CSE 312

CSE 504 Operating Systems

#Final exam project

Jwan hussein

151044078

Page Table :

Page table entry

Belong to	counter	M	R	PA	Page frame number

Belong to field : indicate the owner of the page .

Counter : a counter used when LRU algorithm is applied

M bit : set to 1 when the page is modified

R bit : set to one when the page is referenced

PA : indicate whether the virtual page is in the memory or not .

Page frame number : indicate the physical address of the page

```
struct PAGE_ENTRY{
        int page_frame_number;
        int PA ;
        int M ;
        int R ;
        int counter ;
        char* belongto ;
};
```

Page table :

```
struct PAGE_TABLE{
        struct PAGE_ENTRY *pt ;
        int *pf ;
        int (*pagereplacementalgo)(struct PAGE_TABLE* ,int* , int , int* , char*);
        char *page_replacement_algorithm;
        int size ;
        int frame_size ;
        int diskfd ;
        int phy_size ;
        int pageTablePrintInt;
        int allocpo ;
        struct LL* head  ;
};
```

*pt : pointer to page entry array which represent the virtual memory .

*pf : array of size of the physical memory page number

(*pagereplacementalgo) : pointer to page replacement function (NRU , FIFO , SC , LRU)

char *page_replacement_algorithm : the name of the page replacement algorithm

Size : the size of the virtual memory

Frame_size : size of each page

Diskfd : the file descriptor which will represent the virtual memory

Phy_size : the physical memory size

pageTableprintint : is the interval of memory accesses after which the page table is printed on screen

allocpo : allocation policy (1 = global . 0 = local),

*head : the head of the linked list that will be used when FIFO is applied

Functions :

/* creating a new table   */

struct PAGE_TABLE create_table(int Vsize ,int physize, int fs , char* pagereplacement , char* filename , int pageTablePrintInt , char* allocpo);

/* return the page entry in index */

struct PAGE_ENTRY PT_get_at(struct PAGE_TABLE pt , int index);

/* resets all R bit of the tale, this function is called when the NRU algorithm is applied   */

void reset_R(struct PAGE_TABLE *pt);

/* checks if the virtual page pn is in the memory , return the physical address of the page or -1 if the page is not in the physical memory   */

int is_page_in_table(struct PAGE_TABLE pt , int pn);

/* free the page table */

void free_page_table(struct PAGE_TABLE* pt);

/* this function free all data related to the calling thread from the physical memory

This function used only by the fill thread , but can be used for any thread */

void _free(struct PAGE_TABLE* pt ,int *mem, char*);

/* page replacement algorithms functions , WSClock is not implemented

those functions returns the physical address of the page after replacement */

int NRU(struct PAGE_TABLE* ,int*, int , int*, char*);

int FIFO(struct PAGE_TABLE* ,int*, int ,int*, char*);

int SC(struct PAGE_TABLE* ,int*, int ,int*, char*);

int LRU(struct PAGE_TABLE* ,int*, int ,int*, char*);

/* write_to virtual function is called when a modified page replaced ,

Writes the page back to the disk file */

void write_to_vertual(int fd , int* mem ,int to,int from,int fs);

/* write_to_mem function is called when a page is needed from the virtual memory

Writes the demanded page from the disk to the physical memory */

void write_to_mem(int fd ,int *mem,int from , int to , int fs);

/* return the offset of the given address address */

int get_offset(int index);

/* return the virtual page number of the given address */

int get_Vaddr(int index);

get and set functions :

```
  int get(unsigned int index, char * tName){
        int V_ADDR = get_Vaddr(index);   /* the virtual address */
        int offset = get_offset(index);  /* the offset */
        int physicaladdr = -1;
        int is_written = 0 ;
        int page_miss = 0 ;
        int ret ;
        if(V_ADDR == -2 ){ /* in case of wrong address */
                printf("ERROR :: set() function !!\n");
                return -1;
        }
        mem_access++;          /* memory access counter  */
        if((physicaladdr = is_page_in_table(pt,V_ADDR)) != -1){   /* if the page was found in the physical memory */
                physicaladdr *= pt.frame_size ;                    /*  translating from virtual address to physical address
                physicaladdr += offset ;                           */
                pt.pt[V_ADDR].R = 1 ;                              /* set the R bit of the requested page to 1 */
                if(strcmp(pt.page_replacement_algorithm,"LRU")==0 ){   /* if the LRU algorithm was applied then increase the counter */
                        pt.pt[V_ADDR].counter += 1 ;
                }
                ret = physical_mem[physicaladdr];                  /* the requested value to be returned */
        }else{              /*  the requested page is not in the physical memory    */
                physicaladdr = pt.pagereplacementalgo(&pt,physical_mem,V_ADDR , &is_written , tName );    /* do the page replacement , is written set to 1
                                        only if the page that was replaced was written back to the virtual memory , in other world if the page M bit was 1   */
                physicaladdr *= pt.frame_size ;       /* translating from virtual address to physical address
                physicaladdr += offset ;              */
                pt.pt[V_ADDR].R = 1 ;                /*  set the R bit of the requested page to 1  */
                ret = physical_mem[physicaladdr] ; ];              /* the requested value to be returned */
                page_miss = 1 ;
        }
        if(mem_access >= pt.pageTablePrintInt){
                print_t(&pt);
                if(strcmp(pt.page_replacement_algorithm,"NRU")==0 ){     /*  resetting the R bits when mem_access == pageTablePrintInt */
                        reset_R(&pt);
                }
                mem_access = 0 ;
        }
        if(strcmp(tName,"fill")==0){
                addtostatistics(&s[0] , 1 , 0 , page_miss , page_miss , is_written , page_miss);
        }else if(strcmp(tName,"bubble")==0){
                addtostatistics(&s[1] , 1 , 0 , page_miss , page_miss , is_written , page_miss);
        }else if(strcmp(tName,"quick")==0){
                addtostatistics(&s[2] , 1 , 0 , page_miss , page_miss , is_written , page_miss);
        }else if(strcmp(tName,"merge")==0){
                addtostatistics(&s[3] , 1 , 0 , page_miss , page_miss , is_written , page_miss);
        }else if(strcmp(tName,"index")==0){
                addtostatistics(&s[4] , 1 , 0 , page_miss , page_miss , is_written , page_miss);
        }else if(strcmp(tName,"check")==0){
                addtostatistics(&s[5] , 1 , 0 , page_miss , page_miss , is_written , page_miss);
        }
        return ret ;
}
```

