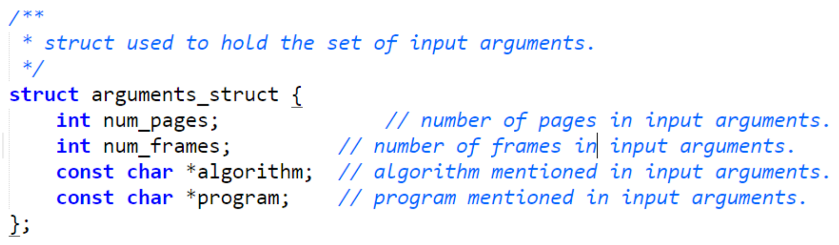
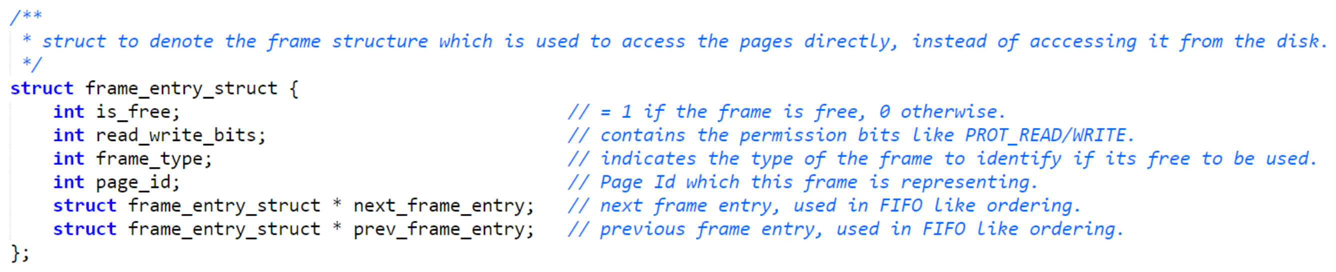
CSCI 5103 – Project 2 (Design Document)

Ravali Kandur (5084769) kandu009@umn.edu
Charandeep Parisineti(5103173) paris102@umn.edu

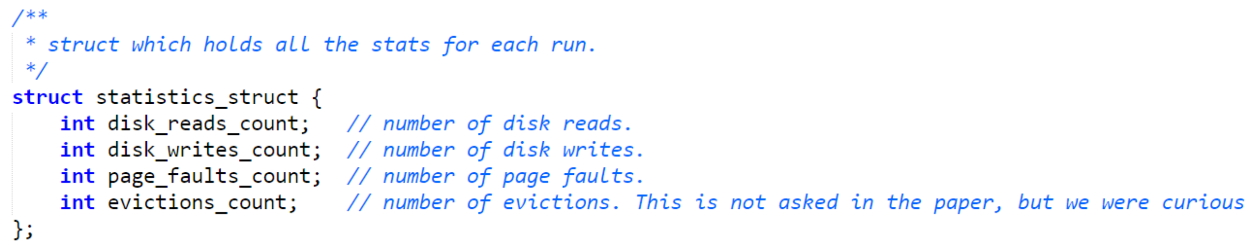
Different Structures Used:



```
/**
 * struct used to hold the set of input arguments.
 */
struct arguments_struct {
    int num_pages;              // number of pages in input arguments.
    int num_frames;         // number of frames in input arguments.
    const char *algorithm;  // algorithm mentioned in input arguments.
    const char *program;    // program mentioned in input arguments.
};
```



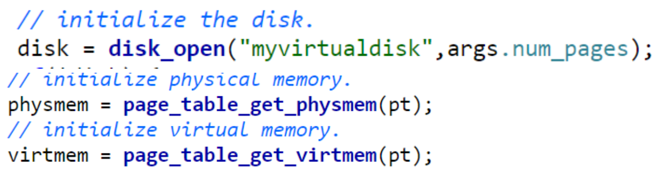
```
/**
 * struct to denote the frame structure which is used to access the pages directly, instead of acccessing it from the disk.
 */
struct frame_entry_struct {
    int is_free;                                    // = 1 if the frame is free, 0 otherwise.
    int read_write_bits;                            // contains the permission bits like PROT_READ/WRITE.
    int frame_type;                                 // indicates the type of the frame to identify if its free to be used.
    int page_id;                                    // Page Id which this frame is representing.
    struct frame_entry_struct * next_frame_entry;   // next frame entry, used in FIFO like ordering.
    struct frame_entry_struct * prev_frame_entry;   // previous frame entry, used in FIFO like ordering.
};
```



```
/**
 * struct which holds all the stats for each run.
 */
struct statistics_struct {
    int disk_reads_count;   // number of disk reads.
    int disk_writes_count;  // number of disk writes.
    int page_faults_count;  // number of page faults.
    int evictions_count;    // number of evictions. This is not asked in the paper, but we were curious
};
```

