LAB_6

Kyle Baxter

2/16/17

1. PThread Example:

a. Here is the Modified program that is going to run 3 threads:

Modified code:

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

using namespace std;

//The thread
void *runner ( 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
            //get the default attributes
              pthread_attr_init ( &attr1 );
                pthread_attr_init ( &attr2 );
                pthread_attr_init ( &attr3);
                  //create the threads
                    pthread_create ( &id1, &attr1, runner, tnames[0] );
                      pthread_create ( &id2, &attr2, runner, tnames[1] );
                      pthread_create ( &id3, &attr3, runner, tnames [2] );
                       //wait for the threads to exit
                          pthread_join ( id1, NULL );
                            pthread_join ( id2, NULL );
                              pthread_join (id3, NULL);
                              return 0;
                              }
```

OUTPUT:

```
[004603663@jb358-2 LAB_6]$ ./pthreads
I am Thread 1
I am Thread 2
I am Thread 3
[004603663@jb358-2 LAB_6]$
```

b. Modification of the SDL Thread program

Code:

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

using namespace std;

//The thread
int runner ( 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, tnames[1] );
               id3 = SDL_CreateThread ( runner, tnames [2] );

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

                      return 0;
                      }
~
~
```

OUTPUT:

```
[004603663@jb358-2 LAB_6]$ ./sdlthread
I am Thread 1
I am Thread 2
I am Thread 3
[004603663@jb358-2 LAB_6]$
```

2. UNIX Semaphore Facilitites

a. When it comes to the running of the program running in the background, It will still input the ‘ie’ consecutively. The reason for this is that when the command is running in the background, the value is less than 1, therefor will continue to output the lowercase form.

OUTPUT:

```
[004603663@jb358-2 LAB_6]$ ./semal &
[1] 11649
[004603663@jb358-2 LAB_6]$ elelelelelelelelelel
11649 finished!
^C
[1]+  Done                    ./semal
```

a. When it comes to running the program with the ‘a’ command, it the value will be greater than 0 and then will input the uppercase values of the program.

OUTPUT:

```
[004603663@jb358-2 LAB_6]$ ./semal a
ELELELELELELELELELEL
12155 finished!
[004603663@jb358-2 LAB_6]$
```

b. Here is the modified code of the program that will take in a command of 1 or 0 and if 0 will be stuck waiting or 1 for the rest of the code to execute

