Project 3 - Light Weight Process

Design

Thread □□

- User Stack : LWP(□□ Thread)□ □□□□ □□□□ □□□□□ □□□□ □□□□. □□□ Thread□ □□□□□ □□□ □□□□ □□□□□, □□□ □□□ □□□□ □□□ □□ page table□ □□□□. □□□, □□□ file descriptor□ □□□□.
- Thread Id : □□ Thread□ □□□□ id□ □□□□ □□□ id□ □□□ □□. □□, Thread□ □□□□ □□□□(`fork`□ □□ □□□)□ Main Thread□□ □□, Main Thread□ id□ 0□□.
- Main Thread : □□ Thread□ Main Thread□ □□□□□, □□□ Thread□ □□ □□□ Thread□ □□ Thread□□ □□□ □□. □□ □□ Thread□ Main Thread□ □□□ □□□□ id□ □□□, □□ □□ □□□□□ □□□□. □□ □□□ Thread□ □□ Thread□ □□□□ □□, □□ Thread□ □□ Thread□ Main Thread□ □□□□□.
- Exec & Exit : Thread□ `exec` □□□ □□ □□ □□□ □□□□□ □□□ □ □□□, `exit` □□□ □□ □□ □□□ □ □□. □□, □□ Thread(Main Thread □□)□ `exec`□□□ `exit`□□□ □□□□, □□ □□□□□ □□ Thread□ □□□□ (`exec`□ □□□ □□□ Thread□ □□□ □□□□□ □□□).

System Call □□

- Thread□ □□□ □□□□ □□□□ □□. Thread□ □□□□□ □□□ □□□□ □□□□□, □□□□□ □□□ □□□ □□ □□□ □ □□□ □□ □□□ □ □□.

Implement

Thread

- □□ □□□ □ □□□ `thread.h` □□ □□□ □□□□ □□.
- `thread_t` : Thread□ id□ □□□□ □□□□□.
- □□□□ □□□□ Thread□ id, Main Thread□ □□□, Thread□ □□□□ □□□ □ □□ □□□ □□□□.

```
// proc.h

// Per-process state
struct proc {
  ...
  int tid;                    // Thread ID
  struct proc *main;          // Main thread
  void* retval;               // Return value
};
```

Thread Creation

return	name	arguments	description
int	_thread_create	thread_t *thread, void *(*start_routine)(void *), void *arg	□□□ Thread□ □□□□.

- `thread`□ □□□ Thread□ id□ □□□ □□□□□.
- `start_routine`□ Thread□ □□□ □□□ □□□□□.
- `arg`□ `start_routine`□ □□□ □□□□.

1. `myproc` □□□ □□ □□ □□ □□ □□□□□□ □□□□.
2. `allocproc` □□□ □□ □□□ Thread□ □□□□. (□□□□□ □□□□□ □□□ □□□ Thread□ □□□□.)
3. □□□ Thread□ `pid`, `tid`, `main`, `pgdir`, `parent`, `tf`□ □□□□.
4. □□□ Thread□ □□ □□□(□□)□ □□ □□ □□□□□ □□□□□ □□□□, Thread `sz`□ □□□□.
5. □□□ Thread□ □□□ `arg`□ □□, `tf`□ `eip`□ `start_routing`□□ □□□□ `esp`□ □□□□.
6. □□□ Thread□ `sz`□ □□□□, □□ Thread□ `sz`□ □□□□.
7. □□ □□□□□□□ □□□□.
8. Thread□ □□□ □□□□, □□□ Thread□ id□ `thread`□ □□□□.
9. □□□ Thread□ `state`□ `RUNNABLE`□ □□□ □□□□ □□ □□□.

Thread Termination

return	name	arguments	description
void	_thread_exit	void *retval	□□ Thread□ □□□□.

- `retval`□ Thread□ `start_routine`□ □□ □ □□ □□□□□.

1. `myproc` □□□ □□ □□ □□ □□ □□□□□□ □□□□.
2. □□ □□ Thread□ Main Thread□□, □□ Thread□ □□□□ □□ □□ □□□ □□□.
3. Thread□ □□□□ □□□□□□ □□□.
4. □□□ Thread□ □□ □□□□□□ `initproc`□ □□□□ □□□□.
5. □□ □□ Thread□ Main Thread□ □□□□ Main Thread□ □□□, □□□ □□ □□□□□□ □□□.
6. □□ Thread□ `retval`□ □□□□, `state`□ `ZOMBIE`□ □□□□.
7. □□□□□ □□□□.

