XINUPaging

Lab 5 Part 1 - Anirudh Pal

Modified/Created Files

Created

- system/createApps.c
- paging/frame.c
- paging/page.c
- paging/pregs.c
- system/vcreate.c
- system/vgetmem.c
- system/vfreemem.c
- system/pfisr.S
- system/pfhandler.c
- system/createVApps.c
- system/hooks.c

Modified

- include/paging.h
- include/prototypes.h
- system/main.c
- system/create.c
- system/initialize.c
- include/process.h
- system/resched.c
- system/kill.c

Implementations

The implementation is completed with no parts missing the following is a list of major files with a break down of all the features that they help support and an explanation of how that enables demand paging.

frame.c

It contains all functions related with manipulating the data structure that tracks the frames called frametab[]. Here are some of the functions:

initFrames()

This function initialize the entries of frametab[]. All of them are thus available.

getDSFrame()

This function is used to get a frame for page tables and page directories. It obtains frames from the first 1000 in frametab[]. It iterates through the frametab[] looking for an empty frame.

getPFrame()

This function is used to get a frame for pages. It obtains frames from the last 2072 in frametab[]. It iterates through the frametab[] looking for an empty frame.

freeFrames(pid32)

This function sets all frames in frametab[] associated to a pid to free so they can be used by others.

page.c

It contains functions that directly manipulating the frame location and set up page tables and page directories.

setSharedPTs()

This function is used to initialize 5 page tables that do the identity map up to frame 4096 and also the devices section.

getPD()

This function is used to get a page directory setup with the shared page directory entries already pre populated.

getPT()

This function is used to get an empty page table.

pregs.c

enablePaging()

This function is used to set the PG bit of CR0 to enable paging.

setPDBR()

This function is used to set the page directory of the current process. It also flushes the TLB.

getCR2()

This function is used to get the CR2 value which store that fault causing virtual address during the page fault.

pfisr.c

The assembly code gets the page fault error code, save the stack, calls the high level handler, restores the stack and then returns to instruction that caused the page fault.

pfhandler.c

This is run when there is a page fault. It uses the CR2 to figure out what page directory entries, page table, page table entries and pages need to be set and then sets them allowing the virtual address to work.

vcreate.c

Modified create() to support private heap. It sets the dummy head of the private heap.

vgetmem.c

Modified getmem() that operates in the virtual heap area.

vfreemem.c

Modified freemem() that operates in the virtual heap area.

initialize.c

This is where we can first turn on paging. It sets up the null process.

initialize_paging()

Initializes frametab[], shared page tables, page directory for null process, sets the CR3 and CR0.

resched.c

Only addition was to switch the CR3 value to the page directory of the process that is being context switched in.

kill.c

Only addition was to delete all frames that have been obtained by the dying process.

Testing

I ran into the kprintf() problem while testing but now that has been resolved. The fix is simple but a little ugly, it required removing sti and cli from pfisr but I use interrupt disabling within pfhandler. I am not sure how, but it work flawlessly within the bounds of my testing suite.

Here is the break down of the test I performed:

1. Tested a proc that uses the standard heap and not the virtual heap.
2. Tested the frame.c and page.c functions without actually having paging on with emulated requests. Output is at the end.
3. Tested a procs that uses virtual heap with vgetmem and freemem that get, set and check the values in the heap. I checked with 1 Page, 2 Pages, 1024 Pages and 2072 Pages.
4. Ran two procs that write to the same virtual address in the heap and made sure the value were as expected.
5. Output of the virtual heap tests are at the end.

Note

I enable hooks with a macro called VERBOSE in include/paging.h. Uncomment it to get hook output.

Commit Log

1. Set good structure, macros and global vars.
2. Removed backing store and shell from main. Added a normal app that uses heap.

3. Added support for initialize, get, set, print and free frames.
4. Testing frames functions.
5. Added support for initialize, get, set and print PDs and PTs.
6. Testing pages functions.
7. Turning on paging and adding all necessary DS to procs.
8. Added support for heap.
9. Testing heap without hooks.
10. Testing heap with hooks (STI/CLI problem).
11. More heap testing.
12. Added kill free frames.
13. Enable all hooks.
14. Added vfreemem() and tests.
15. Added more testing.
16. Added address check in pfhandler().

