Name: Ron Tal

ID: 300890191

<u>Barrier</u>

A barrier is a synchronization method.

A barrier for a group of processes in the source code means that any process must stop at this point and cannot proceed until all other processes reach this barrier.

A Barrier is an object that prevents individual tasks in a parallel operation from continuing until all tasks reach the barrier.

It is useful when a parallel operation occurs in phases, and each phase requires synchronization between tasks.

For example, if I have two processes, each one doing a task.

I need every time an iteration of a task is done, that all the data will be synchronized.

Then, each process that finishes an iteration of his task, will get to the barrier and wait until the other process will get to it - then, they synchronize their data and continue to the next iteration of the task.

<u>First I added these fields to the proc struct:</u>

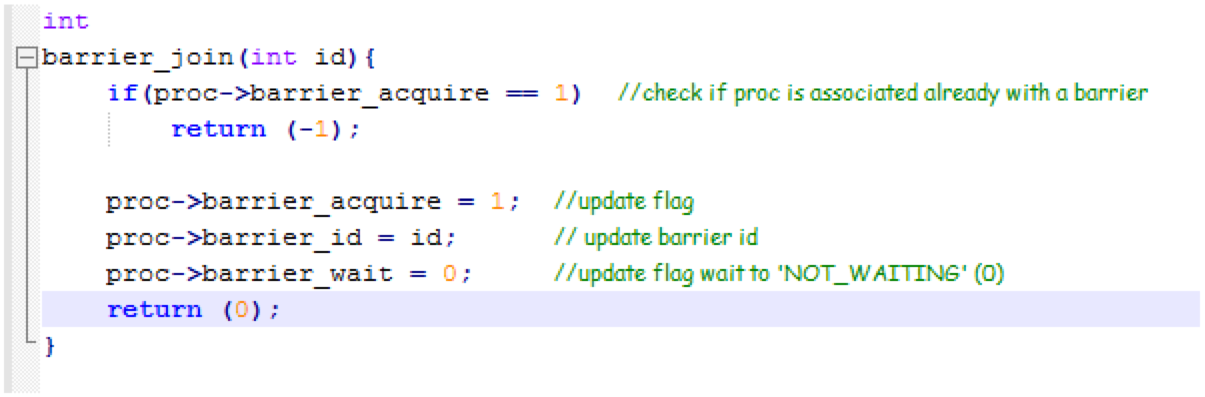
```
int barrier_acquire;        // flag that is indicating if a process is associated with any barrier
int barrier_id;             // the id of the barrier the process is associated with.
int barrier_wait;           // flag that indicates if the process is in 'wait' state in the barrier
```

Then I added the system calls according to the steps we usually do for adding system calls regularly.

(As I did in Mutex and in the previous assignment)