Thread Waiting

return	name	arguments	description
int	_thread_join	thread_t thread, void **retval	□□ Thread□ □□□ □□□ □□□□.

- `thread`□ □□□ Thread□ id□□.
- `retval`□ □□□ Thread□ □□□□ □□□ □□□□□.

1. `myproc` □□□ □□ □□ □□ □□ □□□□□□ □□□□.
2. □□□ Thread□ □□□.
3. □□ □□□ Thread□ □□□□ □□□□, □□ Thread□ `SLEEPING` □□□ □□□□, □□□ Thread□ □□□.
4. □□□ Thread□ □□□□□□, `retval`□ □□□□ □□□□, □□□ Thread□ □□□□.

`proc.c`

`initproc`

- `thread.c`□□ □□□□ □□ `initproc` □□□ `extern`□□ □□□□.

wakeup1

- thread.c□□ □□□□ □□ initproc □□□ extern□□ □□□□.

allocproc

- thread.c□□ □□□□ □□ initproc □□□ extern□□ □□□□.
- Thread□□ □□□□ proc □□□□ □□□ □□□□□ □□□ □□□□□.

```
struct proc *
allocproc(void)
{
  ...
  found:
    p->state = EMBRYO;
    p->pid = nextpid++;

    // EDITED : Thread
    p->tid = 0;
    p->main = p;
  ...
}
```

growproc

- □□ Thread□ □□□ □□□ □□□ □□□ □, proc□ sz□ □□□□□ □□ □□□ □□□ □□□ □□ □□□ critical section□□ □□□□.

```
int growproc(int n)
{
  ...
  acquire(&ptable.lock);
  ...
  release(&ptable.lock);
  ...
}
```

exit

- Thread(Main Thread □□)□ exit □□□ □□ □□□□ □, □□ Thread□ □□□□□ □□□□.

```
void exit(void){
  ...
  for (struct proc *p = ptable.proc; p < &ptable.proc[NPROC]; p++)
  {
    if (p->pid == curproc->pid)
    {
      p->state = ZOMBIE;
      p->cwd = 0;

      for(int fd = 0; fd < NOFILE; fd++){
        if(p->ofile[fd]){
          p->ofile[fd] = 0;
        }
      }
    }
  }
  ...
}
```

`wait`

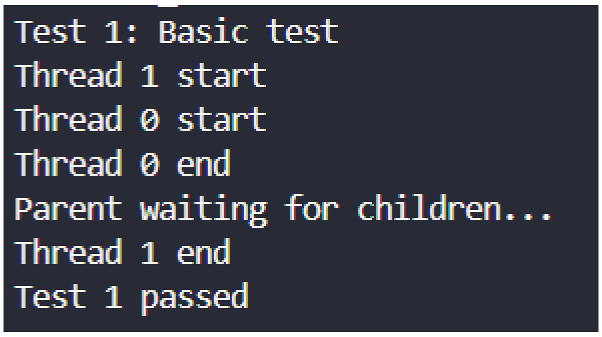
- `ZOMBIE` □□□ □□□□(Thread □□)□ □□ □□□ □□□□□□□ □□□□.

```
int wait(void){
  ...
  for (struct proc *p = ptable.proc; p < &ptable.proc[NPROC]; p++)
  {
    if (p->state == ZOMBIE && p->parent == curproc)
    {
      p->state = UNUSED;
      p->pid = 0;
      p->parent = 0;
      p->name[0] = 0;
      p->killed = 0;
      p->main = 0;
      p->tid = 0;
      p->retval = 0;
      for(; p < &ptable.proc[NPROC]; p++)
      {
        if (p->parent != curproc)
          continue;
        if(p->state == ZOMBIE){
          kfree(p->kstack);
          if (p->tid == 0)
          {
            freevm(p->pgdir);
          }
          p->kstack = 0;
          p->pid = 0;
          p->parent = 0;
          p->name[0] = 0;
          p->killed = 0;
          p->state = UNUSED;
        }
      }
      release(&ptable.lock);
      return pid;
    }
    }
    ...
}
```

