ELL783 Assignment 2

Submitted By,
Lokesh Patel (2017EE10584)
Samarth Gupta (2017EE10608)

Code snapshots for big functions and some insignificant code was omitted to make the report within page limit. Please refer to source code if the code snapshot is not present but needed.

1 Part 1: Paging Framework

1.1 Supplied File Framework

Downloaded the xv6 files with the framework implemented in fs.c.

1.2 Storing Pages in File

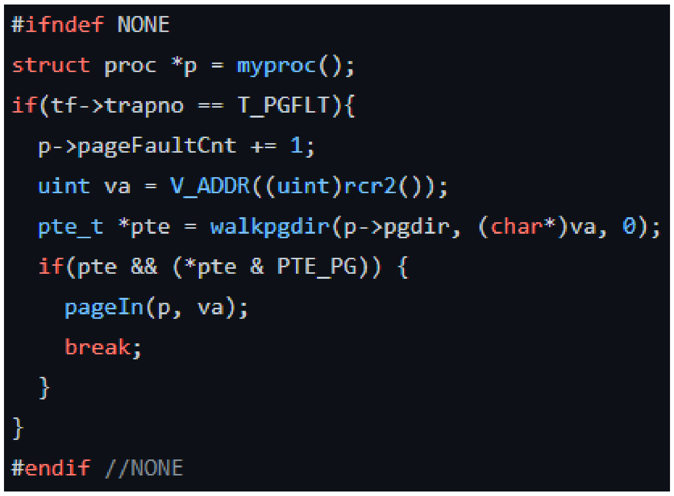
- param.h
  - Defined MAX_PSYNC_PAGES and MAX_TOTAL_PAGES as values 15 and 30 respectively.
- proc.h
  - Added two arrays, one to store metadata about pages in memory, and other for paged out pages. Added some counters for tracking number of pages in memory, number of pages in backing store, number of page faults, and total number of times pages were pages out.
  - The array *memPages* will store the virtual page address along with the access bit in FIFO order. The most significant 20 bits will store the address (because the least significant 12 bits are always 0 for page starting address), and the least significant 12 bits will be used to store the counter value for NFU with aging.
  - The array *pagedOut* will store the virtual page addresses. The address stored at i^{th} position in the array corresponds to the page stored at position i*PGSIZE in swap file. The first 20 bits contain the address, and the left out 12 bits contain the PTE_P flag to denote if the entry is valid.
  - Added two macros for easy access to the address and counter from the two array elements.
- mmu.h
  - Added PTE_PG and PTE_A flag definition.
- vm.c
  - Added five functions for facilitating demand paging
  - *pageOut*: This function removes one page from memory. The page is selected based on the criteria as in section 2. Then the selected page is removed from *memPages* and

moved to a free slot in *pagedOut*. The physical space is freed and frame is copied to swapFile. Further PTE_PG flag is set in the page table.

- ○ *pageIn*: This function moves a page from swap file to memory. And then the respective changes in *memPages, pagedOut,* and page table are carried out. This function also ensures that the MAX_PSYC_PAGES limit is not violated by paging out sufficient pages before paging in the new page.
- ○ *addPageMetaData*: This function simply helps in filling up the meta data whenever a new page is mapped through use of *mappages*. It also ensures that the MAX_PSYC_PAGES limit is not violated, similar to what is done in *pageIn*.
- ○ *removePageMetaData*: This function helps in removing the meta data of in memory pages which are deallocated through the use of *deallocuvm*.
- ○ *removePagedOutMetaData*: This function works similar to above function but here it removes the meta data for paged out pages which were deallocated through the use of *deallocuvm*.

1.3 Retrieving Pages on Demand

Inside the *trap* function in trap.c, check if the trap number is corresponding to page fault (T_PGFLT). In the case of page fault, retrieve the virtual address which caused page fault from the CR2 register, then walk through the page table to check if the page fault is due to the page being in the swapFile. This can be checked by checking the PTE_PG flag in the page table entry. If that is the case, then use the *pageIn* function defined and implemented in 1.2 to load it into memory.

