CS344 : Assignment -1

(Kernel Threads and Synchronization)

<u>Group-M12:</u>

Kura Priyanka – 200123030
Siddam Shetty Sahithi Shresta– 200123054
Vemulapati Sai Lakshmi Swarupa– 200123070

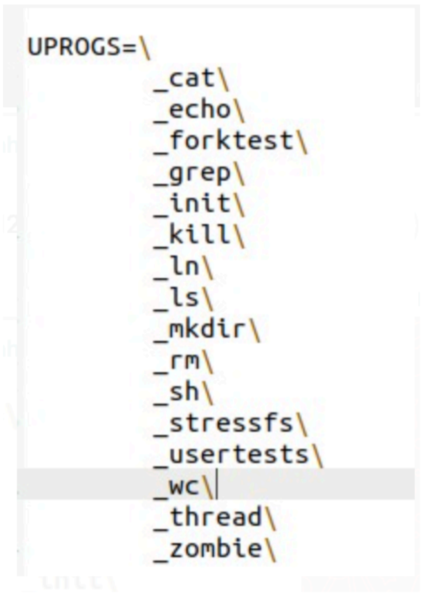
CREATED FILES:

Two new files are created two define locks *lock.h* and *lockFunc.h* .
Lock.h & LockFunc.h
In which definitions of spinlock, spin unlock, mutex lock and mutex unlock and synchronization of threads are included in these files.

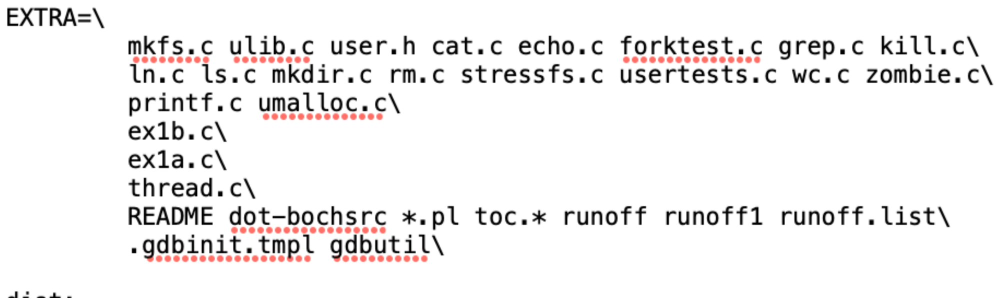
CHANGED FILES:

Makefile
The makefile had to be edited to add the new user programs to test the creation of threads and the concurrent execution of code.

Code file *thread.c* is added in *UPROGS* and in *EXTRA* section.



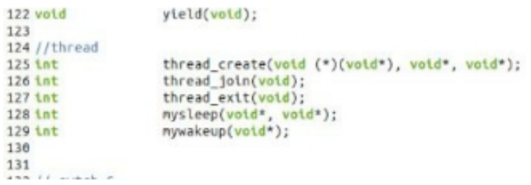
```
UPROGS=\
	_cat\
	_echo\
	_forktest\
	_grep\
	_init\
	_kill\
	_ln\
	_ls\
	_mkdir\
	_rm\
	_sh\
	_stressfs\
	_usertests\
	_wc\
	_thread\
	_zombie\
```



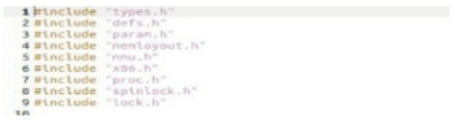
```
EXTRA=\
	mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
	ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
	printf.c umalloc.c\
	ex1b.c\
	ex1a.c\
	thread.c\
	README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
	.gdbinit.tmpl gdbutil\
```

defs.h

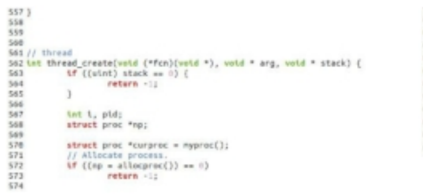
The declarations for thread create, join and exit were created in this file and the declaration for locks mysleep and mywakeup were created in this file and the declaration for locks mysleep and mywakeup were created in this file.

