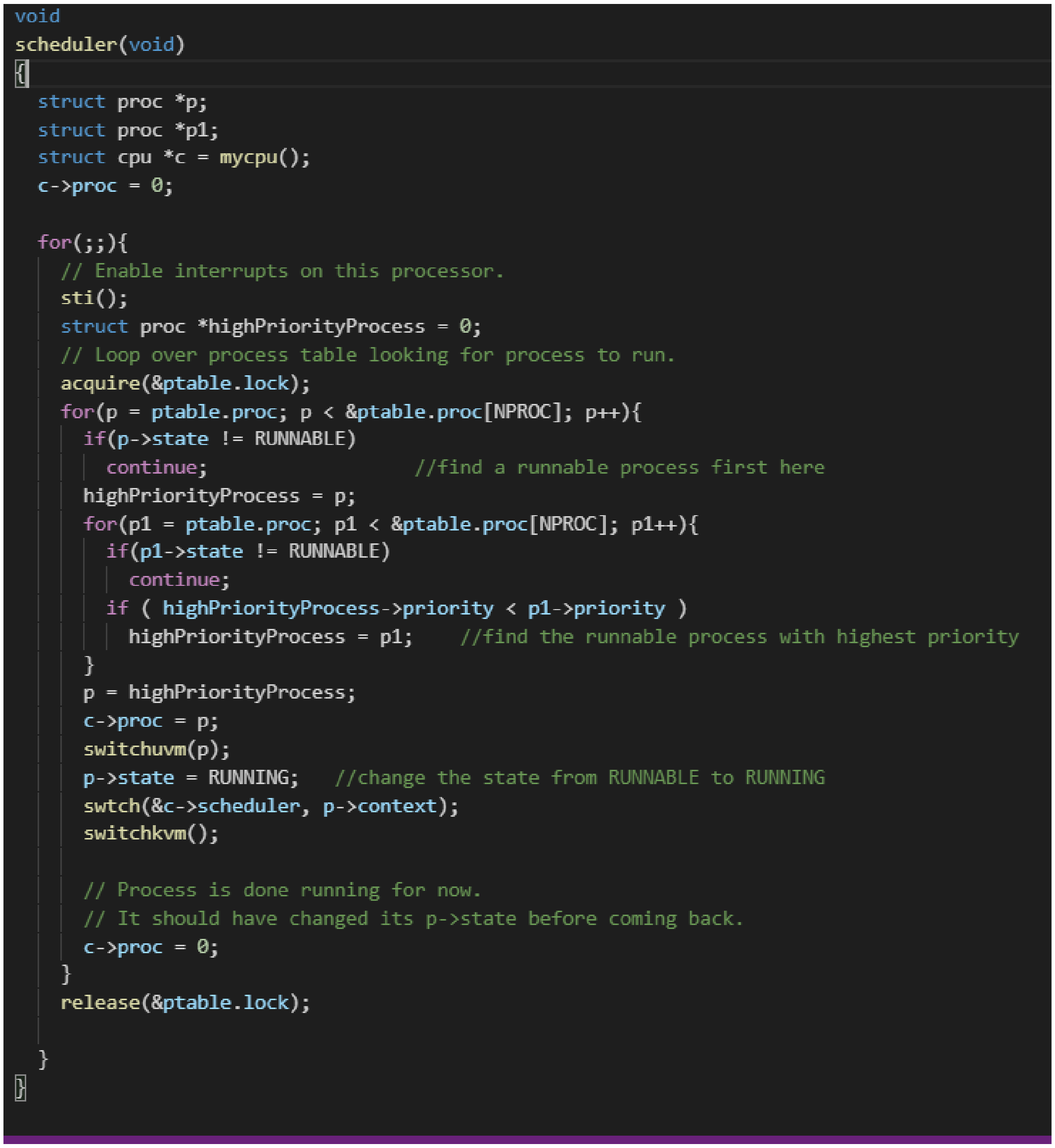
Assignment #4

OPERSTING SYSTEM

Group_04

Team members: Tejendra Khatri, Ankur Lamichhane, Sushil Pandey, Solomon Christiane

#Priority scheduling algorithm:



