CS3523 : OPERATING SYSTEMS 2
Programming Assignment 5

MANOJ KUMAR

ES21BTECH11010

PART A - Demand Paging

Implementation :

1) mydemandPage.c

- Firstly we create **mydemandPage.c** which is responsible to execute when we call the user program **mydemandPage**.
- In this function we declare a global array, based on the pagetables generated for mapping this array to physical memory we do furthur analysis.
- To print the pagetables we call the function **pgtPrint()**.

2) Makefile

```
UPROGS =\
_cat\
_echo\
_forktest\
_grep\
_init\
_kill\
_ln\
_ls\
_mkdir\
_rm\
_sh\
_stressfs\
_usertests \
_wc\
_zombie\
_mydate\
_mypgtPrint \
_mydemandPage \
```

We add mydemandPage as a user program to the OS.

3) exec.c

```
/*
*/
   if((sz    = allocuvm (pgdir , sz ,    ph.vaddr   + ph.filesz))       ==0)
goto  bad;
/*
*/
sz   = sz -ph.filesz+ph.memsz    ;
/*
*/
bad:
   if(pgdir)
```

```
12      freevm(pgdir);
13      if(ip){
14         iunlockput (ip);
15         end_op ();
16         }
17      return   -1;
```

- The function **allocuvm()** is used to allocate a contiguous range of virtual memory pages for a process.
- The function takes three arguments: the process's page directory, the old size of process, the new size of process.
- the **allocuvm()** function returns 0 if it fails to allocate the requested amount of memory for the process. When return 0 occurs control shifts to *bad* codeblock.
- in LINE-3 we change **ph.memsz** to **ph.filesz** in the if-conditional indicating that we are only allocating memory only for initialized memory and not for uninitialized memory.

4) trap.c

```
1  //   Page  fault    detection
2  case  T_PGFLT:
3  ad  = PGROUNDDOWN (rcr2 ());
4  mem= kalloc ();
5  memset(mem ,0, PGSIZE);
6  mappages(myproc () ->pgdir ,( char *)ad ,PGSIZE ,   V2P(mem), PTE_W |   PTE_U);
7  //   Printing    the  address   of  pagefault
8  cprintf("page     fault    occured ,  doing   demandpaging   for   address: 0x%x\n",rcr2 ());
9  break  ;
```

- Pagefault is given a trap number of 14 which is defined in **trap.h**

```
#define  T_PGFLT            14
```

- LINE 2 : We handle that the trap is pagefault trap.
- LINE 3 : The **PGROUNDDOWN()** function is used to round down a given address to the nearest page boundary. **rcr2()** is a function that reads the value of the CR2 control register, which holds the linear address that caused a page fault. This line of code calculates the page-aligned address from the virtual address that caused the page fault.
- LINE 4 : The function **kalloc()** is used to allocate memory in the kernel address space.
- LINE 5 : The function **memset()** is used to initialize the memory to zeros.
- LINE 6 : The fiunction **mappages()** function is used in xv6 to map a page of physical memory to a page of virtual memory in a process's address space.

Observations for gobal array length variation:

Demand paging is done after page fault occurs, accordingly the size of pagetable increases based on the size of the global array declared.

- For no global array declaration

```
$ mydemandPage
Printing final page table:
Entry No. : 1, Virtual address : 0, the Physical address is : dee2000
Entry No. : 2, Virtual address : 2000, the Physical address is : dedf000
```

Number of pages =2.

- For global array of size 3000

```
Printing final page table:
Entry No. : 1, Virtual address : 0, the Physical address is : dee2000
Entry No. : 2, Virtual address : 1000, the Physical address is : dfbc000
Entry No. : 3, Virtual address : 2000, the Physical address is : df76000
Entry No. : 4, Virtual address : 3000, the Physical address is : dfbf000
Entry No. : 5, Virtual address : 5000, the Physical address is : dedf000
Value: 2
```

Number of pages =5.

- For global array of size 5000

```
Printing final page table:
Entry No. : 1, Virtual address : 0, the Physical address is : dee2000
Entry No. : 2, Virtual address : 1000, the Physical address is : dfbc000
Entry No. : 3, Virtual address : 2000, the Physical address is : df76000
Entry No. : 4, Virtual address : 3000, the Physical address is : dfbf000
Entry No. : 5, Virtual address : 4000, the Physical address is : dfc0000
Entry No. : 6, Virtual address : 5000, the Physical address is : dfc1000
Entry No. : 7, Virtual address : 7000, the Physical address is : dedf000
Value: 2
```

Number of pages =7.

- For global array of size 10000

```
Printing final page table:
Entry No. : 1, Virtual address : 0, the Physical address is : dee2000
Entry No. : 2, Virtual address : 1000, the Physical address is : dfbc000
Entry No. : 3, Virtual address : 2000, the Physical address is : df76000
Entry No. : 4, Virtual address : 3000, the Physical address is : dfbf000
Entry No. : 5, Virtual address : 4000, the Physical address is : dfc0000
Entry No. : 6, Virtual address : 5000, the Physical address is : dfc1000
Entry No. : 7, Virtual address : 6000, the Physical address is : dfc2000
Entry No. : 8, Virtual address : 7000, the Physical address is : dfc3000
Entry No. : 9, Virtual address : 8000, the Physical address is : dfc4000
Entry No. : 10, Virtual address : 9000, the Physical address is : dfc5000
Entry No. : 11, Virtual address : a000, the Physical address is : dfc6000
Entry No. : 12, Virtual address : c000, the Physical address is : dedf000
Value: 2
```

Number of pages = 12.

So, from the above outputs it is clear that as the size of global array increases, size of page table increases.