Result

`thread_test`

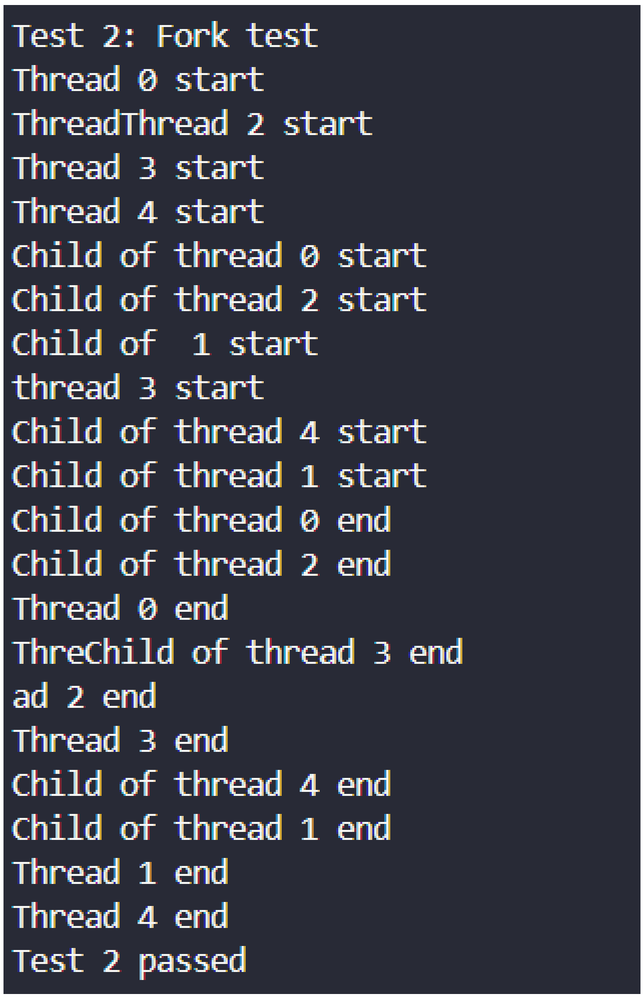
Test 1



```
Test 1: Basic test
Thread 1 start
Thread 0 start
Thread 0 end
Parent waiting for children...
Thread 1 end
Test 1 passed
```

- Thread 0□ 1□ □□□□, Thread 0□ □□□□ 1□ □□ Thread 1□ □□□□.

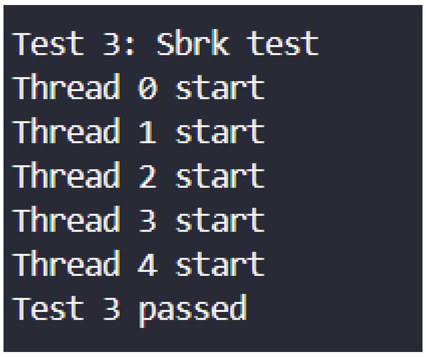
Test 2



```
Test 2: Fork test
Thread 0 start
ThreadThread 2 start
Thread 3 start
Thread 4 start
Child of thread 0 start
Child of thread 2 start
Child of  1 start
thread 3 start
Child of thread 4 start
Child of thread 1 start
Child of thread 0 end
Child of thread 2 end
Thread 0 end
ThreChild of thread 3 end
ad 2 end
Thread 3 end
Child of thread 4 end
Child of thread 1 end
Thread 1 end
Thread 4 end
Test 2 passed
```

- Thread□ `fork`□ □□ □□ □□□□□□ □□□□, □□ □□□□(`int status`)□ □□□□□□ □□□□.

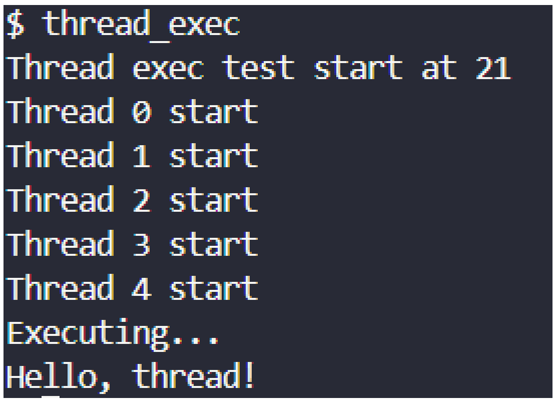
Test 3



```
Test 3: Sbrk test
Thread 0 start
Thread 1 start
Thread 2 start
Thread 3 start
Thread 4 start
Test 3 passed
```

- □□□ Thread□ □□□□□ □□ □□ □, □□□ □□ □□ □□□□□□ □□□□□□ □□□□.