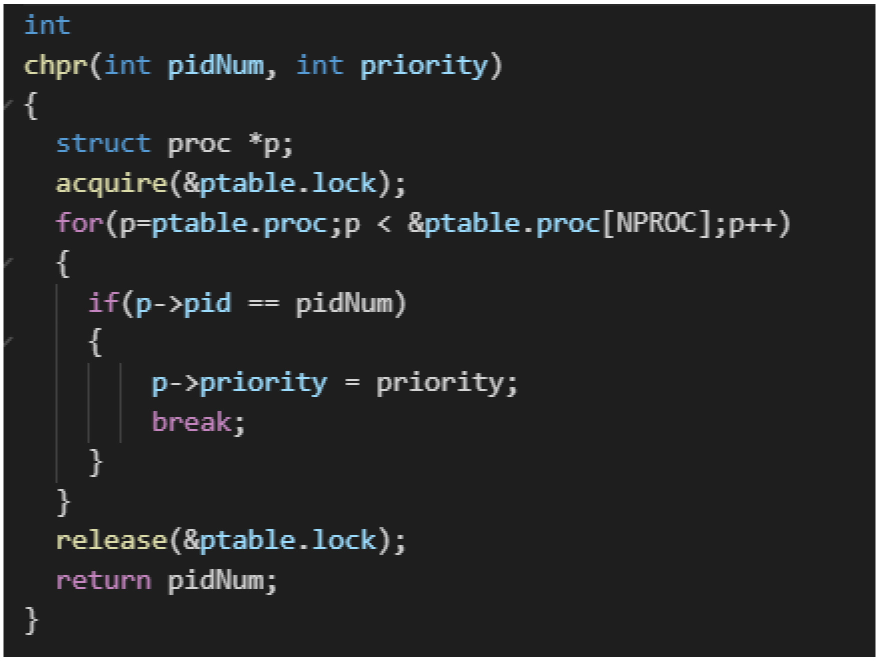
```
void
scheduler(void)
{
  struct proc *p;
  struct proc *p1;
  struct cpu *c = mycpu();
  c->proc = 0;

  for(;;){
    // Enable interrupts on this processor.
    sti();
    struct proc *highPriorityProcess = 0;
    // Loop over process table looking for process to run.
    acquire(&ptable.lock);
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      if(p->state != RUNNABLE)
        continue;                   //find a runnable process first here
      highPriorityProcess = p;
      for(p1 = ptable.proc; p1 < &ptable.proc[NPROC]; p1++){
        if(p1->state != RUNNABLE)
          continue;
        if ( highPriorityProcess->priority < p1->priority )
          highPriorityProcess = p1;    //find the runnable process with highest priority
      }
      p = highPriorityProcess;
      c->proc = p;
      switchuvm(p);
      p->state = RUNNING;   //change the state from RUNNABLE to RUNNING
      swtch(&c->scheduler, p->context);
      switchkvm();

      // Process is done running for now.
      // It should have changed its p->state before coming back.
      c->proc = 0;
    }
    release(&ptable.lock);

  }
}
```

In the schedular function above, we first assign the highest priority to 0 and inside the loop we again assign the highest priority to the first runnable process that we find and then again, we continue to check the highest priority. If the new priority is higher than previous, we replace the highest priority to the new one.

#Files that are modified and description how we modified:

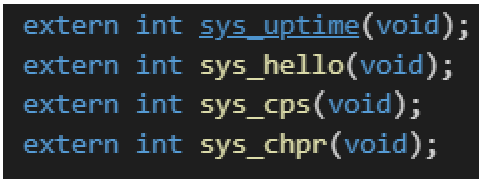
1. proc.c:

   In the proc.c, we added int chpr(int pidNum, int priority) function

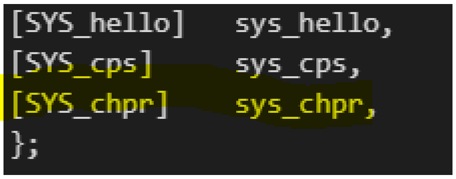


```
int
chpr(int pidNum, int priority)
{
  struct proc *p;
  acquire(&ptable.lock);
  for(p=ptable.proc;p < &ptable.proc[NPROC];p++)
  {
    if(p->pid == pidNum)
    {
      p->priority = priority;
      break;
    }
  }
  release(&ptable.lock);
  return pidNum;
}
```

2. syscall.c: Here, we just added sys_chpr as below:

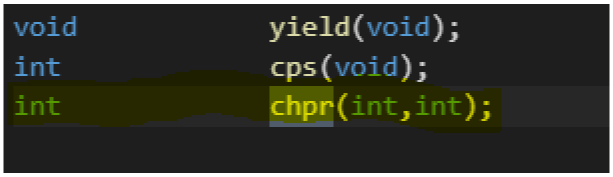


```
extern int sys_uptime(void);
extern int sys_hello(void);
extern int sys_cps(void);
extern int sys_chpr(void);
```



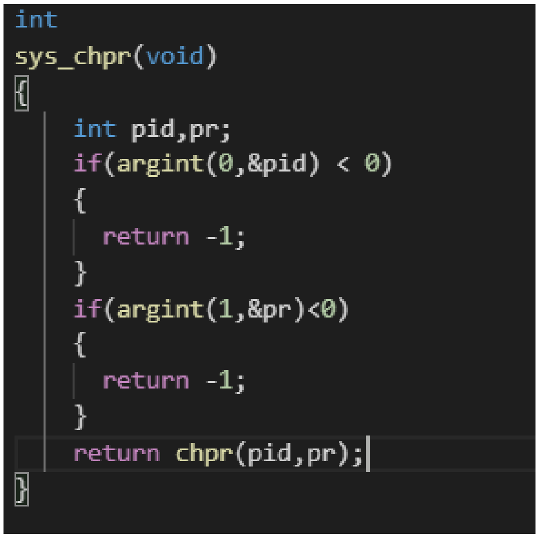
```
[SYS_hello]    sys_hello,
[SYS_cps]      sys_cps,
[SYS_chpr]     sys_chpr,
};
```

3. defs.h: Here, chpr(int, int); is added as below:



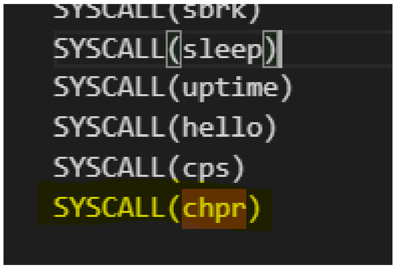
```
void                yield(void);
int                 cps(void);
int                 chpr(int,int);
```

4. sysproc.c: Here, int sys_chpr(void) is added to change the priority as mention in assignment.



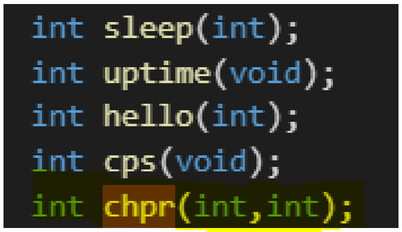
```
int
sys_chpr(void)
{
    int pid,pr;
    if(argint(0,&pid) < 0)
    {
      return -1;
    }
    if(argint(1,&pr)<0)
    {
      return -1;
    }
    return chpr(pid,pr);
}
```

5. usys.s: SYSCALL(chpr) is added.



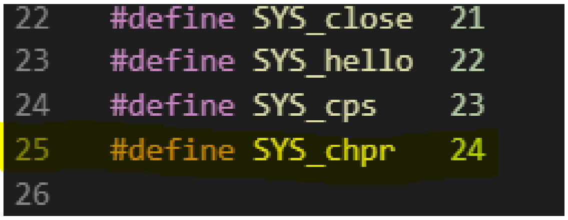
```
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(hello)
SYSCALL(cps)
SYSCALL(chpr)
```

6. user.h: int chpr(int, int) is added in this file



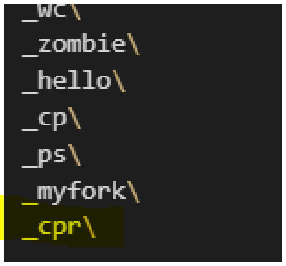
```
int sleep(int);
int uptime(void);
int hello(int);
int cps(void);
int chpr(int,int);
```

6. syscall.h: SYS_chpr is defined here:

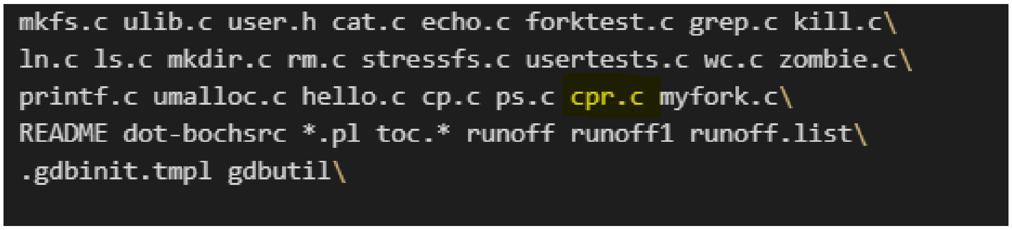


```
#define SYS_close  21
#define SYS_hello  22
#define SYS_cps    23
#define SYS_chpr   24

```

7. Makefile: _cpr\ and cpr.c are added as below:



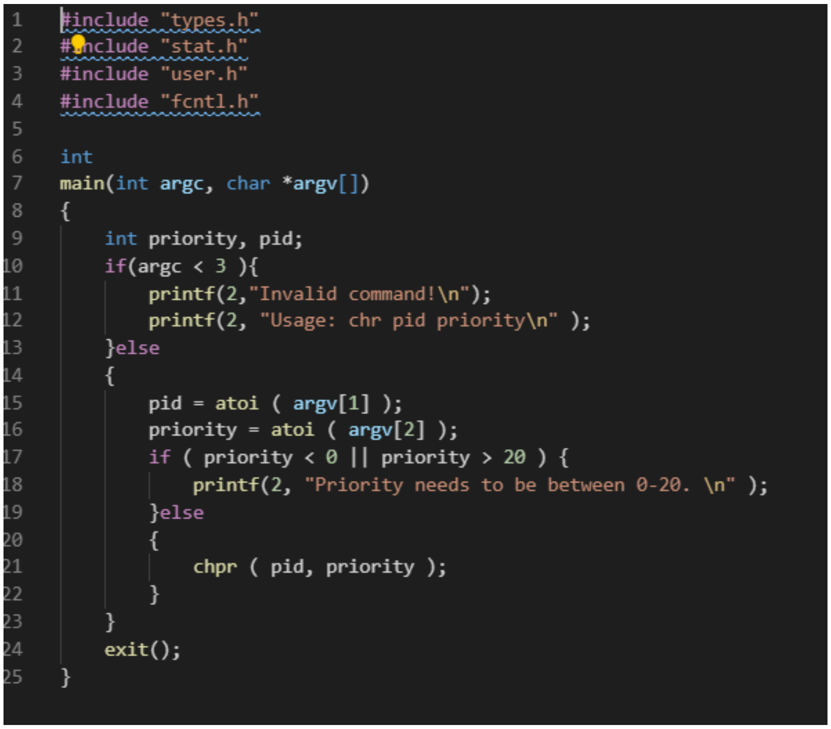
```
_wc\
_zombie\
_hello\
_cp\
_ps\
_myfork\
_cpr\
```



```
mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
printf.c umalloc.c hello.c cp.c ps.c cpr.c myfork.c\
README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
.gdbinit.tmpl gdbutil\
```