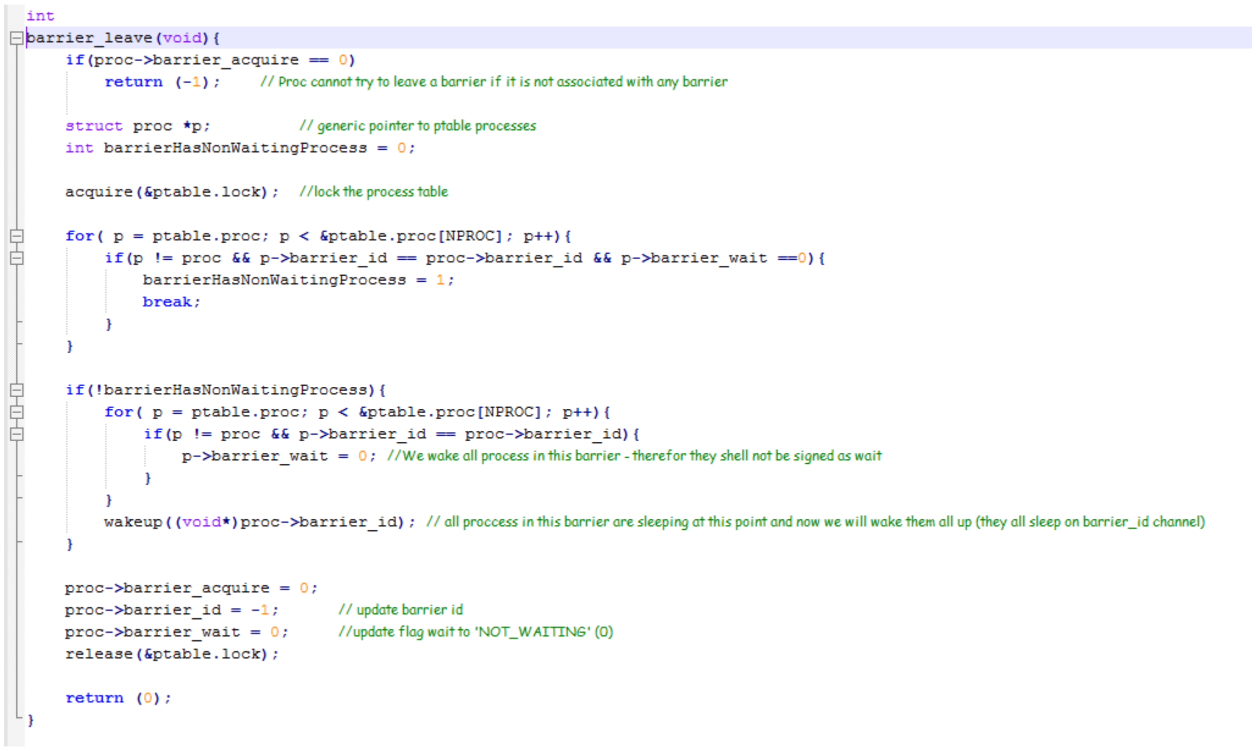
PLEASE NOTE THE COMMENTS & DOCUMENTATION I WROTE INSIDE THE CODE

Here is the implementation of ‘barrier_join’ and ‘barrier_leave’:



```c
int
barrier_join(int id){
    if(proc->barrier_acquire == 1)    //check if proc is associated already with a barrier
        return (-1);

    proc->barrier_acquire = 1;    //update flag
    proc->barrier_id = id;        // update barrier id
    proc->barrier_wait = 0;       //update flag wait to 'NOT_WAITING' (0)
    return (0);
}
```



```c
int
barrier_leave(void){
    if(proc->barrier_acquire == 0)
        return (-1);      // Proc cannot try to leave a barrier if it is not associated with any barrier

    struct proc *p;           // generic pointer to ptable processes
    int barrierHasNonWaitingProcess = 0;

    acquire(&ptable.lock);    //lock the process table

    for( p = ptable.proc; p < &ptable.proc[NPROC]; p++){
        if(p != proc && p->barrier_id == proc->barrier_id && p->barrier_wait ==0){
            barrierHasNonWaitingProcess = 1;
            break;
        }
    }

    if(!barrierHasNonWaitingProcess){
        for( p = ptable.proc; p < &ptable.proc[NPROC]; p++){
            if(p != proc && p->barrier_id == proc->barrier_id){
                p->barrier_wait = 0;  //We wake all process in this barrier - therefor they shell not be signed as wait
            }
        }
        wakeup((void*)proc->barrier_id);  // all proccess in this barrier are sleeping at this point and now we will wake them all up (they all sleep on barrier_id channel)
    }

    proc->barrier_acquire = 0;
    proc->barrier_id = -1;       // update barrier id
    proc->barrier_wait = 0;      //update flag wait to 'NOT_WAITING' (0)
    release(&ptable.lock);

    return (0);
}
```

See the implementation & documentation of ‘barrier_wait’ in proc.c, **please read the comments I wrote in the code**.

Now, let’s update Exit() and Fork() system calls to work properly with the barrier mechanism.

In exit() I added the following code:

//Update the barrier fields of the proc when the proc is exited

proc->barrier_acquire = 0;

proc->barrier_id = -1;

proc->barrier_wait = 0;

In the Fork() system call I added the following code:

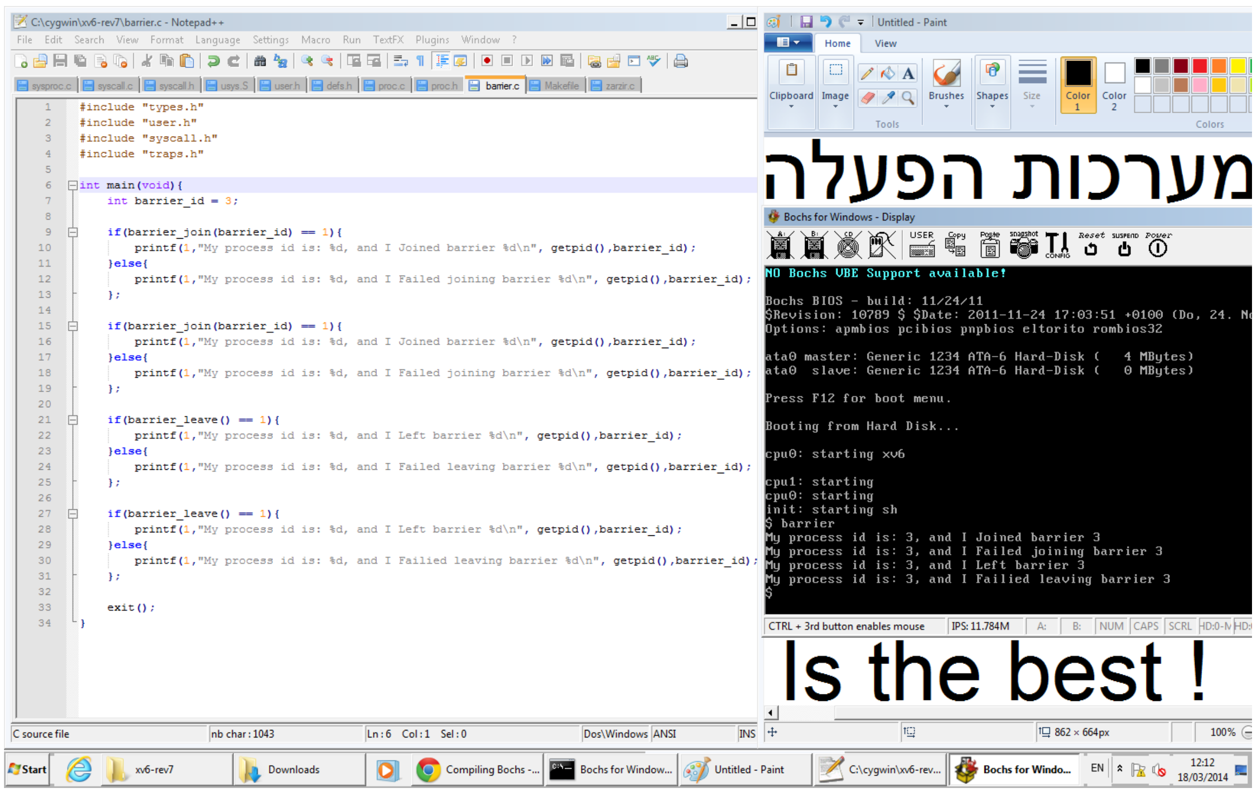
//update the barrier fields of the child, as the fields of the parent

np->barrier_acquire = proc->barrier_acquire;

np->barrier_id = proc->barrier_id;

np->barrier_wait = proc->barrier_wait;

I have added first the user program ‘barrier.c’ - this program just checks the basic functionality of ‘join’ and ‘leave’ with the error codes too.



```c
#include "types.h"
#include "user.h"
#include "syscall.h"
#include "traps.h"

int main(void){
    int barrier_id = 3;

    if(barrier_join(barrier_id) == 1){
        printf(1,"My process id is: %d, and I Joined barrier %d\n", getpid(),barrier_id);
    }else{
        printf(1,"My process id is: %d, and I Failed joining barrier %d\n", getpid(),barrier_id);
    };

    if(barrier_join(barrier_id) == 1){
        printf(1,"My process id is: %d, and I Joined barrier %d\n", getpid(),barrier_id);
    }else{
        printf(1,"My process id is: %d, and I Failed joining barrier %d\n", getpid(),barrier_id);
    };

    if(barrier_leave() == 1){
        printf(1,"My process id is: %d, and I Left barrier %d\n", getpid(),barrier_id);
    }else{
        printf(1,"My process id is: %d, and I Failed leaving barrier %d\n", getpid(),barrier_id);
    };

    if(barrier_leave() == 1){
        printf(1,"My process id is: %d, and I Left barrier %d\n", getpid(),barrier_id);
    }else{
        printf(1,"My process id is: %d, and I Failied leaving barrier %d\n", getpid(),barrier_id);
    };

    exit();
}
```

מערכות הפעלה

```
NO Bochs VBE Support available!

Bochs BIOS - build: 11/24/11
$Revision: 10789 $ $Date: 2011-11-24 17:03:51 +0100 (Do, 24. No
Options: apmbios pcibios pnpbios eltorito rombios32

ata0 master: Generic 1234 ATA-6 Hard-Disk (   4 MBytes)
ata0  slave: Generic 1234 ATA-6 Hard-Disk (   0 MBytes)

Press F12 for boot menu.

Booting from Hard Disk...

cpu0: starting xv6

cpu1: starting
cpu0: starting
init: starting sh
$ barrier
My process id is: 3, and I Joined barrier 3
My process id is: 3, and I Failed joining barrier 3
My process id is: 3, and I Left barrier 3
My process id is: 3, and I Failied leaving barrier 3
$
```

Is the best !