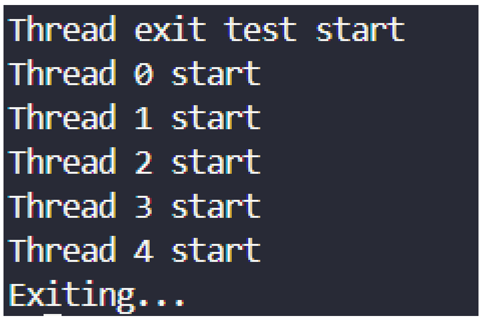
`thread_exec`



```
$ thread_exec
Thread exec test start at 21
Thread 0 start
Thread 1 start
Thread 2 start
Thread 3 start
Thread 4 start
Executing...
Hello, thread!
```

- Thread□ `exec` □□□ □□ □□ □□□ □□□□□□ □□□□□□ □□□□.

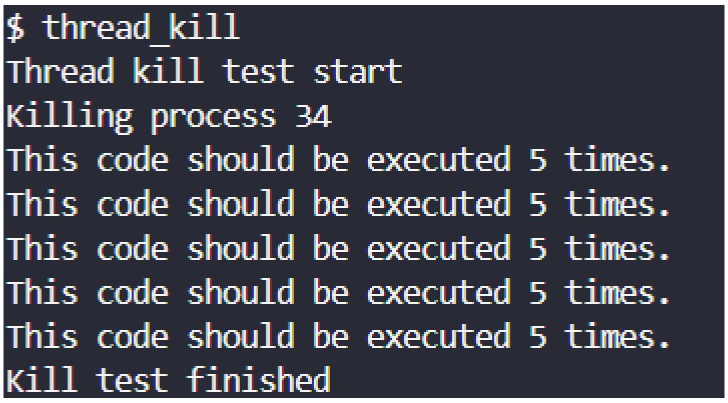
`thread_exit`



```
Thread exit test start
Thread 0 start
Thread 1 start
Thread 2 start
Thread 3 start
Thread 4 start
Exiting...
```

- Thread□ `exit` □□□ □□ □□ □□□□□ □, □□ Thread□ □□□□□□ □□□□.

`thread_kill`



```
$ thread_kill
Thread kill test start
Killing process 34
This code should be executed 5 times.
This code should be executed 5 times.
This code should be executed 5 times.
This code should be executed 5 times.
This code should be executed 5 times.
Kill test finished
```

- Thread□ `kill` □□□ □□ □□ □□□ □, □□ Thread□ □□□□□ □□□□.

Trouble Shooting

`_thread_create`

- Thread□ □□□□□ □□□ □□□□ □□□□□, □□ Thread□ □□□ □ □□□ □□□□□ □□□□ □□□ `allocproc`□□□ □□□□. □□ □ `proc` □□□ □□□ Thread□□ □□□□ □□□□ □□□ □□□ □□□□□ □□□. □□□ `allocproc` □□□ □□□ □□□ `nextpid` □□ □□□□ □□□, □□□□ □□ `pid`□ □□□□.
- Thread□ Stack□ □□□ □, 2 x PGSIZE□ □□□□□ □□. □□ Guard Page□ □□□□ □□□□. □□ `clearpteu` □□□ □□ Guard Page□ □□□□□.

`_thread_exit`

- □□ Main Thread□ □ □□□ □□□□, □□ Thread□ □□□□ □□ □□□ □□□ □□. □□ □□ `kill` □□□ □□□ □□□ □□ Thread □ `killed`□ 1□ □□□□, trap □□□□ `exit` □□□ □□□□□ □□.

`_thread_join`

- □□ □□□ `wait` □□□ □□□□.

`ofile` & `cwd`

- Thread□ □□□□□ □□□ □□ □□□□□□ □□□□□, `ofile`□ `cwd`□ □□□□.
- □□□ □□□ Thread□ `exit`□□□ □□□ □□ Thread□ □□□□□, `exec` □□□ □□□ □ □□□ Thread□ □□ □□□, `ofile`□ `cwd`□ □□□ □ `filedup`□□ `idup`□ □□□□ □□□. □ □□ □□□□□ □□□□, □□□□ `ref` □□□ □□□□□ □□□. □□ Thread□ □□□ □, `fileclose` □□□ `iput` □□□ □ □□□ □□□□.

Locking

- □□□□□ c□ inline assembly□ □□□ atomic swap□ □□□ □ □□.
- x86□□□ `xchg` □□□□ □□□ atomic swap□ □□□ □ □□, arm64□□□ `ldxr`□ `stxr` □□□□ □□□ atomic swap□ □□ □ □ □□.
- atomic swap□ □□□ `compare_and_swap` □□□ □□□□, □ □□□ □□ spinlock□ □□□ □ □□.