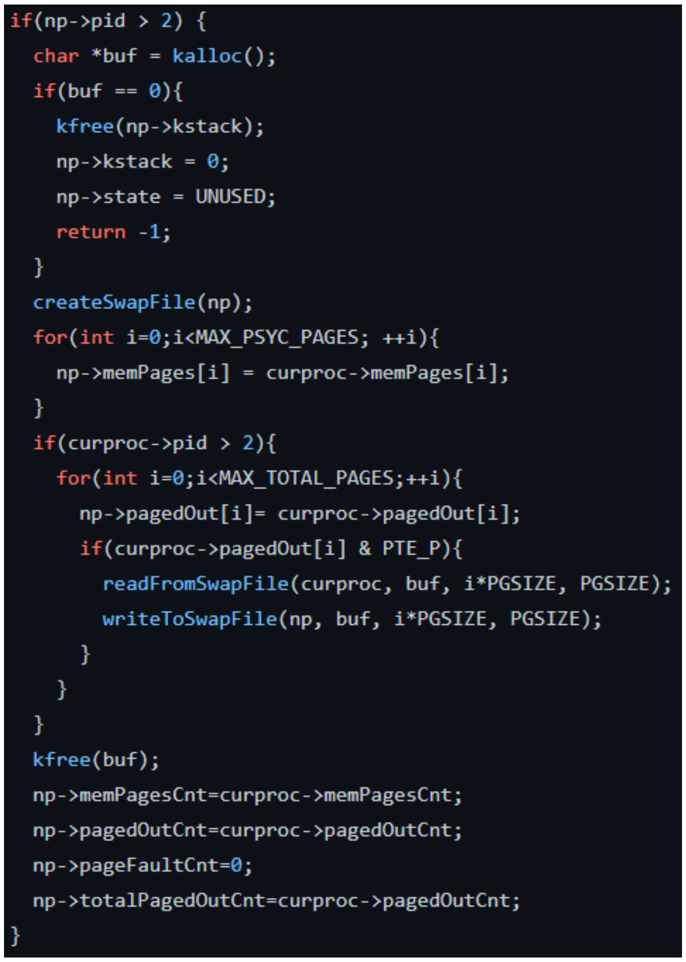


```c
#ifndef NONE
struct proc *p = myproc();
if(tf->trapno == T_PGFLT){
  p->pageFaultCnt += 1;
  uint va = V_ADDR((uint)rcr2());
  pte_t *pte = walkpgdir(p->pgdir, (char*)va, 0);
  if(pte && (*pte & PTE_PG)) {
    pageIn(p, va);
    break;
  }
}
#endif //NONE
```

1.4 Comprehensive Changes in Existing Parts

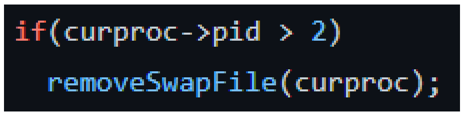
- exec.c
    - ○ Reset the data in page metadata data structure. Reset all the page fault and page out counts.
- proc.c

- *fork*: In fork, we copied the meta data value from parent process to child process and further copied all data from parent's swap file to child's swap file. Only create the swap file if pid > 2 (because pid 0 and 1 correspond to init and shell). Further the page fault count is reset to 0 for the new process because technically it doesn't have any page faults yet, similarly, total paged out count is reset to current paged out count.



```
if(np->pid > 2) {
  char *buf = kalloc();
  if(buf == 0){
    kfree(np->kstack);
    np->kstack = 0;
    np->state = UNUSED;
    return -1;
  }
  createSwapFile(np);
  for(int i=0;i<MAX_PSYC_PAGES; ++i){
    np->memPages[i] = curproc->memPages[i];
  }
  if(curproc->pid > 2){
    for(int i=0;i<MAX_TOTAL_PAGES;++i){
      np->pagedOut[i]= curproc->pagedOut[i];
      if(curproc->pagedOut[i] & PTE_P){
        readFromSwapFile(curproc, buf, i*PGSIZE, PGSIZE);
        writeToSwapFile(np, buf, i*PGSIZE, PGSIZE);
      }
    }
  }
  kfree(buf);
  np->memPagesCnt=curproc->memPagesCnt;
  np->pagedOutCnt=curproc->pagedOutCnt;
  np->pageFaultCnt=0;
  np->totalPagedOutCnt=curproc->pagedOutCnt;
}
```

- *exit*: In exit, after the *freevm* call, we delete the swap file before closing other open files. Other metadata need not be removed because they will get reset during the next time it gets allocated in *allocproc* due to *fork*.



```
if(curproc->pid > 2)
  removeSwapFile(curproc);
```

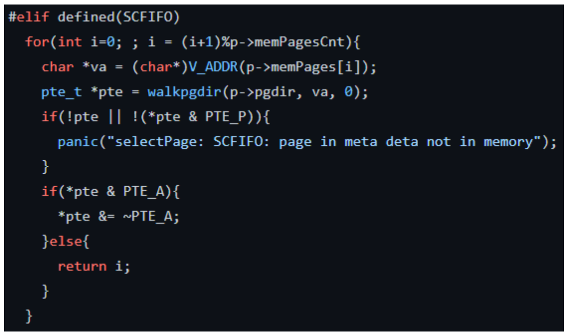
2 Part 2: Page Replacement Schemes

FIFO

The data structure already stored the page meta data in first in first out format. Hence, the page stored at position 0 in the array stores the page that will be replaced.

