Class Assignment - 6

Name - Rajeev Kumar
ID - 12341700

Part 1: Implementation of LRU Page Replacement in xv6

- **Step 1 –Modifications in `proc.h`**

  File Name: `proc.h`
  Code Added: New `frameinfo` structure

```
struct frameinfo {
    uint va;         // Virtual address of page
    pte_t *pte;      // Page table entry
    int ref;         // Reference (optional for CLOCK)
    uint last_used;  // Time of last access (for LRU)
};
```

  Added fields in `struct proc :`

```
struct frameinfo frames[16];
int framecount;
```

- **Step 2 – Modifications in `vm.c`**

  File Name: `vm.c`
  Code Added:

```
extern uint ticks; // Global tick counter from trap.c

void update_lru_access(struct proc *p, uint va) {
  for (int i = 0; i < p->framecount; i++) {
    if (p->frames[i].va == va) {
      p->frames[i].last_used = ticks; // update on access
      break;
    }
```

```
    }
}
```

Replacing existing `allocuvm()` from new function :

```
int allocuvm(pde_t *pgdir, uint oldsz, uint newsz) {
    char *mem;
    uint a;

    if (newsz >= KERNBASE) return 0;
    if (newsz < oldsz) return oldsz;

    a = PGROUNDUP(oldsz);
    for (; a < newsz; a += PGSIZE) {
        mem = kalloc();
        if (mem == 0) {
            cprintf("allocuvm out of memory\n");
            deallocuvm(pgdir, newsz, oldsz);
            return 0;
        }
        memset(mem, 0, PGSIZE);
        if (mappages(pgdir, (char*)a, PGSIZE, V2P(mem), PTE_W | PTE_U)
< 0) {
            cprintf("allocuvm out of memory (2)\n");
            deallocuvm(pgdir, newsz, oldsz);
            kfree(mem);
            return 0;
        }

        // ------------------ LRU Update ------------------
        struct proc *curproc = myproc();
        if (curproc) {
            if (curproc->framecount < 16) {
                curproc->frames[curproc->framecount].va = a;
                curproc->frames[curproc->framecount].pte = walkpgdir(pgdir,
(void*)a, 0);
                curproc->frames[curproc->framecount].last_used = ticks;
                curproc->framecount++;
            } else {
                // Select victim using LRU
                int victim_index = 0;
```

```
                for (int i = 1; i < curproc->framecount; i++) {
                    if (curproc->frames[i].last_used <
                        curproc->frames[victim_index].last_used) {
                        victim_index = i;
                    }
                }

                // Free victim
                pte_t *vpte = curproc->frames[victim_index].pte;
                uint vpa = PTE_ADDR(*vpte);
                kfree(P2V(vpa));
                *vpte = 0;

                // Replace with new
                curproc->frames[victim_index].va = a;
                curproc->frames[victim_index].pte = walkpgdir(pgdir, (void*)a,
0);
                curproc->frames[victim_index].last_used = ticks;
            }
        }
    }
    return newsz;
}
```

- **Step 3 : User Program `mytest.c`**

  File Name: `mytest.c`

  Code Added:

```
#include "types.h"

#include "user.h"

#define MAX_PAGES 5

#define TOTAL_ACCESSES 15


int main(int argc, char *argv[]) {
```

```
    int lru[MAX_PAGES];
    int last_used[MAX_PAGES];
    int i, j, page, hit = 0, miss = 0, time = 0;
    for (i = 0; i < MAX_PAGES; i++) {
        lru[i] = -1;
        last_used[i] = -1;
    }
    printf(1, "Starting LRU page replacement simulation...\n");
    int accesses[TOTAL_ACCESSES] = {0, 1, 2, 3, 4, 1, 5, 0, 6, 1, 2, 7, 3,
8, 4};
    for (i = 0; i < TOTAL_ACCESSES; i++) {
        page = accesses[i];
        time++;
        int found = 0;
        for (j = 0; j < MAX_PAGES; j++) {
            if (lru[j] == page) {
                found = 1;
                last_used[j] = time;
                break;
            }
        }
        if (found) {
            hit++;
            printf(1, "Access page %d: HIT\n", page);
        } else {
            miss++;
```

```
            int replaced = -1;
            for (j = 0; j < MAX_PAGES; j++) {
                if (lru[j] == -1) {
                    lru[j] = page;
                    last_used[j] = time;
                    replaced = j;
                    break;
                }
            }
        if (replaced == -1) {
            int lru_index = 0, min_time = last_used[0];
            for (j = 1; j < MAX_PAGES; j++) {
                if (last_used[j] < min_time) {
                    min_time = last_used[j];
                    lru_index = j;
                }
            }
            printf(1, "Access page %d: MISS, replacing page %d\n",
                    page, lru[lru_index]);
            lru[lru_index] = page;
            last_used[lru_index] = time;
        } else {
            printf(1, "Access page %d: MISS, placed in free frame\n", page);
        }
    }
```