This lines are form the statistics information

```
void set(unsigned int index, int value, char * tName){
        int V_ADDR = get_Vaddr(index);     /* the virtual address */
        int offset = get_offset(index);    /* the offset */
        int physicaladdr = -1;
        int page_miss = 0 ;
        int is_written = 0 ;
        if(V_ADDR == -2 ){ /* in case of wrong address */
                printf("ERROR :: set() function !!\n");
                return ;
        }
        mem_access++;       /* memory access counter  */
        if((physicaladdr = is_page_in_table(pt,V_ADDR)) != -1){    /* if the page was found in the physical memory */
                physicaladdr *= pt.frame_size ;                    /*  translating from virtual address to physical address
                physicaladdr += offset ;                            */
                physical_mem[physicaladdr] = value ;             /*  set the value of the address */
                if(strcmp(pt.page_replacement_algorithm,"LRU")==0 ){  /* if the LRU algorithm was applied then increase the counter */
                        pt.pt[V_ADDR].counter += 1 ;
                }
                pt.pt[V_ADDR].M = 1 ;                             /* set the M bit of the page to 1 */
        }else{                  /*  the requested page is not in the physical memory    */
                physicaladdr = pt.pagereplacementalgo(&pt,physical_mem,V_ADDR ,&is_written , tName);     /* do the page replacement , is written set to 1
                        only if the page that was replaced was written back to the virtual memory , in other world if the page M bit was 1   */
                physicaladdr *= pt.frame_size ;                   /*  translating from virtual address to physical address
                physicaladdr += offset ;                            */
                physical_mem[physicaladdr] = value ;             /* set the value of the address */
                pt.pt[V_ADDR].M = 1 ;                             /* set the M bit of the page to 1 */
                page_miss = 1 ;
        }
        if(mem_access >= pt.pageTablePrintInt){
                print_t(&pt);
                if(strcmp(pt.page_replacement_algorithm,"NRU")==0 ){
                        reset_R(&pt);
                }
                mem_access = 0 ;
        }
        if(strcmp(tName,"fill")==0){
                addtostatistics(&s[0] , 0 , 1 , page_miss , page_miss , is_written , page_miss);
        }else if(strcmp(tName,"bubble")==0){
                addtostatistics(&s[1] , 0 , 1 , page_miss , page_miss , is_written , page_miss);
        }else if(strcmp(tName,"quick")==0){
                addtostatistics(&s[2] , 0 , 1 , page_miss , page_miss , is_written , page_miss);
        }else if(strcmp(tName,"merge")==0){
                addtostatistics(&s[3] , 0 , 1 , page_miss , page_miss , is_written , page_miss);
        }else if(strcmp(tName,"index")==0){
                addtostatistics(&s[4] , 0 , 1 , page_miss , page_miss , is_written , page_miss);
        }else if(strcmp(tName,"check")==0){
                addtostatistics(&s[5] , 0 , 1 , page_miss , page_miss , is_written , page_miss);
        }

}
```

This lines are form the statistics information

Page replacement functions :

int NRU(struct PAGE_TABLE* ,int*, int , int*, char*);

/* the NRU algorithms searches the table for the lowest class page and replace it if the page that is being replaced was modified then the page is written back to disk */

int FIFO(struct PAGE_TABLE* ,int*, int ,int*, char*);

/* the fifo algorithm uses a linked list to keep tracking the pages ,
it replaces the page from the head of the list when page replacement is needed
if the page that is being replaced was modified then the page is written back to disk */

int SC(struct PAGE_TABLE* ,int*, int ,int*, char*);

/* the same as FIFO algorithms with one different , if the page in the head of the list is referenced then the page is being appended to the tail of the list and resetting the R bit to 0 */

int LRU(struct PAGE_TABLE* ,int*, int ,int*, char*);

/* the LRU algorithm is the only algorithm that uses the counter and the belongto fields of the page entry if the allocation policy was global then the algorithm search for the page with the smallest counter to replace it ,

if the allocation policy was local then it searches the table for the page that has the smallest counter value and the page is belong to the thread that is calling the page replacement algorithm */

/* when the LRU algorithm is applied then I treat each thread as a separate process */

/* because as i have understood, with the local allocation policy when a page replacement is needed then the page to be replaced must be in use by the same process that calls the page replacement algorithm*/