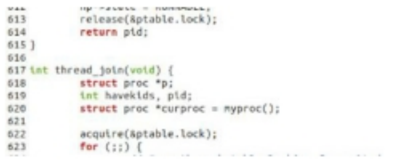

```
void            yield(void);

//thread
int             thread_create(void (*)(void*), void*, void*);
int             thread_join(void);
int             thread_exit(void);
int             mysleep(void*, void*);
int             mywakeup(void*);
```

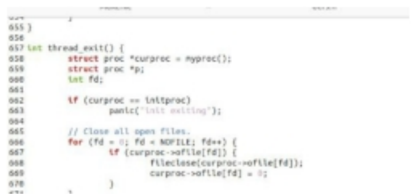
Proc.c
We have to include newly created header file lock.h *(#include "lock.h").*


```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "x86.h"
#include "proc.h"
#include "spinlock.h"
#include "lock.h"
```

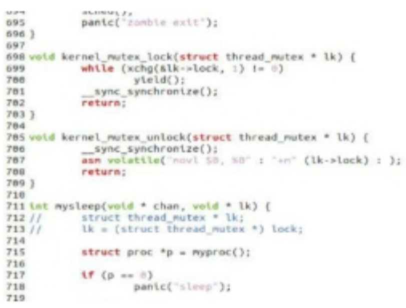
The code definitions for create, join and exit are added to this file. The create function sets up a new process with the given stack arguments, and the join function and exit function scans the process table looking for a zombie child and clears them out.

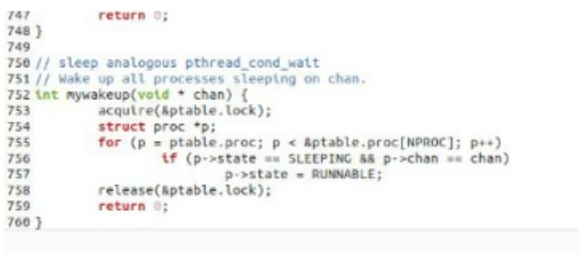

```
}



// thread
int thread_create(void (*fcn)(void *), void * arg, void * stack) {
        if ((uint) stack == 0) {
                return -1;
        }

        int i, pid;
        struct proc *np;

        struct proc *curproc = myproc();
        // Allocate process.
        if ((np = allocproc()) == 0)
                return -1;

```


```
        release(&ptable.lock);
        return pid;
}

int thread_join(void) {
        struct proc *p;
        int havekids, pid;
        struct proc *curproc = myproc();

        acquire(&ptable.lock);
        for (;;) {
```


```
}

int thread_exit() {
        struct proc *curproc = myproc();
        struct proc *p;
        int fd;

        if (curproc == initproc)
                panic("init exiting");

        // Close all open files.
        for (fd = 0; fd < NOFILE; fd++) {
                if (curproc->ofile[fd]) {
                        fileclose(curproc->ofile[fd]);
                        curproc->ofile[fd] = 0;
                }
```

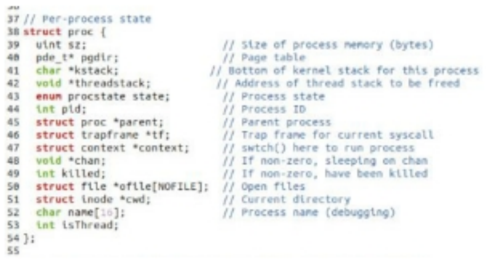
The code definitions for sleep and wakeup are also added in this file. The sleep


```
        panic("zombie exit");
}

void kernel_mutex_lock(struct thread_mutex * lk) {
        while (xchg(&lk->lock, 1) != 0)
                yield();
        __sync_synchronize();
        return;
}

void kernel_mutex_unlock(struct thread_mutex * lk) {
        __sync_synchronize();
        asm volatile("movl $0, %0" : "+m" (lk->lock) : );
        return;
}

int mysleep(void * chan, void * lk) {
//      struct thread_mutex * lk;
//      lk = (struct thread_mutex *) lock;

        struct proc *p = myproc();

        if (p == 0)
                panic("sleep");

```


```
        return 0;
}

// sleep analogous pthread_cond_wait
// Wake up all processes sleeping on chan.
int mywakeup(void * chan) {
        acquire(&ptable.lock);
        struct proc *p;
        for (p = ptable.proc; p < &ptable.proc[NPROC]; p++)
                if (p->state == SLEEPING && p->chan == chan)
                        p->state = RUNNABLE;
        release(&ptable.lock);
        return 0;
}
```