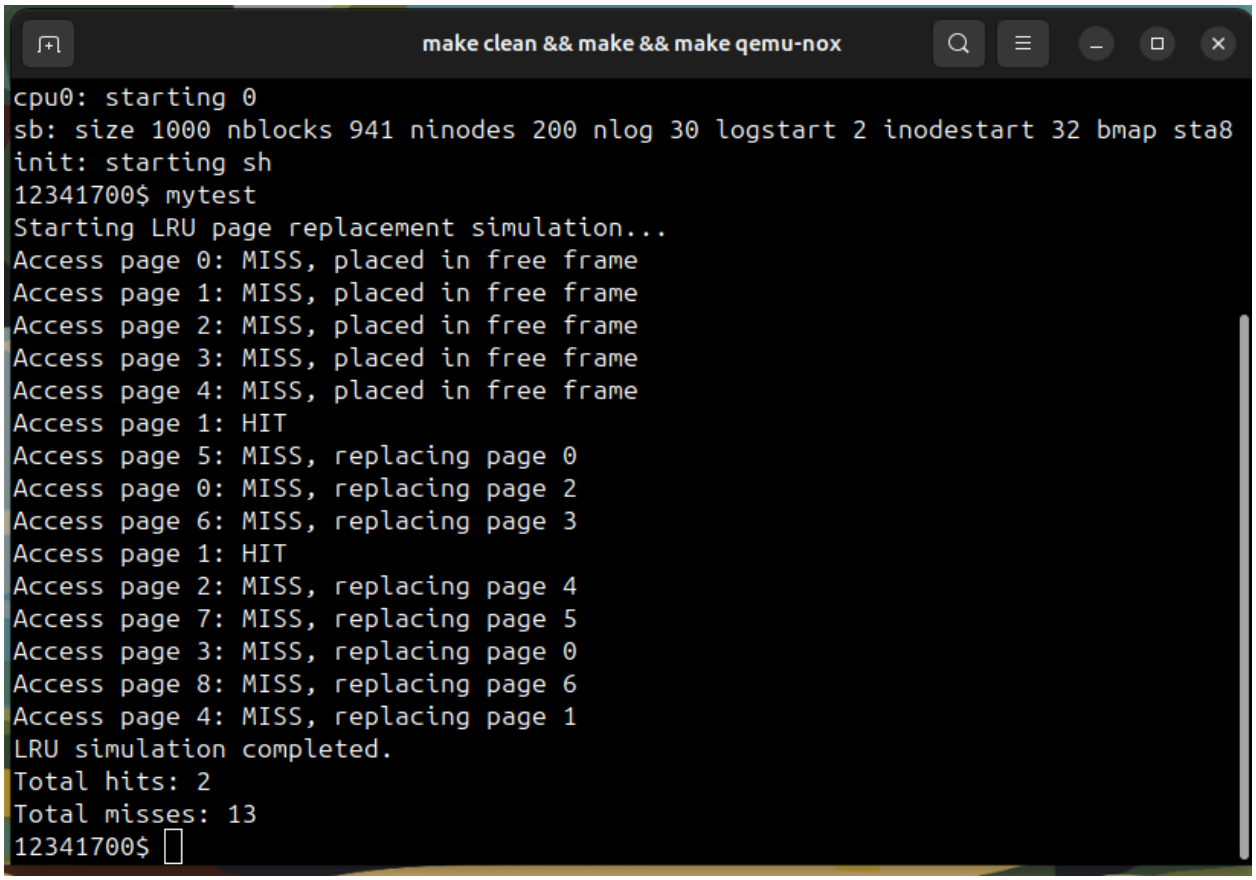
```
    }
    printf(1, "LRU simulation completed.\n");
    printf(1, "Total hits: %d\n", hit);
    printf(1, "Total misses: %d\n", miss);
        exit();
    }
```

- **Step 4 : Makefile Update**

  File Name: Makefile
  Code Added:

```
_mytest\
```

- **OUTPUT :**



Explaination : The output shows the working of the LRU (Least Recently Used) page replacement algorithm.

