Brian Duenas

CSE 460

Lab 6 – Thread Programming and Semaphore Part I and XV6 Scheduling

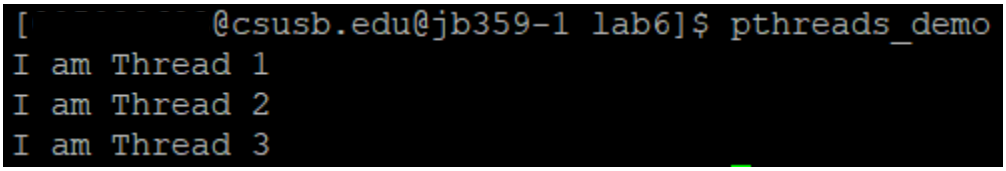
I. Introduction to Thread Programming

Q: Modify the programs so that they run 3 threads and each thread runs a different function, displaying a different message.

Code for pthread:

```
#include <pthread.h>
#include <stdio.h>

using namespace std;

//Thread one
void *runner(void *data)
{
        char *tname = (char *)data;

        printf("I am %s\n", tname);

        pthread_exit(0);
}
//Thread two
void *runner_two(void *data)
{
        char *tname = (char *)data;

        printf("I am %s\n", tname);

        pthread_exit(0);
}
//Thread three
void *runner_three(void *data)
{
        char *tname = (char *)data;

        printf("I am %s\n", tname);

        pthread_exit(0);
}

int main()
{
        pthread_t id1, id2, id3;           //thread identifiers
        pthread_attr_t attr1, attr2, attr3;  //set of thread attributes
        char *tnames[3] = { "Thread 1", "Thread 2", "Thread 3" }; //names of threads


        pthread_attr_init(&attr1);        //get the default attributes
        pthread_attr_init(&attr2);
        pthread_attr_init(&attr3);

        //create the threads
        pthread_create(&id1, &attr1, runner, tnames[0]);
        pthread_create(&id2, &attr2, runner_two, tnames[1]);
        pthread_create(&id3, &attr3, runner_three, tnames[2]);

        //wait for the threads to exit
        pthread_join(id1, NULL);
        pthread_join(id2, NULL);
        pthread_join(id3, NULL);

        return 0;
}
```

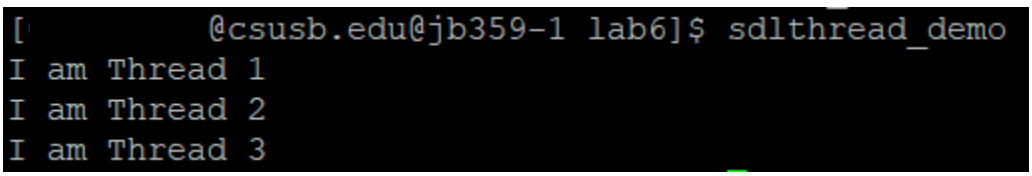
Output:



Code for SDL:

```
#include <SDL/SDL.h>
#include <SDL/SDL_thread.h>
#include <stdio.h>

using namespace std;

//The thread one
int runner(void *data)
{
        char *tname = (char *)data;

        printf("I am %s\n", tname);
        return 0;
}
//The thread two
int runner_two(void *data)
{
        char *tname = (char *)data;

        printf("I am %s\n", tname);
        return 0;
}
//The thread three
int runner_three(void *data)
{
        char *tname = (char *)data;

        printf("I am %s\n", tname);
        return 0;
}

int main()
{
        SDL_Thread *id1, *id2, *id3;                //thread identifiers
        char *tnames[3] = { "Thread 1", "Thread 2", "Thread 3" }; //names of threads

                //create the threads
        id1 = SDL_CreateThread(runner, tnames[0]);
        id2 = SDL_CreateThread(runner_two, tnames[1]);
        id3 = SDL_CreateThread(runner_three, tnames[2]);

        //wait for the threads to exit
        SDL_WaitThread(id1, NULL);
        SDL_WaitThread(id2, NULL);
        SDL_WaitThread(id3, NULL);
        return 0;
}
```

Output:



II. Using Semaphores

Q: Run *./sema1* do you see any printout of letters 'e' and 'l'? Why? Also run *./sema1 a do you see any printout of letters 'E' and 'l'? Why?*

Outputs:

```
[        @csusb.edu@jb359-1 lab6]$ sema1
elelelelelelelelelel
5696 finished!
```

```
[        @csusb.edu@jb359-1 lab6]$ sema1 a
ELELELELELELELELELEL
5695 finished!
```

A: The program prints an "E" when entering and an "L" when leaving a critical section if the program is invoked with one or more parameters, otherwise it prints an "e" and an "l" respectively.

