REPORT

Assignment 3 : Memory managment and introduction to multithreading

Shashwat Johri
12041380

PART A-

The main parts of the code are:
making the paths to proc/pid/pagemap and proc/pid/maps

```
  if(!memcmp(argv[1],"self",sizeof("self"))){
  //check if the entered argument for pid is self
     sprintf(path_pm, "/proc/self/pagemap");
     sprintf(path_m, "/proc/self/maps");
     //contruct the paths
     pid = -1;
  }
  else{
        pid = strtol(argv[1],&end, 10);
        if (end == argv[1] || *end != '\0' || pid<=0){
           printf("PID must be a positive number or 'self'\n");
           return -1;
           }
      }

      if(pid!=-1){
        sprintf(path_pm, "/proc/%u/pagemap", pid);
        sprintf(path_m, "/proc/%u/maps", pid);
      }
```

.

Iterating throught the lines of proc/pid/maps and cleaning it up so that only the starting address can be extracted, we will use this to convert to hex as virt_addr as seen below

```
virt_addr = strtol(file_contents, NULL, 16);
//convert it to hex
read_pagemap(path_pm,virt_addr);
```

After this the finction read_pagemap is called which is responsible for finding the correspoing lines in pagemap to this virt_addr

Inside readpagemap()-

```
   //virtual address is divided by page size to get the page number in
virt_addr and
   //multiplied by size of one entry in page to essentially get the loc
ation in pagemap
   pagemap_loc = virt_addr / getpagesize() * 8;
   status = fseek(f, pagemap_loc, SEEK_SET);
```

after finding the location we seek to that place using fseek

we read the next 8 bytes store it in read_val

```
unsigned char c_buf[8];
for(i=0; i < 8; i++){
    c = getc(f);
    if(c==EOF){
        printf("\nReached end of the file\n");
        return 0;
    }
    if(is_bigendian())
         c_buf[i] = c;
    else
         c_buf[8 - i - 1] = c;
}
for(i=0; i < 8; i++){
    read_val = (read_val << 8) + c_buf[i];
}
```

We check the 63rd bit to check if the pfn exists and then print the PFN\

```
 if(read_val & ((uint64_t)1<<63)) >> 63)
    //if the 64rd bit is 1 then pfn exixts
    printf("Vaddr: 0x%lx, PFN: 0x%llx\n" ,virt_addr,(unsigned long lo
) (read_val & 0x7FFFFFFFFFFFFF)); //0-55 bits are PFN
 else
 //else page is not present
    printf("Vaddr: 0x%lx, Page not present\n",virt_addr);
```

.

PART B-

Part b contains more or less the same logic as in part b but instead of iterating through proc/pid/maps for starting address , the starting address virt_Addr is given to us as an argument

```
     }


     virt_addr=strtol(argv[2],NULL,16);
convert the virt addrr given to hex and send it to find the correspoi
 pagemap below
      read_pagemap(path_pm,virt_addr);

 return 0;
```

inside read_pagemap, some additional checks on the PTE is done, to reveal all the auxillary info encoded in the 64-bit PTE

```
   if(FETCH_BIT(read_val,63))   //if the 63rd bit is 1 then pfn exixts
      printf("Vaddr: 0x%lx, PFN: 0x%llx\n" ,virt_addr,(unsigned long lo
ng) (read_val & 0x7FFFFFFFFFFFFF)); //0-54 bits are PFN
   else//page is not present
      printf("Vaddr: 0x%lx, Page not present\n",virt_addr);
   if(FETCH_BIT(read_val,62))
   //if 62th bit is 1 then page is swapped
        {printf("BIT 62: page is swapped\n");
//if the page is swapped bits 0-5 give the swap type
        printf("swap type: 0x%llx\n",(unsigned long long)(read_val & 0x
1F));
//if the page is swapped bits 5-54 is swap offset
        printf("swap offset: 0x%llx\n",(unsigned long long)(read_val &
0x7FFFFFFFFFFE0));
}
   if(FETCH_BIT(read_val,55))
   //if bit 55 is 1 then pte is soft dirty
        printf("BIT 55 : pte is soft dirty\n");
```

PART C-

code starts with defining global variables for average , min and max as well an array of ints and the size of this array l

```
 6 // Let us create a global variable to change it in threads
 7 int average,min,max;
 8 #define INF 100000000
 9 #define  MAX_LIMIT 100
10 // The function to be executed by all threads
11
12 int mynums[MAX_LIMIT];
13 //this contains the elemets
14 int l;
15 //this is the size of the array mynums
16
```

In the main function we take the input from the user and store it in an array of ints and update the size l

```
54 int main()
55 {
56         int i;
57         pthread_t t1id,t2id,t3id;
58         printf("Enter the elements:\n");
59         int count=0;
60
61         do{
62         scanf("%d", &mynums[count++]);
63
64
65         }while(getchar() != '\n' && count < MAX_LIMIT-2);
66         mynums[count];//resize the array to count
67         //array is made
68         l=count;//update the length of the array
69         // Let us create three threads
70
```

Then we create three threads which call the functions ave0, maximum() and minimum()

```
  // Let us create three threads

  pthread_create(&t1id, NULL, ave, (void *)&t1id );
  pthread_join(t1id, NULL);
  pthread_create(&t2id, NULL, maximum, (void *)&t2id);
  pthread_join(t2id, NULL);
  pthread_create(&t3id, NULL, minimum, (void *)&t3id);
  pthread_join(t3id, NULL);
```

The functions use simple logic to calculate the average , max and min and update the global variables with the answers, also they print the answers along with thread ids .

```
void *ave(void *vargp)
{

        int *myid = (int *)vargp;

        int ctr=0;
        while(ctr<l){
        average=average+mynums[ctr];
        ctr++;
        }
        average=average/l;

        printf("The average value is %d (rounded down)and the thread id
 is %d\n",average,*myid);

        return NULL;
}
```

.

```
34 void *maximum(void *vargp)
35 {
36         int ctr=0;
37         int *myid = (int *)vargp;
38
39         max=-INF;
40         while(ctr<l){
41         int temp=mynums[ctr];
42         if (max<temp) max=temp;
43         ctr++;}
44         printf("The maximum value is %d and the thread id is %d\n",max,
       *myid);
45
46         return NULL;
```

```
49 void *minimum(void *vargp)
50 {
51         int ctr=0;
52         int *myid = (int *)vargp;
53
54         min=INF;
55         while(ctr<l){
56         int temp =mynums[ctr];
57         if (min>temp) min=temp;
58         ctr++;}
59          printf("The minimum value is %d and the thread id is %d\n",min
   ,    *myid);
60
61         return NULL;
```

Then we return back to the main where the three answers are printed.

```
90         //the three global variables average, max, min are set by the t
   hreads
91         printf("AVG:%d MIN:%d MAX:%d and the process id is :%d\n",avera
   ge,min ,max ,getpid());
92
```