- At first, the free frames are filled with pages 0–4, resulting in misses.
- After the frames are full, every new page request may cause either a hit (if the page is already in memory) or a miss (if it is not).
- On a miss, the least recently used page is replaced.
- In this simulation, there were 2 hits and 13 misses in total.

This demonstrates how LRU tries to keep the most recently accessed pages in memory while replacing the least used ones when new pages arrive.

Part 2: FIFO Page Replacement Practical in xv6-public

- **Step 1 –Modifications in `proc.h`**

  File Name: `proc.h`
  Code Added:

```
#define MAX_PAGES 20

struct proc {

int pages[MAX_PAGES];

int page_count;

};
```

  Explaination :

- **Step 2 – Modifications in `vm.c`**

  File Name: `vm.c`
  Code Added:

```
int allocuvm(pde_t *pgdir, uint oldsz, uint newsz) {
  char *mem;
  uint a;

  if(newsz >= KERNBASE)
    return 0;
  if(newsz < oldsz)
    return oldsz;

  a = PGROUNDUP(oldsz);
  for(; a < newsz; a += PGSIZE) {
    mem = kalloc();
    if(mem == 0) {
      cprintf("allocuvm out of memory\n");
      deallocuvm(pgdir, newsz, oldsz);
      return 0;
    }
    memset(mem, 0, PGSIZE);
```

```
    if(mappages(pgdir, (char*)a, PGSIZE, V2P(mem), PTE_W|PTE_U) < 0)
{
      kfree(mem);
      deallocuvm(pgdir, newsz, oldsz);
      return 0;
    }

    struct proc *cur = myproc();

    // Case 1: Still under MAX_PAGES
    if(cur->page_count < MAX_PAGES) {
      cur->pages[cur->page_count++] = (int)a;
      cprintf("Allocated page %d at VA 0x%x\n", cur->page_count-1, a);
    }
    // Case 2: Need FIFO eviction
    else {
      int evict = cur->pages[0];
      char *victim = (char*)P2V(PTE_ADDR(*walkpgdir(pgdir, (void*)evict,
0)));
      kfree(victim);

      // Shift FIFO queue
      for(int i = 1; i < MAX_PAGES; i++)
        cur->pages[i-1] = cur->pages[i];

      cur->pages[MAX_PAGES-1] = (int)a;
      cprintf("Evicted page at VA 0x%x, allocated new page at VA 0x%x\n",
evict, a);
    }
  }
  return newsz;
}
```

- **Step 3 : User Program `mytest.c`**

  File Name: `mytest.c`

  Code Added:

```
#include "types.h"
#include "user.h"

#define FRAME_SIZE 5        // Number of frames in memory
#define TOTAL_ACCESSES 15   // Total number of page accesses

int main(int argc, char *argv[]) {
  int fifo[FRAME_SIZE];       // Stores pages in memory
  int next_to_replace = 0;    // FIFO pointer
  int hit = 0, miss = 0;

  // Page access sequence
  int accesses[TOTAL_ACCESSES] = {0,1,2,3,4,1,5,0,6,1,2,7,3,8,4};
  // Initialize memory slots
  for(int i = 0; i < FRAME_SIZE; i++)
    fifo[i] = -1;
  printf(1, "Starting FIFO Page Replacement Simulation...\n");

  for(int i = 0; i < TOTAL_ACCESSES; i++) {
    int page = accesses[i];
    int found = 0;
    // Check if page is already in memory (HIT)
    for(int j = 0; j < FRAME_SIZE; j++) {
      if(fifo[j] == page) {
        found = 1;
```

```
      break;
    }
  }

  if(found) {
    hit++;
    printf(1, "Access page %d: HIT\n", page);
  } else {
    miss++;
    printf(1, "Access page %d: MISS, replacing page %d\n",
          page, fifo[next_to_replace]);
    fifo[next_to_replace] = page;
    next_to_replace = (next_to_replace + 1) % FRAME_SIZE;
  }
 }
 printf(1, "\nFIFO Simulation Completed.\n");
 printf(1, "Total Hits: %d\n", hit);
 printf(1, "Total Misses: %d\n", miss);

 exit();
}
```