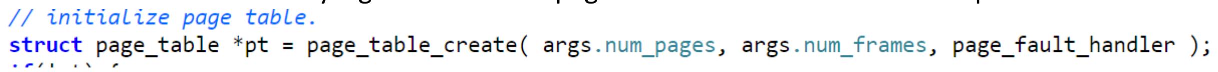
Execution Flow:

1. We initialize the basic Disk, Virtual Memory and the necessary structures shown above.



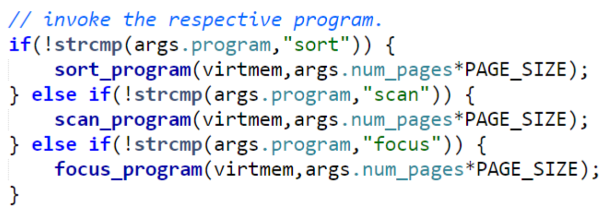
```
 // initialize the disk.
 disk = disk_open("myvirtualdisk",args.num_pages);
// initialize physical memory.
physmem = page_table_get_physmem(pt);
// initialize virtual memory.
virtmem = page_table_get_virtmem(pt);
```

2. Then associate the necessary Signal handler for page fault handlers based on the input choice.



```
// initialize page table.
struct page_table *pt = page_table_create( args.num_pages, args.num_frames, page_fault_handler );
```

3. Invoke the respective program.



```
// invoke the respective program.
if(!strcmp(args.program,"sort")) {
    sort_program(virtmem,args.num_pages*PAGE_SIZE);
} else if(!strcmp(args.program,"scan")) {
    scan_program(virtmem,args.num_pages*PAGE_SIZE);
} else if(!strcmp(args.program,"focus")) {
    focus_program(virtmem,args.num_pages*PAGE_SIZE);
}
```

4. Print Stats

CSCI 5103 – Project 2 (Design Document)

Ravali Kandur (5084769) kandu009@umn.edu
Charandeep Parisineti(5103173) paris102@umn.edu

```
        // print stats.
        print_resource_usage_stats();
```

Master Page Handler which gets invoked globally:

```
// based method which is called when a page fault occurs.
void page_fault_handler( struct page_table *pt, int page ) {
    // track the page faults.
    ++vm_stats.page_faults_count;
    switch (replacement_algo) {
        case RAND: {
            rand_pagefault_handler(pt, page);
            break;
        }
        case FIFO: {
            fifo_pagefault_handler(pt, page);
            break;
        }
        case CUSTOM: {
            custom_pagefault_handler(pt, page);
            break;
        }
        default: {
            printf("unsupported page fault #%d occurred, exiting! \n",page);
            exit(1);
        }
    }
}
```

Individual Page Handlers:

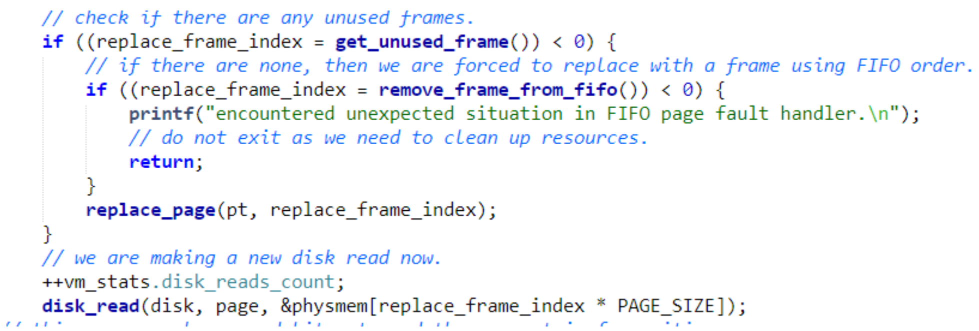
1. rand_pagefault_handler: It follows the principle of randomly choosing a frame to put the current page requested. This does not measure any heuristics which makes this really simple technique

```
// we are forced to replace with something already existing at random.
replace_frame_index = (int)lrand48() % args.num_frames;
if(frames_store[replace_frame_index].is_free != 0) {
  replace_page(pt, replace_frame_index);
}
// we are making a new disk read now.
++vm_stats.disk_reads_count;
disk_read(disk, page, &physmem[replace_frame_index * PAGE_SIZE]);
```