III. XV6 Scheduling

Q: Modify the *proc.c* file to print the pid's of xv6 and add a *foo.c* dummy file.

Code (partial) for *proc.h*:

```
// 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
       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)

                                                        // add timestamps and others
       uint createTime;             // process creation time
       int  sleepTime;              // process sleeping time
       int  readyTime;              // process ready (RUNNABLE) time
       int  runTime;                // process running time
       int priority;                // process priority
       int tickcounter;
       char dum[8];
};
```

Code (partial) for *proc.c*:

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

	for (;;) {
		// Enable interrupts on this processor.
		sti();

		// 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;

			// Switch to chosen process.  It is the process's job
			// to release ptable.lock and then reacquire it
			// before jumping back to us.
			c->proc = p;
			switchuvm(p);
			p->state = RUNNING;

			cprintf("Process %s with pid %d running with createTime %d\n",
			p->name, p->pid, p->createTime);

			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);
	}
}
```

Output from *proc.c*:

```
xv6...
cpu1: starting 1
cpu0: starting 0
Process initcode with pid 1 running
Process initcode with pid 1 running
sb: size 1000 nblocks 941 ninodes 20
Process initcode with pid 1 running
Process initcode with pid 1 running
Process initcode with pid 1 running
Process initcode with pid 1 running
Process initcode with pid 1 running
Process initcode with pid 1 running
Process initcode with pid 1 running
Process initcode with pid 1 running
Process initcode with pid 1 running
Process initcode with pid 1 running
Process initcode with pid 1 running
Process initcode with pid 1 running
```

Output for *foo.c*:

```
$ foo 4
Process sh with pid 2 running
Process sh with pid 2 running
Process sh with pid 4 running
Process sh with pid 4 running
Process sh with pid 4 running
Process sh with pid 4 running
Process sh with pid 4 running
Process sh with pid 4 running
Process sh with pid 4 running
Process sh with pid 4 running
Process sh with pid 4 running
Process sh with pid 4 running
Process sh with pid 4 running
Process sh with pid 4 running
Process sh with pid 4 running
Process sh with pid 4 running
Process sh with pid 4 running
Process sh with pid 4 running
Process sh with pid 4 running
Process sh with pid 4 running
Process sh with pid 4 running
Process sh with pid 4 running
Parent 4 creating child  5
```

Q: Add in the file proc.h the struct proc> the timestamps and the priority

Output:

```
$ foo 4
Process sh with pid 2 running with createTime 17
Process sh with pid 14 running with createTime 5220
Parent 14 creating child  15
Process foo with pid 15 running with createTime 5220
Process foo with pid 15 running with createTime 5220
Process foo with pid 15 running with createTime 5220
Child 15 created
Process foo with pid 14 running with createTime 5220
Process foo with pid 14 running with createTime 5220
Process foo with pid 15 running with createTime 5220
Process foo with pid 14 running with createTime 5220
Parent 14 creating child  16
Process foo with pid 16 running with createTime 5224
Process foo with pid 15 running with createTime 5220
Process foo with pid 16 running with createTime 5224
Process foo with pid 15 running with createTime 5220
Process foo with pid 16 running with createTime 5224
Process foo with pid 14 running with createTime 5220
Process foo with pid 16 running with createTime 5224
Child 16 created
Process foo with pid 14 running with createTime 5220
Process foo with pid 15 running with createTime 5220
Process foo with pid 14 running with createTime 5220
Process foo with pid 15 running with createTime 5220
Process foo with pid 14 running with createTime 5220
Process foo with pid 16 running with createTime 5224
Process foo with pid 14 running with createTime 5220
Parent 14 creating cProcess foo with pid 16 running with createTime 5224
Process foo with pid 17 running with createTime 5231
Process foo with pid 15 running with createTime 5220
Process foo with pid 16 running with createTime 5224
Process foo with pid 14 running with createTime 5220
Process foo with pid 15 running with createTime 5220
Process foo with pid 17 running with createTime 5231
Process foo with pid 14 running with createTime 5220
Process foo with pid 16 running with createTime 5224
Process foo with pid 17 running with createTime 5231
Process foo with pid 14 running with createTime 5220
Process foo with pid 15 running with createTime 5220
Process foo with pid 17 running with createTime 5231
Process foo with pid 14 running with createTime 5220
```

20/20 Total Points