SCFIFO

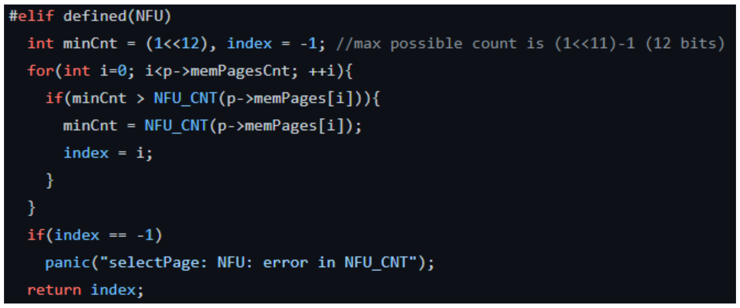
We go through the page metadata array in a cyclic manner. If the page at current index has the access bit set, unset it and move onto the next index (in cyclic manner). Keep doing this till a page is found without the access bit set, which will be replaced.



```
#elif defined(SCFIFO)
  for(int i=0; ; i = (i+1)%p->memPagesCnt){
    char *va = (char*)V_ADDR(p->memPages[i]);
    pte_t *pte = walkpgdir(p->pgdir, va, 0);
    if(!pte || !(*pte & PTE_P)){
      panic("selectPage: SCFIFO: page in meta deta not in memory");
    }
    if(*pte & PTE_A){
      *pte &= ~PTE_A;
    }else{
      return i;
    }
  }
```

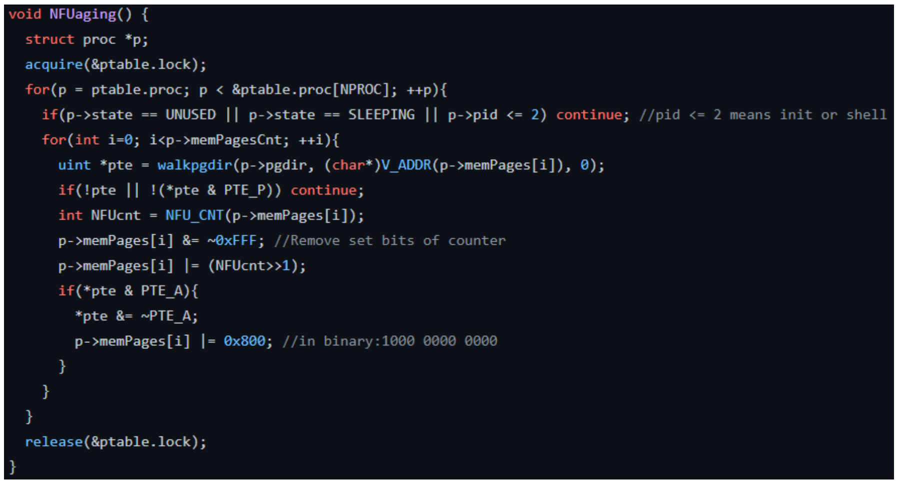
NFU (with aging)

We were already storing the NFU count in the paging metadata. Hence, we iterate through all pages in the metadata and find the index of the page with the lowest count. Then it is selected to be replaced.



```
#elif defined(NFU)
  int minCnt = (1<<12), index = -1; //max possible count is (1<<11)-1 (12 bits)
  for(int i=0; i<p->memPagesCnt; ++i){
    if(minCnt > NFU_CNT(p->memPages[i])){
      minCnt = NFU_CNT(p->memPages[i]);
      index = i;
    }
  }
  if(index == -1)
    panic("selectPage: NFU: error in NFU_CNT");
  return index;
```

The aging is implemented in trap.c. The timer interrupt checks if NFU is defined, if yes then calls the function *NFUaging* which is defined in proc.c. The function goes through all active processes' metadata, and their NFU counter is updated (counter is divided by 2 then if PTE_A bit is set, then 12th LSB is set in counter). Then the PTE_A bit for all these pages is unset.



```
void NFUaging() {
  struct proc *p;
  acquire(&ptable.lock);
  for(p = ptable.proc; p < &ptable.proc[NPROC]; ++p){
    if(p->state == UNUSED || p->state == SLEEPING || p->pid <= 2) continue; //pid <= 2 means init or shell
    for(int i=0; i<p->memPagesCnt; ++i){
      uint *pte = walkpgdir(p->pgdir, (char*)V_ADDR(p->memPages[i]), 0);
      if(!pte || !(*pte & PTE_P)) continue;
      int NFUcnt = NFU_CNT(p->memPages[i]);
      p->memPages[i] &= ~0xFFF; //Remove set bits of counter
      p->memPages[i] |= (NFUcnt>>1);
      if(*pte & PTE_A){
        *pte &= ~PTE_A;
        p->memPages[i] |= 0x800; //in binary:1000 0000 0000
      }
    }
  }
  release(&ptable.lock);
}
```