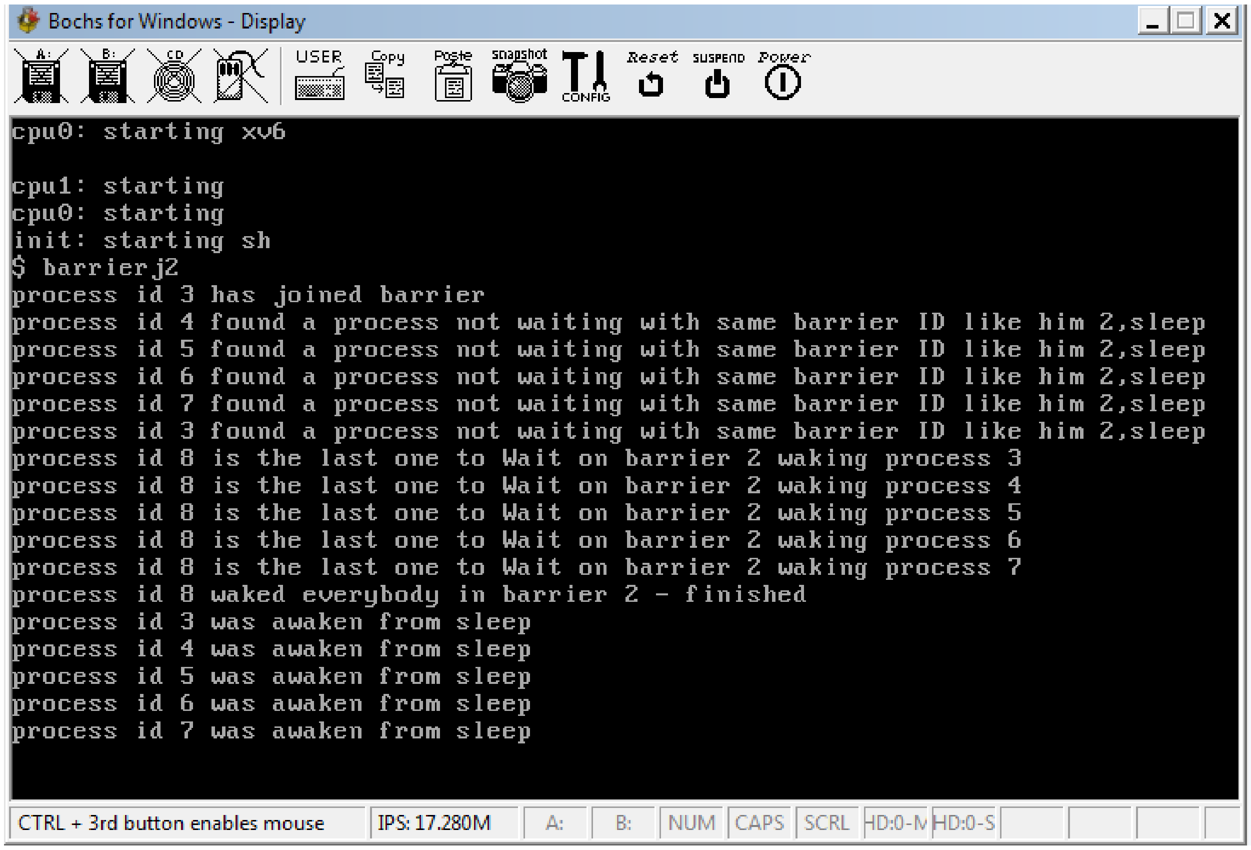
Another program ‘barrierj2.c’, test’s the barrier mechanism.

The function joins the parent process to barrier number 2.

Then the parent process fork’s 5 child processes, each child process gets the barrier data from the parent as expected.

Then, all the child’s and the parent are invoking the barrier wait.

Here is the output in bochs, used cprintf to print inside the kernel.



```
cpu0: starting xv6

cpu1: starting
cpu0: starting
init: starting sh
$ barrierj2
process id 3 has joined barrier
process id 4 found a process not waiting with same barrier ID like him 2,sleep
process id 5 found a process not waiting with same barrier ID like him 2,sleep
process id 6 found a process not waiting with same barrier ID like him 2,sleep
process id 7 found a process not waiting with same barrier ID like him 2,sleep
process id 3 found a process not waiting with same barrier ID like him 2,sleep
process id 8 is the last one to Wait on barrier 2 waking process 3
process id 8 is the last one to Wait on barrier 2 waking process 4
process id 8 is the last one to Wait on barrier 2 waking process 5
process id 8 is the last one to Wait on barrier 2 waking process 6
process id 8 is the last one to Wait on barrier 2 waking process 7
process id 8 waked everybody in barrier 2 - finished
process id 3 was awaken from sleep
process id 4 was awaken from sleep
process id 5 was awaken from sleep
process id 6 was awaken from sleep
process id 7 was awaken from sleep
```

Thats it ! :-)