Project #4 Design

SAURABH LABDE UNITY ID: sslabde

SHRINATH CHERIYANA UNITY ID: scheriy

Physical memory layout

Frame 1048575	**4 GB**
Unmapped	
Frame 14592	**57 MB**
Frame 14591	**57 MB**
Disk Swap Space	
Frame 10496	**41 MB**
Frame 10495	**41 MB**
Free Frame Space	
Frame 8448	**33 MB**
Frame 8447	**33 MB**
Page Directory and Page Table region	
Frame 8192	**32 MB**
Frame 8191	**32 MB**
STATIC REGION: Includes – TEXT, DATA, BSS, HOLE, KERNEL HEAP, STACK. This includes 1 MB of unused space.	
Frame 0	**0 MB**

Initialization of page directories and page tables

Basic setup for initialization:

1) Keep a tab for the frames belonging to each region of the memory. There will be three tabs –
   PD_PT_TAB [256];
   FFS_TAB [2048];
   DSS_TAB [4096];

We are modeling every of frame belonging to these three regions using these three tables. The exact fields of the tables have not been finalized but it will have some kind of reverse mapping indicating which pid is using the frame.

2) A global counter to keep count of the free frames in ffs region. Initially set to 2048 frames.
3) A global vmem free list similar to the memlist to keep tabs of the free frames in the FFS region (This may or may not be needed)

Inside the initialize function, the following should be done.

1) Get a frame from the PD/PT region for the page directory of the null proc. This PD will be shared by all the system processes.
2) Create the page directory and get 15 pages to create page tables for mapping the first 57MB indicated in the memory layout above and fill the pd_base value of each entry with value pointing to each page table.
3) Similarly initialize the page table entries with the appropriate values.
4) Maintain a global array with 15 entries with each entry containing the base address of each of the 15 tables.
5) Enter the base address of the page directory into the PDBR field of proctab[null proc].
6) Write the above value to the CR3 register.

System Initialization

After doing the above steps, setup the page fault handler at exception 14 using the set_evec function.

Call the enable_paging function defined in control_reg.c

The above given operations are done at the end of the sysinit function.

Process Creation

The following changes are to be made in create.c:

1) The PDBR field of proctab[currpid] should be made equal to the PDBR field of proctab[null proc].(This makes sure that every system process uses the same page directory)

The following should be done vcreate.c:

1) Follow the original xinu create functionality except the above step specified for create.c.
2) If the provided hsize is greater than MAX_HEAP_SIZE then return SYSERROR.
3) Subtract hsize from MAX_HEAP_SIZE this reserves space for the virtual heap of the user process.
4) Save the hsize value into the hsize field of proctab[currpid].
5) Get a frame from the PD/PT region for the page directory and create the page directory.
6) Initialize the first 8 PDE to map the static(32MB) region and set the pd_base value of the first 8 PDEs with the first 8 values of the global array created during initialization. So, the page tables for mapping the static region are created only once and shared by all the processes (system as well as user).

7) Save the page directory base address to the PDBR field of proctab[currpid] and write this value to the CR3 register.

Process Termination

Inititally, we created three tables to model each frame in the PD/PT, FFS and SWAP region. This had reverse mapping to the pid that uses these frames. Search these tables and invalidate all the frames used in these regions by the pid to be killed.

So, this will delete all the frames used by the page directory and tables of the current pid including the frames in the SWAP space. This also protects the static region mapping as it should never be deleted as null proc will always remain in the system when no other process exists.

Add the hsize from proctab[currpid] to MAX_HEAP_SIZE.

Context Switch

Before the ctxsw call in resched.c, write the PDBR value of the next process into the CR3 register.

Heap allocation, deallocation and access

Allocation:

1) Get the number of pages to be allocated by rounding of the nbytes to the nearest multiple of 4KB and dividing it by the page size.
2) Get a frame from PD/PT region and create a page table if the previous page table is full. Set the valid bits for PTEs equal to number of pages.
3) Return a virtual pointer pointing to the first allocated PTE.