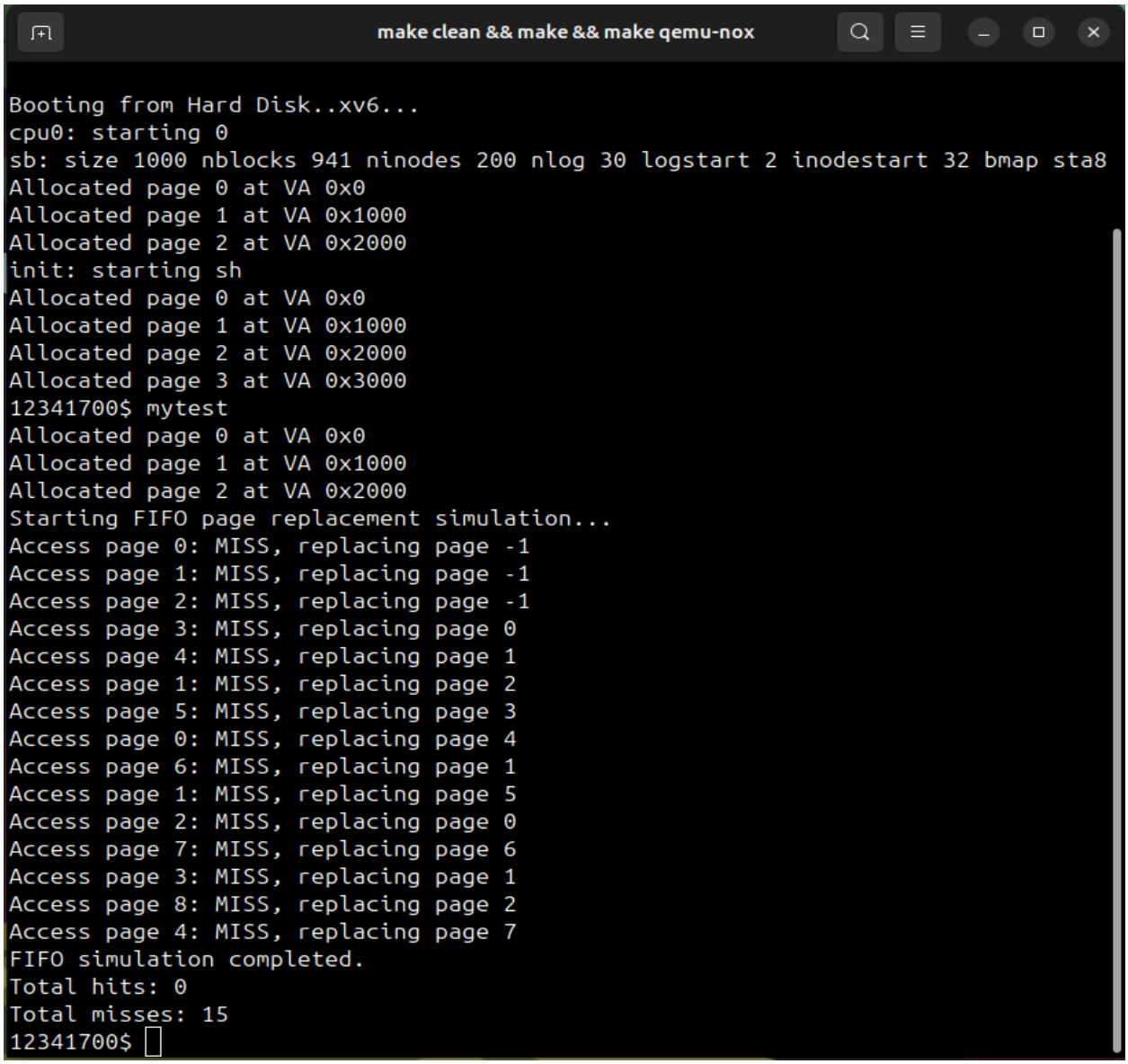
- **Step 4 : Makefile Update**

  File Name: Makefile
  Code Added:

```
_mytest\
```

OUTPUT : FIFO Page Replacement (Access sequence = {0, 1, 2, 3, 4, 1, 5, 0, 6, 1, 2, 7, 3, 8, 4})