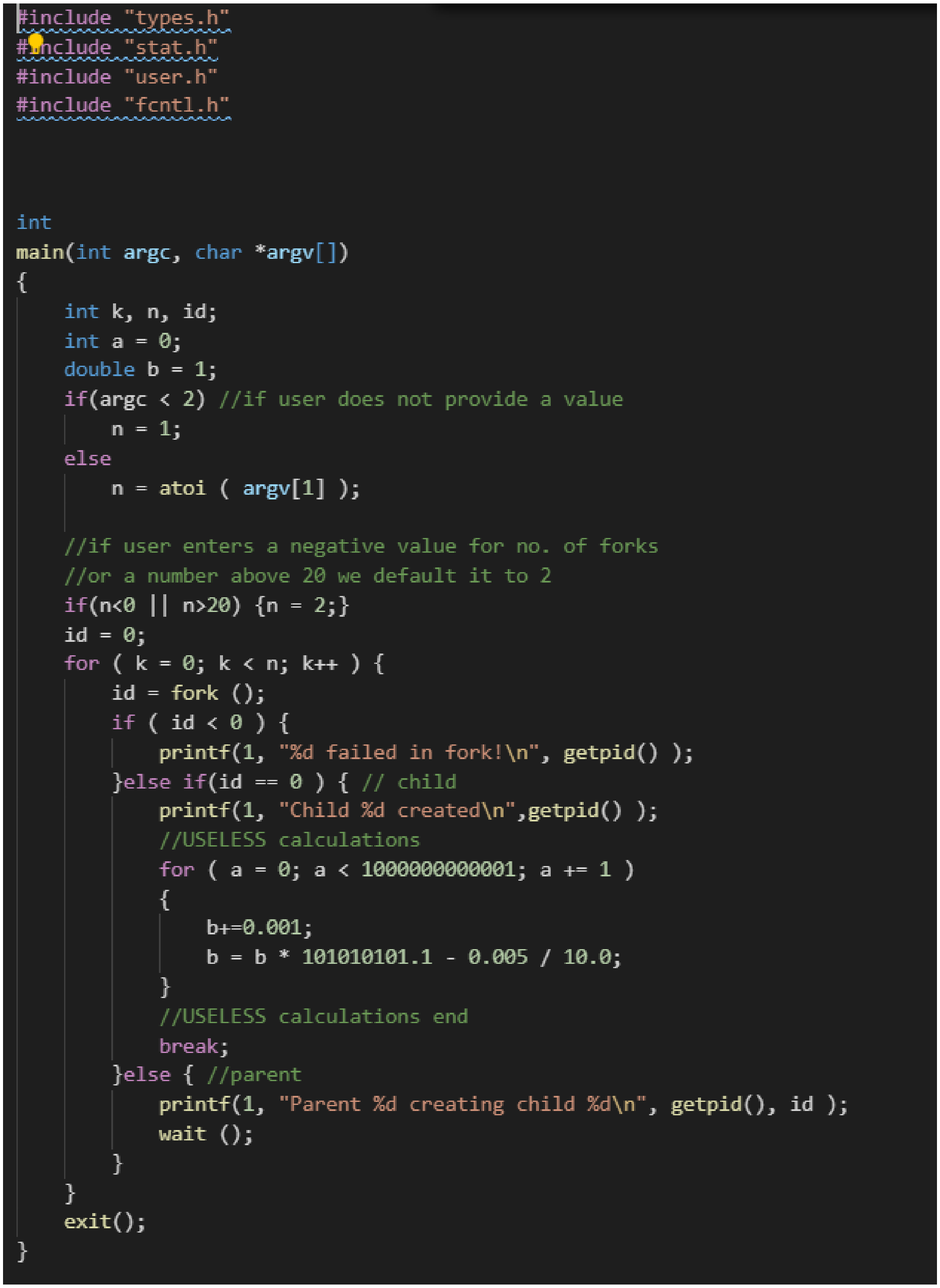
#Files that is/are created:

1.cpr.c: It calls chpr if priority is between 0 and 20



```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"

int
main(int argc, char *argv[])
{
    int priority, pid;
    if(argc < 3 ){
        printf(2,"Invalid command!\n");
        printf(2, "Usage: chr pid priority\n" );
    }else
    {
        pid = atoi ( argv[1] );
        priority = atoi ( argv[2] );
        if ( priority < 0 || priority > 20 ) {
            printf(2, "Priority needs to be between 0-20. \n" );
        }else
        {
            chpr ( pid, priority );
        }
    }
    exit();
}
```

2.myfork.c:

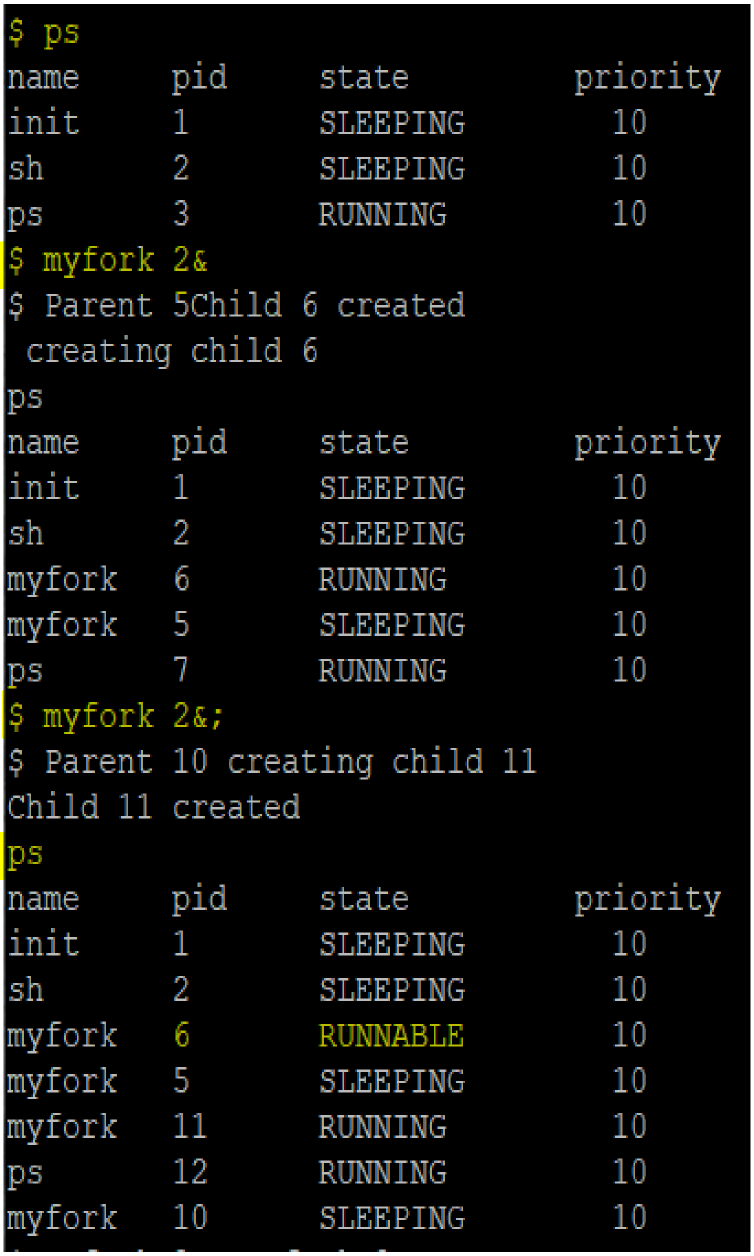


```c
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"



int
main(int argc, char *argv[])
{
    int k, n, id;
    int a = 0;
    double b = 1;
    if(argc < 2) //if user does not provide a value
        n = 1;
    else
        n = atoi ( argv[1] );

    //if user enters a negative value for no. of forks
    //or a number above 20 we default it to 2
    if(n<0 || n>20) {n = 2;}
    id = 0;
    for ( k = 0; k < n; k++ ) {
        id = fork ();
        if ( id < 0 ) {
            printf(1, "%d failed in fork!\n", getpid() );
        }else if(id == 0 ) { // child
            printf(1, "Child %d created\n",getpid() );
            //USELESS calculations
            for ( a = 0; a < 1000000000001; a += 1 )
            {
                b+=0.001;
                b = b * 101010101.1 - 0.005 / 10.0;
            }
            //USELESS calculations end
            break;
        }else { //parent
            printf(1, "Parent %d creating child %d\n", getpid(), id );
            wait ();
        }
    }
    exit();
}
```