proc.h

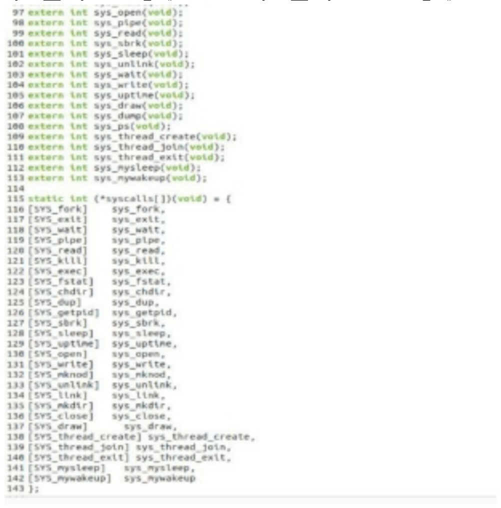
A new property is added to the process data structure to mark the address of the thread stack, titled *threadstack. And variable *isThread* is initialized.



```c
// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;              // Bottom of kernel stack for this process
  void *threadstack;          // Address of thread stack to be freed
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
  int isThread;
};
```

Syscall.c

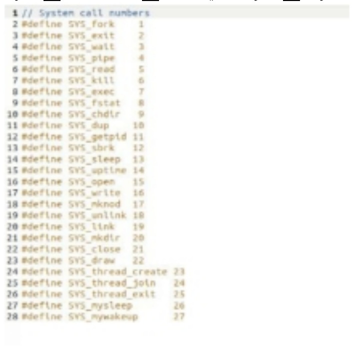
The declaration of functions *sys_thread_create()*, *sys_thread_join(), sys_thread_exit(), sys_mysleep() and sys_mywakeup()* are added to this file.



```c
extern int sys_open(void);
extern int sys_pipe(void);
extern int sys_read(void);
extern int sys_sbrk(void);
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_draw(void);
extern int sys_dump(void);
extern int sys_ps(void);
extern int sys_thread_create(void);
extern int sys_thread_join(void);
extern int sys_thread_exit(void);
extern int sys_mysleep(void);
extern int sys_mywakeup(void);

static int (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_kill]    sys_kill,
[SYS_exec]    sys_exec,
[SYS_fstat]   sys_fstat,
[SYS_chdir]   sys_chdir,
[SYS_dup]     sys_dup,
[SYS_getpid]  sys_getpid,
[SYS_sbrk]    sys_sbrk,
[SYS_sleep]   sys_sleep,
[SYS_uptime]  sys_uptime,
[SYS_open]    sys_open,
[SYS_write]   sys_write,
[SYS_mknod]   sys_mknod,
[SYS_unlink]  sys_unlink,
[SYS_link]    sys_link,
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
[SYS_draw]      sys_draw,
[SYS_thread_create] sys_thread_create,
[SYS_thread_join] sys_thread_join,
[SYS_thread_exit] sys_thread_exit,
[SYS_mysleep]   sys_mysleep,
[SYS_mywakeup]  sys_mywakeup
};
```

Syscall.h

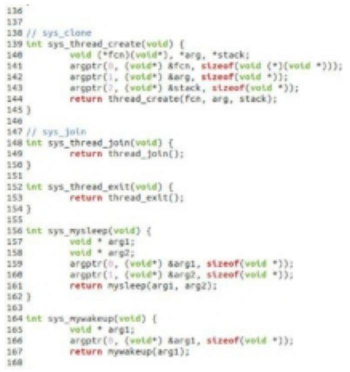
This system calls are assigned to the functions *sys_thread_create()*, *sys_thread_join(), sys_thread_exit(), sys_mysleep() and sys_mywakeup().*



```c
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
#define SYS_mknod  17
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
#define SYS_draw   22
#define SYS_thread_create 23
#define SYS_thread_join   24
#define SYS_thread_exit   25
#define SYS_mysleep       26
#define SYS_mywakeup      27
```

Sysproc.c

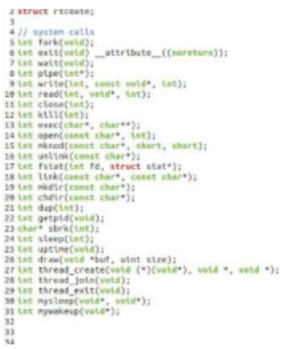
This file contains the definitions of the functions *sys_thread_create()*, *sys_thread_join(), sys_thread_exit(), sys_mysleep() and sys_mywakeup()* functions which call the definitions in proc.c.



```c


// sys_clone
int sys_thread_create(void) {
        void (*fcn)(void*), *arg, *stack;
        argptr(0, (void*) &fcn, sizeof(void (*)(void *)));
        argptr(1, (void*) &arg, sizeof(void *));
        argptr(2, (void*) &stack, sizeof(void *));
        return thread_create(fcn, arg, stack);
}

// sys_join
int sys_thread_join(void) {
        return thread_join();
}

int sys_thread_exit(void) {
        return thread_exit();
}

int sys_mysleep(void) {
        void * arg1;
        void * arg2;
        argptr(0, (void*) &arg1, sizeof(void *));
        argptr(1, (void*) &arg2, sizeof(void *));
        return mysleep(arg1, arg2);
}

int sys_mywakeup(void) {
        void * arg1;
        argptr(0, (void*) &arg1, sizeof(void *));
        return mywakeup(arg1);