2. fifo_pagefault_handler: It follows FIFO order to evict the old pages and make space for new pages. We first check if there are any unused pages first, if there are, then new page goes there. If not, we replace the first in request to make space for new page.

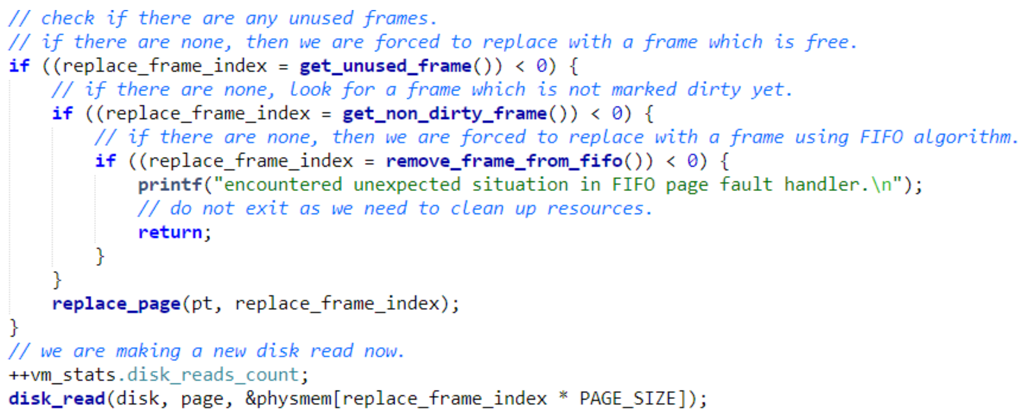
CSCI 5103 – Project 2 (Design Document)

Ravali Kandur (5084769) **kandu009@umn.edu**
Charandeep Parisineti(5103173) **paris102@umn.edu**



```
// check if there are any unused frames.
if ((replace_frame_index = get_unused_frame()) < 0) {
    // if there are none, then we are forced to replace with a frame using FIFO order.
    if ((replace_frame_index = remove_frame_from_fifo()) < 0) {
        printf("encountered unexpected situation in FIFO page fault handler.\n");
        // do not exit as we need to clean up resources.
        return;
    }
    replace_page(pt, replace_frame_index);
}
// we are making a new disk read now.
++vm_stats.disk_reads_count;
disk_read(disk, page, &physmem[replace_frame_index * PAGE_SIZE]);
```

3. Custom pagefault handler: A variant of the FIFO. The order in which the custom algorithm replaces the pages is
    a. Looks for an unused frame if there is one. If there is, we replace that frame.
    b. Looks for a frame which is not marked dirty. If there is one, then it looks for the best possible frame which can be replaced, i.e., some frame which has been written oldest, i.e., hoping that there is nothing much more to be written to it when compared to the latest ones which might have a lot of data to be written. Then we replace that selected frame.
    c. If none of the above are true, then it removes the page according to the FIFO order i..e., replace the page which has been at the head of our FIFO queue.



```
// check if there are any unused frames.
// if there are none, then we are forced to replace with a frame which is free.
if ((replace_frame_index = get_unused_frame()) < 0) {
    // if there are none, look for a frame which is not marked dirty yet.
    if ((replace_frame_index = get_non_dirty_frame()) < 0) {
        // if there are none, then we are forced to replace with a frame using FIFO algorithm.
        if ((replace_frame_index = remove_frame_from_fifo()) < 0) {
            printf("encountered unexpected situation in FIFO page fault handler.\n");
            // do not exit as we need to clean up resources.
            return;
        }
    }
    replace_page(pt, replace_frame_index);
}
// we are making a new disk read now.
++vm_stats.disk_reads_count;
disk_read(disk, page, &physmem[replace_frame_index * PAGE_SIZE]);
```