Paging

This lab follows the first lab about virtual memory. We will reuse the code from the previous and complete it to implement paging.

myAlloc (Paging)

1. We use one of the allocation algorithms from the previous lab to allocate contigous words inside mem

2. For each page used, we map the page in the paging table with a free frame

3. We return the virtual address

```
address_t myAlloc(mem_t *mp, int sz)
{
    int logical_address=FirstFitAlloc(mp, sz); //use the previous lab
allocation
    int first_page=logical_address/PAGE_SIZE;
    int last_page=(logical_address+sz)/PAGE_SIZE;
    int j=0;

    for(int i=first_page;i<=last_page;i++)//for each pages
    {
        if(mp->paging_table[i]<0)//if page isn't attributed
        {
            while(ram.frame[j]!=0 && j<(SIZE/128))//find the first frame which
is free
            j++;

            if(j>=(SIZE/128))//no frame available
            {std::cout<< "No more frame availables";
            return -1;
            }
            ram.frame[j]=getpid(); //Bonus: we write process id in the ramif
we run multiple process
            mp->paging_table[i]=j;// we write the address of the frame
            std::cout<< "page number "<<i<<" allocated to frame "<< j<< "\n";
        }
    }

    return logical_address;

}
```

myFree (Paging)

When we free a space, we also need to free the paging_table and the frame associated with the virtual memory. We retrieve the pages of the adresses we allocated and free the frames associated in the mapping table. Then, we free the paging table.

```
void myFree(mem_t *mp, address_t p, int sz)
{   myContFree(mp,p,sz);
    int first_page=p/PAGE_SIZE;
    int last_page=(p+sz)/PAGE_SIZE;
    int j=0;
    for(int i=0; i< 10;i++)

    for(int i=first_page;i<=last_page;i++)//for each pages
    {

        if(mp->paging_table[i]!=(-1)){//if the page is allocated
            ram.frame[mp->paging_table[i]]=0;// we free the frames in the ram
            mp->paging_table[i]=-1;// we free the paging table
            std::cout << "page number "<<i<<" free \n";
            std::cout << "frame number "<<mp->paging_table[i]<<" free \n";
        }
    }

};
```

Read and Write

In order to write and read, we need to translate the virtual « address » into a physical one using the paging table.

We get the page number by dividing the virtual_ address by the page size.

We multiply the page number by the frame size to get the **frame address**.

We use the modulo of the address and page size to get the **frame offset**.

In myWrite(mem_t *mp, address_t virtual_address, byte_t value), we write the value of the **frame address + frame offset** element in the ram

In myRead(mem_t *mp, address_t virtual_address), we read the value of the **frame address + frame offset** element in the ram

```
// // assign a value to a byte
void myWrite(mem_t *mp, address_t p, byte_t val)
{
    address_t address_frame=mp->paging_table[p/PAGE_SIZE]* PAGE_SIZE;// we
multiply the page number by the frame size to get the frame address
    address_t offset= p%PAGE_SIZE;//we find the offset
    ram.RAM[address_frame+offset]=val;// we write the value inside the write
mempry space
};

// // read memory from a byte
byte_t myRead(mem_t *mp, address_t p)
{
```

```
        address_t address_frame = mp->paging_table[p/PAGE_SIZE]* PAGE_SIZE
;//we find the address fame in the paging table
    address_t offset = p%PAGE_SIZE;// we find the offset
    return ram.RAM[address_frame+offset];
};
```

Execution

We execute the following code :

```
int main()
{
    // initialization of the memory
    mem_t tempMem = initMem();
    mem_t *mem = &tempMem;

    // allocation of 3 addresses
    address_t adr1 = myAlloc(mem, 5);
    address_t adr2 = myAlloc(mem, 10);
    address_t adr3 = myAlloc(mem, 100);

    myFree(mem, adr1, 5);

    myWrite(mem, adr3,543);
    myWrite(mem,adr3+9, 34 );

    int val1=myRead(mem, adr3);
    int val2= myRead(mem, adr3+9);

    std::cout<< val1<< "\n";
    std::cout << val2 << "\n";

    std::cout << "\n";

}
```

Result :

```
page number 0 allocated to frame 0
page number 1 allocated to frame 1
page number 0 free
frame number 0 free

value at adr3 is 543
value at adr3+9 is 34
[1] + Done                       "/usr/bin/gdb" --interpreter
```