```

user.h

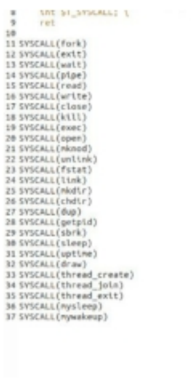
The declaration of new system calls *sys_thread_create()*, *sys_thread_join(), sys_thread_exit(),* sys_*mysleep()* and sys_*mywakeup()* are added to this file. Two new header files are created to define a lock.



```c
struct rtcdate;

// system calls
int fork(void);
int exit(void) __attribute__((noreturn));
int wait(void);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int draw(void *buf, uint size);
int thread_create(void (*)(void*), void *, void *);
int thread_join(void);
int thread_exit(void);
int mysleep(void*, void*);
int mywakeup(void*);



```

usys.S

The declaration of new functions *sys_thread_create()*, *sys_thread_join(),   sys_thread_exit(), sys_mysleep() and sys_mywakeup()* are added to this file.



```
    int $T_SYSCALL; \
    ret

SYSCALL(fork)
SYSCALL(exit)
SYSCALL(wait)
SYSCALL(pipe)
SYSCALL(read)
SYSCALL(write)
SYSCALL(close)
SYSCALL(kill)
SYSCALL(exec)
SYSCALL(open)
SYSCALL(mknod)
SYSCALL(unlink)
SYSCALL(fstat)
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(draw)
SYSCALL(thread_create)
SYSCALL(thread_join)
SYSCALL(thread_exit)
SYSCALL(mysleep)
SYSCALL(mywakeup)
```

thread.c

This user program tests the creation of two different threads and their usage of locks to ensure concurrency is working and thread safety is achieved via locks

TO COMPILE AND RUN:

These are the commands to run:

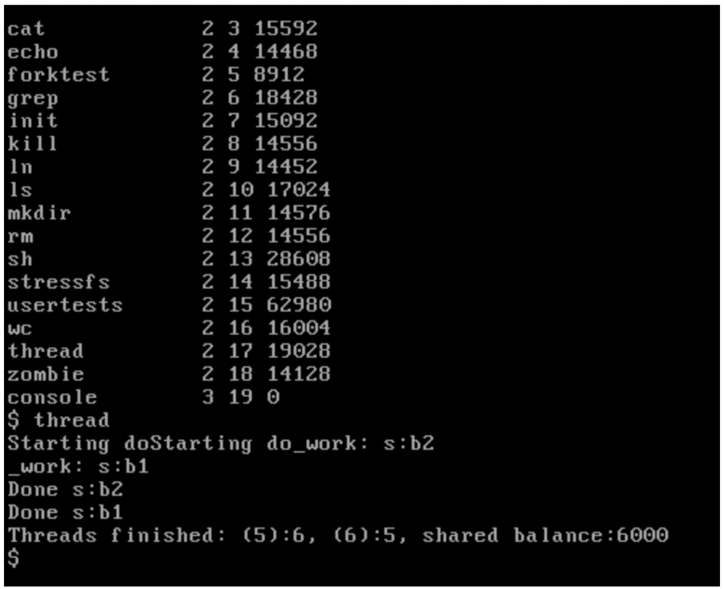
$make clean

$make

$make qemu

$ls

$thread



```
cat            2 3 15592
echo           2 4 14468
forktest       2 5 8912
grep           2 6 18428
init           2 7 15092
kill           2 8 14556
ln             2 9 14452
ls             2 10 17024
mkdir          2 11 14576
rm             2 12 14556
sh             2 13 28608
stressfs       2 14 15488
usertests      2 15 62980
wc             2 16 16004
thread         2 17 19028
zombie         2 18 14128
console        3 19 0
$ thread
Starting doStarting do_work: s:b2
_work: s:b1
Done s:b2
Done s:b1
Threads finished: (5):6, (6):5, shared balance:6000
$
```

b1, b2 are two different balances they start doing work at the same time. After finishing their addresses are shown. As they share balance, we created threads with locks. As we see the pid's of thread 1 for first balance and thread 2 for second balance and the shared balance is shown