NONE

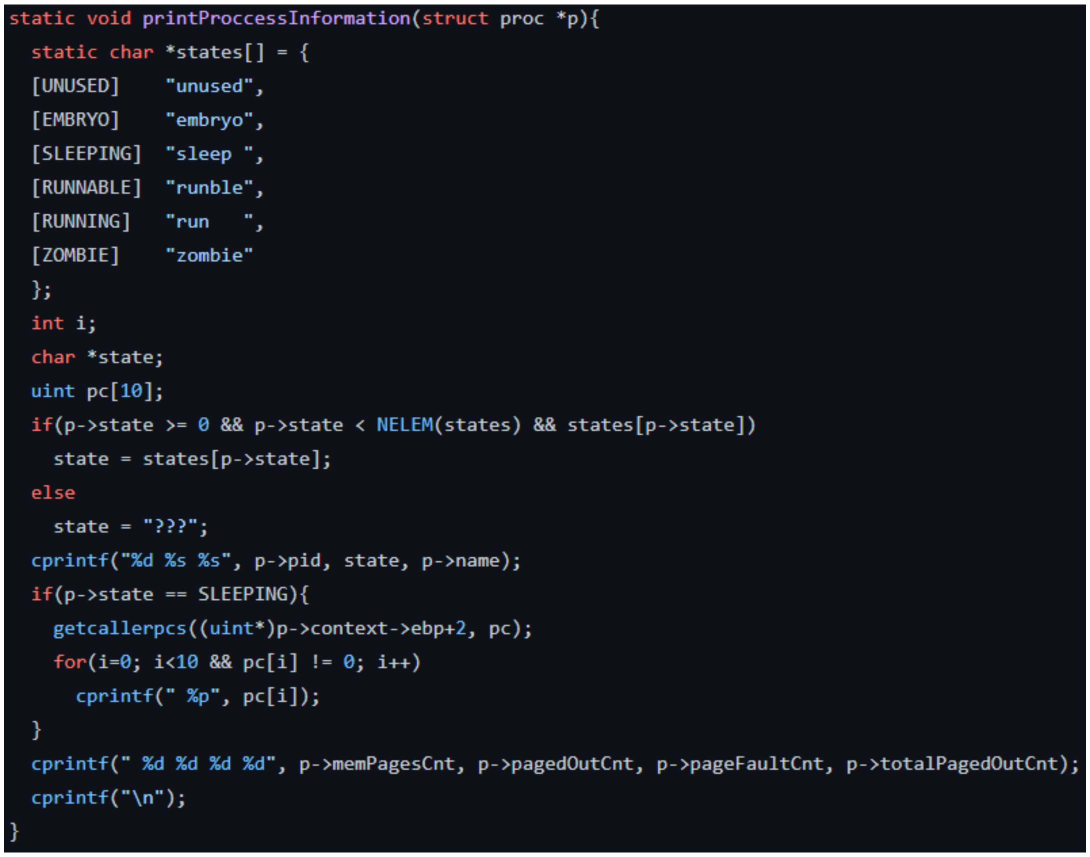
We have placed *#ifndef NONE* among all the paging metadata changing functions so that if NONE is defined, then no pages will be replaced and sent to swapFile. Hence, the original xv6 functionality will be preserved.

Makefile

While compiling we can set which of the page replacement schemes to use. We can do it like this:
make SELECTION=NFU
make qemu SELECTION=NFU
If it is not specified by default NFU is selected. This is achieved by declaring a variable in the makefile with name SELECTION. If it is not defined by the user, then it is set to NFU. Then the option of -D$(SELECTION) is added to compilation flags to set the replacement scheme while compiling.

3 Part 3: Enhanced Process Details Viewer

- **Counting Free Pages in System:**
  In kalloc.c, we created a struct with two counters to count total available pages and currently free pages. *kinit1* and *kinit2* free memory for later allocation, hence start currentPageCnt with 0, and after *kinit2* is finished, that will be total pages. *kfree* will lead to increase in current free pages, and *kalloc* will lead to a decrease. Hence, they help in counting current free pages.
  Then another function is added which computes and returns the percentage of free pages which is used in enhanced process details viewer.
- **Enhancing The Process Details Viewer:**
  Printed the extra info: assigned pages which are in memory, number of pages currently in swap file, total number of page faults, and total number of times pages were swapped to swap file. These info all are present in the metadata, hence easily accessed.
  At the end the percentage of free pages is printed using the function defined in kalloc.c.
  VERBOSE_PRINT macro is added to Makefile, which is by default FALSE. And similar to SELECTION macro, it is also added to compilation flags.
  The info printing part of the process is extracted as another function called *printProcessInformation*, so as to make it easy to print verbose information after the process is killed, if VERBOSE_PRINT was TRUE.