Code:

```
//sema1.cpp
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <iostream>
#include <stdio.h>

using namespace std;

static int sem_id;                  //semaphore id

#if defined(__GNU_LIBRARY__) && !defined(_SEM_SEMUN_UNDEFINED)
  /* union semun is defined by including <sys/sem.h> */
#else
  /* according to X/OPEN we have to define it ourselves */
  union semun {
    int val;                  /* value for SETVAL */
    struct semid_ds *buf;     /* buffer for IPC_STAT, IPC_SET */
    unsigned short *array;    /* array for GETALL, SETALL */
                              /* Linux specific part: */
    struct seminfo *__buf;    /* buffer for IPC_INFO */
  };
```

```
#endif

//initializes semaphore using SETVAL
static int set_semvalue ( int val )
{
  union semun sem_union;// sem_union;

  sem_union.val = val;
  if ( semctl ( sem_id, 0, SETVAL, sem_union ) == -1 ) return ( 0 );
  return 1;
}

//delete semaphore
static int del_semvalue ()
{
  union semun sem_union;// sem_union;

  sem_union.val = 1;
  if ( semctl ( sem_id, 0, IPC_RMID, sem_union ) == -1 ) return ( 0 );
  return 1;
}

static int SEM_DOWN ()
{
  struct sembuf b;

  b.sem_num = 0;
  b.sem_op = -1;         //P(), i.e. down()
  b.sem_flg = SEM_UNDO;
  if ( semop ( sem_id, &b, 1 ) == -1 ) {
    cout << "Semaphore DOWN() failed!" << endl;
    return 0;
  }

  return 1;
}

static int SEM_UP()
{
   struct sembuf b;

  b.sem_num = 0;
  b.sem_op =  1;        //V(), i.e. UP()
  b.sem_flg = SEM_UNDO;
  if ( semop ( sem_id, &b, 1 ) == -1 ) {
    cout << "Semaphore UP() failed!" << endl;
    return 0;
  }
  return 1;
```

```
}
int main ( int argc, char *argv[] )
{
  int i, pause_time;
  char ce = 'e', cl = 'l';

  srand ( ( unsigned int ) getpid() );    //seed RNG with process id

  sem_id  = semget ( (key_t) 1234, 1, 0666 | IPC_CREAT );

  //modified area
  int k;
  k = atoi(argv[1]);


  if ( k > 0 ) {
    if ( !set_semvalue( 1 ) ) {              //process can enter CS
      cout << "Semaphore initialized failed!" << endl;
      exit ( EXIT_FAILURE );
    }
    if ( k == 1 ) {
      ce = 'E';
      cl = 'L';
    }
    sleep ( 1 );
  } else {
    if ( !set_semvalue( 0 ) ) {              //process will be blocked initially
      cout << "Semaphore initialized failed!" << endl;
      exit ( EXIT_FAILURE );
    }
    sleep ( 1 );
  }

  //enter and leave critical section 10 times
  for ( i = 0; i < 10; i++ ){
    if ( !SEM_DOWN () ) exit ( EXIT_FAILURE );
    cout << ce; fflush ( stdout );          //entering critical section
    pause_time = rand() % 3;                //simulate critical section
    sleep ( pause_time );
    cout << cl; fflush ( stdout );          //leaving critical section
    if ( !SEM_UP() ) exit ( EXIT_FAILURE );          //signal other waiting process
    pause_time = rand() % 2;
    sleep ( pause_time );
  }
  cout << endl << getpid() << " finished!" << endl;
  if ( argc > 0 ) {
    sleep ( 2 );
    del_semvalue ();
  }
```

```
  exit ( EXIT_SUCCESS );
}
```

OUTPUT:
[004603663@jb358-2 LAB_6]$ ./semal 1
ELELELELELELELELELEL
13052 finished!
[004603663@jb358-2 LAB_6]$ ./semal 0
^C
[004603663@jb358-2 LAB_6]$

3. XV6 Scheduling

a. Here is the output of the modified proc.c code running with the init with pid runing
OUTPUT:
rocess init with pid 1 runing
Process init with pid 1 runing
Process init with pid 1 runing
Process init with pid 1 runing
Process init with pid 1 runing
Process init with pid 1 runing
Process init with pid 1 runing
Process init with pid 1 runing
Process init with pid 1 runing
Process init with pid 1 runing
init: starting sh
Process init with pid 2 runing
Process init with pid 2 runing
Process init with pid 2 runing
Process init with pid 2 runing
Process init with pid 2 runing
Process init with pid 2 runing
Process init with pid 2 runing
Process init with pid 2 runing
Process init with pid 2 runing
Process init with pid 2 runing

b. on the next step, the dummy program is being introduced.
The commands have been added and modified
OUTPUT:
Process init with pid 2 runing
Process init with pid 2 runing
Process init with pid 2 runing
Process init with pid 2 runing
Process init with pid 2 runing

//Continues to g with pid 2 until
process sh with pid 3 runing
Process sh with pid 3 runing

Process sh with pid 3 runing
Process sh with pid 3 runing
Parent 3 creating child 4
Process foo with pid 4 runing
Child 4 created
Process foo with pid 4 runing
Process foo with pid 4 runing
Process foo with pid 4 runing

//continues with pid 4 until done

Process foo with pid 4 runing
Process foo with pid 4 runing
Process foo with pid 3 runing
Process foo with pid 3 runing
Parent 3 creating child 5
Process foo with pid 5 runing
Child 5 created
Process foo with pid 5 runing
Process foo with pid 5 runing
Process foo with pid 5 runing
Process foo with pid 5 runing
Process foo with pid 5 runing

