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);

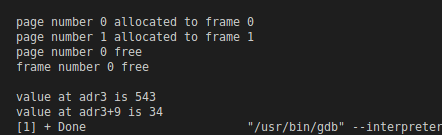
    int val2= myRead(mem, adr3+9);

    std::cout<< val1<< "\n";

    std::cout << val2 << "\n";

    std::cout << "\n";

}

Result :