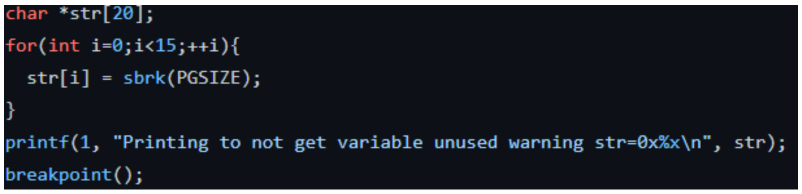
```
static void printProccessInformation(struct proc *p){
  static char *states[] = {
  [UNUSED]    "unused",
  [EMBRYO]    "embryo",
  [SLEEPING]  "sleep ",
  [RUNNABLE]  "runble",
  [RUNNING]   "run   ",
  [ZOMBIE]    "zombie"
  };
  int i;
  char *state;
  uint pc[10];
  if(p->state >= 0 && p->state < NELEM(states) && states[p->state])
    state = states[p->state];
  else
    state = "???";
  cprintf("%d %s %s", p->pid, state, p->name);
  if(p->state == SLEEPING){
    getcallerpcs((uint*)p->context->ebp+2, pc);
    for(i=0; i<10 && pc[i] != 0; i++)
      cprintf(" %p", pc[i]);
  }
  cprintf(" %d %d %d %d", p->memPagesCnt, p->pagedOutCnt, p->pageFaultCnt, p->totalPagedOutCnt);
  cprintf("\n");
}
```

3.1 Sanity Check

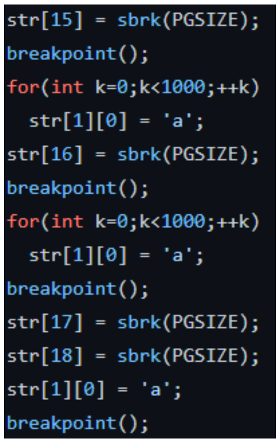
Added user program myMemTest to xv6. We first fill up the memory with 15 pages (page 0 to 14). Then allocate page 15. Then use page 1. Then allocate page 16 (This should differentiate between FIFO and SCFIFO, because PTE_A set due to use of page 1 will prevent it from going to swap file. We cannot be sure about NFU because with aging, the counter could have gone down to 0 before we try looking for page replacement, but theoretically we assume the counter is not 0). Then again use page 1 to check if it causes page fault. Then allocate page 17 and 18. Then use page 1. This should differentiate between SCFIFO and NFU because after allocating page 17, PTE_A in case of SCFIFO will be unset for page 1, then in the next page allocation it will go into the swap file. But if the counter for NFU is not zero yet, then page 1 will remain in memory, not causing any page fault. Again in case of NFU due to aging we cannot be sure of the page fault and it depends whether the counter has gone to zero. Then we fork to test if the swap file in the child process is working correctly. In that case we first free one space in memory by deallocating page 18, then use page 0. If fork is done correctly, then page fault happens and page 0 is brought back from the swap file of the child process. The following table summarizes these steps: (**Mem** is pages in memory, **Swap** is pages in swap file, **PgFlts** is number of page faults)
(For allocating page i we use *str[i] = sbrk(PGSIZE)* system call, and for deallocating, again use *sbrk(-PGSIZE)*. For using a page i, we do *str[i][0]='a'*)

Operation	FIFO			SCFIFO			NFU		
	Mem	Swap	PgFlts	Mem	Swap	PgFlts	Mem	Swap	PgFlts
init	0..14	-	0	0..14	-	0	0..14	-	0
breakpoint1									
alloc15	1..15	0	0	1..15	0	0	1..15	0	0
breakpoint2									
use1	1..15	0	0	1..15	0	0	1..15	0	0
alloc16	2..16	0,1	0	1,3..16	0,2	0	1,3..16	0,2	0
breakpoint3									
use1	3..16,1	0,2	1	1,3..16	0,2	0	1,3..16	0,2	0
breakpoint4									
alloc17	4..16,1,17	0,2,3	1	1,4..17	0,2,3	0	1,4..17	0,2,3	0
alloc18	5..16,1,17,18	0,2,3,4	1	4..18	0,2,3,1	0	1,5..18	0,2,3,4	0
use1	5..16,1,17,18	0,2,3,4	1	5..18,1	0,2,3,4	1	1,5..18	0,2,3,4	0
breakpoint5									
fork (resets page fault count on child process)									
del18	5..16,1,17	0,2,3,4	0	5..17,1	0,2,3,4	0	1,5..17	0,2,3,4	0
breakpoint6									
use0	5..16,1,17,0	2,3,4	1	5..17,1,0	2,3,4	1	1,5..17,0	2,3,4	1
breakpoint7									