//Continues with pid 5 until done

Process foo with pid 3 runing
Process foo with pid 3 runing
Process foo with pid 6 runing
Child 6 created
Process foo with pid 3 runing
Process foo with pid 3 runing
Process foo with pid 3 runing
Parent 3 creating child 6
Process foo with pid 6 runing
Process foo with pid 6 runing
Process foo with pid 6 runing

This pattern continues till it ends on PID 6 and then the Parent (through a fork()) creates another child with a PID of 7 and then continues the process until the child is done and then goes to the parent. There is the consideration that the parent does not wait to check if there are other children before exiting.

b. Modified Code of the Proc.c file:

```
static struct proc*
allocproc(void)
{
  struct proc *p;
  char *sp;
```

```
  acquire(&ptable.lock);

  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
    if(p->state == UNUSED)
      goto found;

  release(&ptable.lock);
  return 0;

found:
  p->state = EMBRYO;
  p->pid = nextpid++;
  p->createTime = ticks;
  p->readyTime = 0;
  p->runTime = 0;
  p->sleepTime = 0;

  release(&ptable.lock);

  // Allocate kernel stack.
  if((p->kstack = kalloc()) == 0){
    p->state = UNUSED;
    return 0;
  }
  sp = p->kstack + KSTACKSIZE;
```

OUTPUT (with the new changes):

```
          cpu1: starting
cpu0: starting
Process initcode with pid 1 running with createTime 0
,Process initcode with pid 1 running with createTime 0
,sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
Process initcode with pid 1 running with createTime 0
,Process initcode with pid 1 running with createTime 0
,Process initcode with pid 1 running with createTime 0
,Process initcode with pid 1 running with createTime 0
,Process initcode with pid 1 running with createTime 0

//This is continuing until

,Process initcode with pid 1 running with createTime 0
,Process initcode with pid 1 running with createTime 0
,Process init with pid 1 running with createTime 0
,init: starting sh
Process init with pid 1 running with createTime 0
,Process init with pid 2 running with createTime 75
,Process init with pid 2 running with createTime 75
,Process init with pid 2 running with createTime 75
```

//the program holds and then foo 4 is then initaited

$ foo 4
Process sh with pid 2 running with createTime 75
,Process sh with pid 2 running with createTime 75
,Process sh with pid 5 running with createTime 15061
,Process sh with pid 5 running with createTime 15061
,Process foo with pid 5 running with createTime 15061
,Parent 5 creating child 6
Process foo with pid 6 running with createTime 15062
,Child 6 created

//upon the same process of creating the same amount of children in the previous program, its not showing the timestamp in seconds that its running,

Process foo with pid 6 running with createTime 15062
,Process foo with pid 6 running with createTime 15062
,Process foo with pid 6 running with createTime 15062
,Process foo with pid 6 running with createTime 15062
,Process foo with pid 5 running with createTime 15061
,Parent 5 creating child 7
Process foo with pid 7 running with createTime 15129
,ChildProcess foo with pid 7 running with createTime 15129
, 7 created
Process foo with pid 7 running with createTime 15129
,Process foo with pid 7 running with createTime 15129
,Process foo with pid 7 running with createTime 15129
,Process foo with pid 7 running with createTime 15129

//continuing
rocess foo with pid 9 running with createTime 15269
,Process foo with pid 9 running with createTime 15269
,Process foo with pid 9 running with createTime 15269
,Process foo with pid 5 running with createTime 15061
,Process foo with pid 5 running with createTime 15061
,Process foo with pid 5 running with createTime 15061
,Process foo with pid 5 running with createTime 15061
,Process foo with pid 5 running with createT

continuing
,Process foo with pid 5 running with createTime 15061
,Process foo with pid 5 running with createTime 15061
,Process foo with pid 5 running with createTime 15061
,Process sh with pid 2 running with createTime 75

NOTE: There is a larger number within running the foo program because I ran foo once without the number and then after that, I ran foo 4 with then continued from there. It still works but just with a larger number

Evaluation:
From this Lab, all of the code, input, and output has been pasted and created with the correct forms. I was enjoying this lab because I had time to go though the process and understand what is going on while getting some guidance from some fellow classmates. Other than that, solid lab.

SCORE: 20/20