Virtual Heap Test Output

```
Testing Normal Heap
PID: 3 -> Using Stack
PID: 3 -> Used Stack, First Add: 0x16a1e0, First Val: 0x123abcef

Testing 1 Page Usage
PID: 3 -> Using 1 Pages
PID: 3 -> Used All Pages, First Add: 0x1000000, First Val: 0x1234abcd, pfCount: 2

Testing 2 Page Usage
PID: 4 -> Using 2 Pages
PID: 4 -> Used All Pages, First Add: 0x1000000, First Val: 0x1234abcd, pfCount: 2

Testing 1024 Page Usage
PID: 5 -> Using 1024 Pages
PID: 5 -> Used All Pages, First Add: 0x1000000, First Val: 0x1234abcd, pfCount: 20905986

Testing 2072 Page Usage
PID: 6 -> Using 2072 Pages
PID: 6 -> Used All Pages, First Add: 0x1000000, First Val: 0x1234abcd, pfCount: 25231362

Testing 2072 Page Usage
PID: 7 -> Using 2072 Pages
PID: 7 -> Used All Pages, First Add: 0x1000000, First Val: 0x1234abcd, pfCount: 25231362

Testing 1 Page Usage
PID: 8 -> Using 1 Pages with freemem()
```

```
Iteration: 99
PID: 8 -> Done Using Heap

Testing 2 Page Usage
PID: 9 -> Using 2 Pages with freemem()
Iteration: 99
PID: 9 -> Done Using Heap

Testing 1024 Page Usage
PID: 10 -> Using 1024 Pages with freemem()
Iteration: 99
PID: 10 -> Done Using Heap

Testing 2072 Page Usage
PID: 11 -> Using 2072 Pages with freemem()
Iteration: 99
PID: 11 -> Done Using Heap

Testing 2072 Page Usage
PID: 12 -> Using 2072 Pages with freemem()
Iteration: 99
PID: 12 -> Done Using Heap

Testing Same Virtual Address
PID: 13 -> Using 1 Byte
PID: 13 -> Used 1 Byte, First Add: 0x1000000, First Val: A
PID: 14 -> Using 1 Byte
PID: 14 -> Used 1 Byte, First Add: 0x1000000, First Val: B
```