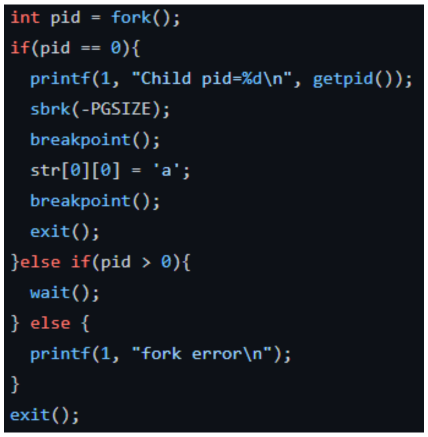
The code for the user program is snapshot below:



```
char *str[20];
for(int i=0;i<15;++i){
  str[i] = sbrk(PGSIZE);
}
printf(1, "Printing to not get variable unused warning str=0x%x\n", str);
breakpoint();
```

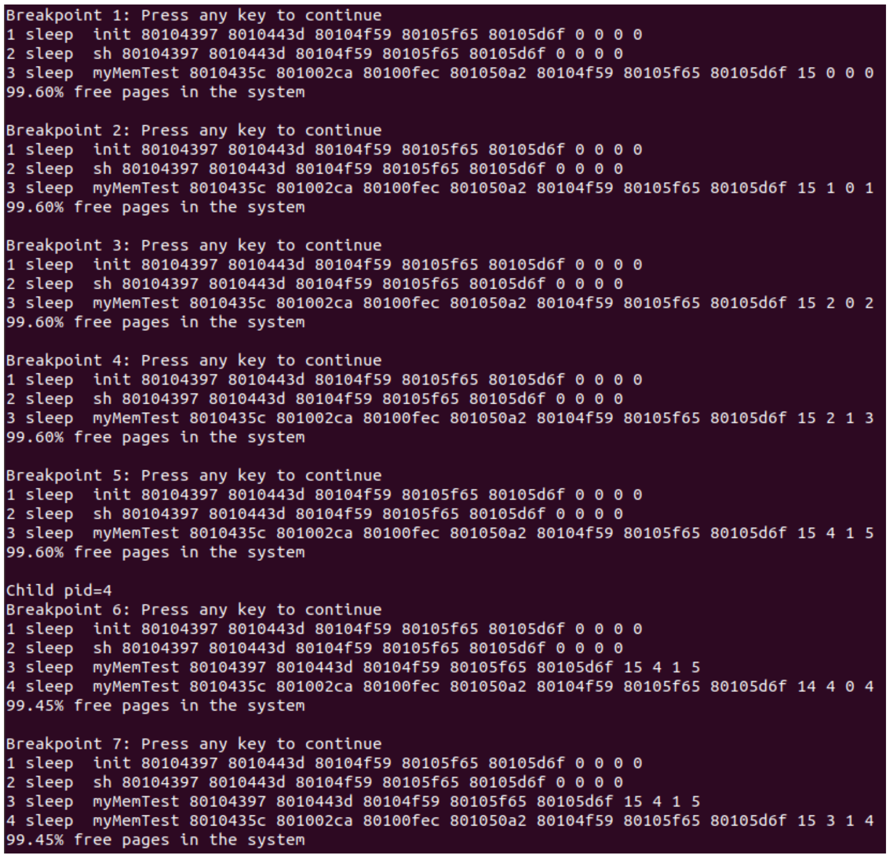


```c
str[15] = sbrk(PGSIZE);
breakpoint();
for(int k=0;k<1000;++k)
  str[1][0] = 'a';
str[16] = sbrk(PGSIZE);
breakpoint();
for(int k=0;k<1000;++k)
  str[1][0] = 'a';
breakpoint();
str[17] = sbrk(PGSIZE);
str[18] = sbrk(PGSIZE);
str[1][0] = 'a';
breakpoint();
```



```c
int pid = fork();
if(pid == 0){
  printf(1, "Child pid=%d\n", getpid());
  sbrk(-PGSIZE);
  breakpoint();
  str[0][0] = 'a';
  breakpoint();
  exit();
}else if(pid > 0){
  wait();
} else {
  printf(1, "fork error\n");
}
exit();
```

