XV6 KERNEL THREADS

AIM: To add thread support to kernel xv6. The thread support is added by means of 2 system calls- clone and join while a thread library interface is defined in the kthreads.h file- consisting of 2 thread functions- thread_create() & thread_join() besides 3 lock functions- init(), acquire(), release().

MODIFICATIONS:

Part A: Kernel Code

To add a system call, we modify the following files:
The 2 system calls are *int clone(void(*fcn)(void*), void arg*, void *stack) & int join(int pid)*
1. syscall.h :add the 2 system call definitions here in the format: #define SYS_clone 22

2. syscall.c :add the 2 system call definitions here in the format: [SYS_clone] sys_clone,
and extern int sys_clone(void);

3. sysproc.c: add the 2 system call definitions here in the format: int clone {…} . The arguments are collected and sent to the system call defined in proc.c.

4. usys.S: add the 2 system call definitions here in the format: SYSCALL(clone)

5. user.h: add the 2 system call definitions here in the format: int clone(void(*fcn)(void *), void *arg, void *stack);

6. defs.h: add the 2 system call definitions here in the format: clone(void(*)(void*), void*, void*);

7. proc.c: The major part of the coding is in this file.

In the clone function:
The input to the clone function are: function to execute, argument for the function, stack pointer. We need to clone the calling process. Similar to a fork, the cloned thread has its name, part of its trap frame (the eax context is set to 0 to force fork return 0), file descriptor and most importantly the page directory and size. The allocation as a new process is done by calling allocproc() function, then unique user and kernel stacks are set. The pid obtained from the allocproc is returned to the calling function. The eip is set to the function passed as a parameter.

In the join function:
The join function borrows logic from the wait function. The join function takes as input the pid of the child process. The parent process is identified and made to wait until the child process exits. When the child process exits, its state is set to ‘zombie’ which is used as a cue for the parent process to finish waiting. Care is taken to not deallocate resources of the parent process (and other threads) since unlike in the wait function, shared resources must not be removed. On success, it outputs a 0 and -1 for failure.

Part B: Thread Library

The thread library, as defined in kthreads.h has 2 intermediate functions for the clone and join functions. The thread_create(routine, arg) function accepts a call from a user process and calls the clone function after. Its essential role is in allocating space (using malloc) for the stack and passing it onto clone. It returns a struct kthread which has 2 attributes : the pid of the cloned function and the stack pointer.

The thread_join takes in a struct k_thread input and passes the pid attribute to the syscall join. On a successful return, it frees the stack and returns a success code.

The locks are defined for critical sections- to prevent race conditions form taking place. The spinlock from the kernel is replicated as necessary. The three functions to provide this functionality are the init()→ initialize a lock, acquire() → get the lock if available and release()→ give up lock after use of shared resource is terminated. All 3 functions accept the kthread_t struct type as input.

TESTING:
The 3 files given for testing were executed successfully. The results are as follows:

1. simple_threads.c

Terminal Output:
```
parent: array[0] = 400
Test1: array[0] = 400
array[10] = 1500
pid = 4
```

2. prod_cons.c

Terminal Output:
```
consumer 0 consumed: 1000000
consumer 1 consumed: 1000000
consumer 2 consumed: 1000000
consumer 3 consumed: 1000000
Remaining products: 6000000
Things made: 10000000
Test passed!
it works!
```

3. testkthreads.c

Terminal Output:
```
consumer 0 consumed: 3000000
consumer 1 consumed: 3000000
Remaining products: 4000000
Things made: 10000000
Test passed!
```