1. **For FRAME SIZE = 3**

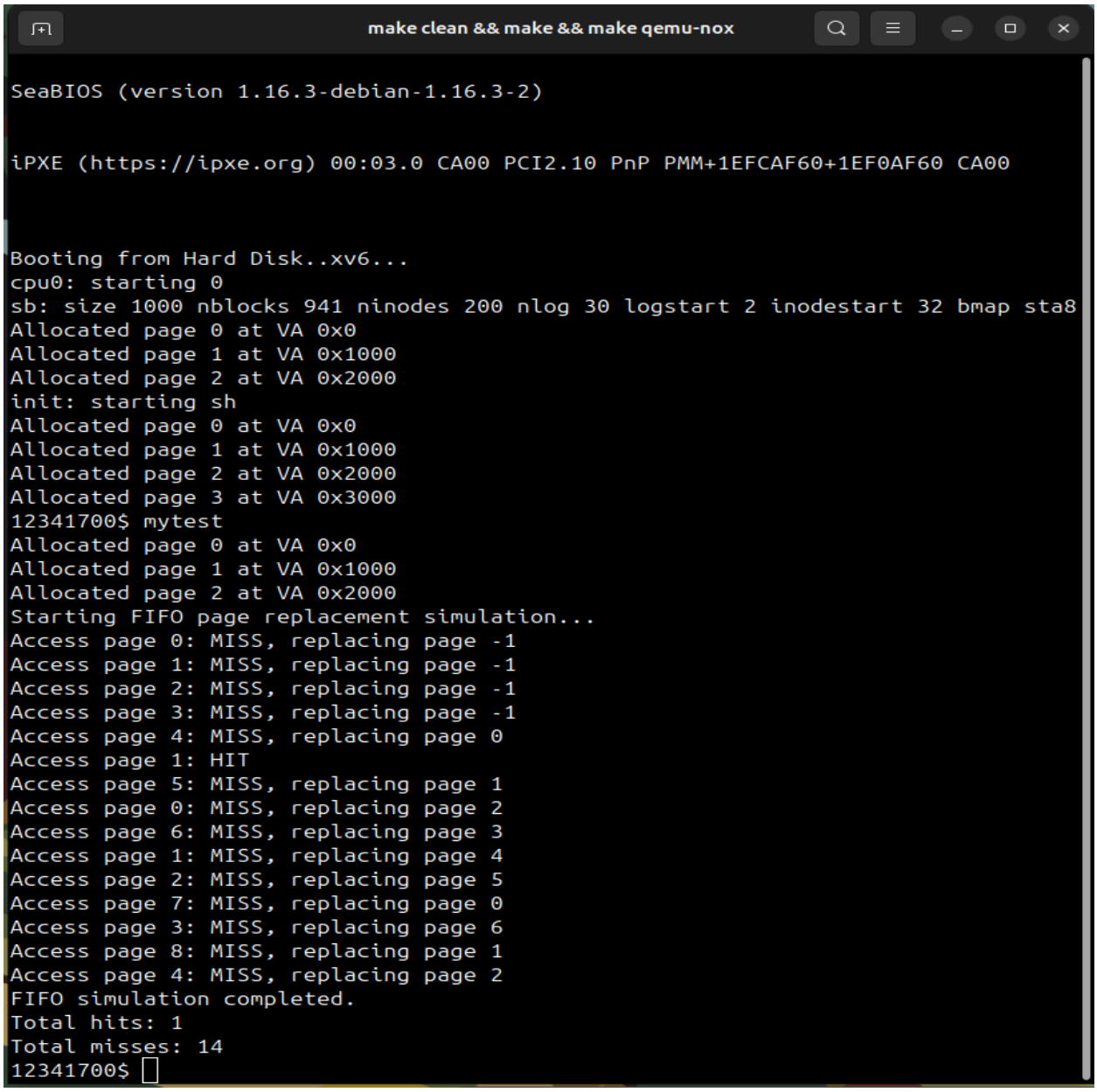


Explaination : Frames = 3

The first three accesses (0, 1, 2) fill the frames. When page 3 is accessed, page 0 (the oldest) is evicted. Next, page 4 evicts page 1. Then, page 1 evicts page 2. When page 5 arrives, page 3 is evicted. The process continues this way, always removing the oldest page.

Drawback: even pages that are still useful can be evicted just because they were loaded earlier (classic FIFO anomaly).