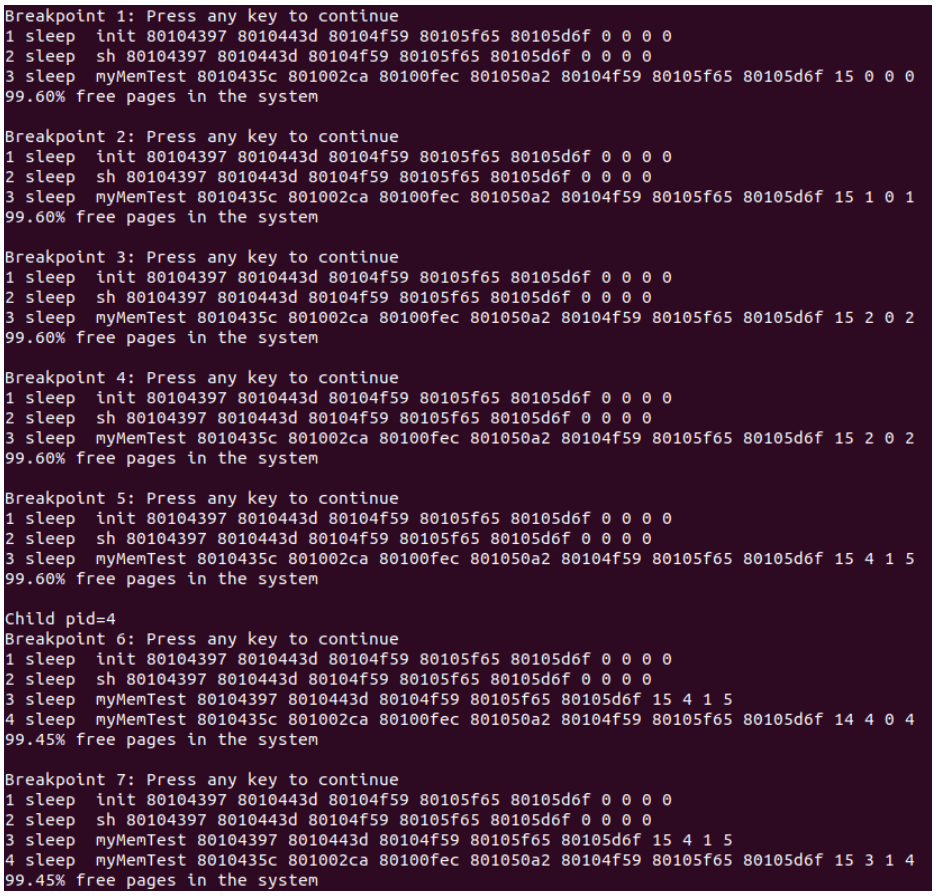
FIFO



```
Breakpoint 1: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 15 0 0 0
99.60% free pages in the system

Breakpoint 2: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 15 1 0 1
99.60% free pages in the system

Breakpoint 3: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 15 2 0 2
99.60% free pages in the system

Breakpoint 4: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 15 2 1 3
99.60% free pages in the system

Breakpoint 5: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 15 4 1 5
99.60% free pages in the system

Child pid=4
Breakpoint 6: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 80104397 8010443d 80104f59 80105f65 80105d6f 15 4 1 5
4 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 14 4 0 4
99.45% free pages in the system

Breakpoint 7: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 80104397 8010443d 80104f59 80105f65 80105d6f 15 4 1 5
4 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 15 3 1 4
99.45% free pages in the system
```

We pressed Ctrl+P at each breakpoint to get process memory details. The second last number for the process represents the count of page faults, while the third last is the count of pages in the swap file. As we can see from the output, it matches exactly as we had predicted in the table.

