Siva Krishna Neppali (SXN180043)

CS 5348.003-Operating Systems and Concepts - Design Document (Project 4)

Overview: Need to replace the simple memory manager with a simulalted virtual memory demand paging scheme.The implementation is completed in "swap.c", "paging.c", and "loader.c" to realize the demand paging scheme. The address space of a process does not have to be fully in memory and need to implement the swap space manager. We need to implement the page replacement policy,specifically, the aging policy by associating an aging vector to each physical memory frame to implement this page replacement based on age of each Frame.

Critical Design Updates:

Simos.h

---> **load_into_memory** new function is added here to facilitate the initial page load for the process implemented in paging.c called from loader.c .

CPU.C

--->Load, Add, Multiplication, Ifgo conditional goto, OPsleep event codes were added as operation on MBR and AC similar as in the previous Project3.
--->Added code to handle page fault and periodical age scan in handle_interrupt function.

LOADER.C

load_process_to_swap: This is called by **load_process ,** which returns number of pages required for that Process

---> Process Page Table for the submitted program File is Initialized and different variables like mType buf, msize, numinstr, numdata etc., were declared as per requirement to be used while loading.

---> Code Added to open the submitted program File to read from and functions load_instruction & load_data were called to print the read instructions and data accordingly.

---> The program was loaded into the buffer according to the pageSize, and the same program was written into swap space by inserting it to swapQ to be placed on swap.disk page by page.

---> The process page table was updated to indicate that the page is not empty but it is on disk (= diskPage). After all the pages were put on to the swap.disk the file is closed.

---> All the progError conditions were defined according to the given limitation in config.sys.

--->load_pages_to_memory: This is called by **load_process ,** after **load_process_to_swap** returns number of pages required for that Process.

--->This function will then finds out #pages to load into the memory = min (loadPpages, numpage = #pages loaded to swap for pid).

--->This will then call **load_into_memory** which was newly coded In Paging.c and defined in simos.h to carry out the loading from swap.disk into the main memory.

SWAP.C

---> **read_swap_page, write_swap_page , dump_Process_swap_page** are added referring to the previous code provided.

---> **dump_swapQ** function was coded in consistency with the print statements that were generated during the execution of simos.exe file provided.

--->**Start_swap_manager** was coded **to initialize_swap_space**, initialize semaphores and create swap thread. **swapThread** is initialized to call the entry function **process_swapQ** which will call process_one_swap to take care of the swapping from disk to memory and vice versa. **End_swap_manager** function coded to join swapThread and close diskfd, accordingly swap_semaq is signalled.

---> **insert_swapQ** was coded which will place each insert request at a swapQtail/swapQtail->next nodes, according to the swapQ status during that insertion, This insertion is protected by Semaphores. These insertions are then Processed by a function defined in **process_one_swap** and the action is based on the defined node->act during the insertion into the swapQ. This function is also put under Semaphore Protection to ensure that insertion and processing of swapQ nodes are in synchronization.

PAGING.C

Trivial Changes:

1.highestAge definition was set to 0x00000080 from the given highestAge 0x80000000 in original paging.c

2. A New variable Data_Exception was defined to differentiate the pageFault occurrence whether during data fetch&data Put or during Instruction Fetch.

3. Initialized the remaining pages, and they are also put in free list.

---> **load_into_memory** is called by loader.c during the initial load and all the pages are put into the swapq from disk to memory and the process state is olnly set to ready at the last page insert.

---> **calculate_memory_address** function is called by get_data, put_data, get_instruction functions and uses offset as an input to calculate the memory address and rwflag is used to categorize different exception cases and thus return the status. Incase there is no exception address is calculated using offset, pageoffsetMask, pagenumShift which are calculated. When ever a particular frame is accessed its current age is being changed and set to HighestAge.

---> **get_data, put_data, get_instruction** functions calls calculate_memory_address function to get the address from where to either fetch or put data and fetch instruction accordingly. If no exceptional statuses were returned back and a legitimate address was returned, respective operations are performed on the got address memory. In case of exceptions the errors were returned back to cpu_excection and then the handle_interrupt takes care of those interrupts accordingly. Data_Exception is set up by get_data,put_data incase of mpFault returned as address. Put_data function makes the frame dirty when a new data is put into that frame. get_instruction calculates the CPU.IRopcode, CPU.IRoperand by taking the instruction and using opcodeShift, operandMask(already defined above in paging.c).

---> **dump_one_frame, dump_free_list, dump_process_pagetable, dump_process_memory** functions were coded following the print statements in consistency with simos.exe provided.

---> **update_frame_info** is called to update the frame metadata fields whe a frame is accessed by particular process and this field is set with the highest age and frame is set as used, addto_free_frame is called by different functions like select_agest_frame, Free_process_Memory etc to add the freed frames to the Free_List and update the metadata of those freed frames as free and other metadata fields are also updated accordingly.

---> **get_free_frame** when called scans all the memframes to check whether free frames exist or not, if the freeframes exist it will then return the free frame and if not it will call the function **select_agest_frame** which is now programmed to find the lowest age and return one frame which is clean back to get_free_frame and add the remaining clean frames with the same lowest age to free frame list. And if there is no clean frame this will return back the first one it encounters with the lowest age.

---> **page_fault_handler** is called by handle_interrupt when pagefault interrupt is set by calculate_memory_address function. This will handle page fault obtaining a free frame or getting a frame with the lowest age, if the frame is dirty, it is inserted with a write request to swapQ to write that frame back on to the disk and a read request is inserted to swapQ to bring the required new page to this frame and the frame metadata is updated and the page tables of the involved processes are also updated at each insert. The scenario where the page is being swapped back is not dirty is handled by updating its pagetable. The scenario where the Data_exception is set by get_data, put_data is used to handle page fault distinctly from the page fault interrupt set by get_instruction.

---> **update_process_pagetable** is called in different insert_swapq scenarios to update the process about the place of existance of a particular page of the process whether on a particular memory frame or on the disk as diskpage.

-->**free_process_memory** will free the entire memory of the Freed process when it is completed or terminated intermittently at an exception or error and adds the freed memory frames to the freelist. This also checks if some frames were already freed and the same will be printed incase if they are free already.

---> **initialize_memory_manager** function coded to initialize memory and add page scan event request, memory_agescan event scans the memory and update the age field of each frame by right shifting it by one bit each time it is called when ageInterrupt is set and handled by handle_interrupt function in cpu.c. This will also add the frames with Zero age to Free frame list.

<u>PROCESS.C</u>

---> **Context_In,Context_out** Codes were added and Required Registers were saved according to the PID during context out, same were reloaded as per requirement during context in.

---> In **submit_process** Initial Time used and NumPF(intial pages loaded into the main Memory) were set during new_PCB allocation for that PID.
--->Code for **execute_process** is added for different CPU.exeStatus definitions, context_in, context_out were called accordingly.

<u>TERM.C</u>

---> Code for Semaphore initialization of Semaq, Mutex is added and sem_wait, sem_post signals were placed to ensure the mutually exclusive access same as in the project3.