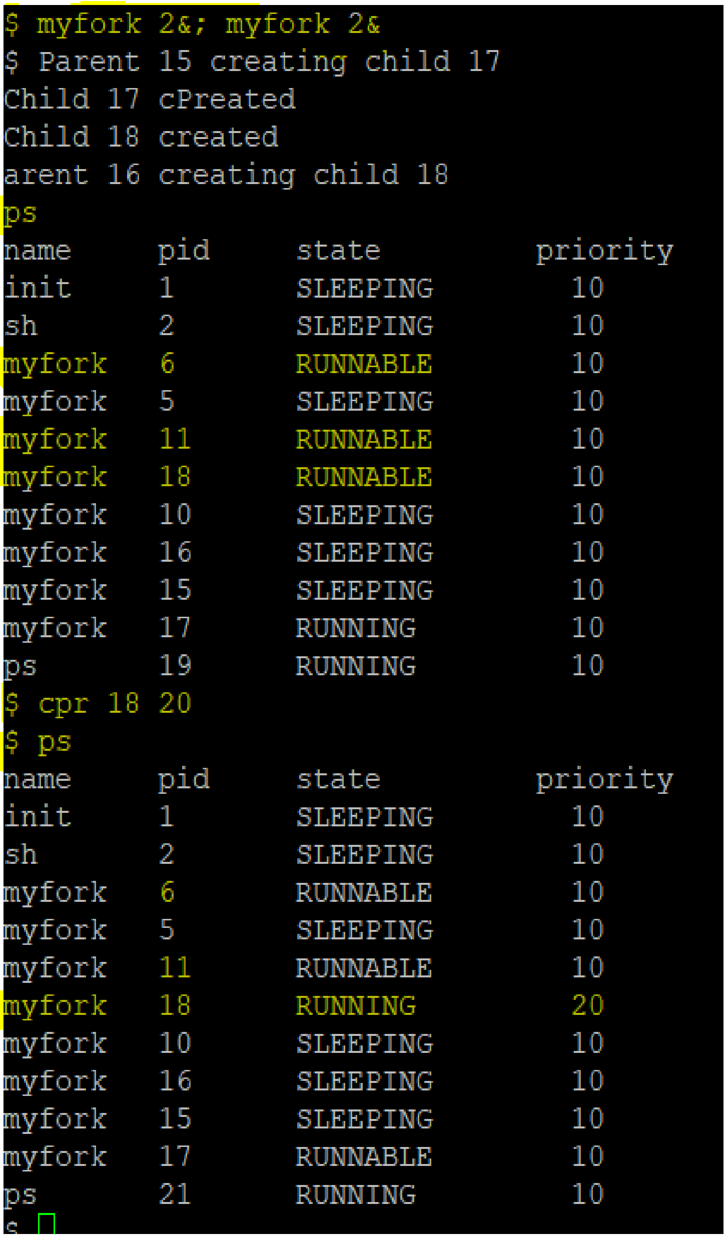
Here, myfork.c creates some child process and consumes some computing tome as per our useless calculation.

Note: In above files, we made a change as we made changes in class lab.

#Observation report:



```
$ ps
name      pid     state          priority
init      1       SLEEPING         10
sh        2       SLEEPING         10
ps        3       RUNNING          10
$ myfork 2&
$ Parent 5Child 6 created
 creating child 6
ps
name      pid     state          priority
init      1       SLEEPING         10
sh        2       SLEEPING         10
myfork    6       RUNNING          10
myfork    5       SLEEPING         10
ps        7       RUNNING          10
$ myfork 2&;
$ Parent 10 creating child 11
Child 11 created
ps
name      pid     state          priority
init      1       SLEEPING         10
sh        2       SLEEPING         10
myfork    6       RUNNABLE         10
myfork    5       SLEEPING         10
myfork    11      RUNNING          10
ps        12      RUNNING          10
myfork    10      SLEEPING         10
```



```
$ myfork 2&; myfork 2&
$ Parent 15 creating child 17
Child 17 cPreated
Child 18 created
arent 16 creating child 18
ps
name      pid     state          priority
init      1       SLEEPING         10
sh        2       SLEEPING         10
myfork    6       RUNNABLE         10
myfork    5       SLEEPING         10
myfork    11      RUNNABLE         10
myfork    18      RUNNABLE         10
myfork    10      SLEEPING         10
myfork    16      SLEEPING         10
myfork    15      SLEEPING         10
myfork    17      RUNNING          10
ps        19      RUNNING          10
$ cpr 18 20
$ ps
name      pid     state          priority
init      1       SLEEPING         10
sh        2       SLEEPING         10
myfork    6       RUNNABLE         10
myfork    5       SLEEPING         10
myfork    11      RUNNABLE         10
myfork    18      RUNNING          20
myfork    10      SLEEPING         10
myfork    16      SLEEPING         10
myfork    15      SLEEPING         10
myfork    17      RUNNABLE         10
ps        21      RUNNING          10
$
```

Here, in the observation, we can see that we change runnable process id 18 to 20 and it’s state change into running.

#Teams roles:

As in the last assignment, every of us involve in self-study and tried our best to write code with possible less error and we compared with each other and made the final best possible version. So, everybody has equal roles and responsibilities in all aspects. Also, all played significant role in making pdf and describing code.

#To compile:

1. User should navigate to the folder where the xv6 files are stored in the terminal
2. Then type “make”, again type “make qemu-nox” and type ps to see process id’s and their state with priorities.
3. Type myfork …& (fill … with integer value) to create more child and parent process
4. Then, type “cpr current_pid new_priority” (for eg: cpr 18 20 as in above screenshots) to change the priority of the runnable process.