NOTE: Here the frames are only allocated in the virtual memory but not mapped to the physical memory thereby following the lazy allocation policy.

Deallocation:

1) Get the number of pages to be deallocated by rounding of the nbytes to the nearest multiple of 4KB and dividing it by the page size.
2) From the given virtual address locate the page table and the corresponding PTEs and invalidate npages number of entries. Delete the entries corresponding to the frames to be deallocated from the table maintained for modeling the FFS and add these to the virtual free list.

Access:

1) If the frame to be accessed is already present in the FFS region i.e valid bit and present bit = 1 then no page fault and access is successful.

NOTE: From the 3 bits provided for the programmer's use, we are selecting one to be a valid bit which is set when a page is allocated.

2) In cases other than the above specified, exception 14 occurs and page fault handler is invoked.

Page Fault Handler Design

NOTE: From the 3 bits provided for the programmer's use, we are selecting one to be a swaptodisk bit which is set when an FFS frame is swapped to disk.

The Hardware raises a page fault in the following scenarios:

1) When virtual memory is allocated but not mapped to physical memory (i.e. Present 0 Valid 1 SwaptoDisk 0)
2) When virtual memory is allocated and mapped to physical memory but the FFS frame has been swapped to disk. (i.e. Present 0 Valid 1 SwaptoDisk 1)
3) When virtual memory is not allocated but accessed. (i.e. Present 0 Valid 0)
4) When tried to write to a page that is read-only. (Protection Fault)

Get the fault address by reading CR2 register and get the corresponding PDE and PTE.

For Case 1:

Check if FFS has any free frame.

If yes, perform the following:

1) Get free frame from the FFS region.
2) Update the PDE and PTE of the current process to appropriate value in order to map this frame.
3) Update the table modelling frames of the FFS region to indicate the new owner of the frame.

If no, perform the following:

1) Use the Swap logic indicated below to swap a frame from FFS to SWAP space.
2) Update the PDE and PTE of the current process to appropriate value in order to map the recently freed frame.
3) Update the table modelling frames of the FFS region to indicate the new owner of the frame.

For Case 2:

Check if FFS has any free frame.

If yes, perform the following:

1) Get frame from SWAP space and map it to the free frame from the FFS region and update the PDE and PTE to appropriate values.
2) Update the table modelling frames of the FFS region to indicate the new owner of the frame.

If no, perform the following:

1) Use the Swap logic indicated below to swap a frame from FFS to SWAP space to make space for frame to be brought in from SWAP.
2) Get the required frame from SWAP region and update the PDE and PTE of the current process to appropriate value in order to map the recently freed frame.
3) Update the table modelling frames of the FFS region to indicate the new owner of the frame.

For Case 3:

Raise a SEGMENTATION FAULT.

For Case 4:

Print “PROTECTION FAULT” and kill the process.

Swapping Design [only students taking course at graduate level]

Case 1: Swapping a frame from FFS to SWAP

1) Get a free frame from the SWAP region.
2) If the SWAP space is full return SYSERROR.
3) Move the data at the FFS frame to the newly acquired swapped frame.
4) From the virtual address of the FFS frame and the PDBR value locate the page directory and the corresponding page table entry and update the pt_base value to the swap address and update the present bit to 0 and swaptodisk bit to 1.
5) Update the table for FFS region to indicate that the frame has been freed and update the table for SWAP region to indicate the new owner pid.

Case 2: Swapping a frame SWAP to FFS

1) From the fault address, get the PDE and the PTE values. The pt_base of the PTE indicates the location of the frame in the SWAP space.
2) If a free FSS frame is available, move the data from the frame in the swap space to the free FSS frame and update the pt_base value in the PTE to the address of the new frame.
3) Update the table for FSS region to indicate the new owner pid of the frame and the table entry for the SWAP region should indicate that it has been freed.
4) If FFS frame is not free, perform case 1 to free a frame from the FFS region and perform steps 2 and 3.