2. **For FRAME SIZE = 4**

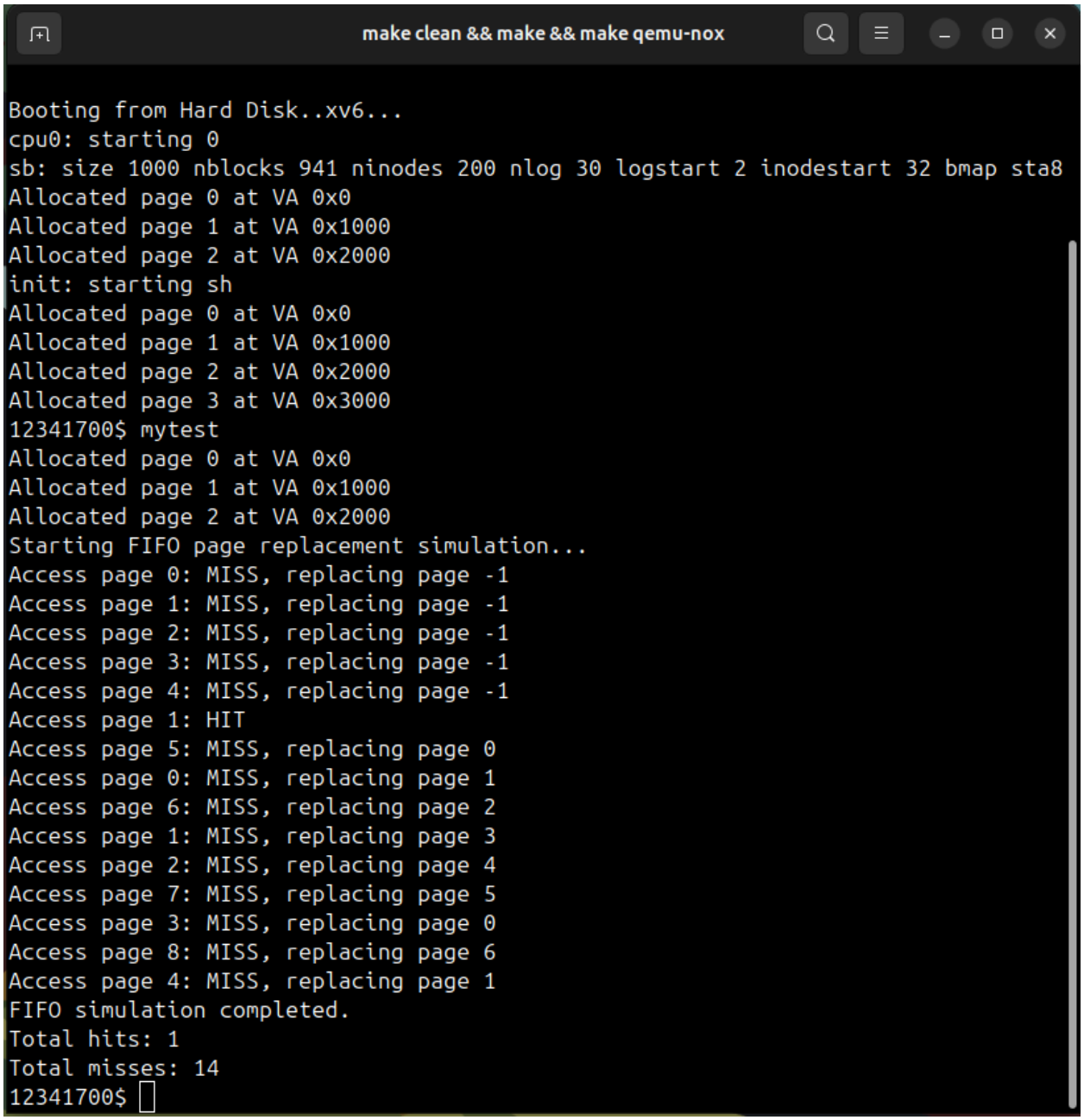


Explaination : Frames = 4

The first four accesses (0, 1, 2, 3) fill the frames. When page 4 is accessed, page 0 is evicted. The next access to page 1 is a hit because it is still in memory. When page 5 arrives, page 1 is evicted. When page 0 is accessed again, page 2 is evicted. The process continues in the same manner.

With four frames, there are fewer evictions compared to three frames. However, FIFO may sometimes show Belady's anomaly, where having more frames can unexpectedly increase the number of misses.

3. _For FRAME SIZE = 5



Explaination : Frames = 5

The first five accesses (0, 1, 2, 3, 4) fill the frames. The next access to page 1 is a hit. When page 5 arrives, page 0 is evicted. Then, page 0 comes back and evicts page 1. Later, page 6 arrives and evicts page 2. The process continues.

With five frames, the number of misses is much lower compared to three or four frames, because more pages can stay resident in memory.