Without Paging Test Output

```
Testing Normal Heap

PID: 3 -> Using Stack

PID: 3 -> Used Stack, First Add: 0x1691e0, First Val: 0x123abcef


Testing Frames

DS frameNum: 2023

frametab[0]: .type = 1 .fnum = 1024 .pid = 4 .addr = 0x400000
frametab[1]: .type = 1 .fnum = 1025 .pid = 4 .addr = 0x401000
frametab[2]: .type = 1 .fnum = 1026 .pid = 4 .addr = 0x402000
.
.
.
```

```
frametab[997]: .type = 1 .fnum = 2021 .pid = 4 .addr = 0x7e5000
frametab[998]: .type = 1 .fnum = 2022 .pid = 4 .addr = 0x7e6000
frametab[999]: .type = 1 .fnum = 2023 .pid = 4 .addr = 0x7e7000

P frameNum: 4095

frametab[0]: .type = 1 .fnum = 1024 .pid = 4 .addr = 0x400000
frametab[1]: .type = 1 .fnum = 1025 .pid = 4 .addr = 0x401000
frametab[2]: .type = 1 .fnum = 1026 .pid = 4 .addr = 0x402000
.
.
.
frametab[3069]: .type = 3 .fnum = 4093 .pid = 4 .addr = 0xffd000
frametab[3070]: .type = 3 .fnum = 4094 .pid = 4 .addr = 0xffe000
frametab[3071]: .type = 3 .fnum = 4095 .pid = 4 .addr = 0xfff000

Testing Pages

frametab[0]: .type = 2 .fnum = 1024 .pid = 5 .addr = 0x400000
frametab[1]: .type = 2 .fnum = 1025 .pid = 5 .addr = 0x401000
frametab[2]: .type = 2 .fnum = 1026 .pid = 5 .addr = 0x402000
frametab[3]: .type = 2 .fnum = 1027 .pid = 5 .addr = 0x403000
frametab[4]: .type = 2 .fnum = 1028 .pid = 5 .addr = 0x404000

frametab[0]: .type = 2 .fnum = 1024 .pid = 5 .addr = 0x400000
frametab[1]: .type = 2 .fnum = 1025 .pid = 5 .addr = 0x401000
frametab[2]: .type = 2 .fnum = 1026 .pid = 5 .addr = 0x402000
frametab[3]: .type = 2 .fnum = 1027 .pid = 5 .addr = 0x403000
frametab[4]: .type = 2 .fnum = 1028 .pid = 5 .addr = 0x404000
frametab[5]: .type = 1 .fnum = 1029 .pid = 5 .addr = 0x405000

frametab[0]: .type = 2 .fnum = 1024 .pid = 5 .addr = 0x400000
frametab[1]: .type = 2 .fnum = 1025 .pid = 5 .addr = 0x401000
frametab[2]: .type = 2 .fnum = 1026 .pid = 5 .addr = 0x402000
frametab[3]: .type = 2 .fnum = 1027 .pid = 5 .addr = 0x403000
frametab[4]: .type = 2 .fnum = 1028 .pid = 5 .addr = 0x404000
frametab[5]: .type = 1 .fnum = 1029 .pid = 5 .addr = 0x405000
frametab[6]: .type = 2 .fnum = 1030 .pid = 5 .addr = 0x406000

PT at Frame 1024
pPT[0]: .pt_pres = 1 .pt_base = 0
pPT[1]: .pt_pres = 1 .pt_base = 1
pPT[2]: .pt_pres = 1 .pt_base = 2
.
.
.
pPT[1021]: .pt_pres = 1 .pt_base = 1021
pPT[1022]: .pt_pres = 1 .pt_base = 1022
pPT[1023]: .pt_pres = 1 .pt_base = 1023

PT at Frame 1025
```

```
pPT[0]: .pt_pres = 1 .pt_base = 1024
pPT[1]: .pt_pres = 1 .pt_base = 1025
pPT[2]: .pt_pres = 1 .pt_base = 1026
.
.
.
pPT[1021]: .pt_pres = 1 .pt_base = 2045
pPT[1022]: .pt_pres = 1 .pt_base = 2046
pPT[1023]: .pt_pres = 1 .pt_base = 2047

PT at Frame 1026
pPT[0]: .pt_pres = 1 .pt_base = 2048
pPT[1]: .pt_pres = 1 .pt_base = 2049
pPT[2]: .pt_pres = 1 .pt_base = 2050
.
.
.
pPT[1021]: .pt_pres = 1 .pt_base = 3069
pPT[1022]: .pt_pres = 1 .pt_base = 3070
pPT[1023]: .pt_pres = 1 .pt_base = 3071

PT at Frame 1027
pPT[0]: .pt_pres = 1 .pt_base = 3072
pPT[1]: .pt_pres = 1 .pt_base = 3073
.
.
.
pPT[1022]: .pt_pres = 1 .pt_base = 4094
pPT[1023]: .pt_pres = 1 .pt_base = 4095

PT at Frame 1028
pPT[0]: .pt_pres = 1 .pt_base = 589824
pPT[1]: .pt_pres = 1 .pt_base = 589825
pPT[2]: .pt_pres = 1 .pt_base = 589826
.
.
.
pPT[1022]: .pt_pres = 1 .pt_base = 590846
pPT[1023]: .pt_pres = 1 .pt_base = 590847

PD at Frame 1029
pPD[0]: .pd_pres = 1 .pd_base = 1024
pPD[1]: .pd_pres = 1 .pd_base = 1025
pPD[2]: .pd_pres = 1 .pd_base = 1026
pPD[3]: .pd_pres = 1 .pd_base = 1027
pPD[576]: .pd_pres = 1 .pd_base = 1028

PT at Frame 1030
```