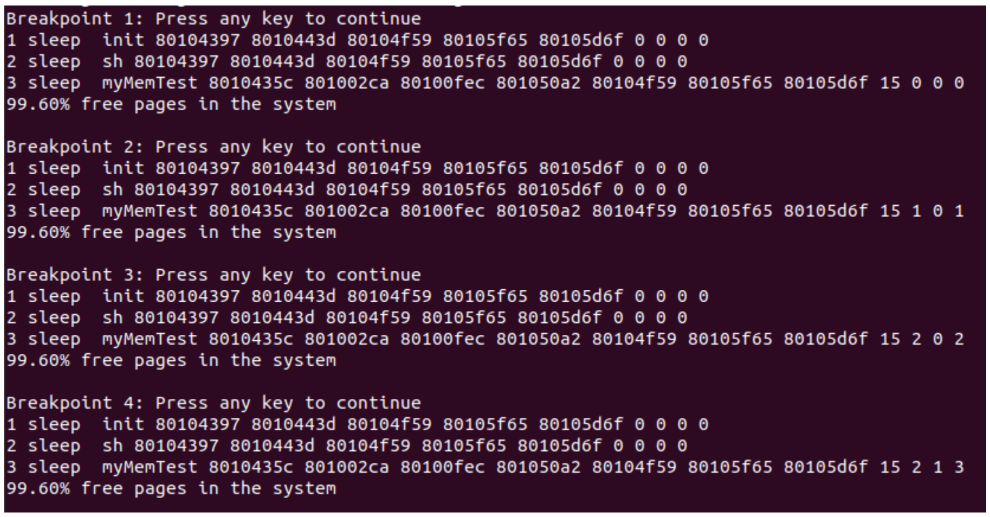
SCFIFO



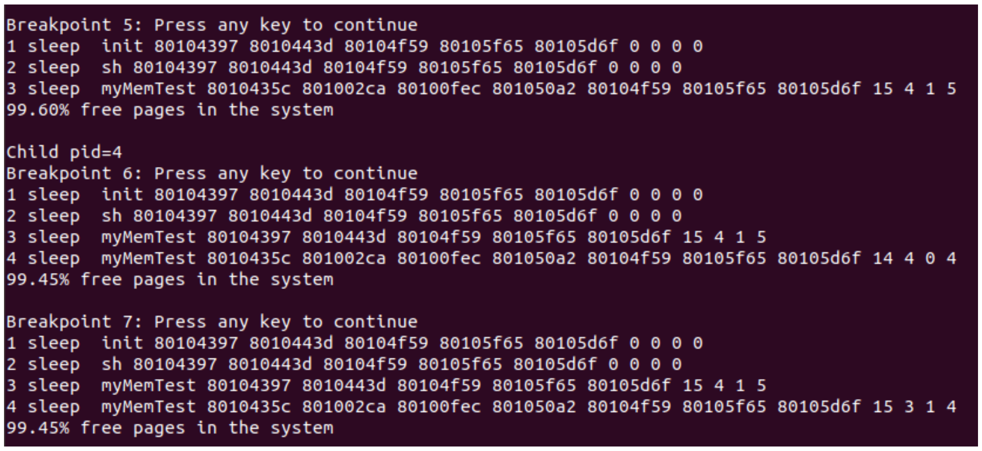
```
Breakpoint 1: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 15 0 0 0
99.60% free pages in the system

Breakpoint 2: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 15 1 0 1
99.60% free pages in the system

Breakpoint 3: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 15 2 0 2
99.60% free pages in the system

Breakpoint 4: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 15 2 0 2
99.60% free pages in the system

Breakpoint 5: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 15 4 1 5
99.60% free pages in the system

Child pid=4
Breakpoint 6: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 80104397 8010443d 80104f59 80105f65 80105d6f 15 4 1 5
4 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 14 4 0 4
99.45% free pages in the system

Breakpoint 7: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 80104397 8010443d 80104f59 80105f65 80105d6f 15 4 1 5
4 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 15 3 1 4
99.45% free pages in the system
```

Similar to the FIFO case, the output is exactly matching the expected output from the table.

NFU



```
Breakpoint 1: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 15 0 0 0
99.60% free pages in the system

Breakpoint 2: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 15 1 0 1
99.60% free pages in the system

Breakpoint 3: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 15 2 0 2
99.60% free pages in the system

Breakpoint 4: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 15 2 1 3
99.60% free pages in the system
```



```
Breakpoint 5: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 15 4 1 5
99.60% free pages in the system

Child pid=4
Breakpoint 6: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 80104397 8010443d 80104f59 80105f65 80105d6f 15 4 1 5
4 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 14 4 0 4
99.45% free pages in the system

Breakpoint 7: Press any key to continue
1 sleep  init 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
2 sleep  sh 80104397 8010443d 80104f59 80105f65 80105d6f 0 0 0 0
3 sleep  myMemTest 80104397 8010443d 80104f59 80105f65 80105d6f 15 4 1 5
4 sleep  myMemTest 8010435c 801002ca 80100fec 801050a2 80104f59 80105f65 80105d6f 15 3 1 4
99.45% free pages in the system
```

In the case of NFU, the output is now different from what is expected from the table. As already suspected, this could happen if aging makes the NFU counter zero before the page selection happens to create space for the new page. Clearly this has happened here, hence making the output same as FIFO. One possible reason for this happening could be that the breakpoint() might take a lot of time to execute which spans many timer interrupts. As we use 12 bits for the NFU counter, 12 timer interrupts are enough to make it zero. This must be happening in our case. Another possible reason is that the managing functions for metadata are very slow, hence, further adding more timer interrupts for reducing NFU counter.

Comparison

From the obtained outputs, we can see that the best replacement scheme is SCFIFO. But clearly this is due to the NFU counter going to zero before selection to make for the allocation of a new page. In a well programmed operating system code for managing the metadata, the timer interrupt will not be able to make the counter zero, hence following the output as expected from the table. Hence, theoretically and on practical systems, NFU is the best replacement policy among these three (in general).
Performance of replacement schemes also heavily depends on the type of program the OS will run. But still in general, due to high temporal locality, SCFIFO and NFU perform better than FIFO. And NFU performs better than SCFIFO because it holds onto the recently used page for a longer amount of time.