released frame, and redefining the page table. At this point control is transferred to the CPU, which will resume its instruction execution.

(IV) REPLACER

Invoked by the SYSTEM, the REPLACER (which is the implementation of our page replacement strategy) is responsible for finding a replacement page if the job already occupies the maximum number of frames that are allotted to it. In this case, the selection of a replacement page (or a victim page) needs to be based on the following priorities (note that here the term "referenced" here means having encountered either p, r, or w):

not referenced	not modified
not referenced	modified
referenced	not modified
referenced	modified

Once a replacement page is found, the reference bit to all pages of the job will be reset to zero. Notice that the modify bit will not be reset (as a simplifying assumption, we will not consider the "latching" or "anchoring" of the pages that are undergoing modification).

Notes:

You are to use the following subqueue quantum sizes for the scheduler in Phase II of the project: 10, 16, 24, and 40. Other system requirements, e.g., the maximum degree of multiprogramming being 15, are the same as in Phase I.

Initially admit until cumulative size in bytes is close to 128 times 256, or a zero is encountered.

There are no explicit execution times. Burst times are implicit in the reference string files.

As in Phase I, you must keep up with class discussions concerning the project specification. As a result of the discussions in class, there may be some changes in the project specification as outlined in this document.

As in Phase I, all deliverables are to be hard copy (i.e., paper copy) submissions. No soft copy submissions such as email or flash drives will be accepted. No handin submissions are required either.

As in Phase I, late submission penalty is 10% per calendar day, and you are to follow the enclosed handouts titled GENERAL GUIDELINES FOR ALL PROGRAMMING ASSIGNMENTS and the PROCEDURE TO OBTAIN A FINAL COPY OF YOUR PROGRAM/OUTPUT.

Your marked and graded Phase I project should be included with your Phase II deliverables.

You must develop and run your program on CSX. If your development platform is different from CSX, you should be aware of and handle potential language and compiler incompatibility problems between your development platform and the execution platform. In short, your project (simulation program) must

compile and run on CSX.

ID Memory needs

arrival file

7	27264	jb07
8	8960	jb08
9	11488	jb09
10	11584	jb10
11	7664	jb11
0	0	
12	3709	jb12
13	11522	jb13
14	4868	jb14
0	0	
15	14464	jb15
16	17472	jb16

p5 → 2 vtv

p5 → 2 vtv

ex) Jb08

r5
p5
p0
p0
p0
p8
w8
p8
p8
p8
p8
p8
p7

GENERAL GUIDELINES FOR ALL PROGRAMMING ASSIGNMENTS

1. External Documentation - At the beginning of the main routine, there should be documentation containing items a through g below. At the beginning of all other routines (i.e., subprograms, functions, methods, modules, classes, etc.), there should be documentation containing items f and g only.

a. Your name.

b. Course number.

c. Assignment title or number.

d. Date

e. A brief description of the global variables, if any.

f. A brief description of what the routine does, i.e., a short explanation of the functionality of the program/subprogram/function/method/module/class.

g. If applicable, depending how thoroughly and accurately the implementation reflects the specification, a critique of the program/subprogram/function/method/module/class indicating the ways in which it does not meet the programming assignment's specification.

2. Internal Documentation - This is the documentation that is "mixed" or interspersed with the code to clarify the potentially obscure segments of the

code, e.g., case/switch statements, loops, conditional statements, and procedure calls/returns.

a. All major variables should be commented in the declarations except those that are patently self-explanatory.

b. The code should also be commented, not too much though. As a general rule, when in doubt, use a comment.

c. Use meaningful names and blank lines to enhance the understandability of your code. Blank lines help isolate code segments as spatial localities.

3. Program Layout and Data Structures -

a. The code should be well-structured, with indentation showing the general program logic and flow. GOTOs (for implementation languages that actually include GOTO among their instructions) and other unconditional transfers of control should be used only for a fairly good reason, otherwise they should be avoided.

b. The program should be modular. Program modules generally should not be larger than the size of a typical monitor screen in terms of the number of lines.

c. Choose your data structures carefully, e.g., linked lists are generally inefficient and should be avoided, use them only if their use is justifiable.

4. Input and Output -

a. Use prompts and echos only when they contribute to the overall user-friendliness of your program. Avoid unnecessary prompts and echos.

b. Your output should be well-formatted, commented, and understandable. To enhance comprehensibility, arrange your output in rows, columns, blocks, etc. with appropriate explanatory headers, within the framework delineated in the programming assignment's specification.

c. The submitted program and the output files should not include any debugging code or debugging comments.

Notes:

(1) Consider the probability that the system (i.e., CSX) could be down, or have system hardware/software problems, a few days before the due date.

(2) Going through the "design/desk-check/redesign cycle followed by coding" is more effective and efficient than going through the "quick design followed by code/output-check/re-code cycle".

(3) It should go without mentioning that a clean compilation with no warnings is called for (applicable to compiled languages and not interpreted ones).

(4) Keep a copy of the deliverables that you submit for your own reference.

==========

PROCEDURE Appendix
[A handout outlining the "procedure to obtain a final copy of your program/output" will be distributed in the near future.]

==========

Find clean code if
replacing.
Only dirty pa
have to
writ